The Road to El Diablo: Towards Secure, High Performance Blockchains

Christopher James Natoli

A thesis submitted to fulfil requirements for the degree of

Doctor of Philosophy

School of Computer Science

Faculty of Engineering

The University of Sydney

June 2021

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.

Christopher James Natoli
June 28, 2021

Authorship Attribution

The results presented in this dissertation were published in the following publications and can be found in the relevant chapters:

(1) Chapter 2 of this thesis contains material under submission and available as a preprint [1]. I was the primary author who proposed the concept and performed the majority of the analysis.

[1] Christopher Natoli, Jiangshan Yu, Paulo Verissimo, Vincent Gramoli, "Deconstructing Blockchains: A Comprehensive Survey on Consensus, Membership and Structure", In: *CoRR abs/1908.08316 (2019). arXiv: 1908.08316.* url: http://arxiv.org/abs/1908.08316.

(2) Chapter 3 of this thesis contains material published in [2] and [3] with an accompanying technical report in [4]. In [2], I was the primary author who proposed the concept, analyses and experimentation. In [3], I was the primary author who proposed the core concept, performed experimental evaluation and assisted with analysis.

[2] Christopher Natoli and Vincent Gramoli. "The Blockchain Anomaly". In: *15th IEEE International Symposium on Network Computing and Applications, NCA 2016, Cambridge, Boston, MA, USA, October 31- November 2, 2016.* IEEE Computer Society, 2016, pp. 310–317.

[3] Christopher Natoli and Vincent Gramoli. "The Balance Attack or Why Forkable Blockchains are Ill-Suited for Consortium". In: *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017, Denver, CO, USA, June 26-29, 2017.* IEEE Computer Society, 2017, pp. 579–590.

(3) Chapter 4 of this thesis contains material published in [5] and [6] (with accompanying technical report [7]). In [5], I performed the integration, automated experimental evaluation and performed the initial analysis. (Also present in Chapter 5). In [6, 7], I had a major contributions to the design and implementation (sections 3, 4, 5) of the blockchain structure including functionality surrounding accounts, blocks/superblocks, transactions and auxiliary assets to interact with the consensus. I also assisted in experimental results in section 7.

[5] Gary Shapiro, Christopher Natoli, and Vincent Gramoli. The Performance of Byzantine Fault Tolerant Blockchains. In: *19th IEEE International Symposium on Network Computing and Applications, NCA 2020, Cambridge, MA, USA, November 24-27, 2020.* IEEE, 2020, pp. 1–8.

[6] Tyler Crain, Christopher Natoli, and Vincent Gramoli. "Red Belly: A Secure, Fair and Scalable Open Blockchain". In: *2021 IEEE Symposium on Security and Privacy, SP May 24-27, 2021.* IEEE Computer Society, 2021

(4) Chapter 5 contains material published in [5] and in a technical report [8] to be published. In [5], I assisted with identification of shortcomings present in current frameworks and suggested resolutions. In [8], I am the corresponding author of this technical report being prepared for submission. I designed and implemented the framework, and I performed the majority of the analysis present in the material.

[5] Gary Shapiro, Christopher Natoli, and Vincent Gramoli. The Performance of Byzantine Fault Tolerant Blockchains. In: *19th IEEE International Symposium on Network Computing and Applications, NCA 2020, Cambridge, MA, USA, November 24-27, 2020*. IEEE, 2020, pp. 1–8.

[8] *Note: Authors in this publication are listed in alphabetical order.* Harold Benoit, Vincent Gramoli, Rachid Guerrauoi, Christopher Natoli. "DIABLO: A Distributed Analytical Blockchain Benchmark Framework Focusing on Real-World Workloads". url: http://infoscience.epfl.ch/record/285731.

Supplementary technical reports with material included in this thesis containing additional information in their respective publications:

[4] (Chapter 3) — Christopher Natoli and Vincent Gramoli. "The Balance Attack Against Proof-Of-Work Blockchains: The R3 Testbed as an Example". In: *CoRR abs/1612.09426 (2016). arXiv:1612.09426*. url: http://arxiv.org/abs/1612.09426.

[7] (Chapter 4) — Tyler Crain, Christopher Natoli, and Vincent Gramoli. Evaluating the Red Belly Blockchain. In: *CoRR abs/1812.11747 (2018). arXiv: 1812.11747*. url: http://arxiv.org/abs/1812.11747

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

Christopher James Natoli
Signature: *Date*: June 28, 2021

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

Associate Prof. Vincent Charles Gramoli
Signature: *Date*: June 28, 2021

(*Auxiliary*) **Dr. Ralph Holz**
Signature: *Date*: June 28, 2021

This thesis is dedicated to my mother, Carmel Natoli,
fufilling the last promise I was ever able to make to her.
Rest In Peace, and thank you for everything.

Acknowledgements

This entire PhD was a monumental journey that has taught me lessons I will never forget, especially finishing up in the middle of a lockdown. Although unconventional, I have quite a long list of acknowledgements, as this thesis is a product of support and encouragement. I am extremely grateful for all of those who helped and without you I would not have made it through.

Firstly, I would like to thank my Mum (Carmel), brother (Daniel), and especially Dad (John) for their unending support and patience and getting me through this. Without you I do not know where I would be. My mother, Carmel, was my motivation through this entire journey. Although not with us, provided me unending strength to keep going. I would especially like to thank my brother who would not only provide me with 2am food, but support and reminders of the outside world.

I would like to thank my supervisor, Vincent, for the incredible journey we have been on which includes pioneering the Blockchain research at the University and going through an entire roller coaster of challenges. You have taught me a lot, and I am extremely grateful for your patience. I would also like to thank my auxiliary supervisor, Ralph, for the unending jokes of how blockchain is dying and the incredible support towards the end of the thesis. It was a shame both of you were out of the university for the last year, but what an adventure it was. Thank you both for getting me through this and for the support throughout. Thank you also to my reviewers for their efforts and tremendous reviews.

I would like to thank my entire extended family. Every single one of you played a part in keeping me happy and supporting me: first, my Nonni for teaching me the values of hard work and never giving up; my aunties and uncles, for the laughter, support and absolute love, especially Zio Joe Natoli (for the stories, biscotti and supportive messages), Zio Joe and Zia Gina (for the Thursday dinners, love and support), 1UT, Zia Debbie, Zio Joe, Zia Lisa, Zio Rick, and Zia Rose; all my amazing cousins who kept a smile on my face and helped me through — Adrienne and Bill, Sebastian, Aurora, Luca (my climbing partner!), Sabrina, Matthew and Claire (especially for all the amazing support through lockdown, food and coffee!).

I would also like to thank my brother's wife, Georgia, and her entire family for their love and support through this, especially Irene for the words of wisdom and many laughs.

To Sigma Prime, especially Paul, Mehdi and Age to whom I owe a lot of this thesis. Thank you for bringing me on board, helping me learn more and the incredible support with the many many adventures. Thank you beyond what I can write here.

The two Postdocs who changed my life, I cannot thank enough: Tyler Crain, for starting this journey with me and giving me a whole new outlook on life, and Gauthier Voron, who changed my mind and taught me valuable lessons on life, salsa, research and the power of perl. Without you both, my journey would have been incredibly different, so I am extremely grateful to have met you and worked with you both. I look forward to meeting again in future! This extends to the entire research group, both current and past, who have gone through this journey and provided interesting discussions with coffee through lockdown, especially Gauthier, Alejandro, Deepal, Pierre, Yiding and Daniel.

Collaboration is a fundamental learning experience, so I would like to thank Jiangshan and Paulo for their patience and hard work through our collaboration, and look forward to future endeavours. I would also like to thank Harold Benoit, the first student who worked with me on Diablo and produced amazing work.

To the staff at the University who believed in me and supported me all the way from undergrad with a myriad of challenges, especially Alan, Uwe, Bryn, Simon, Josiah, Andre, Julian, Joachim, William, Martin, Evelyn, Katie, Greg, Will and Kam and everyone in the SACT group.

To the entire crew at Toby's Estate coffee (Danika, Lulu, Mai, Taylor, Bryan, Jack, Kevin, Ashley and Harry), for keeping my caffeine levels high, and giving me a space to concentrate, laugh and start to appreciate the amazing journey of coffee and latte art.

Truly, thank you to everyone in Team Zazzle, especially Kanykey, Scott, Elie, Kristy, Erica, Nick, Marcus, Johnno, Roger, Candice, Sammi, Justin, Emily, and all others who have constantly supported me, checked up on me, helped with internships and taught me about everything, thank you. To BWHENO and partners, who distracted me with all kinds of crazy, and lots of food. Thank you for the support since before undergrad even began. I thank all my other friends who I am extremely blessed to run out of space writing this, especially; Alan, Zuman, Kumar, Johan, Jarrod, Chris, Felix (for keeping me healthy being an awesome gym partner for the entire 3 years!), Omid, and all the rest for their amazing support. A huge thank you to Luke for mentoring me and for keeping my head aligned, setting up many aspects of my life and going beyond to keep me up, thank you!

To my students, who have become lifelong friends. Thank you, especially Sam, Ashu, Costas, Rebecca (especially for the care packages and kind support), and Tom (especially for my epic motorbike distractions) for their unending support.

I would like to thank my climbing crew, especially Baptiste, Martin $\times$ 2, Jess, Andrew, Emily, Pierre and Luca for keeping my mind straight and getting me away from my desk.

I would like to thank Kanykey and Scott for their monumental effort supporting me through all of this, especially during lockdown where they would organise running and travel up to see us.

Thank you to my wonderful friends Shaun and Laura who kept me powering through with constant visits, climbing and keeping me happy.

To my partner's family, Jen, Stephen, Virginia and especially Paul and Michele, for their love and amazing journeys we have had together through this time. Thank you for the endless support throughout all of this and proof reading the entire thesis within a week; I appreciate it beyond words.

To my dear friend Pouya, the "cleanista", who would visit my desk daily, be crazy enough to make coffees with me at University at 2am and make the University Hawaii on weekends. Thank you so much my friend.

Finally, saving the best for last, my partner Natalie, who not only has tolerated my craziness, but has shown unending love and support throughout this entire journey. Thank you for being my distraction, keeping me smiling and alive.

Abstract

It is no exaggeration to say that Bitcoin constituted the reboot of research efforts in distributed systems and Byzantine fault tolerance. However, along with rising popularity a flurry of proposals were made, each aiming to address problems in the initial specification proposed by Bitcoin or introduce new functionality. This added considerable complexity to the blockchain ecosystem, amplified by the absence of detail in accompanying documentation, such as whitepapers or blog posts. Although moving towards technological maturity, the blockchain presents new environments for distributed systems with implicit assumptions, with the developers often overlooking critical details that lead to vulnerabilities and mistaken guarantees when deployed in practice.

This dissertation presents fundamental contributions towards secure, high performance blockchains. Firstly, we evaluate the impact of misrepresented assumptions, identifying anomalous behaviour as a result of chain reorganisation, indicating weaknesses of probabilistic blockchain consensus. We then develop and propose the Balance Attack, a novel attack that utilises communication across multiple subgroups of nodes to double-spend, emphasising the importance of critical assumptions and guarantees, such as synchrony and committing transactions. This provides the foundations for the Red Belly Blockchain, a secure, high performance blockchain which we present to mitigate risks posed by anomalies and attacks such as the Balance Attack. Experimental evaluation shows that the Red Belly Blockchain improves upon latency and throughput offered by other Byzantine Fault Tolerant blockchains while scaling to hundreds of consensus nodes. Finally, we developed Diablo, a modular, distributed benchmark framework capable of facilitating fair and accurate comparisons of blockchains through dynamic, real world application workloads. This provides integral performance validation of our Red Belly Blockchain design and paves the way for future analysis and further comparison of new blockchains and workloads.

Foreword

The title, *The Road to El Diablo*, stems from a childhood movie, *The Road to El Dorado*, depicting an adventure to find the fabled city of *El Dorado*, dubbed the city or empire of gold. Numerous expeditions were led throughout history to find this city, however, have been unsuccessful thus far. This road, and expedition, is analogous to the work in this thesis with the search for high performance and secure blockchains.

The secondary aspect, *Diablo*, not only denotes our benchmark framework presented in this thesis, but is also the name in many cultures for the *Devil*. In our search for greatness in the blockchain, judgement was unleashed. Thus, DIABLO's benchmark provides fair judgement upon blockchains, allowing a comparable analysis and evaluation of blockchains to further provide insight into the quest for high performance and secure blockchains.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The era of blockchains began in 2008 with the introduction of Bitcoin [9], pioneering a technological disruption that changed the way the world viewed digital finance. This phenomenon reignited interests in the many areas of distributed and peer-to-peer systems, leading to numerous alternatives and extensions proposed to ameliorate this new technology. The nature of the blockchain provided insights into ungoverned distributed systems, leading to a re-evaluation of decentralisation. From its unnoticed beginnings, the blockchain has risen to become a complex online payment system with considerable potential yet to be revealed, capable of executing business logic and powering a diverse range of infrastructure.

In this thesis, we present one overarching core research objective; to design a secure, high performance blockchain. The objective is then split into three key objectives, detailed below in Section 1.1, in which the contributions (Section 1.2) are the resulting outcomes of the objectives. We have identified issues surrounding transient forks in blockchain deployments, impacting both performance and safety. To illustrate, we reveal the presence of an anomaly, where transaction existence in the chain is probabilistic and may be rearranged in periods of long delays. A novel attack was presented to highlight the importance of these issues, utilising characteristics identified in the anomaly to double-spend through manipulation of the network between nodes. We argue that a solution to such problems lies with the guarantees provided by the system, whereby consistency should be held above availability. These shortcomings are improved upon to present a high performance blockchain built around foundations of safety, unforkable under network partitions. The deterministic consensus allows for maximal resource utilisation of the participating nodes, where block proposals are combined to provide higher throughput than more traditional approaches. To evaluate our solution, we develop a benchmark framework that allows for fair comparison of blockchains through use-case scenarios.

A blockchain is described as a fully distributed, append-only ledger containing transactions between participants of the system. Each transaction in the system originates from at least one account and is signed by the respective private key(s) belonging to the owner(s). The transaction can contain a simple value transfer of assets, or as the technology progresses, data to invoke code running on the chain. Transactions are batched in blocks, which are then stored

on the ledger, and each block appends to the hash-chain [10] of the ever growing ledger.

The underlying foundation of the blockchain is distributed State Machine Replication (SMR), the concept of which multiple machines coordinate to uphold a global system state through replication and communication [11]. This requires the machines to reach an agreement on the global state and the transitions to new global states. The core of this agreement is performed through distributed consensus, where machines reach agreement through communicating proposals and voting amongst one another. Although agreement is the trivial concept of having the majority of the votes, complications arise with the presence of faulty or corrupted machines [12, 13] producing unwanted results. To account for machines that fail to respond, from crashes or by omitting a response, the total number of machines (n) must be greater than twice the number of faults (f), namely $n > 2f$, to ensure a majority of the votes have been received, given assumptions on the messaging delays. Algorithms that successfully tolerate crashes are denoted Crash Fault Tolerant (CFT). This becomes increasingly difficult when dealing with arbitrary faults. The problem was first depicted in the analogy of the "Byzantine Generals' Problem" [14], where the generals of the Byzantine army are coordinating an attack on an enemy city and can only succeed if all generals agree on a common plan of action. Traitorous generals will try to confuse others by sending arbitrary messages, causing difficulty to reach a global agreement. In this setting, additional assumptions are required on both the number of participants n with respect to the number of faults and the behaviour of the messages sent in the network. To ensure a strict majority, it is common in many environments that $n > 3f$, to ensure the strict majority agrees on the same state and unwanted behaviour can be tolerated [12, 14, 15, 16], whereas other environments show that $n > 2f$ can still tolerate Byzantine faults if stricter assumptions on the messages are placed. Consensus algorithms that can provide agreement despite these arbitrary faults are categorised as *Byzantine Fault Tolerant (BFT)* consensus algorithms. Research into these algorithms has provided numerous variants that can be integrated into the blockchain.

The complexity of the problem of faults is further compounded with the acknowledgement of message delays and the time taken for message transmission. A model is *synchronous* if there exists a fixed, known upper bound, Δ, to receive messages. This allows for all machines to determine whether messages have been received to ensure the threshold of faults has been met. However, messaging over the Internet may yield unknown delays, making a fixed Δ an inappropriate assumption. An *asynchronous* model states that there is no upper bound on message delay, proving difficult to recognise whether a node has crashed or is suffering from long delays. In 1985, Michael Fischer, Nancy Lynch and Mike Paterson produced a fundamental result proving it impossible to reach consensus in a fully asynchronous environment in the presence of even one fault, dubbed the FLP Impossibility [17]. This result has impacted the way consensus is understood even today, as techniques must be used, such as redefining assumptions or models to circumvent the result of the impossibility. One common approach is adding probabilistic guarantees to the consensus through techniques such as coin flipping or randomisation [18, 19, 20, 21, 22]. This allows for consensus to be provided in an asynchronous

environment at the cost of weakening some of the guarantees. Alternatively, another approach is to assume a *partially synchronous* model [23], which provides two variants for assumptions on bounding message transmission and delay. One variant states that there exists an upper bound on all communication that is unknown to all actors in the system a priori. The other states that there exists a *Global Stabilisation Time (GST)* in which all communication from this point becomes synchronous, but the occurrence of the GST is unknown.

The blockchain is no exception, requiring techniques to overcome the FLP Impossibility result. However, blockchain payment systems like Bitcoin present a unique environment where consensus and state machine replication would occur over an unknown, dynamic set of machines that were able to join and leave at any time. This environment was termed *Permissionless*, denoting the dynamic network where machines can decide to participate at any time. Unlike traditional environments, the blockchain required consensus to be reached over a seemingly infinite number of participants, or nodes, making known consensus algorithms unfitting as they require a known n for the assumption of faults, or would struggle with increased message complexity. To circumvent the FLP Impossibility, Bitcoin introduced a new style of consensus where nodes reach conclusions based on local calculations from the data they had available [9]. This algorithm, dubbed *Nakamoto's Consensus*, expressed consensus by selecting the longest succession of blocks as the correct canonical chain, which, as long as the majority of the network is honest and following the rules, the global state would progress reaching majority agreement. One problem remained, also prevalent in traditional consensus mechanisms, which was trusting that node identity was unique, and that the computations presented are correct.

In the absence of a trusted authority for identification, there is no means of distinguishing the uniqueness of identities of machines. This gives rise to Sybil Attacks [24], where a single adversary can assume multiple node identities and skew the calculation of n. This problem becomes superimposed in the permissionless model, such as Bitcoin, where nodes are already unknown and significant assumptions are placed on the available data. To provide resilience against this, Bitcoin imposed a challenge for all nodes wishing to contribute to the system as a disincentive for adversaries. This challenge was in the form of Proof-of-Work (PoW) [25], where nodes must perform computational work to solve a puzzle that can be trivially verified by anyone. By doing so, Bitcoin strengthened its resilience to Sybil attacks and adopted the assumption that Byzantine adversaries are required to have less than half the computational resources of the entire network. Bounding the power of the Byzantine nodes allows Bitcoin to assume that the length of the chain is directly proportional to the computational power attached, and can safely agree that the longest chain would reflect the majority of the network being honest.

With Bitcoin maturing, there was fast-growing interest in the blockchain and new alternatives started taking shape. Many of these initial alternatives, such as Litecoin[1], Z-Cash [26]

[1] Available online: https://litecoin.org/

and Ethereum [27], also used the Proof-of-Work model together with Nakamoto Consensus to provide distributed state machine replication. Rising popularity resulted in the discovery of a number of shortcomings and limitations, particularly the tremendous power consumption from Proof-of-Work. This prompted research into alternatives to Proof-of-Work, aiming to provide similar guarantees at a lower resource cost.

Today, as the blockchain continues to evolve, the core foundations and principles must be studied and validated to ensure purposeful progression. The problems identified throughout this thesis are fundamental in highlighting the existence of issues and future possibilities that require further exploration.

1.1 Objectives

We now outline the research objectives of this thesis; one overarching goal which is then presented as three separate objectives. Throughout this thesis, the aim is to evaluate the blockchain landscape, both theoretically and experimentally, providing improvements on safety and performance.

Goal: Design and develop a secure, high performance blockchain

Rapidly growing interest in the blockchain has resulted in a plethora of advancements aiming to solve shortcomings or to provide functionality to integrate into existing systems. The core objective of this thesis is to design and develop a blockchain that provides security and high performance, motivated by the transformations of the blockchain as it matures. Our focus is to provide a solution which is suitable for integration into mission-critical systems, such as critical finance or infrastructure.

Objective 1. Investigate weaknesses of blockchain consensus

The original specification released by Bitcoin [9] was presented in a whitepaper, leaving room for interpretation and development. This paved the way for new advancements in functionality, such as the introduction of "Smart Contracts" through Ethereum [27], sparking an explosion of blockchain use through various tokens and contract-related integration. This resulted in a number of implicit assumptions being propagated through new blockchain variants. Identifying any underlying weaknesses is crucial for the future of blockchain adoption. To assist in the investigation, the objective is split into two sub-objectives:

1.1: *To analyse and assess the assumptions of blockchains in the context of their deployment environments.*

1.2: *To investigate the strength of consensus in an adversarial scenario.*

The blockchain contains a myriad of complex components, each presenting unique requirements and assumptions that play a role in the overall safety and operation of the system. The focus of this dissertation lies in the assumptions with the consensus, or fork choice, as previous work [28, 29, 30, 31, 32, 33, 34, 35, 36] has highlighted numerous weaknesses of blockchains in practice. Our focus through adversarial perspectives will reveal potential techniques to mitigate such shortcomings, in the hope of providing more safety to the blockchain.

Objective 2. Design and Implement a secure, high performance blockchain

Through evaluation of the blockchain landscape, the challenge then becomes: design a secure blockchain that provides safety under reasonable assumptions in practice. A number of blockchain proposals [37, 38, 26, 39] utilise foundations of Bitcoin and extend the design to incorporate further enhancements, whereas others [27, 40, 41, 42] build new models entirely. Three sub-objectives are presented:

2.1: *To assess and evaluate current blockchain designs, with a focus on designing a secure, high performance blockchain.*

2.2: *To design and develop a blockchain providing high performance that is highly resilient, especially in adversarial scenarios.*

2.3: *To analyse and evaluate, experimentally, the performance of the implementation and competing designs.*

This objective's aim is to evaluate the existing systems to formulate a design that best accommodates the goals of security, safety and performance. While the previous research objective provides insight into weaknesses and vulnerabilities, the findings will be used to motivate better design and provide input to evaluate our design. By providing a design with foundations of safety and reasonable assumptions, many risks associated with partition and network-related attacks are mitigated, whilst also aiming to improve performance. To confirm the effectiveness of our design, and therefore compare against competitors, experimental evidence must be provided to highlight any performance improvements.

Objective 3. Compare and evaluate available blockchains to investigate suitability to applications

Comparative evaluations are critical to ongoing efforts in performance improvements. With the number of readily available blockchain systems increasing over time, it begs the question whether there are benefits and advantages to certain blockchains being used in different environments and scenarios. Thus, it is imperative that fair comparisons and evaluations are performed to derive the blockchains suitable for certain environments and applications. To this end, this objective comprises of two key sub-objectives to reach fair evaluations:

3.1: *To design, develop and evaluate a benchmark framework to measure blockchain performance under real world application workloads, in particular focusing on fair evaluations.*

3.2: *To evaluate, experimentally, the performance of blockchains through investigation of real world application workloads.*

Optimal analysis requires a framework that provides fairness and transparency with appropriate comparisons. However, there is a scarcity of publicly available blockchain benchmark information, as many results are often tailored in environments optimal for the system presenting the results, or lack details such as deployment information or workload details [43, 44, 45, 46]. The aim is to evaluate the available frameworks [47, 48, 49, 50] and, if necessary, provide improvements to perform critical but fair evaluations of blockchains by capturing accurate performance metrics. This will not only assist in the evaluation of performance, but provide a foundation for future blockchain evaluation, leading to a better understanding of performance in various deployment environments and added transparency.

1.2 Contributions

Throughout this dissertation, four novel contributions are presented as a result of the research objectives, working towards secure, high performance blockchains. The blockchain is first analysed to present a novel attack, highlighting how an adversary is able to utilise synchrony assumptions to cheat the system. Secondly, a new safety-focused blockchain is presented, providing consistency and high performance with scalability. Finally, a novel benchmarking platform is proposed and developed, aimed at a fair comparison amongst blockchains through various use-cases with distributed workloads to simulate real-world usage.

To summarise, this dissertation presents the following contributions:

1) **Balance Attack:** Objective 1
   During the analysis of the blockchain, we observed the effects of the synchrony assumptions not clearly stated in available whitepapers at the time, in which we demonstrate through the Blockchain Anomaly [2]. These assumptions are then utilised to propose an attack where an adversary is able to partition the network into subgroups to double-spend the same coins with high probability. This highlights the impact of implicit synchrony assumptions in practice, and how various network disruptions can have adverse effects. Numerous attacks have been proposed from inspirations of the attack, utilising network and messages to double-spend.

2) **Deconstruction of Blockchains:** Objective 1 Objective 2
   The goal to produce a secure, high performance blockchain was fuelled by carefully analysing current state of the art to determine which components matched our criteria. To do so, we propose a unique deconstruction of the blockchain that allowed viewing

of the interleaving components and their impact on operation. This deconstruction provides the framework to categorise blockchains and understand the positives and negatives of different component interactions, as well as the suitability of different techniques when applied in new environments.

3) **Red Belly Blockchain:** Objective 2
We use our findings from the Balance Attack to motivate and present the Red Belly Blockchain (RBBC), a blockchain constructed from the foundations of the Democratic Byzantine Fault Tolerance (DBFT) algorithm [51] to provide safety over liveness in a partially synchronous environment. We then provide an evaluation of the RBBC to verify that no loss of performance had been observed by increasing the safety bounds.

4) **Diablo:** Objective 3
We introduce a new, distributed blockchain benchmark focused on providing a fair comparison against blockchains with real-world workload generation and integration. We aim to provide a fully distributed workload generation, allowing us to simulate real-world traffic against dynamic workloads to diligently evaluate blockchains in specific deployments.

1.3 Outline

The remainder of this dissertation is organised as follows, with Figure 1.1 presenting a visual representation of the chapters and their relation to the objectives:

- **Chapter 2** introduces preliminary blockchain concepts and components as well as a discussion of related literature.
- **Chapter 3** presents the Blockchain Anomaly, our finding surrounding the implicit synchrony assumption of blockchains. We then propose a novel attack, The Balance Attack, highlighting how an adversary can use these assumptions to double-spend with high probability if they have influence on the network.
- **Chapter 4** introduces the Red Belly Blockchain, a secure blockchain offering scalability and high performance. The performance of the Red Belly Blockchain is evaluated experimentally and compared to other blockchains.
- **Chapter 5** presents a new benchmark framework, Diablo that improves upon shortcomings identified in current benchmark frameworks. We perform experimental evaluation of 6 blockchains against application-based workloads to demonstrate the capabilities.
- **Chapter 6** concludes the dissertation and presents potential future work.

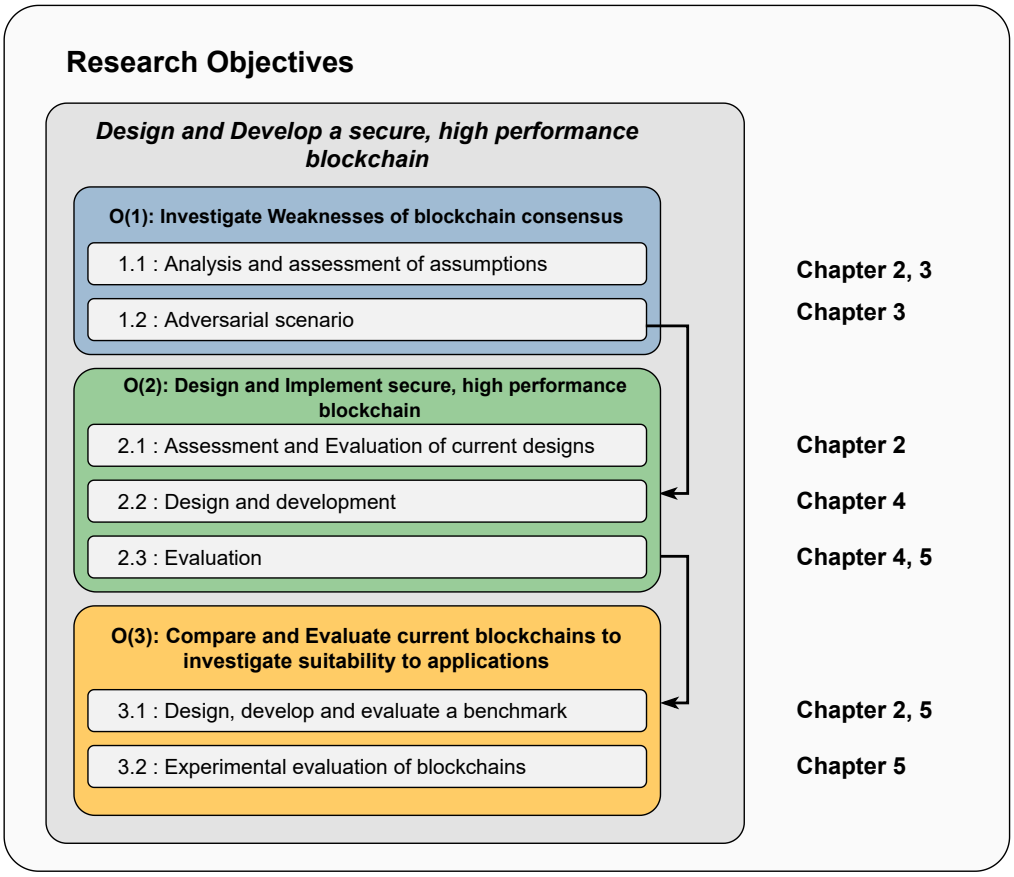


Figure 1.1: Visual representation of the research objectives and how they are presented throughout the thesis.

Chapter 2

Background

There is no doubt that the blockchain ecosystem has evolved rapidly from the initial Bitcoin specification of simple value transfer transactions being recorded in a distributed ledger. The vast flurry of proposals have added considerable complexity, often superimposed by the absence of detail in the whitepapers, wikis and websites in which they were presented [52, 37, 40, 53, 54, 55]. The evolution of the blockchain resulted in deviations from Bitcoin's initial specification, resulting in an overuse of the term "blockchain". The term Distributed Ledger Technology (DLT) has been adopted to describe these numerous proposals, capturing the vast differences from the original structure proposed with Bitcoin.

2.1 Preliminaries

Although there are inherent differences with the variety of blockchains available today, they often conform to similar primitives. These primitives form the foundations of the blockchain, which are then specialised per implementation.

Transaction (tx) The most rudimentary aspect of all blockchains is the notion of a transaction, which is a transfer of information from one entity to another. Most often, this transaction is either an asset transfer, or an invocation of code running on the ledger. Transactions often include numerous properties that contain metadata, such as information about the account, extra bytes for execution specific to implementation, or, other data fields. To avoid ambiguity, throughout this dissertation we adopt the term "transaction" to refer to a set of commands that are executed resulting in update, or "write", of the global state [27, 9, 56, 57, 58]. Alternatively, we use the term "request" to denote operations that interact with the state (executing a "read") but results in no writes to the state, performed as an application-level call to the underlying node. It should be noted that some implementations and literature use the term "transaction" inclusively to denote operations interacting with Smart Contracts, defined below, sent as fully formed transactions recorded on the chain.

Block A block is a collection of transactions that have been executed and are written onto the distributed ledger. A block represents the new state of the world, where each block added

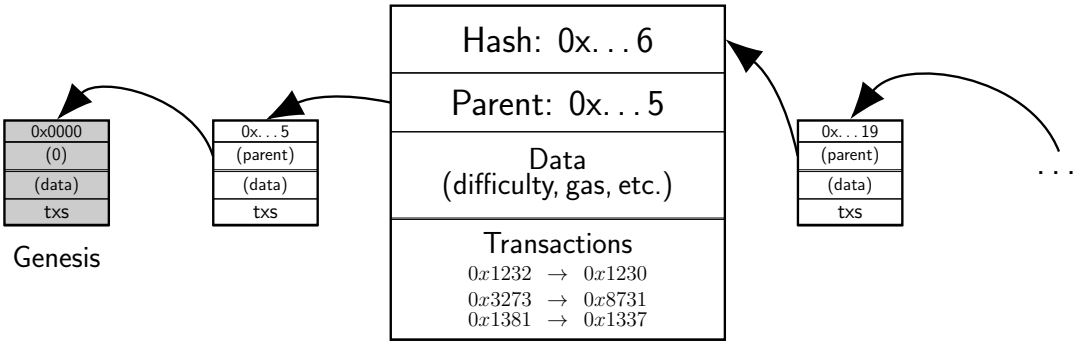


Figure 2.1: A simple blockchain example, where each block b points to the previous block, which is referred to as the parent of b, by using hash forming a cryptographically linked hash chain. The Genesis block is the only block that has no parent, as it defines the initial state of the blockchain.

to the ledger acts as a global state transition. Block composition varies from system to system based on specifications of the components, but often contain a set of transactions, a unique identifier (usually the block hash), an index representing the height or depth of the chain at that block, and a pointer to its predecessor, or the previous block, that it builds upon and is referred to as the "parent" of the block.

An *Uncle Block* is the term first introduced by Ethereum [27], denoting a block not accepted in the canonical chain, but whose parent block is. These blocks are used in some calculations and for fork choice resolution (as discussed in Section 2.2.2).

Blockchain The term "blockchain" originates from the structure proposed by Bitcoin, where each block is cryptographically linked to the previous block forming a chain of blocks from the current state to the beginning. Simply put, a blockchain can be represented as a directed acyclic graph (DAG), $blockchain = DAG\langle B, R\rangle$, consisting of blocks (B) and pointers (R), where each block points to the previous block. The pointer is stored in the current block and is constructed using the hash of the previous block. A special block, the *Genesis*, is the only block not pointing to any previous block and serves as the global "first block", commonly at index 0 of the chain. As depicted in Figure 2.1, each block b points to the previous block, which is referred to as the parent of b, by using a hash of the previous block, and contains information denoting system parameters as well as the batch of transactions to apply in order to transition to the next state.

Account The original Bitcoin specification outlined an account composed of a public and private key pair. The hash of the public key identifies the account of the key owner, forming an *address*. This public address is used in transactions to receive coins, whereas the private key of an account is used to sign and authorise spending. Payment between accounts occurs with a "payer" (or "payers") signing a transaction with their respective private key(s), transferring

Table 2.1: Relation between Difficulty and expected hashes. Higher difficulty, more leading 0's required in the target hash.

Difficulty	Expected Number of Hashes	Approximate Target
1	4.295e+09	0x00000000FFFF0000000000000000000000...
1000	4.295e+12	0x00000000004188F5C28F5C2800000000...
10000	4.295e+13	0x0000000000068DB22D0E560400000000...
100000000	4.295e+17	0x000000000000002AF2F2D14354120000...
1000000000000	4.295e+21	0x00000000000000000119787E99468E30...

assets to the address of the "payee" (or "payees"). Most blockchain variants have adopted similar concepts, having a public/private key pair and the hash of the public key as an address with differing cryptography.

Smart Contract First introduced in Ethereum [27, 59], a Smart Contract refers to conditional payment performed through execution of Turing-complete code on the blockchain. This facilitates complex interactions running on the blockchain, where developers have the ability to programmatically define actions based on variable inputs. Often written in a high-level language, the code is compiled to instructions that are run on a virtual machine integrated with the blockchain. The code will be run "on the blockchain", denoting that its execution will occur on the decentralised set of machines, and the output state will be updated on all nodes. Most blockchain based applications, known as Decentralised Applications (DApps) interact with these smart contracts through libraries in common languages such as JavaScript, Java, Python, Go and Rust. Often, these are tied with user interfaces such as web pages or mobile applications. This feature allows the blockchain to be directly integrated with business logic and has immense potential for future use cases. Now, other blockchain systems, such as HyperLedger [60], EoS [52] and Diem [40, 61] also provide the capability for smart contracts.

Gas To prevent infinite execution, Ethereum introduced the concept of *gas*, providing a fuel for running code. Each instruction on the virtual machine is assigned a monetary cost. For a transaction to interact with a smart contract, it must dedicate a certain amount of gas to run the code, where running out of gas will halt the execution. The gas model has a large impact [62] on the blockchain operation and security. This was observed in a Denial-of-Service which occurred because of an operation that was priced cheaper than its execution and was abused, causing extremely long execution times for a large portion of the network [63].

Tokens and Coins The introduction of smart contracts paved the way for unique application developments. One outcome was the introduction of *Tokens*, currencies that are defined as part of a smart contract with the ability to trade and transfer. *Coins*, on the other hand, denote the native currency of the given blockchain. Various exchanges have been developed to trade these tokens for monetary value, which prompted widespread use by various investors. The evolution of tokens led to the development of NFTs, or Non-Fungible Tokens, a unique token that points

to a distinct resource, such as an artwork, traded on the blockchain through a smart contract.

Difficulty Difficulty is the concept relating directly to Proof-of-Work (discussed below in Section 2.2.1), defining the complexity of the cryptographic puzzle to be solved. Introduced into the blockchain by Bitcoin, inspired by the seminal paper of Proof-of-Work [25], the difficulty denotes the average number of hashes required to achieve a target. The difficulty provides a delay in block proposals, as each proposed block must have a verified, correct Proof-of-Work solution. This target difficulty is often included in the block, as it can be verified to ensure that the Proof-of-Work attached to the block meets the criteria. This has been adopted by other Proof-of-Work blockchains with varying hash techniques. Table 2.1 provides an example of the difficulty and its relation to the target denoted in the block and the expected number of hashes.

2.1.1 Openness and permissions

One fundamental change introduced through Bitcoin was the use of an open participation system, allowing nodes to join, participate and leave on their own accord. This provided unprecedented possibilities for scalable consensus algorithms to flourish, as they would be dealing with large, dynamic sets of participants. However, as the blockchain evolved, it was deployed in a variety of environments, where node participation was known and membership is strictly authorised. These restrictions form the rules on how nodes are able to participate and how committees to perform consensus are selected. This is a fundamental aspect in the design of the blockchain, as the openness depicts its suitability for certain environments [64, 65]. Combinations and hybrids of these models started taking shape with variations on the permissions of nodes, but can all be classified into the following categories:

Permissionless A permissionless model often permits open participation and has no restriction on the set of candidates and their behaviour. That is, the size and identification of the consensus committee is often not fixed, nor predefined. This provides nodes with the ability to leave, join or participate at any given time. Bitcoin introduced this model to the blockchain, which has been adopted by the popular public blockchains, such as Ethereum [27], ZCash [66, 67] or Monero [37, 68].

Permissioned Conversely, a permissioned, or private, model has complete restriction on the nodes participating in the system. A node must join the system through authorisation, by being accepted as a known member. The consensus committee is often predefined, or, operates under the assumption that all nodes are known to all.

Open-Permissioned An open-permissioned model is often seen in consortium scenarios, where there are certain restrictions present on either the consensus members or the node participation. Some systems allow nodes to freely join, but only a predefined subset of nodes form the consensus; other systems require nodes to be accepted in the system but allow for any node to become a member of the consensus.

2.1.2 Consensus problem

Byzantine Agreement defines the problem of reaching agreement in the presence of faulty processes, as depicted through the *Byzantine Generals' Problem* [14]. A process is correct if it is non-faulty and following protocol. All messages between correct processes are eventually delivered. During the consensus, proposals refer to inputting to the consensus. A decision refers to the value output by this procedure. A system wishing to reach agreement in the presence of faults must adhere to the following properties [69, 70, 71, 72, 73];

1) **Termination**: Each correct process eventually decides.

2) **Agreement**: No two correct processes decide different values.

3) **Validity**: If all non-faulty processes propose the same value, no other value can be decided.

These properties are critical in all systems wishing to solve the consensus problem. In the context of the blockchain, *termination* guarantees that a transaction, or block, will have been decided upon and the global state can progress, ensuring the commit status of a transaction. *Agreement* ensures that no two correct nodes decide different blocks for the same index of the chain. The property of *Validity*, however, has been adopted in a number of variations. The concept of *External Validity* [74] was defined such that any honest party that terminates on a value v for a consensus instance, v must be valid for some computed predicate. Viewing through the lens of the blockchain context, in a simple, canonical chain, validity ensures that if all non-faulty processes propose a block for a given height, then no other block will be decided. External validity, however, can be viewed as blocks, or transactions, being valid for some predicate before being decided. The Set Byzantine Consensus (SBC) problem, defined as part of Chapter 4 in Definition 7, deals with block proposals as sets of transactions, where the validity property states that "*a decided set of transactions is a valid non-conflicting subset of the union of the proposed sets;*".

2.1.3 CAP Theorem / Brewer's Theorem

Similar to the above properties of consensus, a distributed state machine often provides guarantees for the finalisation and storage of state amongst participants. However, Brewer's Theorem [75, 76], now more commonly known as the *CAP Theorem*, highlighted that a distributed service cannot provide more than two of the following guarantees:

1) **Consistency**: There must exist a total order on all operations such that each operation looks as if it were completed at a single instance.

2) **Availability**: Every request by a non-faulty node in the system must result in a response.

3) **Partition Tolerance**: No set of failures less than total network failure is allowed to cause the system to respond incorrectly.

These guarantees are heavily tied into the above properties of *Termination* and *Agreement*. Evident in the blockchain context, *Consistency* will guarantee the total order of transactions and blocks that occur, *Availability* will ensure that a transaction broadcast to the network will be added to a block and appended to the chain, and *Partition Tolerance* would result in the blockchain providing the above in the presence of network or node faults. It was proved that a system will only be able to display at most two of these guarantees [75].

2.1.4 Incentives

A major segment of the public blockchain is the ability for all nodes to participate in the creation of blocks and the progression of the system. However, this requires the consumption of resources, or, some risk as nodes may not be guaranteed a reward if they are slower. To balance this, blockchains often employ incentive schemes to motivate participation and demotivate any unwanted or malicious behaviour. Incentives are often classified into two groups, rewards and punishments, and when effective, they introduce *rationality* and thus influence node participation and heavily constrain any Byzantine behaviour.

Rewards Reward measures the benefit that one gets as a result of successful contribution to the system. Often, it can be a combination of either *block rewards*, a fixed value reward for successfully appending an accepted block to the chain, or, *transaction fees*, variable rates paid by users for their transaction to be executed and included into the chain.

Punishment Punishment, on the other hand, can be categorised in a number of ways dependent entirely on the system. This can be strictly stated as part of the protocol, or, as a byproduct of the reward scheme where the punishment is the absence of a reward if the misbehaviour leads to a negative outcome. Regardless of the selected technique, the result is a loss of resources for unwanted or malicious behaviour.

Obtaining a balance between rewards and punishments is vital in such decentralised systems, as motivating participation will help strengthen the execution of the protocol, but may also lead to strategies where nodes optimise for maximal reward in ways that are not helpful to the overall progress of the system.

2.1.5 Forks

With numerous nodes working to participate in the block creation as a race to gain rewards, there is increased chances of simultaneous proposals at the same index as depicted by block index 1337 in Figure 2.2. For traditional blockchains that follow a single, canonical chain, the concurrent proposal creates a transient branch, known as a *fork*, in which two or more proposed blocks share a common predecessor, and are able to continue on their own separated path. This results in disagreement in the network, as nodes would be conflicted about which is the correct state transition to accept. Due to the nature of the gossip-based propagation, nodes will receive the blocks in differing order [77, 78, 79], naturally accepting the first block they have seen until

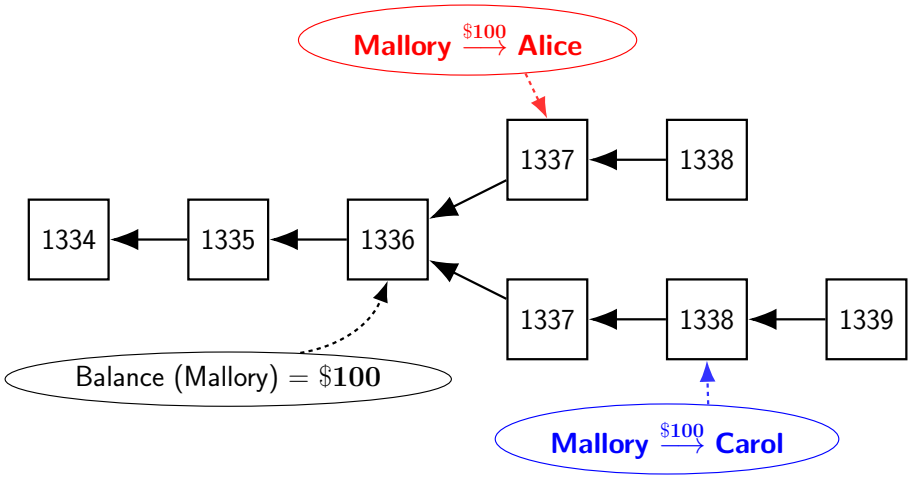


Figure 2.2: In the presence of a fork, an adversary can issue two conflicting transactions, each to different branches. In this example, Mallory is sending her entire balance to Alice in one branch and Carol in the other, meaning upon resolving the fork, one will fail to have received the funds.

proven otherwise incorrect through the consensus.

However, forks are also critical in the maintenance of a blockchain, as they are used for major protocol upgrades and changes, known as *hard forks*. To perform a protocol change, the system will diverge from a given block index and all nodes that accept the new protocol upgrade and run the latest version of the software will be on the same branch, whereas others not running the latest version will continue in a separate branch. If enough nodes do not wish to participate in the protocol upgrade, it can lead to a separation of currencies, which can be observed from famous hard forks such as Bitcoin with Bitcoin Cash [39], and Ethereum [27, 59] with Ethereum Classic [80].

New proposals introduce *Unforkable* blockchains, where deterministic guarantees mitigate transient forks. Commonly, this is due to the blockchain adhering to consistency, halting progress in the event that a significant percentage of the network is unresponsive or faulty. Such blockchains commonly use traditional consensus, where proposals are voted upon and a unique decision is output as a result.

Double-Spending The presence of a fork often means that there are nodes which believe a different global state exists, leading to a split horizon. Not only does this result in the waste of resources spent on block creation, as well as impacting transaction finality, it can also be used negatively leading to cases of double-spending [29] where the same coin is spent multiple times on different branches. If a node is able to propose two conflicting transactions to different branches in the fork, they can effectively use the same coin in those transactions and spend it multiple times. Figure 2.2 depicts a scenario where an adversary can send two conflicting transactions to different branches. Once a merchant performs an irreversible service, or a product is transferred

off chain, then this can no longer be reversed. Upon fork choice rule resolving the fork, one of the recipients will fail to receive the funds as they have been double spent.

2.1.6 Child chains and side chains

However, the observation of forks led to insight into potential scaling techniques, as a fork often represents concurrent work being processed. One such technique is the usage of *side chains* [81, 82, 83, 84, 85] and *child chains* [86], which are chains beginning at one block of the main branch, stemming off to a separate branch, executing in parallel. When the side or child chain requires interaction with the main chain, the state is updated on the main chain and the interaction can be facilitated. Although these parallel branches of the same blockchain system interact, they are not required to utilise the same consensus.

2.1.7 Sharding

With the flourishing potential of side chains and child chains, the feasibility of sharding started to take shape [87, 88, 89]. The core aim of future blockchains is to provide higher throughput while minimising the impact and requirements of storage and resource consumption, which could be accomplished if multiple chains ran in parallel. Although the sharding technique has been seen in a number of areas [90, 91, 92, 93], the compatibility with blockchains is still unknown. The fundamental problem to be solved with sharding is allowing progression of global state, while maintaining the properties provided by a single, canonical chain. A number of blockchains are developing alternate chain structures to indicate how sharding is viable for the blockchain.

2.1.8 The Bitcoin blockchain model as an example

The initial specification of Bitcoin [9] denoted a fully decentralised electronic cash system on a peer-to-peer network. This facilitated pseudonymous payment between two parties, relinquishing the requirement for a third party, such as a bank. There were many similarities between the Bitcoin release and Wei Dai's BMoney [94], a text document outlining the preliminary concepts of cryptographic digital cash. The aim of Bitcoin was to provide a fully distributed system whereby there is no single entity controlling it and people have power over their assets. Transactions on Bitcoin originally represented transfer of coins from the "payer(s)" to the "payee(s)". The "payee(s)" are then able to claim and use the coins by using the private key associated to the address, this is done by using the private key to sign the transaction wishing to spend the coins.

Transactions are broadcast to the network using the Peer-to-Peer gossip model, where each node broadcasts to its neighbours eventually propagating to the entire network. A transaction that is received is verified, ensuring correctness of the signature and that all metadata matches the state known by the node. For a received verified transaction, the node can utilise it as input to create a block and append to the blockchain. After a block has been created, through the necessary processes, it will contain a set of verified transactions, a cryptographic link to its

predecessor and any required metadata. At that point, it is then broadcast and propagated to the network. If that block is valid and has been decided as the correct extension of the chain, the block will be accepted and the global state transition will occur. All nodes in the network have the ability to participate in this process of creating blocks and batching transactions, with reward as an incentive for their efforts.

The seminal work introduced by Bitcoin revealed a scalable consensus with active membership selection, as well as the application of a distributed timestamping service for validity on a ledger. The distributed ledger of cryptographically-linked blocks can be represented as a Replicated State Machine [11], where the state machine is a hashchain [10]. The assembly of ideas presented the foundations of a new technology, causing a paradigm shift in digital payments and providing numerous possibilities for this technology to evolve.

2.2 Deconstruction of blockchains

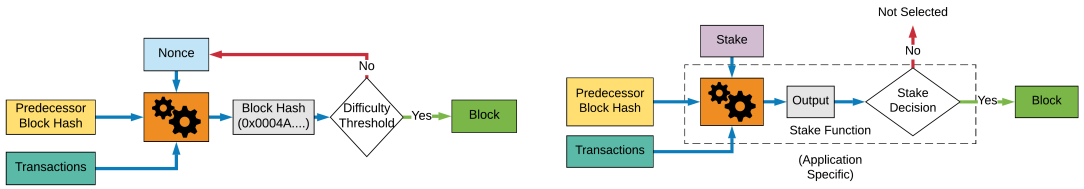
Crucial to the analysis of the blockchain was investigating and categorising proposals to better evaluate its goals, assumptions and requirements. This led to a deconstruction of the blockchain into three simple, critical components common to most known systems: *membership selection*, *blockchain consensus* and *blockchain structure*. The membership selection is tasked with providing a subset of nodes to participate in the consensus, providing the consensus with requirements such as delayed proposals and a select number of chosen nodes. The blockchain consensus, also denoted fork choice rule in some cases, is responsible for deciding the correct block for a particular place in the chain. The structure dictates the representation of data, and provides strict requirements for the consensus on the expected output decision, for example a single canonical chain expects one block per index, whereas a graph structure may require multiple blocks per index. A visual representation is depicted in Appendix A.

2.2.1 Membership selection

A critical innovation introduced through Bitcoin was the ability to select a proposer from a seemingly infinite number of nodes. The primary purpose of *Membership Selection* is to pass a subset of nodes to participate in key tasks for the consensus, this may involve verifying, batching and proposing transactions, or, voting and deciding on the future indices of blocks.

Proof-of-Work

Proof-of-Work (PoW) is the mechanism introduced to blockchains through Bitcoin and became the most prominent selection technique for mainstream blockchains today. In the blockchain context, it was proposed to delay proposals and address concerns of Sybil Attacks [24], where an adversary creates multiple identities to dominate the consensus committee. PoW addresses this by requiring nodes to prove they have performed work at some non-negligible cost, such as time, hardware or consumed resources, as shown in Figure 2.3a. This was heavily derived from the work proposed by Dwork et al. to combat junk mail [25] and later adopted by HashCash [95].



(a) Proof-of-Work (b) Proof-of-Stake

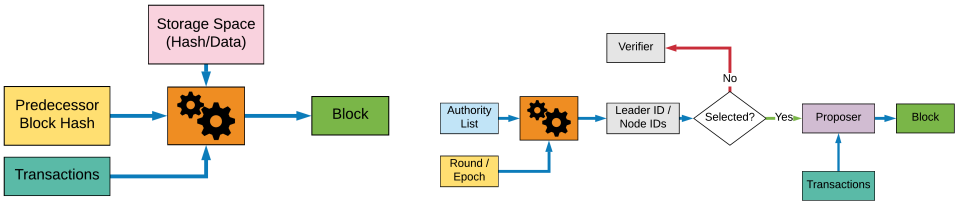
Figure 2.3: Membership Selection Techniques I.

There are two main variants, the CPU-bound approach introduced through Bitcoin [9] comprising of performing numerous hashes, and the memory-bound approach [96, 97, 98] which relies on random memory accesses rather than hashing speed. Adaptations, such as Proof-of-Reputation [99], build upon the primitives introduced through Bitcoin to improve the security in the face of attacks as detailed in Section 2.4.

Proof-of-Stake

Proof-of-Stake (PoS) aims to address two major impediments of Proof-of-Work; the extreme energy consumption and the low throughput [100, 57]. The key concept is that a node must bet, or *stake*, something of value that will be removed if they have been found guilty of malicious actions. The selection process is highlighted in Figure 2.3b. The first noted proposal on a stake-based voting system is mentioned in Wei Dai's b-money [94] and later proposed on the BitcoinTalk forum [101]. The concept of PoS provides flexibility for implementations, as the *stake* can be represented by a variety of resources and calculated if different ways. The main concern is that this may lead to centralisation as wealthy actors could form factions to control the system.

There are a number of implementations, beginning with PPCoin, or Peercoin [102] where it proposed the age of coins as the staked "value". Proof-of-Burn [103, 104] adopted by SlimCoin [105] requires coins to be destroyed to propose a block. Tendermint [106, 107] requires nodes who wish to participate in the consensus to lock, or freeze, coins. Algorand [42] selects through a Verifiable Random Function (VRF) using the node's balance as a selection parameter. Ouroboros [108, 109] presents a lottery-like method for selection with balance-weighted probabilities. Ethereum 2.0 [87] implements PoS by requiring nodes to deposit a set amount, and rotates via random selection. Delegated Proof-of-Stake (DPoS) [110, 111] has been adopted by a number of chains, such as BitShares [110], Lisk [112], and EoS [52], which allows voting on *Delegates* to perform critical actions, where votes are weighted by balance, and the stake is reputation in the system. Waves [113] introduced Leased Proof-of-Stake that allows stake to be leased to others to combat the centralisation fears brought by wealthy entities in the network.



(c) Proof-of-Capacity (d) Proof-of-Authority

Figure 2.3: Membership Selection Techniques II.

Proof-of-Capacity

Similar to the reasoning behind the development of Proof-of-Stake, there is ample motivation to provide a utilisation of resources spent during the membership selection of Proof-of-Work. Proof-of-Capacity aims to harvest the computational resources by providing a selection mechanism that uses the selection technique to maintain the blockchain network. Figure 2.3c highlights the key concept of Proof-of-Capacity. A number of variants, such as Proof-of-Retrievability [114, 115] and Proof-of-Space [116, 117], focus on storing data of the blockchain across multiple nodes and utilise proofs to show that files have been successfully stored. This not only improves the usability of resources of the blockchain, but empowers nodes to maintain the network by sharding data across nodes, reducing overall replication and wasted computation. This also provides ways for applications, such as FileCoin [118], to operate a decentralised file system on blockchain primitives.

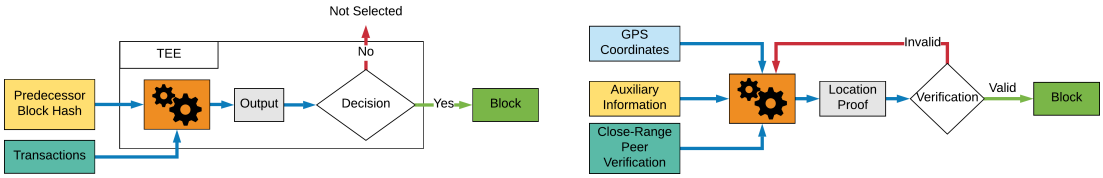
Proof-of-Authority

Proof-of-Authority (PoA) was initially proposed as an addition to the Ethereum blockchain [119, 120] for Testnet usage as a response to the shortcomings of Proof-of-Work in smaller networks. To ensure minimal waste of energy, and provide progress in a small network, PoA employs *Authorities* that are selected to propose and seal blocks. The blocks are created periodically by the chosen leader for that round, and are signed by others. A depiction of this can be found in Figure 2.3d. Authorities are the only chosen, trusted nodes rotated to propose and create blocks. There are two main implementations, Go-Ethereum's Clique [120] progresses in epochs where each has an elected leader. The other is OpenEthereum's[1] AuRa [119] where a trusted leader proposes blocks for the round. The main difference is that Geth's Clique allows multiple authorities to propose a block, whereas AuRa requires the leader to propose for that epoch.

TEE-Based

Technological advancements in hardware provide new possibilities for membership selection that can be adopted by blockchains, as shown in Figure 2.3e. A Trusted Execution Environment (TEE) is a secure hardware module that guarantee code running will honestly follow pre-

[1]OpenEthereum was formerly known as Parity's Rust implementation, but has since diverged from Parity Technologies.



(e) TEE-based selection (f) Proof-of-Location

Figure 2.3: Membership Selection Techniques III.

defined policies, such as *sleep* waiting for the correct amount of seconds. This allows trust to be placed on hardware, making room for algorithms that use randomness and waiting time. Hyperledger Sawtooth's Proof-of-Elapsed-Time (PoET) [121, 122], uses the trust in hardware to "sleep" for randomly defined periods of time before proposing, where the first node that wakes is selected as the proposer. Proof-of-Luck [123] uses randomisation in a lottery-like selection process that elects a winner by selecting a random number proposed by all other nodes. Proof-of-Useful-Work [124] performs desired workloads to contribute to real-world computations, where a generated proof verifies the work done by the miner at the completion of the workload.

Proof-of-Location

Another technological advancement is the emerging use of Internet of Things (IoT) and mobile devices, which have slowly been introduced to the blockchain [125, 126, 127]. However, mainstream blockchains still require high computation or large storage, impractical for low-powered devices. Proof-of-Location [128, 129] aims to utilise the dynamic nature of mobiles to create and verify proofs based on the location of devices and surrounding nodes. Originally proposed for collusion resistance [130] and later adopted by the Platin blockchain [131], the location verification utilises geographic locations to sign and verify transactions, as shown in Figure 2.3f. Location verification aims to stop geographic spoofing by utilising low-range communication channels.

2.2.2 Blockchain consensus

Contrary to popular belief, the consensus in the blockchain is not purely tied to Proof-of-Work, rather the concept of fork resolutions, such as the "Longest Chain" [9] in Bitcoin. Consensus holds the responsibility of deciding the next accepted block on the chain. This is vital to the operation of the blockchain, as it effectively produces the state transitions that govern the final global state and progression of the system. Traditionally, blockchain consensus operates in a permissionless, open network where there is a seemingly unlimited number of nodes participating in the consensus. However, newer deployments in permissioned environments exhibit similar properties to traditional distributed systems environments. The distributed agreement problem has been heavily studied in distributed computing, Byzantine Fault Tolerant (BFT), also known as Byzantine Agreement (BA), protocols were proposed to achieve consensus in the presence of potentially malicious distributed participants. This problem was detailed in the *Byzantine*

Generals' Problem [14], which has been heavily studied for over 40 years. However, traditional BFT protocols were ill-suited for the new permissionless environment introduced by Bitcoin, and require changes to form a valid committee.

Nakamoto's consensus

Nakamoto's Consensus introduced through Bitcoin was the first consensus to be applied in a permissionless, blockchain context and has been adopted by many other blockchains [26, 27]. The core aspect is the *Longest Chain* rule, used in the event where multiple blocks are proposed at the same index and the chains have progressed in a forked state. The rule states that nodes following the protocol must select the longest chain with the highest block number as the single, canonical chain, as it is believed to have the most work performed. This adheres to the assumption that the strict majority of the network are honest participants, and therefore the majority of the computational power. In the event that two chains are equal length, the nodes perform work on the first valid block they receive, abstaining from confirming conflicting blocks until one branch becomes the longest.

In the permissionless, blockchain context, Nakamoto's Consensus can only guarantee eventual consistency [132], that is, if there are no new blocks proposed then all nodes eventually see the latest block at the height of the chain. During the time the blockchain experiences a transient fork, consistency will not be met and fork resolution will eventually occur through Nakamoto's consensus. This consensus provides security under the assumptions that the strict majority is honest and that blocks are propagated through the network prior to any new proposals. With high block creation speed, there will be an increase in conflicts as information would be propagated slower than a proposal is created [79]. This results in an increasing number of blocks that would fail to extend the *longest chain*, which has a number of implications for performance and leaves the door open for potential attacks. With increasing node presence in the network, the number of possible conflicting proposals increases, resulting in the need for limiting proposals with satisfactory delay. Bitcoin paired this consensus with Proof-of-Work [25], acting not only as resilience to Sybil's, but also sufficiently delaying proposals adjusted through changing difficulty of the cryptographic puzzle. However, if block proposal speed exceeds propagation time, the probability of conflicting proposals increases resulting in a waste of mining power, and, at large, resources, as not all blocks are "accepted" [30, 133]. This also leads to a number of attacks that leverage the undecided state of a fork [29, 100, 31, 32, 33, 34, 35], where merchants and recipients of transactions would believe what they see until the fork gets resolved.

Nakamoto's consensus operates correctly assuming a synchronous network, as it requires all nodes to have knowledge of the latest blocks to correctly maintain and extend the chain. This results in consensus terminating completely when all blocks for a given height are seen by the entire network. In practice, this provides a probabilistic guarantee as increasing block height represents more of the network agreeing on the canonical chain to extend, but there is always a non-zero possibility of unseen chains being revealed after sufficient time. The pitfalls

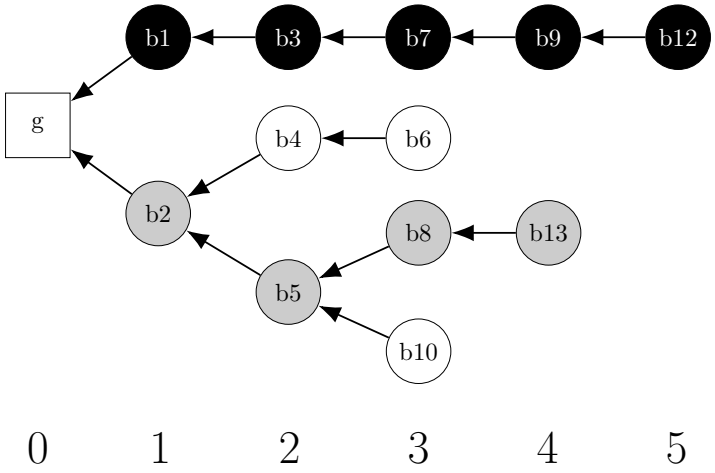


Figure 2.4: Nakamoto's consensus selects the main branch as the longest branch (in black) whereas the GHOST consensus protocol follows the heaviest subtree (in grey).

of probabilistic consensus occur during delays where the time bound of synchrony is exceeded, resulting in either agreement or termination not being reached. To account for this, Nakamoto's consensus employs an adversary model that places heavy assumptions on the block proposals. The assumption is that honest nodes produce new proposals quicker than adversarial nodes, such that the honest nodes will always be proposing a longer, valid chain. This is achieved through ensuring that the majority of the resources are owned by the honest nodes.

Ghost

Inherent impediments caused by Nakamoto's consensus prompted a proposal for a new consensus mechanism that could cater for higher speed proposals. The *Greedy Heaviest Observed Sub-Tree*, or GHOST, protocol [134] was proposed as a new consensus for fork resolution in the blockchain, aiming to alleviate wasted effort from the longest chain approach. Rather than discarding wasted efforts on transient forks, like in the longest chain, GHOST iteratively selects the heaviest sub-tree rooted at block 0, up to the current block at the leaves. Through inclusion of blocks not on the canonical chain, GHOST provides resilience to forks occurring in high throughput environments, maintaining the 50% adversary threshold at higher throughput and varying block sizes. Figure 2.4 highlights the example of the fork choice rules between GHOST and Nakamoto's consensus.

Similar to Nakamoto's consensus, GHOST operates under the assumption of synchrony, where blocks are received by all nodes. Under conditions where synchrony is not met, the heaviest tree cannot be observed and it may fail to reach agreement or terminate, as it will observe a stale state of the tree. The adversary model heavily follows Nakamoto's consensus, where the majority of the power belongs to the honest nodes, ensuring that the heaviest branch belongs to the honest nodes.

Complexity and Usage One concerning aspect of GHOST is the requirement for the sub-tree to be calculated from block 0, where it must re-calculate every possible branch. These specifications meant that long-running chains quickly became computationally expensive to perform the evaluation, so modifications were made to utilise it in practice. Ethereum originally utilised a simplified GHOST in the calculations of rewards [59], giving a percentage of reward to blocks not mined on the canonical chain but still contributing to the heaviest weighted sub-tree. As Ethereum moves away from Proof-of-Work, eliminating the requirement for computational puzzle solving, the new proposition is to use GHOST as a fork choice rule based on messages [135], performing only necessary calculations on the latest messages, rather than entire chain.

BFT consensus

With growing understanding of blockchains, and their suitability to business use-cases and inter-business settlements, there has been an adoption of consortium and permissioned blockchain networks. These environments operate with known node participation and are often lower in numbers, opening the door for traditional BFT consensus algorithms. However, to utilise BFT protocols in an open, Bitcoin-like context, some problems must be addressed. First, most traditional applications of BFT protocols require a set of participants forming a committee to exchange communication through the protocol, however, in the blockchain context this committee is generally not fixed, nor predefined. Secondly, the selection technique to find eligible participants cannot be directly taken from the original specifications of Proof-of-Work designed for Nakamoto's/GHOST. However, the chosen technique should incur some cost, to alleviate the possibility of Sybil attacks [24] that would disrupt the BFT consensus from reaching agreement or terminating. Thirdly, due to the open nature of blockchains, participants may join and leave at arbitrary times, unless specified in a consortium or private network, making quorum sizes [136] required for agreement not constant. The majority of these questions can be addressed through applying appropriate techniques for selecting participants, making BFT applicable in a number of blockchain use cases.

PBFT The Practical Byzantine Fault Tolerance (PBFT) [16] algorithm achieves consensus through leader-based communication over three phases, the *pre-prepare*, *prepare* and *commit* and was designed to bring practicality to BFT consensus. The three phases include message transfers beginning with a leader broadcasting a sequence number to all nodes in the *pre-prepare*. Once all nodes receive the *pre-prepare*, they enter the prepare phase and multicast to all nodes. If a given node receives $2f$ prepare messages, where f is the expected number of potentially faulty nodes, matching the pre-prepare, it enters the commit stage and multicasts a commit message. Upon receiving $2f + 1$ commit messages that match the pre-prepare, the message is committed. The phases are used to totally order messages and ensure that commits are ordered across all nodes.

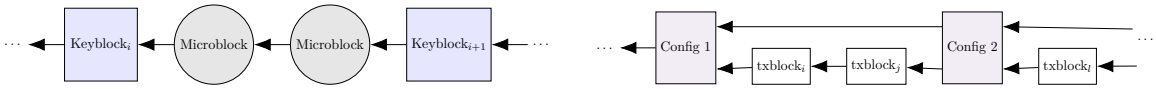
To bring BFT to the blockchain, a number of systems such as PeerCensus [28], ByzCoin [137] and Solida [138], utilise variations of PBFT tailored to their requirements to achieve consensus.

Both PeerCensus and ByzCoin assume a partially-synchronous environment and require less than a third of the nodes to exhibit Byzantine behaviour, whereas Solida assumes a synchronous environment. PeerCensus utilises a secure group membership protocol [139], allowing members to join and leave. The leader performs PBFT-based consensus and is responsible for committing the transactions and blocks. ByzCoin builds upon this by adding scalability through a collective signing scheme, CoSi [140] and performing PBFT-based consensus in a fixed committee. The modified PBFT runs using a two-phase PBFT protocol. Solida, however, uses Proof-of-Work to provide a rotating committee, where old members are evicted and new miners are added into the committee.

Tendermint Similarly, Tendermint [106, 107, 141, 142] approaches BFT consensus through a leader-based Byzantine Agreement, where selected nodes take turns proposing a new block. Each round consists of the proposal of a block, nodes voting on the proposal, and finally, if over two-thirds of the nodes vote on the same block, it gets committed and appended to the blockchain. The consensus requires a locking mechanism to ensure that a selected node cannot double-vote. A node will lock their vote if they have pre-voted or pre-committed a specified block but can unlock and change their vote if they see greater than two-thirds voting for a different block in the same round. The vote change is permitted to protect liveness, but must be executed such that it does not compromise safety, therefore only performed after two-thirds of the committee have voted to commit a block. Like PBFT, Tendermint operates under partial-synchrony, where it utilises increasing timeouts for the proposal rounds. It assumes that more than two-thirds of the participants are honest, resulting in $3f+1$ nodes, halting if this condition is not met due to network partitions, or otherwise.

Algorand BA$\star$ Algorand [42, 143] utilises the BA$\star$ providing efficient probabilistic Byzantine Agreement. The members of the consensus committee are chosen through randomisation provided by a blockchain-based common coin, which is also utilised in the consensus rounds. BA$\star$ provides two phase consensus under synchrony assumptions, terminating in 6 rounds on average with worst case of 13 rounds utilising the common coin when decisions are not met. The first phase consists of a leader proposing a block, which is then reduced to a binary decision to run binary consensus. The second phase is the voting on the binary value to reach agreement, assuming that more than two-thirds, or $2f+1$ nodes are honest. Although requiring synchrony for agreement, it can achieve safety in periods of weak synchrony where the network is asynchronous for a bound of time.

HoneyBadgerBFT Unlike others, the HoneyBadger BFT [144] aims at providing consensus specific to permissioned environments with an asynchronous network. However, although the network delay is unbound, HoneyBadger assumes that there exists reliable channels between nodes that guarantee the delivery of a message once it has been placed into the queue. HoneyBadger utilises a reduction of multi-value consensus to a binary consensus, using a common coin for randomisation as a circumvention technique to utilise the asynchronous network. The protocol begins by each node selecting a random set of uncommitted transactions, encrypt-



(a) Bitcoin-NG Block Approach (b) ComChain (Community Blockchain)

Figure 2.5: Heterogeneous block structures.

ing with threshold-based encryption [145] and disseminating it to all nodes through reliable broadcast [22]. For each message, the node produces its own part of the decryption share and broadcasts it. Once receiving $f + 1$ shares, the node performs the threshold-based decryption which, if succeeds, can be treated as accepted and committed to the canonical chain. This BFT protocol, alike others, assumes that the honest nodes are greater than two-thirds of the participants, therefore being $n > 3f$. However, it was found that does not terminate under an adversarial scheduler [146].

DiemBFT The DiemBFT [147] is a leader-based BFT protocol that builds upon concepts from PBFT [16], Tendermint [106, 107] and Ethereum's Casper [148] and is heavily founded on HotStuff [149]. The DiemBFT utilises a three phase consensus protocol, namely *prepare*, *pre-commit* and *commit*, where a leader collects votes to progress through the stages. The collection of votes are used to form a *Quorum Certificate*, which is signed data containing the proof of message reception and votes to progress the rounds and eventually commit the data. The BFT applied to a blockchain requires a new leader for each block, where the leader proposes a block that is then voted upon by other members of the committees. The DiemBFT allows for a reconfigurable consensus committee to change in defined epochs. As this protocol is heavily derived from HotStuff and PBFT, it requires the honest nodes to be greater than two thirds of the participants, so $n > 3f$, to progress and reach agreement.

IstanbulBFT Introduced through an Ethereum Improvement Protocol [150], Istanbul Byzantine Fault Tolerance (IBFT) aimed to provide a BFT consensus to Ethereum for deployments in small private or consortium environments. The core goal was to provide a modification of PBFT [16] that was tailored for the blockchain, meaning no one single client is sending requests, rather the participating validators are able to propose blocks, and the leader is chosen in a round-robin fashion. Now integrated into the Quorum blockchain [55], the IBFT protocol provides a tailored consensus for a small consortium blockchain, where changing membership occurs through reconfiguration. Following the three phase commit protocol, *pre-prepare*, *prepare* and *commit*, the participants engage in a leader-based all-to-all communication to progress through rounds. As in PBFT, IBFT tolerates at most f faulty nodes, where $n > 3f$, requiring honest participants to account for over two thirds of the total. However, external analysis of Liveness [151] and Correctness [152] highlighted that IBFT failed to provide liveness in an eventually synchronous network, and there has been proposed improvements to the protocol, such as IBFT 2.0 [153].

RapidChain's BFT Although other Byzantine Fault Tolerant consensus for blockchains is widely applicable to changing environments, RapidChain [154] tailored a consensus appropriate for blockchains that apply the technique of sharding. Heavily iterated from Ren et al.'s Practical Synchronous Byzantine Consensus [155, 156], the RapidChain consensus provides a leader-based four round consensus tolerating f Byzantine members where $n > 2f$. The consensus begins with a *propose* and *echo*, where all participants receive messages from all others. In the case a participant receives a threshold of matching proposals, it will move to accept the proposal and progress through the rounds, however, in the case that it receives equivocating proposals from a faulty leader, it will propose to commit an empty root. To further improve the performance of the consensus, the authors propose to pipeline, which allows for blocks to be proposed at every round, rather than waiting for all rounds to complete.

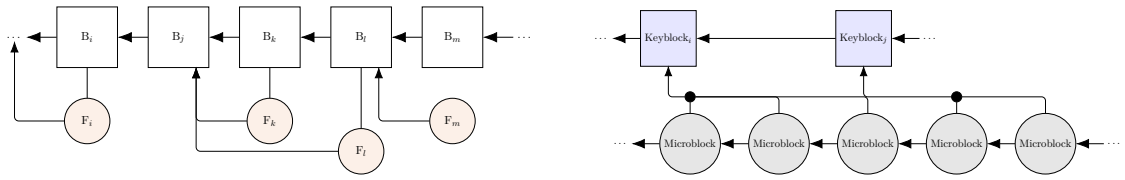
Avalanche/Snow Unlike other blockchain BFT consensus, Avalanche [157] presents a family of leaderless BFT consensus with probabilistic guarantees. Inspired by Nakamoto's consensus in Bitcoin, Avalanche utilises a Directed Acyclic Graph (DAG) structure where each node calculates a local consensus achieving safety with probability $1 - \epsilon$, where ϵ is a security parameter defined by the system. However, the Avalanche/Snow consensus proposal was not proven to work in the partially-synchronous model, as most other listed BFT solutions are. The protocol states that a node uniformly selects random nodes in the network, interacting and gossiping their chosen decision. The node maintains a counter to see if a threshold of other nodes have selected the decision, and will decide once confident that the node set has reached a specified threshold. The DAG is used as the data structure to store transactions and gain confidence on decisions.

2.2.3 Structure

Originally, the Bitcoin blockchain introduced a single, canonical structure to provide the progression of state. All transactions in the system were batched into blocks, where all blocks were a homogeneous type. This meant that transactions were limited to the size and speed of block production. Transient branches, or forks, must be resolved to form a single, canonical chain which requires calculations and resources. New proposals, however, provide variety in the structure by adding new block types or alternative structures to modify the operation of the blockchain. The structure provides the foundation for the consensus and membership, as it sets the requirements of what is decided upon, how the decision is made, and how the system progresses.

New block types

Some proposals focus primarily on modifying the block types, moving from a single, homogeneous block type, as described in the seminal Bitcoin chain, to heterogeneous blocks with alternate compositions.



(a) FruitChains (b) ByzCoin Microblocks

Figure 2.6: Alternative Structures I.

Bitcoin-NG Bitcoin-NG [158], depicted in Figure 2.5a, introduced the context of two block types, namely *KeyBlocks*, produced through Proof-of-Work containing no transactions but used for leader election to batch transactions, and *Microblocks*, containing transactions and produced without Proof-of-Work proposed by the elected leader linked to the Keyblock.

ComChain ComChain [159, 160], proposes a *configuration block* that is used to form the next consensus committee, as shown in Figure 2.5b. The new committee is proposed, and once agreed is formed into a block whereby the new committee is tasked to batch and agree upon transactions.

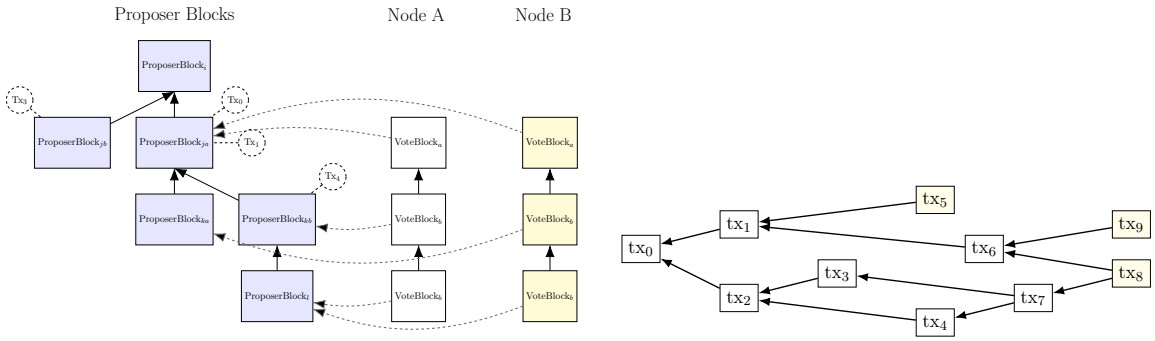
New structures

Rather than proposing only changes to block types, a number of proposals turn their focus to structure, diverging from a single canonical chain of blocks. This presents new opportunities for parallel block processing and alternate communication patterns between nodes for potential performance gains.

FruitChains Fruitchains [161] introduces an alternative structure by decoupling transaction processing from blocks. A *fruit* contains a batch of transaction that hangs from one block, referring to a recent block to show its context. Figure 2.6a depicts the referencing of the fruit to the chain allowing transaction processing to be separated from blocks.

ByzCoin and RepuCoin ByzCoin [137] was inspired by the work in Bitcoin-NG [158] providing a decoupled structure for microblocks and keyblocks. The microblocks are separated into their own chain, and relate back to the keyblock, as shown in Figure 2.6b. RepuCoin [99] also uses these features by decoupling transaction blocks using the microblocks for transactions and keyblock selecting the consensus committee.

Prism Prism [162] deconstructs the blockchain into separate structures introducing proposer and voter blocks. Each node builds its own block tree. The proposer blocks are appended into a block tree, similar to that of Bitcoin's blockchain, but transactions are partially decoupled from the proposed blocks. Voter blocks are used to attest to the proposer blocks and gossiped around the network. An example is depicted in Figure 2.6c.



(c) Prism DAG with 2 block trees (d) IOTAs Tangle

Figure 2.6: Alternative Structures II.

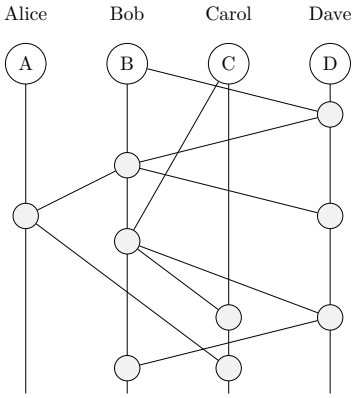
Tangle Iota presents the *Tangle* [53] which forms a DAG and removes blocks. As illustrated by Figure 2.6d, the transactions form vertices in the graph and require each vertex to have an edge between two (or more) prior transactions. A transaction's weight, or confirmation, is proportional to the number of transactions that it references, with respect to the recency and weight of the other referenced transactions.

HashGraph Swirlds [163] proposes a graph structure, where each transaction batch has multiple children and parents. The *HashGraph* was introduced to reduce wasted effort of blocks, so each block, or event, gossiped around the network is then voted upon by other nodes. Figure 2.6e represents the relationship between events, showing that events should reference previous events to vote for them. Although nodes may learn of events in differing places of their graph, an event is said to be consistent if it has equivalent ancestors in all node's graphs.

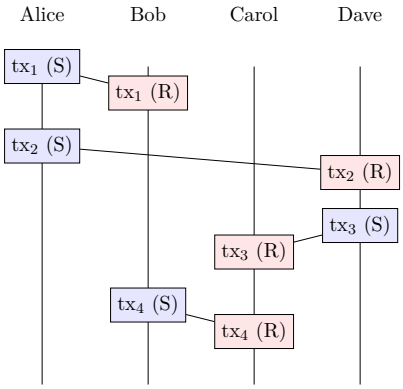
Block Lattice RaiBlocks [54] introduces the block lattice, as depicted in Figure 2.6f. The distinguishing feature is that each account maintains its own personal blockchain, the DAG ledger consists of the global set of accounts and their relevant blockchain. Each transaction sent from user to user requires two transactions, a "send" transaction signed by the transaction sender, and a "receive" transaction, signed by the party receiving the transaction. Similarly, each party must create a block on their respective chain, a "send block" and "receive block".

Parallel Chains Parallel Chains [88] proposes the idea of sharding by distributing the blockchain into multiple concurrent chains. Transactions are spit amongst the shards, as shown in Figure 2.6g. Similarly, Omniledger [89], and RapidChain [154] also use this concept of multiple chains operating in parallel. Although multiple chains operating in parallel facilitate concurrent transaction execution, the system must provide mechanisms for cross-shard interactions.

Beacon Chain Ethereum 2.0 [87], also known as Ethereum Serenity, introduced the model of the *Sharded Beacon Chain*, where concurrent shard chains run in parallel, communicating through a central beacon chain in a hub-and-spoke design as depicted in Figure 2.6h. DFinity [41] envisioned a chain where a set of nodes are selected through a *decentralised randomness*

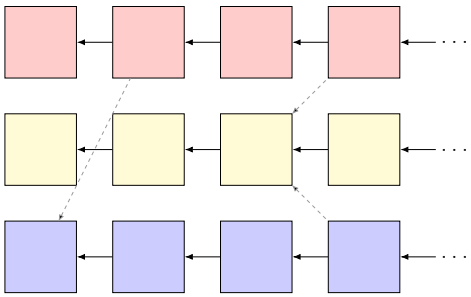


(e) HashGraph Gossip

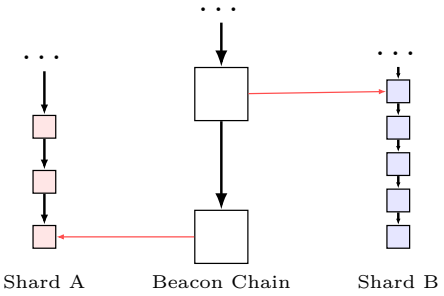


(f) Block Lattice

Figure 2.6: Alternative Structures III.



(g) Shard Chains running concurrently



(h) Ethereum Serenity's Beacon Chain

Figure 2.6: Alternative Structures IV.

beacon and were tasked to notarise and propose blocks. Ethereum utilises a similar beacon concept to shuffle and reassign validators to shards, where the beacon chain offers cross-shard communication and a central point of synchronisation for the entire system.

2.3 Performance evaluation

With a vast flurry of blockchain proposals, evaluation of their impact is of paramount importance. To achieve accurate evaluations, it is common to perform experiments that present performance metrics. These often consist of three axes; *throughput*, *latency* and *scalability*.

Throughput. Throughput measures the volume of transactions a system is able to process, often expressed in *Transactions Per Second (TPS)*. The throughput represents the number of transactions that are committed to the chain, where committed is often denoted as the transaction existing on the blockchain at a point where it will not be reversed or removed[2].

[2]In some blockchains that have probabilistic guarantees, committed refers to the point in time where the transaction will not be reversed with extremely high probability.

Latency. Latency expresses the duration of time for a transaction request to be committed. It is often measured from the time the request is sent, to the time that the transaction has been committed and acknowledged by the requester.

Scalability. Scalability often presents itself as the ability for a system to offer the same, or increased, performance with an increasing number of nodes. This is often measured with respect to the other two axes, showing that larger node sets lead to no degradation of performance.

2.3.1 Benchmark systems

This burst of proposals often leads to confusion when deciding upon a system to use in a given environment. Performance evaluation assists in highlighting advantages of specific systems as well as providing metrics for quantitative comparison. Most of these evaluations, however, are tailored specifically for the system under test, where it targets optimal environments most suited for the deployment. This leads to issues when comparing systems, as published results, both in academic papers and social media [43, 44, 45, 46], would be based on varying environments and can lead to misleading conclusions.

To remedy this issue, similar fields were examined, providing previous studies for both performance evaluation and fair comparison. This quickly highlighted previous work performed in both systems programming [164, 165], distributed systems and databases [166, 167, 168] where specific benchmark platforms were used as the basis for comparison. The aim was to procure techniques to provide benchmarks at a macro and micro level. Macro benchmarks provide system-wide tests, where the entire system is tested end-to-end in environments mimicking deployment environments. Micro benchmarks, however, provide a more granular approach targeting specific aspects of the system measured in controlled environments, allowing metrics of core building blocks to be evaluated.

Similarities between database systems and blockchains were quickly revealed, as both systems present themselves as a forefront to perform user requests, writing or reading the state of the global system and performing actions based on this state. Conveniently, distributed database systems also looked at the measurement of throughput and latency and provide extensive benchmarking solutions. Workloads and frameworks such as the *Yahoo! Cloud Service Benchmark* (YCSB) [166] and various TPC-*[3] [167, 168] benchmarks are provided as a standard. The workloads in these benchmark suites aim at providing real-world testing scenarios, designed to stress various aspects of the systems under test to highlight the key differences. In conjunction with strict testing rules, the results can be used for fair comparisons to easily determine which systems were most suitable for certain workloads, and can be applied in their respective use-cases. These heavily researched benchmarks and workloads were attractive for blockchains, as the complexity produced by numerous proposals made it difficult to measure and understand their suitability to certain environments.

[3] Available online: http://www.tpc.org

Blockbench One of the most notable benchmarking frameworks for blockchains is Blockbench [50], a framework that provides a number of micro and macro benchmarks, but more notably ports YCSB from the databases field to blockchains. Aiming at private blockchains, Blockbench evaluates the different layers of the blockchain with different workloads, allowing granular execution of test data to measure the effectiveness of different implementations on the specific layers of the blockchain. The evaluation metrics allow for throughput and latency to be measured with respect to scalability of increasing nodes, and fault tolerance with injected delays, crashes and message corruption.

However, the requirements introduced from the complexity in Blockbench leads to rigidity in workload definitions and the integration of other blockchains. This rigid design also introduces difficulty when performing benchmarks of high throughput, as the workload is sent from a single point of the benchmark.

Chainhammer The extensive number of Ethereum adaptations led to the development of Chainhammer [48], a benchmark tool focused exclusively on high-load testing of Ethereum-based blockchains. The high-load workload generation aims to stress test the blockchain to measure the maximal number of transactions it can process under constant load. All transactions originate from a single machine, which measures throughput and latency of the blockchain under test. However, there is minimal flexibility to adopt different testing scenarios or alternative workloads, leading to a rigid design. As Chainhammer focuses exclusively on Ethereum, it can perform post-benchmark analysis that provides information about critical Ethereum infrastructure, such as block and transaction distribution as well as various intricacies with transaction processing with respect to gas and execution costs.

Hyperledger Caliper The Hyperledger foundation developed Caliper [47], a benchmarking suite that focuses on custom use cases on a number of blockchains. While still under active development, Caliper comes preconfigured with various blockchains, including Hyperledger Burrow [169], Hyperledger Besu [170] and Hyperledger Fabric [171, 172] and Ethereum [27]. New blockchains can be easily integrated through implementation of generic interfaces, factories and transaction definitions. The output produced from Caliper presents information about throughput and latency, and the active development is further integrating more results. Development is shifting towards distributed load generation across multiple machines, as the current implementation only permits multiple threads being run on a singular machine to interact with the system under test.

Twins Alongside performance evaluation, understanding system behaviour while they experience unexpected conditions is critical. The Twins [173] approach emulates Byzantine behaviour by running multiple clones of a node, with the same identity, in the network. By generating interesting behaviour, such as multiple proposals or equivocation, it demonstrates the resilience of systems, highlighting the ways varying protocols break. This performance evaluation method

demonstrates the ability to discover protocol flaws in BFT consensus that could hinder the system during operation.

BFT Bench BFT Bench [174] provides a framework to evaluate the performance of BFT consensus protocols. The framework provides a number of pre-defined BFT consensus algorithms, allowing for testing under varying deployment settings. It also allows for the injection of unwanted behaviour by reporting and observing metrics on performance and reliability. While not directly evaluating blockchains, the BFT bench framework tests an number of consensus protocols used for agreement of blocks in a number of implementations. Through the measurement of critical metrics, such as latency and throughput during periods of unwanted behaviour, it provides insight into the resilience of a system under faults and how they can be applied in practice.

BlockSIM Unlike benchmark frameworks, BlockSIM [175] provides a practical simulation tool to aid in the design of new blockchain systems. The core focus is to present an extensible framework allowing for a range of system models to be simulated through a python system. The simulator breaks the blockchain into three layers, the network layer, the consensus layer and the incentives layer, each playing a critical part in the design. The simulation presents an architecture that interacts with the blockchain, providing measurements on the operation of the blockchain in the simulated environment.

2.4 Attacks against blockchains

Although performance is a critical factor motivating research and development in blockchains, there numerous safety concerns that must be addressed due to assumptions that lead to limitations in practice. Various attacks have been theorised, detailing scenarios that affect a range of blockchain protocols, which allow an adversary to double-spend coins. The attacks were categorised into three key areas; mining power, communication and stake attacks.

2.4.1 Mining attacks

Mining power attacks target systems that utilise work-based protocols, such as Proof-of-Work, especially chains that are susceptible to forks. These attacks focus on an adversary gaining sufficient amounts of mining power to control perceptions of the truth. To improve a miner's chance of profiting from mining, they often collude to form pools, allowing cost sharing of profits and losses.

51% Attack The most notable attack is the majority mining attack, commonly referred to as the *51% Attack* [9, 100]. It depicts an adversary that overcomes assumptions that the majority of the mining power belongs to honest nodes. This gives the adversary the ability to produce valid blocks quicker than other nodes on the network, meaning they are able to control the behaviour of the entire chain, including reverting and rewriting history.

Bribery Attack (Flash Attack) In large blockchain networks, gaining significant power is often infeasible, requiring extensive investments of hardware and infrastructure to support a large scale attack. However, the Bribery Attack [35], or Flash Attack, consists of an adversary temporarily gaining the majority power, providing a window where they may launch an attack. The methodology describes an adversary bribing miners to lend computational power to the adversary, while the adversary misleads a merchant. The adversary, as they have obtained enough mining power, can now produce alternate chains to ensure that the transaction to the merchant is overwritten and the coins have been double-spent.

Selfish Mining Often, however, mining pools control significant percentages of mining power due to pooled resources. The Selfish Mining technique [176] portrays mining pools producing blocks in secret, keeping newly produced blocks from others in the network. The pools may selectively release information of blocks to waste the honest miners computational power on producing old blocks. Extensions to this work [177] optimise the strategies for selfish mining to show optimal and sub-optimal conditions for success. These strategies enable the mining pools to create alternative chains within their own pool and release a chain longer than the public branch.

Stubborn Mining Selfish mining alone may be impractical for real-world applications, with high requirements for lower profit margins. Stubborn Mining [31] builds upon selfish mining techniques introducing more strategies to increase probability of success. The strategies include *Lead Mining*, where the adversary leads in secret by k blocks and releases blocks at the same time as the network, and *Trail Mining*, where the miners continue to mine until they fall behind some threshold of blocks. These strategies, if combined with other network attacks, can further increase the percentage of success.

Rosenfeld's Attack A variant of strategic mining is Rosenfeld's proposed attack [34], where nodes withhold block production from other nodes until convenient. The adversary solo mines a branch in secret, keeping information from other nodes. Once the adversary interacts with a selected merchant, they issue a transaction for some irreversible action. Once the merchant has completed the action, the adversary releases the longer, solo-mined branch conflicting with the main branch, which will be accepted with high probability as it is longer. The adversarial branch does not contain the transaction to the merchant and thus invalidates the payment to the merchant.

Finney Attack Alternatively, non-pooled miners may choose strategies to deceive merchants. The Finney Attack, proposed in an early Bitcoin forum [32], illustrates an attack where an adversary double-spends to a merchant that does not wait for block confirmation. The adversary engages with the merchant and sends coins for the service or goods that are irreversible. At this point, the merchant completes the trade. Simultaneously, the adversary then creates a transaction to themselves and mines a block, which includes this transaction. The adversary then quickly broadcasts the block with the transaction to themselves. If quick enough, the

transaction to the merchant will conflict, and the transaction back to self will be confirmed in a block and accepted by the network.

Fast-Payment Double-Spending Similar to the above, Double-Spending attacks on Fast-Payments [29] leverages a merchant's acceptance of transactions with 0 block confirmations. *Fast Payments* are payments where a transaction is considered accepted within a minute of it being proposed, whereas *Slow payments* are those that require a block confirmation threshold. The adversary connects to the merchant as a peer, sending a transaction for the merchant's irrevocable action. As the adversary is a direct peer, it can send a transaction directly to the merchant while simultaneously sending a conflicting transaction to others in the hope that the conflicting transaction is propagated quicker and mined into a block.

Vector76 Attack The user "vector76" on BitcoinTalk forums proposed an attack [33] where an adversary wishes to double-spend against a merchant that has a 1 block confirmation threshold. The adversary interacts with a merchant, and secretly mines the transaction to the merchant into a block. When the adversary learns of a block, they quickly send their block to the merchant directly, who confirms the block threshold and performs the irrevocable action. As the merchant learns of new forked blocks, which will discard the adversary's transaction in an alternate chain, the adversary would have successfully double spent the coin.

Block Withholding Attack against Mining Pools The Block Withholding Attack [178] targets mining pools where an adversarial miner aims to cause disruption. Mining pools operate by producing full or partial Proof-of-Work. An adversary participating in the pool can determine the validity of their Proof-of-Work hash, and thus can withhold it from the pool. If the adversary aims to disrupt the pool, they can withhold valid hashes to cause the pool to miss out on block production, lowering overall mining pool rewards.

2.4.2 Communication attacks

Alternative to mining power attacks, adversaries can target assumptions on the peer-to-peer network of the blockchain to attack a wider variety of systems. The assumptions placed on communication between nodes and message delivery can often be misleading, and if abused by an adversary can lead to loss of assets.

BGP Routing Attack The Border Gateway Protocol (BGP) [179] is the routing protocol responsible for connecting systems and routing IP packets through the internet. As blockchain utilises peer-to-peer connections over the internet, BGP is inherently relied upon to route packets between nodes. However, in BGP, the validity of route advertisements are unchecked, as systems promote routing paths between each other. The BGP Routing Attack [180] proposed against the Bitcoin blockchain exploits the lack of BGP verification to isolate a selected group of nodes. Once a group of nodes has been successfully isolated, the adversary controlling the hijacked BGP route can delay block propagation or leave nodes uninformed to use their mining

power for their benefit.

However, BGP attacks are considerably difficult, as controlling an autonomous system, or successfully maintaining a hijacked route, can be seen through various tools and acted upon [181]. A number of solutions for blockchains, such as SABRE [182], FIBRE [183], and Erlay [184], provide overlay networks to provide alternate communication channels if BGP routes are interrupted.

Erebus Attack Alternative to BGP hijacking, the Erebus attack [185] utilises an Autonomous System to change outgoing peer connections on the Bitcoin node. This does not require BGP route hijacking, and remains stealthy as it performs connection rerouting through *shadow IPs*, which are old IPs harvested from other autonomous systems. The adversarial Autonomous System will continuously attempt to reroute a Bitcoin node's peering such that it places itself in the middle. By doing so, it will be able to insert, modify, remove or delay any incoming messages and effectively control the victim node's communication to others.

Eclipse Attack BGP hijacking is a considerably difficult task as the adversary is required to maintain a hijacked route between subsets of isolated nodes. However, the blockchain utilises a decentralised peer-to-peer network, where all nodes communicate through a gossip network and build a list of peers to form direct connections. A connected node stays connected to a limited number of peers, in Bitcoin 125, and Ethereum 25, which are evicted and reconnected depending on their performance. The Eclipse Attack [36] describes a way in which a node can become eclipsed and isolated from the network. The attack method follows an adversary filling the peer tables of a victim node, spoofing as many IPs as possible. If the adversary has successfully populated a large percentage of the IPs in the peer table, the node will connect all peers to the adversary. The adversary can now control the information seen by the victim, and is now able to execute double-spending attacks or delay attacks. Eclipse attacks against Ethereum have been proposed [186, 187, 188] targeting a number of network assumptions with the discovery protocol or the rotation of connected peers. Countermeasures have been suggested and implemented [189, 36], to become resilient to the attack.

Man-in-the-Middle Attack The Man-in-the-Middle (MitM) attack [190] proposed against Ethereum illustrates the applicability of BGP hijacking to double-spend on public blockchains. Through BGP hijacks, the adversary is able to manipulate communication delays between nodes and form subgroups. Once the adversary is successfully in control of the communication, they can leverage this to perform other attacks. Similarly, the Man-in-the-Middle attack is also effective in the private blockchain context using other spoofing techniques to gain control of routing, such as ARP spoofing in a private context.

Attack of the Clones Proof-of-Authority blockchains [27, 119, 120] are increasingly popular for small private or consortium environments. It operates by rotating between elected authority nodes, requiring the set of nodes to sign off and seal the proposed blocks. The Attack of

the Clones [191] is carried by segregating the network into smaller subgroups, but cloning an instance of their machine so that a threshold is met on each side to produce valid blocks. At this point, the adversary is able to double-spend on each partition, as the segregated groups of nodes are producing valid blocks.

Tampering Delay Attack Gervais et. al [192] highlight how an adversary can successfully delay blocks and transactions to victim Bitcoin nodes for a considerable amount of time. The adversary provides information to the victim nodes before they are able to receive any information from others. With optimisations in Bitcoin, this means that the victim nodes will request additional information and await a response from the adversary. The success of the attack relies on the adversary being able to broadcast information and interact with victim nodes quicker than others. The result is that victim nodes may have blocks delayed up to 20 minutes.

2.4.3 Stake attacks

With the growing interest in Proof-of-Stake, active research is searching for vulnerabilities which would allow adversaries to gain power or profit. Although Proof-of-Stake shows immediate improvements to traditional Proof-of-Work blockchains, it does, however, suffer from a different breed of attacks.

Nothing At Stake One of the first attacks on Proof-of-Stake was dubbed the "Nothing at Stake" attack [193], where rational validators strategically play on the protocol's specification for profit. Proof-of-Stake relies heavily on validators following protocol to earn rewards, and they must collaborate to progress the chain's global state. However, in the case of a fork, the most optimal strategy for validators is that they should attempt to validate all possible chains for a higher chance for reward. Adversaries are able to leverage this behaviour, and with non-zero probability successfully execute a Double-Spending attack. The adversary creates a transaction to a merchant, while simultaneously creating a fork in the chain with a conflicting transaction. If validators are following the above strategy to validate all forks, the adversary's chain may be selected as the correct chain. To mitigate these problems, validators must commit stake to sign and create blocks, acting as a disincentive for the validators to sign on all forks, and they are punished for signing on multiple forks. Alternatively, unforkable blockchains also present a viable solution, preventing forks from being created.

Discouragement In many Proof-of-Stake implementations, the entire validator group is punished during sub-optimal conditions, such as disagreements. An adversary can utilise this in a Discouragement Attack [194], where an adversary purposefully acts maliciously to punish the entire validation set. With the adversary broadcasting their intention to harm the other validators, honest validators will become aware of their potential loss of stake and would withdraw themselves from the committee. Eventually, the number will diminish to the point where an adversary gains the majority and can perform other attacks.

Censorship Rather than validators acting for personal gain, adversarial collusion can lead to validators dictating the blockchain. Censorship [195] can occur where validators censor certain transactions from being added to blocks. The ramification may lead to the collapse of the blockchain ecosystem, as it breaks the fundamental principles founding the blockchain. If adversarial validators are only a subset of the validators, they can leverage discouragement techniques to lower the number of honest validators. Once the adversary makes up the majority of power, they are able to censor the transactions.

Stake Grinding Various implementations of Proof-of-Stake, such as Peercoin, or PPCoin [102], and NXT [196], select validators to propose blocks based on calculations that can be precalculated ahead of time. The Grinding Attack [108, 197], or precomputation attack, outlines the ability for an adversary to influence the selection of validators by precomputing and "grinding" outcomes to increase their chance of selection. Known solutions exist [102, 197], such as improved distributed randomness techniques with secret sharing or threshold signatures, or, having validators stake more coins for their selection.

Stake Bleeding To mitigate some issues faced by Proof-of-Stake, checkpointing was introduced as a measure to "solidify" the blockchain state eliminating the probability of forks prior to that checkpoint. Chains that fail to integrate checkpointing can become vulnerable to Stake Bleeding Attacks [198], where adversaries can double-spend by convincing the network of an alternate chain. Dependent upon fork choice rule, or consensus mechanism, an adversary creates a parallel chain and creates blocks periodically. When the adversary is selected to propose in the "honest" chain, it refuses to produce a block and waits for the timeout, which will inevitably slow the chain down. Over time, although the adversary is losing, or bleeding stake, the adversary's private chain will catch up, and once it surpasses the honest chain, it can relay the blocks and the information to the remainder of the network. Due to the fork choice rule, they may be able to convince the network about the adversarial chain.

Although checkpointing is effective, other mitigations [198, 108, 102] exist, including context sensitive transactions, chain density measurements, or coin age which requires an investment of time.

Long Range The Long Range Attack [199, 200] was originally theorised during early Ethereum development. The attack details events where an adversary can double-spend by mining a forked chain from an incredibly early block, such as block 1, and mine a longer chain to become accepted as the superior fork. In traditional Proof-of-Work scenarios, this attack would be incredibly costly, as it requires a majority of the networking power to overthrow the chain from such a long distance. The adversary can slow the progress of the main chain by submitting work to a smart contract that has a long function execution and high computation for a predetermined output known by the adversary. The adversary can continuously produce blocks whereas the honest chain will be required to perform the computation. Proof-of-Stake, however, removes the large hardware requirements, and an adversary with as little as 1% of the coins is

able to perform a long range attack.

Mitigations [200, 197] exist, such as using timestamps as rejection criteria, or a larger threshold, such as over 30%, are required to sign blocks periodically to progress.

Resource Exhaustion Although Proof-of-Stake reduces the strain on hardware in comparison to Proof-of-Work, there are a number of calculations and costly operations that must be performed to uphold the chain. The Resource Exhaustion Attack [201] demonstrates how a number of Proof-of-Stake implementations fill RAM and disk space when dealing with forks and potential future blocks. To ensure that a correct fork choice rule is being followed, the blockchain nodes are required to store the potential blocks mapping the indices to the block heights. This can quickly lead to resource exhaustion, quickly filling space on the machine and leading to a denial of service of the node.

Balancing Attack on Gasper The Balancing Attack on Gasper [202] depicts an attacker that strategically releases votes to ensure a growth of a forked chain. By balancing votes amongst two forks, honest nodes will be stuck in a tie-break scenario, where they either (1) vote for a single branch, or (2) sway between branches in an attempt to get more than two-thirds of the votes. If the adversary can sufficiently maintain the tie-break, this will delay block finalisation and lead to liveness issues.

Bouncing Attack Similarly, the Bouncing Attack targets finalisation in Ethereum 2.0 [203], where an adversary directly influences the finalisation mechanisms. If an adversary has enough vote power, such that they have the final say whether to ensure a block is committed, they can submit their votes at any time to influence which branch is considered correct. The attack scenario depicts a fork that has sufficient votes for a given checkpoint (C) on a single branch, and on the same branch is about to have sufficient votes for the second checkpoint (C“). At this point, the adversary releases votes for an alternative checkpoint (C‘) on the opposing branch, causing conflict between honest validators who see a swap in the voting for branches. This will cause the honest validators to bounce between branches and cause liveness issues in the system.

Chapter 3

The Balance Attack

The introduction of blockchain systems through whitepapers and non-scientific channels can lead to implied assumptions not clearly stated in writing. In Proof-of-Work blockchains, nodes are competing amongst each other to produce a block, with each node having potentially different sets of transactions as input to the computation. Due to the highly distributed nature of these operations, concurrent proposals may lead to completely distinct blocks being proposed for the same index of the chain, resulting in a forked state. If an adversary is able to solve Proof-of-Work puzzles fast enough to grow a targeted branch, they will not only have the ability to impact the accepted state of the system, but potentially increase the amount of wasted blocks, delay block release, and slow the overall growth of the system. These delays present risks, where an adversary may be able to double-spend coins spent in other transactions [34]. As highlighted in Chapter 2, a chain that experiences transient forks as a result of disrupted networking can be vulnerable to a number of double-spending attacks.

A simple attack, the *Balance Attack*, is presented. An adversary transiently disrupts communication between subgroups of nodes of similar mining power to forcefully introduce forks in the chain. At this point, the adversary actively submits conflicting transactions to separate subgroups while assisting a targeted chain to produce more blocks, manipulating the fork choice rule to select this specified branch. The feasibility of such an attack on smaller networks is demonstrated both theoretically and experimentally, highlighting that an adversary can compensate lower mining power through the message delays.

In summary, the work presented in this chapter is:

(i) We highlight the potential impact of implied synchrony assumptions when in conjunction with relaxed agreement in the blockchain context.

(ii) We present the Blockchain Anomaly, a resultant of chain reorganisation experienced after periods of long delay where a fork resolution is required.

(iii) We show that both Nakamoto's consensus and Ghost are *corruptible*, or vulnerable to double-spending when faced with delays in communications. We introduce the Balance Attack that influences branch selection rather than an adversary solo-mining.

(iv) We illustrate experimentally and theoretically in the context of the R3 Ethereum testnet if a single node is able to delay communications. The impact of heterogeneous machines that are exacerbated in small consortium environments is highlighted.

(v) We discuss the trade-off between mining power and required communication delay to perform the Balance Attack against Ethereum. This suggests that combinations of network and application-level attacks increases the probability of success.

(vi) We argue for non-forkable blockchain designs in the face of network partitions to protect against attacks similar to the Balance Attack. This not only provides complete resilience against fork-based exploits, but increases the applicability of digital ledgers to critical sectors where forking and loss cannot be tolerated.

Chapter Outline This chapter is organised as follows. Section 3.1 presents a simple distributed model for the blockchain, describing the communication network and the blockchain graph. Section 3.2 discusses the impact of consensus and termination on transactions and introduces the Blockchain Anomaly. Section 3.3 introduces the Balance Attack. Section 3.4 discusses the Balance Attack when applied to the GHOST fork choice rule. Section 3.5 discusses the feasibility of the attack in small and large networks. Section 3.6 provides our analysis of the attack when applied to a small blockchain network. Section 3.7 provides experimental analysis of the Balance Attack. Section 3.8 discusses potential solutions. Section 3.9 summarises the chapter and concludes.

3.1 A simple distributed model for blockchains

Consider the blockchain network as a communication graph, $G = \langle N, E \rangle$, where N represents the nodes of the blockchain, and all nodes are connected through communication links, or edges (E). Nodes are either *miners* trying to combine transactions in a block to progress system state, or, passive and interacting with the blockchain by issuing transactions. All nodes are part of a blockchain system (S) where $S \in \{\text{bitcoin}, \text{ethereum}, \text{etc.}\}$. For simplicity, it is considered that each node possesses a single account, and each *transaction* issued by some node $p_i \in N$ is a transfer of digital assets from the source account p_i to the destination account $p_j \neq p_i$. Each transaction is uniquely identified and broadcast to all nodes in a best-effort manner through gossip communication. We assume that a node wishing to transfer multiple times can create many distinct transactions.

To the computational power of a miner is referred to as its *mining power*, denoting the total mining power of the network (Ψ) as the sum of the mining power of all miners. All miners try to batch a set of transactions T it learns about through gossip into a block $b \subseteq T$ as long as the transactions of b do not conflict with one another, and, that account balances remain non-negative as a result of applying this state transition from the block.

For the sake of simplicity, the graph G is static, meaning that no node can join and leave

the system, however, nodes may fail as described in Section 3.1.3. Considering a network potentially shared by other applications and subject to contention, it is assumed that the system is *partially synchronous* in that there is a bound on the delay of messages but that this bound can be large and is not known by the algorithm [23].

In the remainder of this chapter, the blockchain DAG is referred to as a *tree* rooted at g, the Genesis, where child blocks point to their parents.

3.1.1 Mining goals

As there is no known best strategy to solve the Proof-of-Work crypto-puzzle, the miners simply keep testing randomly chosen numbers in an attempt to solve the puzzle. The mining power is thus expressed in the number of hashes the miner can test per second, or H/s for short. The *difficulty* of this crypto-puzzle is defined by a moving threshold, limiting the rate at which new blocks can be generated by the network.

3.1.2 Communication Network

The blockchain network consists of nodes communicating through gossip communication patterns in a best-effort manner. Each node broadcasts messages and communicates directly with a set of connected nodes, named peers. The node gossips both transactions and block information, receiving new information on blocks or transactions from the direct node connections. A node maintains a limited number of connections to send and retrieve information, where a node may disconnect at any time.

3.1.3 The failure model

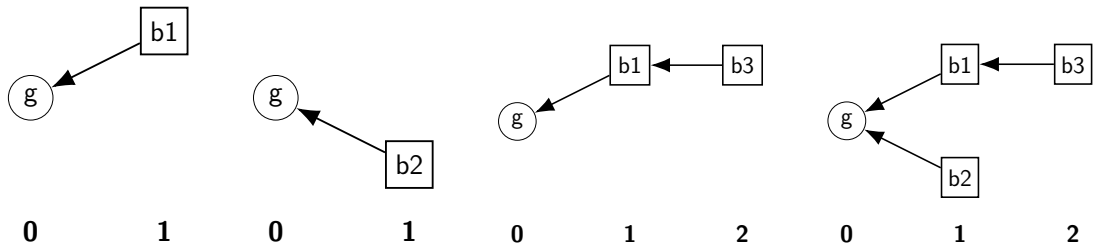
We assume the presence of an *adversary* who can control *attacker* or malicious nodes that together own a relatively small fraction $0 < \rho < \frac{1}{2}$ of the total mining power of the system. The nodes controlled by the adversary may not follow the protocol specification, however, they cannot impersonate other nodes while issuing transactions[1]. A node that is not malicious is deemed *correct*.

Network failure may be observed through (1) disconnection of nodes leaving information unattainable by others, or (2) network partition, segregating critical information from other parts of the network from ever knowing.

3.1.4 The forkable blockchain abstraction

Let the *blockchain* be a directed acyclic graph (DAG) $\ell = \langle B, R \rangle$ such that blocks of B point to each other with pointers R (pointers are recorded in a block as a hash of the previous block) and a special block $g \in B$, called the *Genesis block*, does not point to any block but serves as the common first block.

[1] This is typically ensured through the public key cryptography employed through the blockchain.



(a) view ℓ_1 (b) view ℓ_2 (c) view ℓ_3 (d) global state $\ell_0 = \ell_1 \cup \ell_2 \cup \ell_3$

Figure 3.1: The global state ℓ_0 of a blockchain results from the union of the distributed local views ℓ_1, ℓ_2 and ℓ_3 of the blockchain.

Algorithm 1 Blockchain construction at node p_i.

1: $\ell_i = \langle B_i, R_i \rangle$, the local blockchain at node p_i is a directed acyclic
2: graph of blocks B_i and pointers R_i

3: receive-blocks$(\langle B_j, R_j \rangle)_i$: ▷ *upon reception of blocks*
4: $B_i \leftarrow B_i \cup B_j$ ▷ *update vertices of blockchain*
5: $R_i \leftarrow R_i \cup R_j$ ▷ *update edges of blockchain*

Algorithm 1 describes the progressive construction of a forkable blockchain at a particular node p_i upon reception of blocks from other nodes by simply aggregating the newly received blocks to the known blocks (lines 3–5). As every added block contains a hash to a previous block that eventually leads back to the Genesis block, each block is associated with a fixed index. By convention, the Genesis block is considered at index 0, and the block at j hops away from the Genesis block as the block at index j. As an example, consider the simple blockchain $\ell_1 = \langle B_1, R_1 \rangle$ depicted in Figure 3.1a where $B_1 = \{g, b_1\}$ and $R_1 = \{\langle b_1, g \rangle\}$. The Genesis block g has index 0 and the block b_1 has index 1.

Forks as disagreements on the blocks at a given index

As depicted by views ℓ_1, ℓ_2 and ℓ_3 in Figures 3.1a, 3.1b and 3.1c, respectively, nodes may have different view of the current state of the blockchain. In particular, it is possible for two miners p_1 and p_2 to mine almost simultaneously two different blocks, say b_1 and b_2. If neither block b_1 nor b_2 was propagated early enough to nodes p_2 and p_1, respectively, then both blocks would point to the same previous block g as depicted in Figures 3.1a and 3.1b. Because network delays are not predictable, a third node p_3 may receive the block b_1 and mine a new block without hearing about b_2. The three nodes p_1, p_2 and p_3 thus end up having three different local views of the same blockchain, denoted $\ell_1 = \langle B_1, R_1 \rangle$, $\ell_2 = \langle B_2, R_2 \rangle$ and $\ell_3 = \langle B_3, R_3 \rangle$.

The *global blockchain* is referred to as the directed acyclic graph $\ell_0 = \langle B_0, R_0 \rangle$ representing the union of these local blockchain views, denoted by $\ell_1 \cup \ell_2 \cup \ell_3$ for short, as depicted in

Figure 3.1, and more formally defined as follows:

$$\begin{cases} B_0 & = \cup_{\forall i} B_i, \\ R_0 & = \cup_{\forall i} R_i. \end{cases}$$

The point where distinct blocks of the global blockchain DAG have the same predecessor block is called a *fork*. As an example Figure 3.1d depicts a fork with two branches pointing to the same block: g in this example.

3.2 Decisions and termination

In the event of a fork, the system must invoke a rule whereby the fork is resolved and a single, canonical chain is chosen as the correct chain to extend. A variety of approaches have been proposed to resolve forks, but this decision must be agreed upon by all correct nodes.

Bitcoin and other systems employ Nakamoto's Consensus, as described in Section 2.2.2, which selects the longest branch, whereas an alternative solution, GHOST, selects the heaviest subtree.

3.2.1 Main branch in Nakamoto and Ghost

Building upon the generic construction (Alg. 1), two selection techniques are presented: Nakamoto's consensus protocol (Alg. 2) present in Bitcoin [9] and the GHOST consensus protocol (Alg. 3).

Nakamoto's consensus algorithm The difficulty of the crypto-puzzles used in Bitcoin produces a block every 10 to 15 minutes in expectation. The advantage of this long period, is that it is relatively rare for the blockchain to fork because blocks are rarely mined during the time others are propagated to the rest of the nodes.

Algorithm 2 depicts the Bitcoin-specific pseudocode that includes Nakamoto's consensus protocol to decide on a particular block at index i (lines 8–18) and the choice of parameter m (line 6) explained later in Section 3.2.2. When a fork occurs, Nakamoto's protocol resolves it by selecting the longest branch as the main branch (lines 8–15) by iteratively selecting the root of the deepest subtree (line 11). When process p_i is done with this pruning, the resulting branch becomes the main branch $\langle B_i, R_i \rangle$ as observed by the local process p_i. Note that the pseudocode for checking whether a block is decided and a transaction committed based on this parameter m is common to Bitcoin and Ethereum, it is thus deferred to Alg. 4.

The GHOST consensus algorithm

As opposed to the original Bitcoin protocol, other blockchain variants utilise smaller block intervals. While throughput may be improved, in regards to transactions batched into blocks per second, it increases the probability of forks as miners are more likely to propose new blocks without having heard about the latest mined blocks. To avoid wasting effort in fork resolution,

Algorithm 2 Nakamoto's consensus protocol at node p_i.

$m = 5$, the number of blocks to be appended after the block containing
    tx, for tx to be committed in Bitcoin

get-main-branch()$_i$: ▷ *select the longest branch*
    $b \leftarrow$ genesis-block(B_i) ▷ *start from the blockchain root*
    while $b.next \neq \perp$ **do** ▷ *prune shortest branches*
        $block \leftarrow \text{argmax}_{c \in children(b)}\{\text{depth}(c)\}$ ▷ *root of deepest subtree*
        $B \leftarrow B \cup \{block\}$ ▷ *update vertices of main branch*
        $R \leftarrow R \cup \{\langle block, b\rangle\}$ ▷ *update edges of main branch*
        $b \leftarrow block$ ▷ *move to next block*
    return $\langle B, R\rangle$ ▷ *returning the Bitcoin main branch*

depth$(b)_i$: ▷ *depth of tree rooted in b*
    if $children(b) = \varnothing$ **then return** 1 ▷ *stop at leaves*
    else return $1 + \max_{c \in children(b)}$ depth(c) ▷ *recurse at children*

the Greedy Heaviest Observed Subtree (Ghost) consensus algorithm accounts for the total number of blocks in all branches of the subtree. In contrast to Nakamoto's protocol, Ghost iteratively selects the root of the subtree that contains the largest number of nodes (c.f. Algorithm 3).

Algorithm 3 The Ghost consensus protocol at node p_i.

$m = 12$, the number of blocks to be appended after the block containing
    tx, for tx to be committed (Ethereum as an example)

get-main-branch()$_i$: ▷ *select the branch with the most nodes*
    $b \leftarrow$ genesis-block(B_i) ▷ *start from the blockchain root*
    while $b.next \neq \perp$ **do** ▷ *prune lightest branches*
        $block \leftarrow \text{argmax}_{c \in children(b)}\{\text{num-desc}(c)\}$ ▷ *root of heaviest tree*
        $B \leftarrow B \cup \{block\}$ ▷ *update vertices of main branch*
        $R \leftarrow R \cup \{\langle block, b\rangle\}$ ▷ *update edges of main branch*
        $b \leftarrow block$ ▷ *move to next block*
    return $\langle B, R\rangle$. ▷ *returning the main branch*

num-desc$(b)_i$: ▷ *number of nodes in tree rooted in b*
    if $children(b) = \varnothing$ **then return** 1 ▷ *stop at leaves*
    else return $1 + \sum_{c \in children(b)}$ num-desc(c) ▷ *recurse at children*

The primary difference between Nakamoto's consensus protocol, and the Ghost consensus is depicted in 2.4, where the main longest branch is selected by Nakamoto's consensus whereas the heaviest subtree is selected in Ghost for the main chain. While Ghost may provide resilience to high forking blockchains, it has a heavy computational cost as it must iterate through many combinations of the subtrees. Ethereum utilises a modified Ghost in the Proof-of-Work chain for mining rewards, to account for the effort of all miners. As Ethereum moves to Proof-of-Stake, they further modified the Ghost protocol to derive the heaviest subtree based

Algorithm 4 Checking transaction commit at node p_i.

19: is-committed$(tx)_i$: ▷ *check whether transaction is committed*
20: $\langle B'_i, R'_i \rangle \leftarrow$ get-main-branch() ▷ *pick main branch* Alg. 2 or 3
21: **if** $\exists b_0 \in B'_i : tx \in b_0 \wedge \exists b_1, ..., b_m \in B_i :$ ▷ *tx in main branch*
22: $\langle b_1, b_0 \rangle, \langle b_2, b_1 \rangle ..., \langle b_m, b_{m-1} \rangle \in R_i$ **then** ▷ *enough blocks*
23: **return** true
24: **else return** false

on messaging [204, 134].

3.2.2 Termination of consensus and committed transactions

In the context of the blockchain, the resolution of forks and the termination of consensus indicates that a block has been decided for a certain index. By relaxing the agreement property of consensus, the blockchain is able to guarantee deterministic termination. We say that all transactions in a block that has been decided are *committed*[2]. Observing that a block had been mined is generally not sufficient to guarantee that it is decided, due to chain reorganisation; this block could be part of a branch in a transient fork that is not accepted by the network without consensus being reached.

A blockchain system S must define when the block at an index is agreed upon, thus different blockchain systems adopt different values of m to define termination. More precisely, there must exist a parameter m provided by S for an application to consider a block as *decided* and its transactions as *committed*. Note that these two choices do not lead to the same probability of success [100] and different numbers are used by different applications. In Bitcoin (BTC), $m_{btc} = 5$, meaning that the block at index i is decided—consensus for index i terminates—when the $m_{btc} + 1 = 6$ blocks at indices $i, ..., i + 5$ have been successfully mined. As previously mentioned, a new block is decided every 10 minutes in Bitcoin, hence it takes $(m_{btc} + 1) * 10\, min = 1\, hour$ for a transaction to be committed in Bitcoin. In Ethereum (ETH), $m_{eth} = 12$ is recommended, meaning that the block at index i is decided—consensus for index i terminates—when the blockchain depth reaches $i+12$. Hence it takes $(m_{eth}+1)*15\, sec = 3\, min$ for transactions to be committed in Ethereum on average.

Hence, it is said that a system S has to define a point in its execution where a prefix of the main branch can be "reasonably" considered as persistent[3].

Definition 1 (Transaction commit)**.** *Let $\ell_i = \langle B_i, R_i \rangle$ be the blockchain view at node p_i in system S. A transaction tx is* committed *if there exists a node p_i such that tx is* locally

[2]The term "committed" is used rather than "confirmed", a commonly used term in blockchain literature. Frequently, a transaction that is "confirmed" denotes that it has been successfully mined into a block, or in some cases, $m + 1$ blocks dependent upon literature. The term "committed" is used in this chapter to denote $m + 1$ blocks have been mined (the block containing the transaction, and, m successor blocks).

[3]In theory, there cannot be consensus on a block at a particular index [17], hence preventing persistence. However, applications have successfully used Ethereum to transfer digital assets based on parameter $m_{ethereum} = 12$

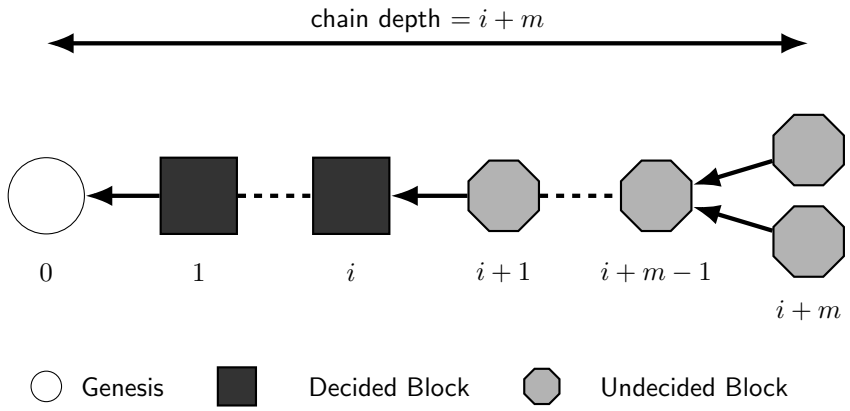


Figure 3.2: A typical blockchain structure, starting with a Genesis block at index 0, where each subsequent block appended to the chain is linked to its predecessor. A new block is decided at index i iff there are m successors appended to the chain and the depth reaches $i + m$.

committed.

For a transaction tx to be locally committed *at p_i, the conjunction of the following properties must hold in p_i's view ℓ_i:*

1. *Transaction tx has to be in a block $b_0 \in B_i$ of the main branch of system S. Formally, $tx \in b_0 \wedge b_0 \in B'_i : \langle B'_i, R'_i \rangle = \mathsf{get\text{-}main\text{-}branch}()_i$.*

2. *There should be a subsequence of m blocks $b_1, ..., b_m$ appended after block b. Formally, $\exists b_1, ..., b_m \in B_i : \langle b_1, b_0 \rangle, \langle b_2, b_1 \rangle, ..., \langle b_m, b_{m-1} \rangle \in R_i$. (In short, we say that b_0 is* decided*.)*

Property (1) is needed because nodes eventually agree on the main branch that defines the current state of accounts in the system—blocks that are not part of the main branch are ignored. Property (2) is necessary to guarantee that the blocks and transactions currently in the main branch will persist and remain in the main branch. Before these additional blocks are created, nodes may not have reached consensus regarding the unique blocks b at index j in the chain. This is illustrated by the fork of Figure 3.1 where nodes consider, respectively, the pointer $\langle b_1, g \rangle$ and the pointer $\langle b_2, g \rangle$ in their local blockchain view. By waiting for m blocks, where m is given by the blockchain system, the system guarantees with a reasonably high probability that nodes will agree on the same block b. Note that the property is not prefix-closed in the sense that get-main-branch may return a chain that is not necessarily a prefix of another branch returned later.

Figure 3.2 depicts the progression of the termination of consensus for a given block at index i. The arrow pointing from right to left indicates that the block references the hash of its predecessor immediately located on the left. Newly mined blocks are added to the right end of the blockchain that may fork transiently if multiple blocks referring to the same predecessor get mined concurrently. Forks are only transient and their resolution depends on the blockchain system in use. The consensus for an index i terminates when participants decide on the new block to be assigned at index i. The decision upon the block at index i occurs for all $i > 0$ when the

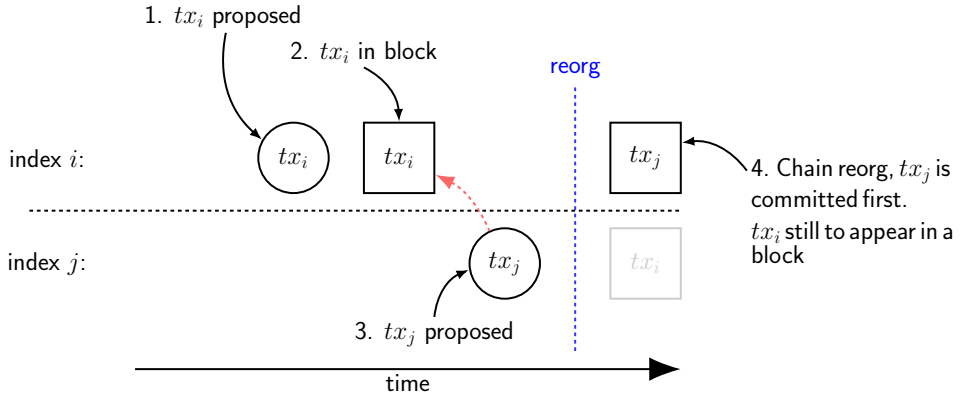


Figure 3.3: The Blockchain anomaly: a first client issues tx_i that gets successfully mined and committed then a second client issues tx_j commencing some off-chain service or interaction, with tx_j being conditional to the commit of tx_i (note that $j \geq i + m$ for tx_i to be committed before tx_j gets issued), but the transaction tx_j gets finally reorganised and successfully committed before tx_i, hence violating the dependency between tx_i and tx_j.

blockchain depth reaches $i+m$, where $m \geq 0$ is a constant dependent on the Blockchain system.

For example, consider a fictional blockchain system with $m_{fictional} = 2$ that selects the heaviest branch (Alg. 3, lines 8–15) as its main branch. If the blockchain state was the one depicted in Figure 2.4, then blocks b_2 and b_5 would be decided and all their transactions would be committed. This is because they are both part of the main branch and they are followed by at least 2 blocks, b_8 and b_{13}. (Note that the Genesis block is omitted as it is always considered decided but does not include any transaction.)

3.2.3 The Blockchain Anomaly

The Blockchain Anomaly represents an example of possible actions that may occur due to probabilistic agreement in blockchain systems, illustrating the effect of unwanted chain reorganisations based on the termination of consensus, or, probabilistic agreement.

Causes of the Blockchain Anomaly

The problem stems from the asynchrony of the network, in which message delays cannot be bounded and the consensus fails to terminate. Although two miners mine on the same chain starting from the same Genesis block, a sufficiently long delay in messages between them could lead to having the miners seemingly agree separately on different branches containing more than m blocks each, for any m. This anomaly has the potential to be dramatic as it can lead to simple attacks within any private network where users have an incentive to maximise their profits; in terms of coins, stock options or arbitrary ownership.

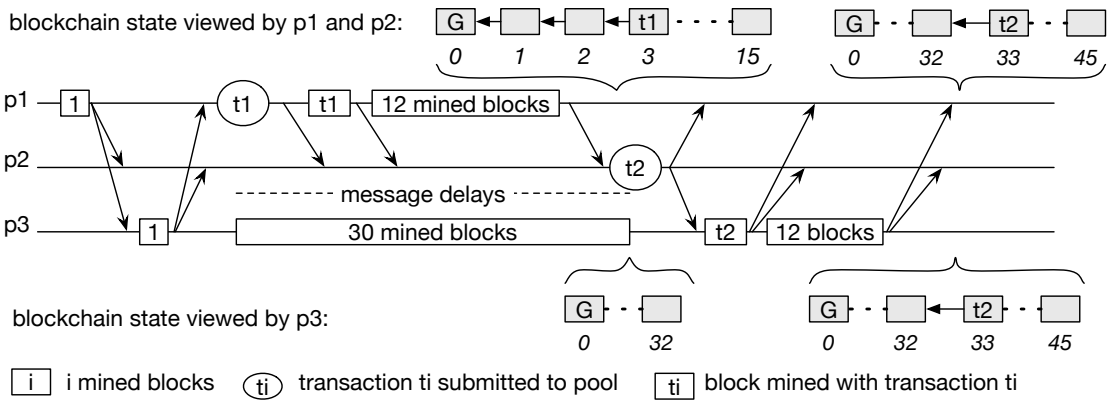


Figure 3.4: Execution scenario leading to the Blockchain anomaly: p_3 mines a longer chain than p_1 without including tx_1 and without disseminating new blocks until it forces a reorganisation that imposes tx_2 to be committed while tx_1 appears finally uncommitted.

Uncommitting transactions

Figure 3.3 depicts the Blockchain anomaly, where a transaction tx_i gets committed as part of slot i from the standpoint of some nodes. Based on this observation, one proposes a new transaction tx_j knowing that tx_i was successfully committed. Again, one can imagine a simple scenario where "Bob transfers an amount of money to Carole" (tx_j) only if "Bob had successfully received some money from Alice" (tx_i) before. However, once these nodes get notified of another branch of committed transactions, they decide to reorganise the branch to resolve the fork. The reorganisation removes the committed transaction tx_i from slot i. Later, the transaction tx_j is successfully committed in slot i.

The anomaly stems from the violation of the dependency between tx_j and tx_i: tx_j occurred meaning that Bob has transferred an amount of money to Carole, however, tx_i did not occur meaning that Bob did not receive money from Alice. Note that in Bitcoin, transaction tx_i gets discarded whereas in Ethereum transaction tx_i may in some cases be committed in slot j.

Tracking Blockchain Anomalies

Another dramatic aspect of the Blockchain anomaly is that it can go undetected if not monitored carefully. More specifically, the Blockchain anomaly relies on a wrongly committed state of the blockchain. Once the wrongly committed state gets uncommitted, there is no way to a posteriori observe this problematic state and to notice that a blockchain anomaly occurred. Although it is possible to observe that a peer mined several blocks in a row, there is no way to track down the beneficiaries of the Blockchain anomaly. This dangerously incentivises participants of the private chain to leverage the Blockchain anomaly to attack the chain.

Implications of the anomaly

Unsurprisingly, chain reorganisations resulting in experiencing the Blockchain Anomaly may have larger effects on the network. One such implication is frontrunning [205], where blockchain anomalies provide trading with alternative executions of transactions. Similarly, for systems that have implicit ordering, an anomalous execution may prove fatal if interacting with off-chain services or products that are irrevocable [206]. An adversary may utilise this uncertainty in transaction and transaction ordering for malicious purposes.

Facilitating a double-spending attack

One dramatic consequence of the Blockchain anomaly is the possibility for an adversary to execute a *double-spending* attack: converting, for example, all her coins into goods twice. The scenario consists of the attacker issuing a first transaction tx_1 that converts all her coins into goods in block i and starting mining blocks after block $i-1$ in isolation from the network. As part of this mining, the attacker mines another transaction tx_2 that also converts all her coins into goods. The attacker then waits for the blockchain depth to reach $i+m$ after which she can collect her goods as a result of transaction tx_1, then it publicises its longer chain without tx_1 so that the chain gets adopted by the rest of network. tx_2 gets committed in block j and after the chain depth reaches $j+m$, the adversary can collect her goods for the second time. Note that even if one tries to re-commit tx_1 later, the transaction will be invalidated because the balance is insufficient, however, the double-spending already occurred. An example of this execution is presented in Figure 3.4.

3.3 The Balance Attack

In this section, the Balance Attack, a novel form of attacks that affect forkable Proof-of-Work blockchains is presented, using Ethereum as an example. Its novelty lies in identifying sub-groups of miners of equivalent mining power and delaying messages between them rather than entering a race to mine blocks faster than others.

The Balance Attack demonstrates a fundamental limitation of main Proof-of-Work systems in that they are not immutable.

Definition 2 (Corruptibility)**.** *A blockchain system is* corruptible *if an adversary can:*

1. *make the recipient of a transaction tx observe that tx is committed and*
2. *later remove the transaction tx from the main branch,*

with probability $1-\varepsilon$*, where* ε *is a small constant defined by the adversary.*

Note that the adversary selects ε to maximise the chances of the attack based on the delay of messages as explained in line 14 of Algorithm 5.

The Balance Attack is simple: while the attacker disrupts communications between correct subgroups of equivalent mining power, it simply issues transactions in one subgroup. The attacker then mines sufficiently many blocks in another subgroup to ensure with high probability that the subtree of another subgroup outweighs the transaction subgroup's. Even though the transactions are committed, the attacker can rewrite with high probability the blocks that contain these transactions by outweighing the subtree containing this transaction.

Note that one could benefit from delaying messages only between the merchant and the rest of the network by applying the eclipse attack [36] to Ethereum [187, 188, 186]. Eclipsing one node of Bitcoin appeared, however, sufficiently difficult: it requires restarting the node's protocol in order to control all the logical neighbours to which the node will eventually try to connect. While a Bitcoin node typically connects to 8 logical neighbours, an Ethereum node typically connects to 25 nodes, making the problem even harder. Techniques have been proposed to achieve this [187, 188] by using subtleties in the networking protocol to fill connections and eclipse a node. Another option would be to isolate a subgroup of smaller mining power than another subgroup, however, it would make the attack only possible if the recipients of the transactions are located in the subgroup of smaller mining power. Although possible this would limit the generality of the attack, because the attacker would be constrained on the transactions it can override.

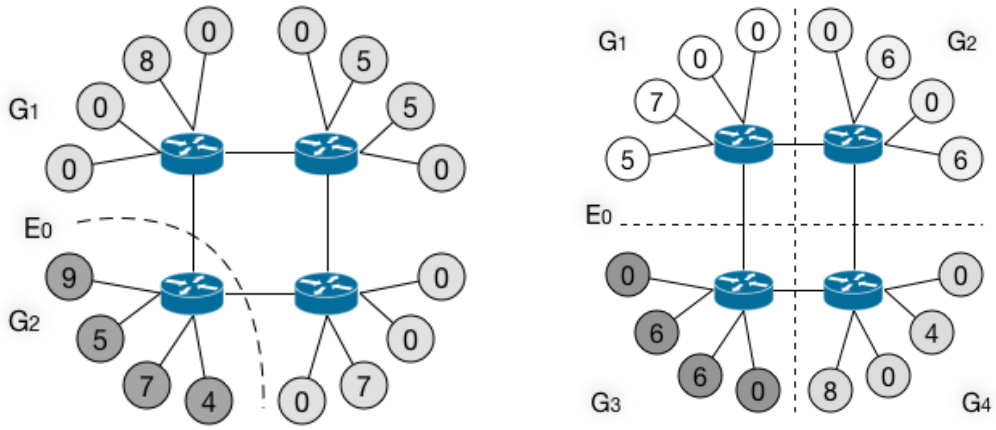
Note that the Balance Attack inherently violates the persistence of the main branch prefix and is enough for the attacker to double-spend. The attacker has simply to identify the subgroup that contains merchants and create transactions to buy goods from these merchants. After that, it can issue the transactions to this subgroup while propagating its mined blocks to at least one of the other subgroups. Once the merchant has shipped goods, the attacker stops delaying messages. Based on the high probability that the tree seen by the merchant is outweighed by another subtree, the attacker could reissue another transaction transferring the exact same coin again.

3.3.1 Attacker model

Finally, as in the Dolev-Yao model [207], it is assumed the attacker, who has the control over the Byzantine participants, can intercept messages, delay, or delete them. However, it is assumed the attacker cannot modify messages as it do not have enough power to forge signatures. We assume the adversary has access to, or controls, network infrastructure that performs routing for the blockchain network. This threat model feasibility is discussed in more detail in Section 3.3.4.

3.3.2 Executing a Balance Attack

For the sake of simplicity, let us fix the number of subgraphs $\kappa = 2$ and postpone the general analysis for any $\kappa \geq 2$ to the Appendix B. Subgraphs $G_1 = \langle N_1, E_1 \rangle$ and $G_2 = \langle N_2, E_2 \rangle$ of the communication graph $G = \langle N, E \rangle$ are considered so that each subgraph has half of the mining power of the system as depicted in Figure 3.5a whereas Figure 3.5b illustrates the variant when



(a) Example of the selection of edges E_0 delayed between $\kappa = 2$ subgraphs of 25 units of mining power each

(b) Example of the selection of edges E_0 delayed between $\kappa = 4$ subgraphs of 12 units of mining power each

Figure 3.5: Two decompositions of communication graphs into subgraphs by an attacker where E_0 represents the cut of communication edges linking the subgraphs.

$\kappa = 4$. Let $E_0 = E \setminus (E_1 \cup E_2)$ be the set of edges that connect nodes of N_1 to nodes of N_2 in the original graph G. Let τ be the communication delay introduced by the attacker on the edges of E_0. The lower bound on τ is selected with sufficient probability, later explained in the proof of Theorem 5.

As indicated in Algorithm 5, the attacker can introduce a sufficiently long delay τ during which the miners of G_1 mine in isolation from the miners of G_2 (line 14). As a consequence, different transactions get committed in different series of blocks on the two blockchains locally viewed by the subgraphs G_1 and G_2. Let b_2 be a block present only in the blockchain viewed by G_2 but absent from the blockchain viewed by G_1. In the meantime, the attacker issues transactions spending coins C in G_1 (line 15) and mines a blockchain starting from the block b_2 (line 17). Before the delay expires, the attacker sends his blockchain to G_2. After the delay expires, the two local views of the blockchain are exchanged. Once the heaviest branch that the attacker contributed to is adopted, the attacker can simply reuse the coins C in new transactions (line 20).

3.3.3 Exploiting the knowledge about the network

As indicated by the state of Algorithm 5, an attacker has to be knowledgeable about the current settings of the blockchain system to execute a Balance Attack. In fact, the attacker must have information regarding the logical or physical communication graph, the mining power of the miners or pools of miners and the current difficulty of the crypto-puzzle. In particular, this information is needed to delay messages sufficiently long enough between selected communication

Algorithm 5 The Balance Attack initiated by attacker p_i.

State:
  $G = \langle N, E \rangle$, the communication graph
  pow, a mapping from of a node in N to its mining power in $\mathbb{R}^+$
  Ψ, the total mining power of the system
  $\ell_i = \langle B_i, R_i \rangle$, the local blockchain at node p_i is a directed acyclic graph of blocks B_i and pointers R_i
  $\rho \in (0; 1)$, the portion of the mining power of the system owned by the attacker p_i, with $0 < \rho < 0.5$
  d, the difficulty of the crypto-puzzle currently used by the system

balance-attack$(\langle N, E \rangle)_i$: ▷ *starts the attack*
  Select $\kappa \geq 2$ subgraphs $G_1 = \langle N_1, E_1 \rangle, ..., G_\kappa = \langle N_\kappa, E_\kappa \rangle$:
    $\sum_{\forall v \in N_1} \mathsf{pow}(v) \approx ... \approx \sum_{\forall v' \in N_k} \mathsf{pow}(v')$
  Let $E_0 = E \setminus \cup_{\forall 0 < i \leq \kappa} E_i$ ▷ *attack communication channels*
  Stop communications on E_0 during $\tau \geq \frac{(1-\rho)6d \log(\frac{4}{\varepsilon})}{\rho^2 \Psi}$ seconds
  Issue transaction tx crediting a merchant in graph G_i with coins C
  Let b_2 be a block appearing in G_j but not in G_i
  Start mining on ℓ_i immediately after b_2 ▷ *contributed to correct chain*
  Send blockchain view ℓ_i to some subgraph G_j where $j \neq i$
  When τ seconds have elapsed, stop delaying communications on E_0
  Issue transaction tx' that double-spends coins C

subgraphs.

Information about Ethereum is often public and can be found online[4]. In particular, the difficulty of the crypto-puzzle, the propagation delay, block uncles and the times between blocks are shown. Also, any nodes that choose to register voluntarily show their mining power, Ethereum client and version, and their latency.

It is more difficult, however, to gather information regarding the communication network. In particular, work by Ekparinya et al. [190] demonstrated that finding enough connectivity information about miners through public web pages is significantly difficult in comparison with deployments in LAN maintained in the cloud, or, between a consortium of partners connected through the Internet. Note that some tools exist to retrieve information regarding the communication topology of blockchain systems. The interesting aspects of the Balance Attack is that it can apply to the logical overlay used by the peer-to-peer network of the blockchain system or to the physical overlay. While there exist tools to retrieve the logical overlay topology, like AddressProbe [208] which finds some information regarding the peer-to-peer overlay of Bitcoin, it can be easier for an attacker of Ethereum to run a Man-in-the-Middle Attack rather than a BGP hijacking [209, 210] or rerouting [185].

[4] Available through https://ethstats.net, https://ethstats.io, and https://etherscan.io/

3.3.4 Delaying networking communications

While disrupting the communication between subgroups of a blockchain system may look difficult, there have been multiple attacks successfully delaying blockchains messages in the past.

In 2014, a BGP hijacker exploited access to an ISP to steal $83000 worth of Bitcoin by positioning itself between Bitcoin pools and their miners [209]. Some known attacks against Bitcoin involved partitioning the communication graph at the network level [210] and at the application level [36]. At the network level, a study indicated that autonomous systems can simply intercept a large amount of Bitcoin and it evaluated the impact of these network attacks on the Bitcoin protocol [210]. At the application level, some work showed that an attacker controlling 32 IP addresses can "eclipse" a Bitcoin node with 85% probability [36]. Similar work has been done on Ethereum [186, 187, 188], where an attacker can sufficiently fill the connections of a node to isolate them.

Although a number of mitigations have been implemented making network-level eclipse attacks sufficiently difficult [36], there are a number of new techniques that highlight the possibility of new weaknesses that can be used in combination to perform this attack [185, 211]. Similarly, a number of eclipsing techniques on Ethereum [187, 188] highlight that some mitigations cannot be patched due to compatibility concerns.

Starting in September 2016, Ethereum experienced a Denial-of-Service attacks that forced miners to spend a long time accessing memory while executing smart contracts. While as far as it is known, it did not lead to double-spending, but rather slowed down the entire network [63]. More generally, there are varieties of network attacks, known as Man-in-the-Middle attacks and spoofing attacks, that can be exploited to lead to similar results by relaying the traffic between two nodes through the attacker.

3.4 Vulnerability of the GHOST protocol

In this Section, it is shown that the Balance Attack makes a blockchain system based on GHOST (depicted in Alg. 3) corruptible. A malicious user can issue a Balance Attack with less than half of the mining power by delaying the network. Previous and new notations for the analysis are summarised in Table 3.1.

For the sake of simplicity in the proof, it is assumed that $\kappa = 2$ and $\sum_{\forall v \in N_1} \mathsf{pow}(v) = \sum_{\forall v' \in N_2} \mathsf{pow}(v')$ so that the communication is delayed between only two communication subgraphs of equal mining power. The proof for the general case where $\kappa \geq 2$ to Appendix B. As there is no better strategy for solving the crypto-puzzles than random trials, it is considered that subgraphs G_1 and G_2 mine blocks during delay τ. During that time, each of G_1 and G_2 performs a series of $n = \frac{1-\rho}{\kappa}\Psi\tau$ independent and identically distributed Bernoulli trials, each returning 1 in case of success with probability $p = \frac{1}{d}$ and 0 otherwise. Let the sum of these outcomes for subgraphs G_1 and G_2 be the random variables X_1 and X_2, respectively, each with

Table 3.1: Notations of the analysis.

Ψ	total mining power of the system (in million hashes per second, MH/s)
d	difficulty of the crypto-puzzle (in million hashes, MH)
ρ	fraction of the mining power owned by the malicious miner (in percent, %)
κ	the number of communication subgraphs
τ	disruption time of communication between subgraphs (in seconds, s)
μ_c	mean of the number of blocks mined by each subgraph during τ
μ_m	mean of the number of blocks mined by the attacker during time τ
Δ	the maximum difference of mined blocks for the two subgraphs
pow	A mapping from a node in N to its mining power in $\mathbb{R}^+$

a binomial distribution and mean:

$$\mu_c = np = \frac{(1-\rho)\Psi\tau}{2d}. \tag{3.1}$$

Similarly, the mean of the number of blocks mined by the malicious miner during time τ is

$$\mu_m = \frac{\rho\Psi\tau}{d}.$$

3.4.1 Analysis of the Balance Attack

The Balance Attack relies on the adversary outnumbering the difference in blocks mined by each subgraph G_1 and G_2.

Thus, the probability $\Pr[\mu_m > \Delta]$ that the expected number of blocks μ_m mined by the attacker is greater than the difference $\Delta = |X_1 - X_2|$ in blocks mined by the two subgraphs G_1 and G_2 is lower-bounded.

Let us first recall the Bernoulli's inequality.

Fact 3 (Bernoulli's inequality)**.** *$1 + nt \leq (1 + \Psi)^n$ for $n \geq 1$ and $\Psi \geq -1$.*

The probability $\Pr[\mu_m > \Delta]$ is now lower-bounded as follows.

Lemma 4. *If the attacker owns a portion $0 < \rho < \frac{1}{2}$ of the total mining power (Ψ), then at time τ:*

$$\Pr[\mu_m > \Delta] > \left(1 - 2e^{-\frac{\rho^2}{3(1-\rho)^2}\mu_c}\right)^2. \tag{3.2}$$

Proof. By Eq. 3.1, it is known that:

$$\Pr[\mu_m > \Delta] \quad = \quad \Pr\left[\Delta < \frac{2\rho}{1-\rho}\mu_c\right]. \tag{3.3}$$

At time τ, the communication is re-enabled (cf. line 19 of Alg. 5). At this point in the execution, the probability that the numbers of blocks mined by each subgraph are within a $\pm\delta$

factor from their mean, is bound for $0 < \delta < 1$ and $i \in \{1, 2\}$ by Chernoff bounds [212]:

$$\begin{cases} \Pr\left[X_i \geq (1+\delta)\mu_c\right] & \leq e^{-\frac{\delta^2}{3}\mu_c}, \\ \Pr\left[X_i \leq (1-\delta)\mu_c\right] & \leq e^{-\frac{\delta^2}{2}\mu_c}. \end{cases}$$

Thus, probability that the number of blocks mined by a subgraph diverges from its mean can be upper-bounded:

$$\Pr[|X_i - \mu_c| < \delta\mu_c] \quad > \quad 1 - 2e^{-\frac{\delta^2}{3}\mu_c}. \tag{3.4}$$

Observe that the probability that the two random variables X_1 and X_2 are both within $\pm\delta\mu_c$ is lower than the probability that their difference Δ is upper-bounded by $2\delta\mu_c$, hence:

$$\left(\Pr\left[|X_i - \mu_c| < \delta\mu_c\right]\right)^2 \leq \Pr[\Delta < 2\delta\mu_c].$$

It follows from Eq. 3.4 that:

$$\Pr[\Delta < 2\delta\mu_c] > \left(1 - 2e^{-\frac{\delta^2}{3}\mu_c}\right)^2. \tag{3.5}$$

Since $0 < \rho < \frac{1}{2}$ by line 8 of Alg. 5, we have $0 < \frac{\rho}{1-\rho} < 1$ so $\delta = \frac{\rho}{1-\rho}$ is fixed which leads to the result.

Theorem 5. *At time τ, the probability $\Pr[\mu_m > \Delta]$ that the expected number of blocks μ_m mined by the attacker is greater than the difference $\Delta = |X_1 - X_2|$ in blocks mined by the two subgraphs G_1 and G_2 is $\Pr[\mu_m > \Delta] > 1 - \varepsilon$, where ε is user-defined.*

Proof. As $\mu_c \geq 0$ we have:

$$\begin{aligned} -\frac{\rho^2}{3(1-\rho)^2}\mu_c & \leq 0, \\ -e^{-\frac{\rho^2}{3(1-\rho)^2}\mu_c} & \geq -1, \end{aligned}$$

and Bernoulli's inequality can be applied (Fact 3) to Eq. 3.2:

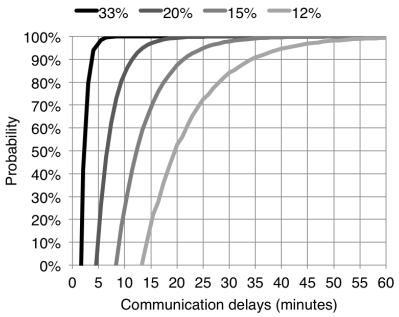
$$\Pr[\mu_m > \Delta] > \left(1 - 2e^{-\frac{\rho^2}{3(1-\rho)^2}\mu_c}\right)^2 \geq 1 - 4e^{-\frac{\rho^2}{3(1-\rho)^2}\mu_c}. \tag{3.6}$$

From line 14 of Alg. 5, it is known that $\tau \geq \frac{(1-\rho)6d\log(\frac{4}{\varepsilon})}{\rho^2\Psi}$ which leads to:

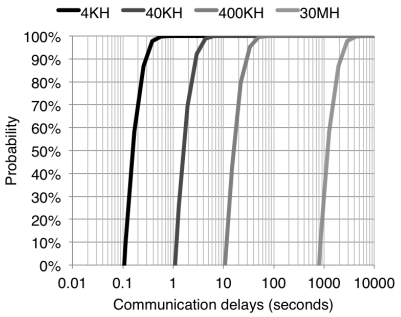
$$\begin{aligned} \tau & \geq \frac{(1-\rho)6d\log(\frac{4}{\varepsilon})}{\rho^2\Psi}, \\ \frac{(1-\rho)\Psi\tau}{2d} & \geq \frac{3(1-\rho)^2\log(\frac{4}{\varepsilon})}{\rho^2}. \end{aligned}$$

Number of nodes	50
Number of miners	15
Total mining power (MH/s)	20
Mining power of the most powerful miner (MH/s)	2.4
Difficulty (MH)	30

(a) The R3 settings used in the analysis as observed in June 2016



(b) Probability of the Balance Attack in the R3 network as the communication delay increases for different portions of the mining power controlled by the adversary



(c) Probability of Balance Attack in the R3 network (with 20 MH/s of total mining power, 12% of the mining power at the attacker) for different difficulties as the communication delay increases

Figure 3.6: Simulation of the Balance Attack with the difficulty of R3 and with the maximum mining power of R3.

By Eq. 3.1:

$$\begin{aligned}
\mu_c &\geq \frac{3(1-\rho)^2 \log(\frac{4}{\varepsilon})}{\rho^2}, \\
\frac{\rho^2 \mu_c}{3(1-\rho)^2} &\geq \log\left(\frac{4}{\varepsilon}\right), \\
-\frac{\rho^2 \mu_c}{3(1-\rho)^2} &\leq \log\left(\frac{\varepsilon}{4}\right), \\
e^{-\frac{\rho^2}{3(1-\rho)^2}\mu_c} &\leq \frac{\varepsilon}{4}, \\
1 - 4e^{-\frac{\rho^2}{3(1-\rho)^2}\mu_c} &\geq 1-\varepsilon.
\end{aligned}$$

Replacing this expression in Eq. 3.6 leads to the result where ε can be decided by the attacker and τ can be adjusted.

Corollary 6. *A blockchain system that selects a main branch based on the* Ghost *protocol (Alg. 3) is corruptible.*

Proof. By lines 17–18 of Alg. 3, it is known that Ghost counts the mined blocks to compute the weight of a subtree, and to select one blockchain view and discard the other.

Since by Theorem 5, the expected number of blocks mined by the attacker at time τ is greater than the difference Δ with probability $1-\varepsilon$, we know that when the timer expires at line 19 of Alg. 5, the attacker can make the system discard the blockchain view of either G_1 or G_2 with probability $1-\varepsilon$ by sending its blockchain view to the other subgraph, hence making the blockchain system corruptible.

3.5 Feasibility of the Balance Attack

The Balance Attack relies on the ability for an adversary to access and manipulate communication channels between subgroups of nodes. This requires the adversary to maintain information delay such that there is no information leakage about blocks or transactions, ensuring subgroups stay isolated form each other. As is highlighted in Section 3.3.4, blockchain networks experience communication issues through service-level outages and attacks that may form conditions suitable for an adversary wishing to perform such an attack. A number of other attacks [210, 180, 36, 190, 186, 185] discussed in Chapter 2 (Section 2.4), provide discussions of techniques to eclipse or manipulate communication between subgroups of nodes.

In small private or consortium networks, where a blockchain is being run over a single network such as a data centre or business infrastructure, an adversary may gain such privileges through *Address Resolution Protocol* (ARP) spoofing [213] techniques, allowing them to disrupt and control communication in smaller networks. This provides techniques to perform the attacks for blockchains stored in single datacenters or small businesses [190].

In larger networks, however, this attack becomes increasingly difficult. Border Gateway Protocol (BGP) [179] is responsible for routing of autonomous systems that make up the Internet connections known today. Hijacking BGP has been effective [209] in disrupting communication and may provide the possibility to launch the Balance Attack using various techniques as accessories [190]. Other techniques have been proposed in attacks [185], where the connection properties are utilised to re-route nodes and perform Man-in-the-Middle attacks at a larger scale to control and manipulate connections. However, maintaining disrupted communication, especially with multihoming of nodes, may prove difficult in a number of circumstances [180], especially with techniques providing defences against such attacks [182, 184]. Numerous patches and countermeasures [211, 36, 187] have been applied to various blockchains to strengthen against such network attacks, but are still vulnerable to large state-level actors or extreme service outages which may provide more probable conditions for an attacker.

The Balance Attack is most effective in small private, or consortium networks with limited number of nodes such that an adversary can manipulate connections and provide sufficient resources to execute the attack. Whilst the Balance Attack is feasible on the Internet at a larger scale, it requires an adversary with a significantly higher number of resources and the ability to manipulate communication at a global scale. The analysis below provides key insights on the feasibility of the Balance Attack against a consortium blockchain network, using Ethereum as used in the public setting as an example.

3.6 Analysis of the R3 Network

The statistics of the R3 network were obtained through a private deployment of `eth-netstat` at the end of June 2016. R3 is a consortium of more than 50 banks that has tested blockchain

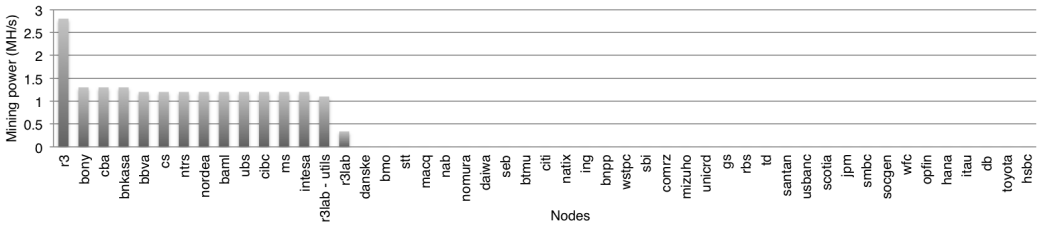


Figure 3.7: The mining power of the R3 Ethereum network as reported by `eth-netstats` as of June 2016.

systems and in particular Ethereum in a consortium private chain context over 2016 [214]. The network consisted at that time of $|N| = 50$ nodes among which only 15 were mining. The mining power of the system was about 20 MH/s, the most powerful miner mined at 2.4 MH/s or 12% of the total mining power while the difficulty of the crypto-puzzle was observed close to 30 MH as summarised in Figure 3.6a.

3.6.1 How the most powerful node could double-spend

It is assumed that the attacker is the `r3` node with $\rho = 12\%$ of the mining power as depicted in Figure 3.7 and that it delays communication between subgraphs G_1 and G_2, each with mining power $\frac{1-\rho}{2}\Psi = 8.8$ MH/s. The probability p of solving the crypto-puzzle per hash tested is $\frac{1}{30\times10^6}$ so that the mean is $\mu_c = \frac{(1-\rho)\Psi\tau}{2d} = \frac{8.8\times10^6\times1180}{2\times30\times10^6} = 346.13$ with a wait of 19 minutes and 40 seconds, i.e., 1180 seconds. The attacker creates, in expectation, a block every $\frac{30}{2.4} = 12.5$ seconds or $\lfloor\frac{1180}{12.5}\rfloor = 94$ blocks during the 19 minutes and 40 seconds. Hence let $\delta = \rho/(1-\rho) = 0.136$. The probability that the attack is a success is 53%.

3.6.2 A coalition of 33% of mining power needs a 4 minute delay to attack with 94% of success

Malicious nodes may have an incentive to form a coalition in order to exploit the Balance Attack to double-spend. In this case, it allows the attacker to control a larger portion of the mining power of the system which, in turn, increases the chances of success of the Balance Attack.

It is assumed that the adversary controls $\rho = \frac{1}{3}$ of the total mining power, which represents $\rho\Psi = 6.7$ MH/s. In this case the adversary delays communications between two subgraphs G_1 and G_2 with mining power of $\frac{1-\rho}{2}\Psi = 6.7$ MH/s each. With a 4 minute wait, i.e., 240 seconds, each isolated graph and adversary would mine $6.67 * 10^6 * 240/(30 * 10^6) = 53.4$ blocks. The probability that the attack succeeds would become $1 - e^{-\rho^2/3(1-\rho)^2*53.4}$, which is around 94%.

3.6.3 Tradeoff between communication delays and mining power

To illustrate the tradeoff between communication delay and the portion of the mining power controlled by the attacker, the R3 network is considered. With a 30 MH total difficulty and a 20 MH/s total mining power and the probability as the communication delay increases for

different portions of the mining power controlled by the attacker is plotted. Figure 3.6b depicts this result. As expected, the probability increases exponentially as the delay increases, and the higher the portion of the mining power controlled by the attacker, the faster the probability increases. In particular, in order to issue a balance attack with 90% probability, 35 minutes are needed for an attacker controlling 12% of the total mining power whereas only 11 minutes are sufficient for an attacker who controls 20% of the mining power.

3.6.4 Tradeoff between communication delays and difficulties

Another interesting aspect of a Proof-of-Work blockchain is the difficulty parameter d. As already mentioned, this parameter impacts the expected time it takes for a miner to succeed in solving the crypto-puzzle. When setting up a private chain using Proof-of-Work, one has to choose a difficulty to make sure the miners would mine at a desirable pace. Too high a difficulty reduces the throughput of the system without requiring leader election [158] or consensus sharding [215]. Too low a difficulty increases the probability for two correct miners to solve the crypto-puzzle before one can propagate the block to the other, a problem of Bitcoin that motivated the Ghost protocol [134].

Figure 3.6c depicts the probability of the Balance Attack when the communication delay increases for different difficulties without considering the time for a block to be decided. Again, consider the R3 Ethereum network with a total mining power of 30 MH/s and an attacker owning $\rho = 12\%$ of this mining power and delaying communications between $\kappa = 2$ subgraphs of half of the remaining mining power ($\frac{1-\rho}{2} = 44\%$) each. The curve labelled 4 KH indicates a difficulty of 4000 hashes, which is also the difficulty chosen by default by Ethereum when setting up a new consortium blockchain. This difficulty is dynamically adjusted by Ethereum at runtime to keep the mining block duration constant in expectation, however, this adaptation is dependent on the visible mining power of the system. The curve labelled 30 MH indicates the probability for the difficulty observed in the R3 Ethereum network. It can be clearly seen that the difficulty impacts the probability of the Balance Attack. This can be explained by the fact that the deviation of the random variables $X_1, ..., X_k$ from their mean μ_c is bounded for sufficiently large number of mined blocks.

3.7 Experimenting the Balance Attack on a Private Ethereum Blockchain

In this section, the attack on an Ethereum private chain is experimentally produced, involving up to 18 physical distributed machines. To this end, a realistic network is configured with 15 machines dedicated to mining as in the R3 Ethereum network described in Section 3.6 and 3 dedicated network switches.

All experiments were run on 18 physical machines of the Emulab environment where a network topology was configured using ns/2 [216] as depicted in Figure 3.8. The topology

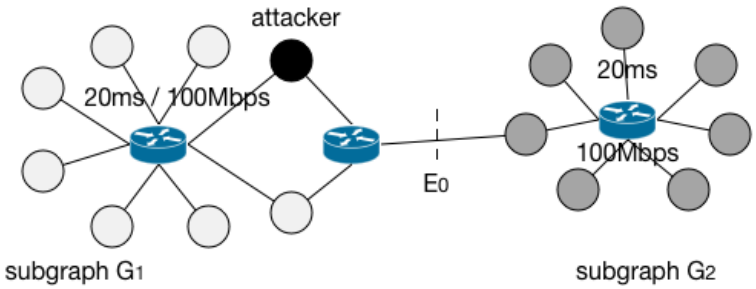


Figure 3.8: The topology of the experiment involving 15 miners with subgraph G_1 including the attacker depicted in black and subgraph G_2 depicted in grey.

consists of three local area networks configured through a ns/2 configuration file with 20 ms latency and 100 Mbps bandwidth. All miners run `geth` [217] Ethereum client v.1.3.6 and the initial difficulty of the crypto-puzzle is set to 40 KH. The communication graph comprises the subgraph G_1 of 8 miners that includes the attacker and a subgraph G_2 of 7 correct miners.

3.7.1 Favouring one blockchain view over another

The first experiment was run lasting 2 minutes. The link E_0 was delayed for 60 seconds so that both subgraphs mine in isolation from each other during that time and end up with distinct blockchain views. A snapshot is taken after the delay, at time t_1, of the blocks mined by each subgraphs and the two subgraphs start exchanging information normally leading to a consensus regarding the current state of the blockchain. At the end of the experiment, after 2 minutes, another snapshot t_2 is taken of the blocks mined by each subgraph.

Table 3.2: Number of blocks in the main branch (excluding uncles) mined by the subgraphs G_1 and G_2; the adversary influences the selection of branches and keeps blocks from G_1 but discards blocks from G_2.

	# blocks at t_1	# blocks discarded at t_2	# blocks kept at t_2	retention
G_1	52	39	13	25%
G_2	58	58	0	0%

Table 3.2 lists the number of blocks (excluding uncles) of the blockchain views of G_1 and G_2 at times t_1, while the two subgraphs did not exchange their view, and at time t_2, after the subgraphs exchanged their blocks. Note that uncle blocks are not represented, to provide focus on the main branches. It is observed that the blockchain view of the subgraph G_1 was adopted as the valid chain while the other blockchain view of the subgraph G_2 was not. In particular,

13 blocks of the main branch of G_1 at time t_1 were retrieved in the main branch selected at t_2. As expected, all the blocks of G_2 at time t_1 were discarded from the main branch by time t_2.

3.7.2 Blocks mined by an attacker and two subgraphs

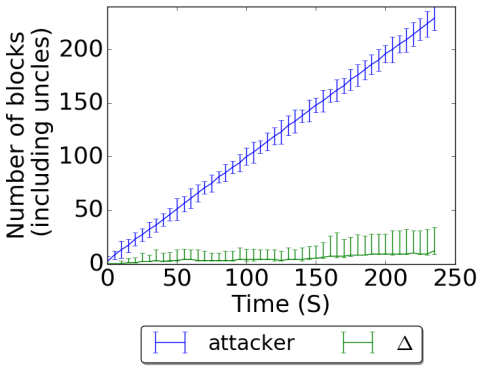
The total number of blocks mined are now reported, focusing on the creation of uncle blocks. More precisely, the number of blocks mined by the attacker are compared against the difference of the number of blocks Δ mined by each subgraph. It is known from the analysis that it is sufficient for the attacker to mine at least $\Delta + 1$ blocks in order to be able to discard one of the κ blockchain views, allowing for double-spending. The experiment is similar to the previous experiment in that Emulab with the same ns/2 topology was used, however, delays were not introduced and results were averaged over 10 runs of 4 minutes each.

Figure 3.9a depicts the minimum, maximum and average blocks obtained over the 10 runs. The vertical bars indicate minimum and maximum. First, it is observed that the average difference Δ is usually close to its minimum value observed during the 10 runs. This is due to having a similar total number of blocks mined by each subgraph in most cases with few rare cases where the difference is larger. As can be seen, the total number of blocks (including uncles) mined during the experiment by the attacker is way larger than the difference in blocks Δ mined by the two subgraphs. This explains the success of the Balance Attack as was observed in Section 3.7.1.

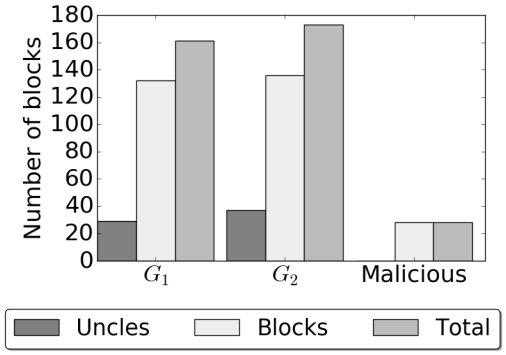
3.7.3 Relating connectivity to known blocks

Figure 3.9d illustrates the execution of two subgraphs resolving connectivity issues and adopting a chain. This experiment outlines one of the fundamental aspects of the balance attack, in which the chosen subgraph resolves the network delay and attempts reconnection with another subgraph. At this point, the subgraphs will initiate the consensus protocol and select the branch to adopt as the main branch. The experiment was set up with two subgraphs G_1 and G_2 where $|N_1| = |N_2| = 7$. The attacker selects a subgraph and delays messages between this subgraph and another, enforcing an isolated mining environment. Once the delay is set, the attacker joins one of the subgraphs and begins to mine onto the current chain. The attacker then delays the messages until there is a sufficient amount of blocks mined onto the isolated blockchain for it to be adopted as the correct chain by the other subgraph. In this experiment, at $t = 60\,s$, the delay between subgraphs is resolved, and the subgraphs maintain a connection. Upon reconnection, the subgraphs invoke the consensus protocol to select and adopt the correct chain. In this case, using the Ghost protocol, the heaviest chain is selected for both subgraphs, meaning the chain mined by G_1 is chosen, to which the attacker contributed.

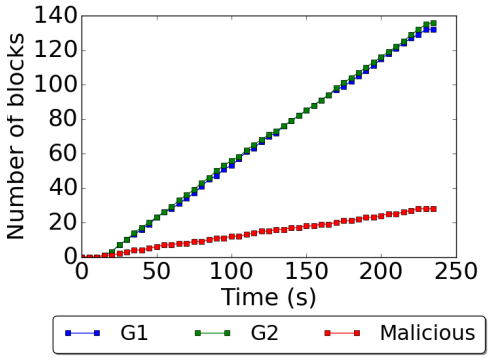
This result reveals that the adoption of a chosen blockchain is plausible, given that the attacker is able to sufficiently delay messages between subgraphs.



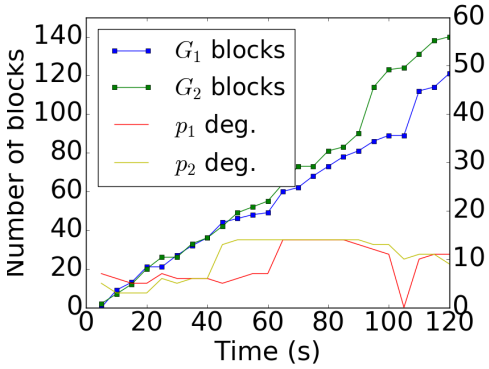
(a) Number of blocks mined by the attacker and the difference Δ in the number of blocks mined by the subgraphs



(b) Number of blocks mined by the attacker and the two subgraphs G_1 and G_2



(c) The depth of the blockchains mined by the attacker and two subgraphs G_1 and G_2



(d) Evolution of blocks and degrees where the attacker in G_1 delays the communication between G_1 and G_2 for one minute

Figure 3.9: Distributed experiments performed on a blockchain with 15 physical machines connected by a 100 Mbps network.

3.8 Proposal: solution with non-forkable blockchains

Corruptibility is a general problem that may affect other forkable blockchains, especially if they require messages to be propagated within a time bound, or may experience disagreement with a non-null probability [143, 42, 158].

3.8.1 Other forkable blockchains

First, it is important to note that the corruptibility problem is not restricted to the Ghost algorithm or to Proof-of-Work mechanisms, but could also apply to other forkable algorithms as well.

For example, the Balance Attack can be modified slightly to work in Bitcoin, or any derivative that uses Nakamoto's consensus as well. While in the example of Ghost, running on Ethereum, it was sufficient for the attacker to mine on any branch of the blockchain view of G_j after the block b_2 (Alg. 5, line 16), in Bitcoin the attacker has to mine at the top of the blockchain view of G_j. By doing so, the attacker increases the length of the Nakamoto's main branch in graph G_j. Considering that each correct miner mines at the top of the longest branch of their subgroup with the same probability q, the mean of the number of blocks added to the main branch will become $\mu_c^{bitcoin} = \frac{(1-\rho)\Psi\tau}{2dq}$. Two binomial random variables X_i' and X_j' can be defined for the expected number of blocks in the main branch of G_i and G_j, respectively, and a similar argument is applied as in Section 3.4.

The absence of Proof-of-Work does not imply immunity against the Balance Attack. The original Casper FFG [218] was an early Proof-of-Stake alternative proposed for the Ethereum blockchain, but would still experience forks and have vulnerabilities to such an attack. Validators were incentivized to bet some coin amount, called *value-at-loss*, on the block that they believe would be decided first among multiple ones. Once the aggregated value-at-loss bet by all validators for a particular block reaches a threshold, then the block gets decided and the winning gambler gets rewarded. For the sake of availability, this threshold has to be adjusted depending on the responding participants. Like in the Balance Attack, an adversary could delay communication between groups of nodes to alter the responsiveness of the participants and the threshold definition. By contrast with Proof-of-Work, the adversary would have to exploit some value-at-loss rather than its computational power to outweigh the bets and influence the selection of the main branch. As blockchains employ new techniques with alternatives, such as Proof-of-Stake, it is imperative that considerations into attack vectors are thoroughly checked. Assumptions and guarantees on transaction commits are vital to understanding susceptibility to such attacks. Ethereum 2.0's LMD Ghost [87, 204] counts a block's finality based on a threshold of the network attesting to this value. Partitions of a considerable duration may cause instability on these values and must be investigated.

3.8.2 Non-forkable blockchains

The crux of the problem of corruptibility stems from a blockchain favouring availability over consistency. Non-forkable, or unforkable, blockchains ensure that no two nodes disagree due to the outcome of consensus. The well-known CAP theorem (Section 2.1.3), initially mentioned by Brewer at PODC [76] and proved by Gilbert and Lynch [75], stating the impossibility for a distributed service to provide *(i)* consistency: returning the right response to a request; *(ii)* availability: returning a response to each request; and *(iii)* partition-tolerance: supporting message delays and losses. Intuitively, it appears natural for a distributed system to stop working as soon as communication is no longer ensured between the distributed participants. What is less clear is the consequence of communication being delayed. As presented in this chapter, the consequence for forkable blockchains can be asset losses.

There exists a large body of solutions in the distributed computing literature favouring consistency over availability. As consortium and private blockchains involve a typically known number n of participants, Byzantine consensus algorithms could be used to cope with the Balance Attack provided that strictly less than $\frac{n}{3}$ processes are under the control of the adversary. Several blockchains tried to build upon the classic Byzantine consensus implementation PBFT [15]. Examples include Hyperledger Implementations [172, 169], Tendermint [106] or Solida [138]. Unfortunately, PBFT does not perform well with a growing set of participants [219]. Hierarchical approaches were proposed to improve the performance of PBFT [215], but assumes synchrony. Other tentative approaches, such as HoneyBadger [144], Diem [40], Mir [220], or DBFT [51], and Algorand's BA$\star$ [42] provide improved performance using Byzantine Consensus algorithms designed for blockchain use cases. Byzantine Consensus yields major potential for blockchains and is a promising direction for safety and high performance.

3.9 Summary

In this chapter, a weakness was identified in the inherent synchrony assumptions of selected blockchains in practice, highlighting the presence of the Blockchain Anomaly that occurs during periods of long delay followed by chain reorganisations. These findings were used to highlight the importance of probabilistic consensus. To demonstrate the impact of such weaknesses, the Balance Attack was proposed, a simple attack demonstrating the power of an adversary with the ability to manipulate connectivity amongst subgroups of nodes. The Balance Attack utilises both mining power and network connectivity to double-spend on forkable blockchain protocols that rely on Nakamoto's or Ghost fork choice rules, where availability is favoured over consistency. The Attack provides a methodology for an adversary to convince subgroups of nodes of a correct fork, leading to eventual disregard of other transient branches leading to a double-spend. The theoretical analysis of the attack was presented, analysing the tradeoff between communication delay and mining power on the probability of success in a consortium network. This theoretical analysis highlights the importance of chain selection in small private or consortium

environments and provides motivation for secure alternatives.

In conclusion, unforkable blockchains under network partitions are proposed as a solution to the Balance Attack and problems of a similar nature. By selecting a deterministic consensus, mitigating the probability of transient forks, it will provide safety against double-spending in these scenarios and ensure that consistency is maintained in the presence of network delays. These mitigations not only provide higher security for participants, but improves the suitability of blockchains for critical infrastructure.

Chapter 4

The Red Belly Blockchain

Banking and critical infrastructure are yet to integrate the blockchain into their workflows, with varying levels of hesitation due to current unknowns, performance issues or problems presented in the current systems that have yet to be solved. The findings through the Blockchain Anomaly, Balance Attack (Chapter 3), and other related work [29, 30, 31, 32, 33, 34, 35, 210, 178, 176, 177] highlight the importance of the termination of consensus and motivates the need for a safe, high performance blockchain.

In the context of the blockchain, State Machine Replication (SMR) is performed through gossip communication over a peer-to-peer network, where all geodistributed participants replicate the system state. Permissionless blockchains allow a dynamic set of participants, as nodes can join and leave at any time, mostly suited to public systems and follow the traditional context set by Bitcoin. Conversely, permissioned blockchains provide a subset of nodes to participate, and often are required to go through a vetting process to join. New designs are moving towards an approach where subsets of nodes perform critical operations and rotate through dynamic sets. For example, Ethereum 2.0 [87] provides an open approach where nodes participate by offering coins for Proof-of-Stake, whereas other chains, such as Algorand [42], Stellar [221] and Tendermint [106] operate and build upon BFT consensus. While these proposals offer improvements in safety, the performance gain is limited when deployed to large networks with the magnitude of hundreds of nodes as seen in traditional permissionless deployments today. Transaction verification and execution is often computationally intensive whereas agreement on a block is communication intensive.

An ideal decentralised blockchain would propose scaling to hundreds of consensus nodes, where each participates in serving the requests of more nodes. The *Red Belly Blockchain (RBBC)* is proposed. The RBBC is the first secure blockchain with the ability to scale to hundreds of nodes, in that throughput does not drop as it reaches participation of hundreds of consensus nodes. The RBBC is secure, whereby it prevents double-spending by resolving conflicts and not forking, even when experiencing asynchrony. It provides optimal resilience in that, among the n nodes executing each of its consecutive consensus instances, up to $f < n/3$ can be Byzantine [14]. The foundations of the RBBC follow the Democratic Byzantine Fault

Tolerance (DBFT) consensus [51], meaning it is time optimal [222] and has been proved correct through model checking [223] for any number of nodes. RBBC also provides levels of censorship resilience and fairness, whereby all *correctly* requested transactions are eventually committed which implies blockchain liveness [224]. A consequence of censorship resilience is mitigating a series of problems, such as unfairness [225], anomalies [2] as described in Chapter 3, frontrunning [205, 226] or oligarchy [206] by ordering transactions by age. Complimentary to this, RBBC exchanges $O(n^3)$ bits to commit a single transaction, similar to lightweight leader-based implementations [149], which can be particularly appealing at the application level for well-formed, non-conflicting transactions.

To achieve scalability, RBBC offers a new balancing method that totally orders all transactions while assigning them to distinct groups of *proposer* and *verifier* nodes. *(i)* Its leaderless design balances the communication load on multiple proposers, hence avoiding the congestion and slowdown induced by the least responsive node. As opposed to classic Byzantine consensus protocols that rely on a leader to propose transactions, RBBC's multiple proposers combine distinct sets of transactions into a *superblock* to solve the new *Set Byzantine Consensus* problem and commit more transactions per consensus instance. *(ii)* Its verification sharding balances the computation load across verifiers. As opposed to existing blockchains where all n nodes typically verify every transaction, each of our transaction signatures is verified by between $f+1$ and $2f+1$ *verifiers*. Table 4.1 highlights the scalability and performance of RBBC and other large scale high performance blockchains. The effectiveness of RBBC's scalability mechanisms are evident when comparing the results obtained in current literature. The global experimental analysis of RBBC also revealed that these mechanisms allow performance to be observed even under attack.

Extensive experiments were conducted to analyse the performance of the solution and identify any pitfalls. The first set of experiments were performed with hundreds of Virtual Machines (VMs) spread over 10 countries and 4 continents, and was compared to traditional leader-based PBFT [16, 15] and HoneyBadger [144]. Secondly, the Caliper benchmark framework [47] was used to compare against two blockchains; the Burrow [169] blockchain which utilises the Tendermint [106] BFT consensus algorithm built on the Ethereum blockchain [27] and Quorum [55] which is a blockchain extension built on top of Ethereum running the Istanbul Byzantine Fault Tolerance (IBFT) consensus algorithm [150].

A summary of the contributions presented in this chapter is as follows;

(i) We present the Red Belly Blockchain (RBBC), a secure, high performance blockchain capable of scaling to hundreds of nodes.
(ii) We provide an analysis on the correctness, scalability and durability of the RBBC.
(iii) We present, experimentally, that the RBBC is able to scale to hundreds of consensus nodes.
(iv) We compare against two other BFT-based blockchain through a public benchmark framework, Caliper, showing that RBBC is able to handle sending rates of an order of magnitude

Table 4.1: Scalable blockchain experiments — the peak throughput of Elastico is obtained at 100 nodes with 14 1MB-sized blocks in 700 seconds [215] or at larger scale by producing 16 of them within 800 seconds [154]. Omniledger achieves 3500 TPS when tolerating 25% of adversarial power for 512-byte transactions but goes up to $4 \cdot 10^5$ TPS when the adversarial power is 1%. RapidChain peaks at 7384 TPS for 512-byte transactions but needs smaller blocks to achieve a 9-second latency, which leads to 7000 TPS [154]. The throughput of Algorand is 750 MB/h=208 KB/s or 327 MB/h=90 KB/s for a 22-second latency and the impact of attacks on its performance seems negligible [42]. The throughput of Mir is from 0, when a leader stalls after another, to 60,000 TPS, when all $n = 100$ proposers are correct, with 500-byte payload and no durability, and peaks at 40,000 TPS when $n = 4$ with 3500-byte payload [220]. The throughput of RBBC peaks at 660,000 TPS for 400-byte transactions with $n = 300$ (Fig.4.5) but varies from 1900 TPS under a 33% coalition attack (Fig.4.12) to 30,684 TPS at max scale, where its latency is 3 s (Table 4.3).

System	Deployment	Network	Throughput			Latency	#nodes	#machines
			Peak	Max Scale	Under Attack			
Elastico [215]	country-wide	emulated	20 KB/s	20 KB/s	N/A	800 sec	1,600	800
Algorand [42]	country-wide	emulated	208 KB/s	90 KB/s	∼	22 sec	50,000	1000
Omniledger [89]	datacenter	emulated	205 MB/s	1.8 MB/s	N/A	14 sec	1,800	60
RapidChain [154]	datacenter	emulated	4.2 MB/s	3.8 MB/s	N/A	9 sec	4,000	32
Mir [220]	world-wide	real	140 MB/s	30 MB/s	0	5 sec	100	100
RBBC	world-wide	real	264 MB/s	12.3 MB/s	760 KB/s	3 sec	8,560	1000

higher than others.

Chapter Outline The remainder of the chapter is as follows: Section 4.1 introduces the threat model and provides motivation for the work; the goals and assumptions are introduced in Section 4.2; in Section 4.3 the Red Belly blockchain and the architecture surrounding the two main novelties, the sharded verification and the unified consensus are presented; Section 4.4 details the implementation; Section 4.5 explains why the Red Belly Blockchain scales with the number of nodes while presenting information on correctness and durability; experimental evaluation is presented in Sections 4.6 and 4.7; Section 4.8 concludes the chapter. Proofs of correctness are deferred to Appendix D.

4.1 Threat model

Byzantine Fault Tolerant blockchains often aim to solve the traditional consensus problem, as described in Chapter 2. This may lead to exposing themselves to a number of threats summarised below.

Double-Spending Solving the traditional consensus problem in the context of the blockchain aims to guarantee that all correct nodes agree on a unique block at a given index [12]. Uniqueness avoids forks, where an attacker could spend two of the same coin in different branches [34],

typically found in blockchains with probabilistic guarantees [9, 27, 215, 42, 89, 154]. Although this provides improvements for scalability, as depicted in Table 4.1, the probability of consensus failure grows with the number of blocks that need to be agreed upon.

A blockchain based on deterministic consensus ensures that consensus is reached among correct nodes if less than a third of all nodes are Byzantine, so $n < 3f$. The traditional definition, however, unnecessarily limits the scalability of the blockchain [65, 107, 227]: most blockchains decide *at most one* of the proposed blocks [107, 227]. Indeed, the leader, whose proposal is eventually decided, needs to propose requests to many nodes, its network interface thus acting as a bottleneck [228, 229]. By contrast, RBBC solves the *Set Byzantine Consensus problem* by deciding up to $\Omega(n)$ proposals (Theorem 9) when $n > 3f$ and no forks are possible even when the communication is asynchronous.

Incorrectly signed transactions To decide a superblock that combines all proposed blocks, one may think of solving a variant of the consensus problem to instead combine proposals, into a decision [230, 231, 51]. For example, Agreement on a Core Set (ACS) [230], Interactive Consistency (IC) [14] and Vector Consensus (VC) [231] all require at least $f+1$ (either $n-f$ or $f+1$ with $n > 3f$) proposed values to be decided. In blockchain, however, there may not even be $f+1$ compatible proposed blocks: when among $2f+1$ blocks one transaction per block is not correctly signed or transactions of distinct blocks *conflict* in that they are concurrent and withdraw assets from an account that has insufficient balance. This is why, the *Set Byzantine Consensus* problem is introduced (Section 4.2) that ensures that all correct nodes extract the correctly signed and non-conflicting transactions from the same set of proposals. This allows RBBC to combine proposed valid blocks into a *superblock* (Section 4.4.3) for its performance to increase with the system size.

Unfairness Discarding transactions is required to provide resilience to anomalies as described in Chapter 3, frontrunning [205, 226] and oligarchy [206]. This means that a correct requester's transaction will never remain uncommitted due to invalid transactions always being committed first. Hyperledger Fabric [171] suffers from censorship due to spamming attacks denying service by keeping the ordering service busy with invalid transactions, as acknowledged by the authors. FastFabric [232], an optimised version, mitigates this issue by being approximately 7 times faster, however, due to the nature of the ordering service, none of these versions tolerate malicious failures (including Denial-of-Service attacks). Even its Byzantine-fault tolerant ordering service [227] suffers from this issue as it cannot detect whether a transaction is valid. RBBC favours older requested transactions to achieve censorship-resistance (Theorem 11).

Network Attacks Many mainstream blockchains assume synchrony, where all messages must be delivered within some time bound [9, 27, 215, 42, 89, 154]. The main drawback is vulnerability to various network-related attacks that abuse this synchrony assumption to facilitate double-spending [36, 3, 190, 191]. Other blockchains [138, 89, 137, 157] assume synchrony, but use partially-synchronous consensus. Ouroboros reached 247 TPS [108] and assumes synchrony,

but its Praos version [109], which does not, has no evaluation at the time of writing. RBBC only assumes partial synchrony to tolerate unknown delays.

Adversarial Schedulers As an alternative to assuming synchrony, other recent blockchains solve consensus probabilistically [144, 69, 70, 221]. Stellar [221] proposes a probabilistic leader election for the consensus. Honey Badger's BFT Consensus (HBBFT) [144] solves ACS building upon an asynchronous binary Byzantine consensus [69] that does not terminate under an adversarial scheduler [146]. Highlighted in Section 4.6.2, HBBFT is too costly for our needs because each of its nodes creates $n-1$ erasure coded messages and $n-1$ signatures, and verifies $\Omega(n^2)$ signatures. BEAT [233] and Dumbo [234, 235] improve over HBBFT when transactions are respectively less than 10 bytes and 250 bytes but build upon the same consensus algorithm [69]. Some consensus alternatives relax this assumption but require more messages, which risks to increase the overhead [70]. RBBC does not assume a fair scheduler.

Faulty Leaders Various systems [15, 236, 237, 238, 239, 240, 241, 149, 220] assume partial-synchrony to avoid the cost associated with randomisation, such as common coin, or synchrony. Unfortunately, however, they rely on a leader and will revert to a costly recovery mechanism in the event of a failure [242, 243, 244]. Tendermint [106, 107] uses a variant of PBFT but cannot scale beyond tens of nodes, and suffers from issues [142, 245, 57] such as liveness and termination. However, newer additions [141, 142] propose updates to reduce message complexity for leader changes and potential scalability properties. SBFT [246] uses threshold signatures to reduce communication complexities of PBFT, evaluated in a wide-area network as a key-value store and as a blockchain peaking at 172 TPS. ByzCoin [137] relies on PBFT using multicast trees to reduce messages to $O(n)$, also relying on a leader.

Other leaderless consensus algorithms [247, 243, 248, 249] are unfortunately incomplete, impractical or cannot be easily extended to implement a blockchain with sharded verification.

Single Point of Slowdown Requiring a leader places heavy load on the bandwidth and performance of that node, often becoming a bottleneck with higher workloads. HotStuff [149] is a Replicated State Machine (RSM) that tries to reduce the leader load by sending only the digest of each request. Its implementation[1] exchanges asymptotically as many bits as RBBC per committed transaction but requires clients to send their transactions to all correct consensus participants. Mir [220] is a deduplicating total order broadcast protocol that builds upon our sharded verification. Although not formally stated, it could potentially solve the Set Byzantine Consensus (SBC) problem (Def. 7) as it also leverages multiple proposers. The key difference is that it builds upon the PBFT leader-based consensus algorithm by combining n PBFT proposers (called 'leaders' in Mir) and one leader (called 'primary' in Mir). In Mir, verification sharding may fail due to a single faulty or slow replica, in which case it reverts to full verification. In RBBC, faulty or slow nodes do not stop verification sharding as verification priorities are

[1] Online, available: https://github.com/hot-stuff/libhotstuff

assigned to replicas so that faster replicas simply verify on behalf of the faulty or slow replicas. The throughput of HotStuff and Mir drops to 0 when their leader is faulty or slow as was reported separately for HotStuff [149] and Mir [220, Table 4.1]. As a full-fledged blockchain, RBBC ensures durability (Section 4.5) and tolerates failures by always deciding proposals.

Human Errors These consensus algorithms can appear extremely complex, especially dealing with monolithic architecture [14, 15, 236]. Blockchain consensus requires tolerating conflicting proposals [250], as transactions, from different nodes which can lead to dramatic losses with erroneous consensus implementations. Some suffer from known errors [251], others may be formally specified [248] but too large to be machine checked and may appear erroneous [252]. Such errors have already affected blockchains, for example the HBBFT randomisation was found to be non-terminating [146]. Attempting to limit blockchain errors lies in theorem provers [253, 254], but they do not check algorithms.

In contrast, the consensus algorithm at the heart of RBBC, DBFT [51], was formally proven to be safe and live for any parameters of n and $f < n/3$ using complete model checking [223]. To achieve this, the multivalue consensus protocol decomposes into reliable broadcast [22] and binary consensus using the reduction by Ben-Or et al. [230]. The reliable broadcast was formally verified with model checking [223]. Although these verifications rely entirely on the correctness of the model checker, compiler, language, and other factors, it considerably reduces human errors.

4.2 Goals and assumptions

The goal is to implement a blockchain system whose performance scales with a number of consensus participants that treat (verify cryptographically and totally order) a large amount of transactions sent by requesters all over the world and mitigates issues highlighted in Chapter 3. The communication model is the classic resilience optimal model with partial synchrony [23] and $f < n/3$ [15].

4.2.1 Open permissioned system

An "open" distributed system is considered. It consists of nodes that do not need any permission to join, called *replicas* and *requesters* ('requester' is equivalent to 'client' in the distributed computing literature but differs from the notion of 'client' of the Ethereum documentation). The replicas receive blocks from other nodes and maintain a copy of the current state of the blockchain whereas the requesters simply act as clients, requesting balances and issuing transactions. This is called an "open permissioned" system because nodes need some permission to play the roles of *proposers* and *verifiers*. Each proposer collects a set of requested transactions and proposes it periodically as a batch whereas the verifiers check the transaction signatures, a procedure called *verification*. This open permissioned model is appealing for assigning permissions based on Proof-of-Stake in Ethereum 2.0 or for revoking permissions upon misbehaviour's

in LLB [255]. Similar to Ethereum 2.0; Polkadot [256], EoS [52] and Cosmos [257] employ similar open-permissioned systems where specific permissioned nodes play the roles of proposers and verifiers.

4.2.2 Transaction model

Let T be the set of all possible transactions and let any transaction $tx \in T$ be a tuple of $\langle a, e, j, \sigma \rangle$ that represents a transaction with non-forgeable signature σ transferring amount j from account a to account e (implemented with UTXO as explained in Section 4.3). We use SSL handshakes with certificates listed within blocks for secure channels, and new blockchain accounts create new key pairs that they use after they receive coins. Let a proposal $s \subset T$ be a set of transactions. Let $S = T^*$ be the set of possible proposals, or the set of possible sets of transactions. A transaction $\langle a, *, j, \sigma \rangle$ is *provisioned* if the balance of a is larger than j; it is *valid* if it is provisioned and σ is the signature of the owner of a. A set of transactions is valid if all its transactions are valid; it is *non-conflicting* if no subset of its transactions are cumulatively withdrawing more from any account than its balance. Note that the transactions could be multisigned as their execution is not sharded.

Initially, all nodes have a copy of a special block called the *Genesis* block of the blockchain that indicates the initial balance of some addresses (i.e., accounts) and the identities of the permissioned nodes identified by their public keys. Note that because permissioned nodes are listed in blocks, they do not have to be "pre-selected" as in traditional consortium blockchains [64] but can instead change periodically to avoid bribery attacks [160].

4.2.3 Failure model

The failure model is Byzantine in that *faulty* nodes can fail arbitrarily [12]. Non faulty nodes are referred to as *correct*. More specifically, among the n permissioned nodes there are at most $f < n/3$ faulty nodes. Among these n permissioned nodes, the set V of verifiers contains at least $f+1$ correct nodes and the set P of proposers contains at least one correct proposer, which is easily ensured with $|V| = |P| = n$. Note that any number of requesters and replicas that are not part of these permissioned nodes can be faulty. To ensure termination it is assumed that the communication is partially synchronous [23] in that there exists an unknown global stabilisation time after which all messages sent are delivered in less than a fixed amount of time. In order to guarantee that the system is censorship-resistant (Theorem 11) despite Byzantine nodes, the following assumptions are made:

1. *sequential-transaction-requests:* a correct requester does not issue invalid transactions or two conflicting transactions;
2. *bounded-requesting-rate:* the rate at which transactions enter the *Mempool* (memory pool) of proposers is lower than the rate at which transactions get proposed by proposers.

Note that assumption (1) only applies to correct nodes in order to guarantee that their transaction will be eventually treated by the system; the system cannot guarantee that transactions

issued by Byzantine requesters will be treated (as their transactions may be invalid). Assumption (2) precludes Denial-of-Service (DoS) attacks where correct proposers would receive too many requests to keep track of them within their bounded memory. The likelihood of a DoS attack is reduced by showing empirically that RBBC treat a large volume of transactions for a long period (Section 4.6).

4.2.4 Goal

Our goal is to implement a censorship-resistant Replicated State Machine (RSM). Censorship-resistant, means that any transaction issued properly by a requester gets committed by the system, hence preventing censorship (Section 4.1). Replicated State Machine, refers to a way of totally ordering sets of transactions in the form of a blockchain, starting from the Genesis block, so that all transactions are provisioned and linearizable [258]. To this end, it is required to solve a variant of the BFT consensus problem to agree on an enumerable valid subset of the union of the proposed values as follows.

Definition 7 (Set Byzantine Consensus)**.** *Assuming that each correct node proposes an enumerable set of transactions as a proposal, the* Set Byzantine Consensus *(SBC) problem is for each of them to decide on a set in such a way that the following properties are satisfied:*

(1) SBC-Termination: every correct node eventually decides a set of transactions;

(2) SBC-Agreement: no two correct nodes decide different sets of transactions;

(3) SBC-Validity: a decided set of transactions is a valid non-conflicting subset of the union of the proposed sets;

(4) SBC-Nontriviality: if all nodes are correct and propose an identical valid non-conflicting set of transactions, then this subset is the decided set.

The SBC-Termination and SBC-Agreement properties are common to many Byzantine consensus definition variants, while SBC-Validity is different: it states that transactions proposed by Byzantine proposers could be decided as long as they are valid and non-conflicting, where conflicting transactions refer to those that consume the same UTXO as input. SBC-Validity is inspired by the external validity property [74, 259] that requires a decision to be valid and the idea of deciding at least $f + 1$ proposed values [14, 230, 260], however, SBC-Validity cannot result from any combination of these properties. For example, the union of strict subsets of all proposed values is a possible SBC decision. SBC-Nontriviality prevents a trivial algorithm that always outputs an empty set from solving the problem.

4.3 The Red Belly Blockchain

The architecture surrounding the two main novelties of the RBBC are presented, namely; its use of few computational resources when verifying transactions and the consensus that leverages communication to commit more transactions as the system grows in size but bandwidth becomes limited.

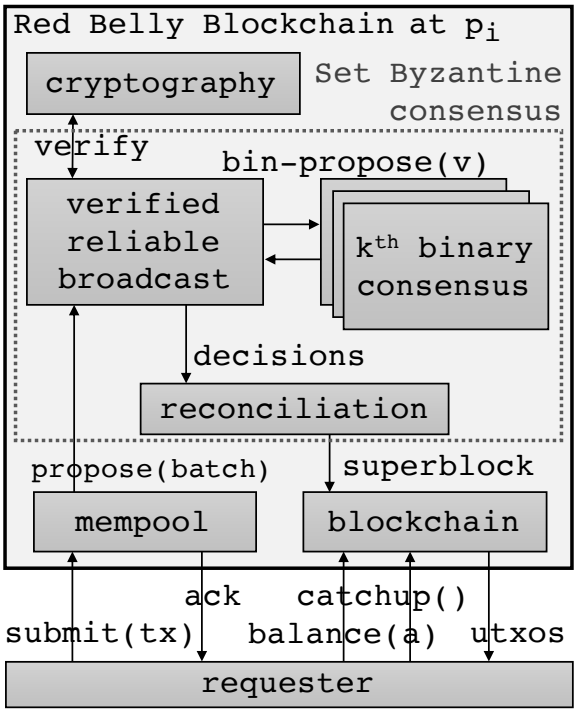


Figure 4.1: RBBC accepts transactions *tx*, balance and catchup requests. Its mempool batches transactions and proposes to the Set Byzantine Consensus, which invokes the verified reliable broadcast and multiple consensus instances. Once all binary consensus instances have decided, reconciliation is invoked to group proposals into a decided superblock stored on the blockchain.

4.3.1 Architecture

Figure 4.1 shows an overview of the architecture of the RBBC, it contains a memory pool (*mempool*) for transactions, a Set Byzantine Consensus with DBFT, a cryptographic component for verification and communication, a reconciliation component and the blockchain assets that interact and store the superblocks and related state information. The Set Byzantine Consensus contains: (i) a verified variant of the Reliable Broadcast [22], (ii) a reduction from the multivalue consensus problem to the binary consensus problem, (iii) a binary consensus and (iv) a reconciliation protocol used to compose a superblock containing multiple sets of transactions. Periodically, the $|P|$ proposers extract a subset of transactions from their mempool and propose to the multivalue consensus.

4.3.2 Reducing the computation at small scale

Verification is required to guarantee that Unspent Transaction Outputs [9] are correctly signed. Signature verification is CPU-intensive, and can notably affect performance (as is experimented in Section 4.6.1), therefore by distributing the verification, or sharding the verification, the verifying nodes will verify subsets of the transactions without reducing the security. The expected result is twofold. Firstly, it improves performance as it reduces each verifier's computational

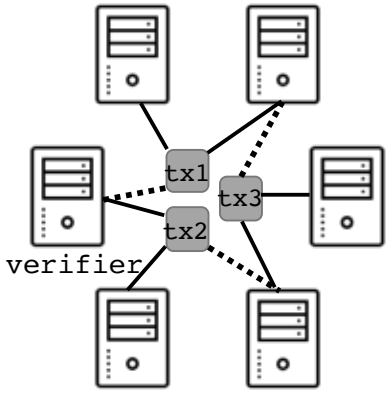


Figure 4.2: Each transaction is verified by $f+1$ primary verifiers (solid lines), before being verified if necessary by f secondary verifier nodes.

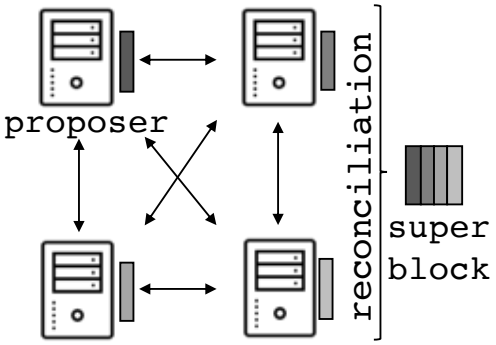


Figure 4.3: $n = |P| = 4$ proposals depicted with grey rectangles are all combined in the decided superblock.

load. Secondly, it helps scaling by further reducing computational load on all verifiers as the number increases. It is confirmed empirically in Section 4.6.1 that verification sharding divides verification load by at most 3×.

As f verifier nodes can be Byzantine, each transaction signature must be checked by at least $f+1$ verifier nodes. If all results are unanimous, then the signature check will pass as there is at least one correct node that has verified the signature and it follows that the transaction is correctly signed. However, if the results are not unanimous, given that f may be Byzantine, a transaction may need to be verified up to $2f+1$ times before $f+1$ equal responses are computed. As depicted in Figure 4.2, each transaction is mapped to $f+1$ *primary verifiers* who eagerly verify this transaction, while f *secondary verifiers* lazily verify this transaction if necessary, hence summing to $2f+1$ total verifications.

4.3.3 Leveraging bandwidth at large scale

At a large scale, geodistributed proposers are likely to receive different sets of transactions originating from requesters located within their vicinity. Rather than a consensus that accepts only one batch of transactions in a single block, while discarding all other proposals, RBBC combines all sets of transactions proposed by distinct proposers into a unique superblock to improve the performance (as we quantify in Section 4.6.2).

In particular, RBBC decides upon the union of the proposed sets of transactions. To illustrate why this is key for scalability, consider that each of the n consensus participants have $O(1)$ transactions to propose. Traditional Byzantine consensus will decide $O(1)$ transactions, whereas the RBBC can decide up to $\Omega(n)$ transactions. The typical communication complexity of Byzantine consensus is $O(n^4)$ bits [15, 51], it results that $O(n^3)$ bits are required per committed transaction in the RBBC (c.f. Theorem 9), instead of $O(n^4)$ and without suffering

from a leader bottleneck. Some leader-based approaches limit the communication complexity by relying on threshold encryption [246, 259, 149], whereas our goal is to avoid additional cryptography to reduce the CPU load.

Figure 4.3 depicts how RBBC achieves this optimisation, consider $n = 4$ permissioned nodes proposing unique sets of transactions, but all decide a value that is a super block containing a union of all sets of transactions that are proposed. It results from this optimisation that the number of decided transactions grows linearly with n, as long as each transaction is proposed by a constant number of correct proposers (c.f. Theorem 9). Note that some of these transactions may not be executable as they conflict, hence the requirement for a reconciliation procedure (Section 4.4.3).

4.3.4 Assigning roles to nodes

It is now explained how the nodes are assigned roles of verifiers and proposers using a deterministic function. For each consensus instance, there is an ordered set of P permissioned node identifiers, where $f < |P| \leq n$, indicating that nodes play the role of primary or secondary proposers for all transactions. Note that this determinism does not imply predictability, as one can change proposers [42] directly, and such changes can also be decided deterministically and anonymously by the consensus nodes themselves with a recent voting protocol also based on DBFT [261]. Although $2f < |P| \leq n$ is necessary to achieve censorship-resistance, a requester never needs to send requests to more than $f + 1$ proposers.

For a requester to identify the proposers responsible to propose a given transaction (tx) to the consensus, it executes a deterministic function $\eta(a)$ that takes as input the source account, a, of transaction tx and returns the identifier of a node $p_i \in P$ called the *Primary proposer* for tx. To guarantee that a transaction is proposed (despite a faulty primary), between f and $n - 1$ secondary proposers distinct from p_i are also selected deterministically. The number of proposers of each transaction tx is at least $f+1$ to guarantee that tx will be proposed by at least one correct proposer (c.f. Theorem 11). The number of proposers can be as large as n, however, fewer proposers result in potential lower latency, whereas more proposers increase throughput as is highlighted experimentally in Section 4.6.2.

As each transaction must be verified by between $f+1$ and $2f+1$ verifiers, each proposer p_i is also mapped to a set of $f+1$ $primary\_verifiers_{p_i}$ and a set of f additional $secondary\_verifiers_{p_i}$. The $primary\_verifiers_{p_i}$ include p_i itself, and verify upon reception of the signatures for tx; so the basic design makes some nodes both proposers and verifiers. If the verification returns the same $f + 1$ results, then it becomes clear whether the signature of tx is correct. If not, f additional verifications are required to identify the majority $f+1$ identical responses to indicate whether the signature of tx is in fact correct. This is why the $secondary\_verifiers_{p_i}$ set includes nodes of P that are distinct from the $primary\_verifiers_{p_i}$ and that verify tx in case one or more of the primary verifiers are faulty or slow. One could choose to have more verifiers, however,

this would result in wasted CPU on extra unnecessary verification (c.f. Section 4.6.1).

4.4 Design and implementation

The design and implementation of the RBBC are detailed, describing communication and the flow of execution. Requesters request the balance of an account, or, can send a transaction to $f+1$ proposers. All communication is exchanged through SSL-encrypted channels. The following methods are exposed by the permissioned nodes through a JSON RPC to the requesters, as they are able to submit requests directly:

- `submit`(tx) submits a transaction, running on the proposer node taking a transaction (tx) and a set of UTXOs averaging 400 bytes from a requester. If tx is provisioned, does not exist and does not conflict with any transaction in the mempool, it is then placed in the mempool and `true` is returned to acknowledge recipient of the transaction. A correct requester calls this method to $f+1$ unique proposers.
- `balance`(a) takes an account a as input, returning the valid UTXOs for that account. A requester performs this operation by contacting different proposers until $f+1$ equal notifications are received. Not only does it allow small devices to securely consult the state without downloading the blockchain, it also guarantees integrity of the ledger.
- `catchup()` is a method for lagging, or recovering, replicates to update and find information about the current index of the blockchain.
- `getblock(`$index$`)` takes a number $index$ representing the index of the blockchain, returns the super block at that index.
- `getblockhash(`$b.Hash$`)` takes the unique hash id of a block, $b.Hash$, and retrieves the block with that hash.

4.4.1 Normal consensus execution

Figure 4.4 depicts a normal consensus execution of $n = |P| = 4$ and $f = \lceil n/3 \rceil - 1 = 1$, where a single requester sends a request to $f+1$ proposers for simplicity[2], and each of the proposers executes the following:

❶ **Request.** A requester computes $\eta(a)$ with the source account a of transaction tx to retrieve the mapped proposers and verifiers of tx and sends the request for transaction tx to the $f+1$ primary proposers and verifiers. Upon reception, tx is added to the mempool and is verified by the primary verifiers.

❷ **Propose.** Each proposer selects, from their mempool, transactions (i) for which it is the primary proposer and (ii) in decreasing order of their *age* or the number of blocks appended to the chain since these transactions arrived. This batch is proposed to the consensus by sending it in an INIT message to all other permissioned nodes. (Note that a proposer with an empty mempool starts the consensus with an empty proposal if it receives a non-empty proposal from another node to guarantee sufficiently many participants.)

[2]Many requesters typically request proposers in parallel.

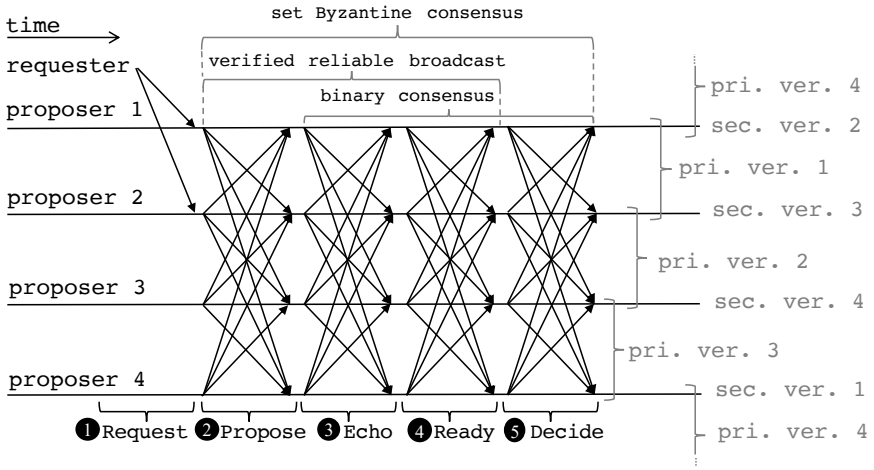


Figure 4.4: The typical message pattern of the consensus protocol between each proposer (i) and the permissioned nodes that play the role of primary verifiers (pri. ver. i) and secondary verifiers i (sec. ver. i) for proposer i.

❸ **Echo.** Upon reception, permissioned nodes broadcast a digest of each received proposals in ECHO messages.

❹ **Ready.** Upon reception of $n-f$ equal ECHO messages, verifiers for the received proposals verify them (if not already done in the request step) and send the result in a READY message (Section 4.4.2) to all proposers. Upon reception of $f+1$ equal READY messages, a node broadcasts the READY message if it has not done so already. (This step is represented by a single message exchange in Figure 4.4 due to the fast Ready-Decide optimisation presented below.)

❺ **Decide.** Upon reception of $2f+1$ equal READY messages (Section 4.4.2) for a particular proposal, the nodes store this proposal in an array indexed by the sender id and input 1 to a corresponding binary consensus instance. This is repeated for other proposals until $|P|-f$ binary consensus instances decide 1, after what a reconciliation (Section 4.4.3) combines into a superblock every valid transaction of proposals for which a binary consensus output 1.

Good execution complexity and scalability In good executions, RBBC differs from previous work by committing securely $\Omega(n)$ transactions within 4 message delays (❷–❺) thanks to the *Ready-Decide* optimisation: In the *Ready step*, if the verifiers are fast enough to verify before the reception of $f+1$ equal READY messages, then they broadcast a READY message. As a result, all nodes receive $2f+1$ equal READY messages in a single message delay allowing them to enter the *Decide step* by inputting 1 to a binary consensus instance directly. At the end, the proposers pick the transactions from the $\Omega(n)$ proposals for which the binary consensus decided 1 and *reconciliate* them into a superblock. The complexity and throughput of RBBC are computed in Section 4.4.3 and Theorem 9.

4.4.2 Verified all-to-all reliable broadcast

For proposers to exchange verified proposals, `secp256k1` *Elliptic Curve Digital Signature Algorithm (ECDSA)* verification is added to the reliable broadcast, which is originally a 3-step one-to-all communication abstraction exchanging INIT, ECHO and READY messages where any message delivered to a correct proposer gets eventually delivered to all correct proposers [22].

1: verified-reliable-broadcast(v): ▷ *verified variant of reliable broadcast at p_i*
2: broadcast(INIT, v) ▷ *broadcast value v to all*
3: **upon** receiving a message (INIT, v) from p_j:
4: broadcast(ECHO, $h(v), j$) ▷ *echo the hash digest of v*
5: **upon** receiving $n - f$ (ECHO, $h(v), j$) msgs and not having sent READY:
6: **if** $p_i \in$ *primary_verifiers*(v) **then** *verif* $\leftarrow$ verify(v)
7: **if** $p_i \in$ *secondary_verifiers*(v) **then** wait(Δ); *verif* $\leftarrow$ verify(v)
8: broadcast(READY, *verif*, $h(v), j$) ▷ *piggyback verifications*
9: **upon** receiving $f + 1$ (READY, *verif*, $h(v), j$) and did not send READY:
10: stop-verify(v) ▷ *prevent unnecessary verifications*
11: broadcast(READY, *verif*, $h(v), j$) ▷ *piggyback verifications*
12: **upon** receiving $n - f$ (READY, *verif*, $h(v), j$) and not delivered from j:
13: **if** is-verified(v, *verif*) **then** deliver(v, j) ▷ *deliver if sufficiently verified*

Our verified variant of the reliable broadcast adds a verification function verify (lines 6 and 7) before the broadcast of the READY message. A proposer broadcasts an INIT message with a proposal v (line 2). Upon reception, its digest $h(v)$ is broadcast in an ECHO message (line 4); we use the SHA256 digest to save bandwidth as proposals can contain thousands of transactions (Section 4.6.2). Upon reception of $n - f$ equal ECHO messages, the verification of the proposal starts first at the *primary_verifiers*(v) and later, if necessary, at the secondary ones. After the verification, a list *verif* of integers indicating the indices of invalid transactions in the proposal is appended to the READY message, which is then broadcast (line 8) with the digest of the corresponding proposal. After receiving the same *verif* field for $h(v)$ from $f + 1$ distinct processes (line 9), a node knows which transactions of v are valid. Upon reception of $n - f$ equal READY messages, v is delivered if it contains valid transactions (line 13). The proof that the verified reliable broadcast ensures the properties of reliable broadcast for each valid value (and discards invalid values) relies on the fact that the sets Q of correct proposers and Q' of verifiers are such that $Q \cap Q' \geq f + 1$, guaranteeing that the READYmessage with the same *verif* will be sent at line 11 [7]

4.4.3 Agreeing on a superblock

Ben-Or, Kelmer and Rabin's reduction (lines 14–23) of the multi-value consensus problem is modified to the binary consensus problem [230] to solve the Set Byzantine Consensus (Section 4.2) by replacing the reliable broadcast by our verified reliable broadcast (Section 4.4.2) and invoking a reconciliation to decide a superblock of non-conflicting provisioned transactions (Section 4.2). Symbol $\rightarrow$ indicates that the array *props* gets populated in the background by all concurrent reliable broadcasts (line 15).

For each of the first proposals delivered at p_i by the verified reliable broadcasts (Section 4.4.2), p_i proposes 1 to a binary consensus instance (line 19). Proposer p_i proposes 0 to the remaining binary consensus instances (line 21) after a timer expires and $|P| - f$ consensus return 1. This timer (line 16) increases with the age of the oldest transaction of the mempool to potentially decide it.

```
14: propose(val):                                   ▷ set Byzantine consensus at p_i with val a batch of txs
15:   verified-reliable-broadcast(val) → props      ▷ exclude invalid txs Section 4.4.2
16:   start-timer(age of oldest tx in mempool)      ▷ give time to slow ones
17:   while |{k : bitmask[k] = 1}| < |P| − f or timer did not expire do
18:     for all k such that props[k] has been delivered   ▷ for all delivered
19:       bitmask[k] ← bin-propose_k(1)             ▷ propose 1 to k^th binary consensus
20:   for all k such that props[k] has not been delivered   ▷ for undelivered
21:     bitmask[k] ← bin-propose_k(0)               ▷ propose 0 to k^th binary consensus
22:   wait until bitmask is full and ∀ℓ, bitmask[ℓ] = 1 : props[ℓ] ≠ ∅
23:   reconciliate(bitmask & props)                 ▷ combine into a superblock
```

All binary consensus instances proceed in parallel (their invocation is non-blocking). The decisions of these binary consensus instances constitute a *bitmask* that is applied to the set of potentially decidable proposals (line 23). Although the array of verified proposals may differ across correct nodes, the bitmasks of all correct nodes are guaranteed to contain 1s (Lemma 21) and be identical due to the agreement properties of the binary consensus. Note that even though the proposal may not be known yet for some of these indices, it is guaranteed by the reliable broadcast to be eventually delivered at all correct proposers (Section 4.4.2). Each correct proposer waits until a proposal has been delivered at each of these indices (line 22), then each correct proposer obtains the same set of proposals after applying the *bitmask*. The unselected proposals are stored in the mempool for later.

Reconciliation

To fill the superblock, we reconciliate the decidable set of proposals (lines 24–29). Correct proposers extract deterministically the transactions by going through all proposals and through each of their transactions one-by-one, to add the non-conflicting provisioned ones to the superblock. With the UTXO model, one can easily ensure all transactions are provisioned and non-conflicting by implementing conflict$(tx, *)$ that executes transaction tx and returns false if all the UTXOs tx consumes exist and otherwise returns true. For the sake of fairness (i.e., to not favour any particular proposer), correct proposers traverse the proposals from the index number ($k \mod n$) to index number ($k-1 \mod n$) where k is the index of the last superblock in the blockchain. This prevents the proposer with the lowest index number from having its proposed transactions added to the superblock with a higher priority than the transactions of other proposers. (In Section 4.6.2 it is shown that the computation needed for reconciliation is negligible compared to the signature verification.)

```
24: reconciliate(props):                                  ▷ combine proposals, initially superblock = ∅
25:   for i = 0..(n − 1) do                                ▷ use the last block index k for fairness
26:     for tx ∈ props[(k + i) mod n] do                  ▷ starting from the first tx of proposal
27:       for ctx ∈ superblock do                         ▷ for all committed transactions
28:         if ¬conflict(tx, ctx) then superblock ← superblock ∪ {tx}
29:   decide(superblock)                                  ▷ decide superblock of non-conflicting transactions
```

Binary consensus

To solve the binary consensus deterministically, the binary consensus of DBFT [51] was chosen because it is resilience optimal, time optimal and was recently verified with model checking [146, 223]. The binary Byzantine Consensus (BBC) problem is for each process to decide on some value $V \in \{0, 1\}$ in such a way that the following properties hold [51]:

- **BBC-Termination**: Every non-faulty process eventually decides on a value.
- **BBC-Agreement**: No two non-faulty processes decide on different values.
- **BBC-Validity**: If all non-faulty processes propose the same value, no other value can be decided.

```
30: bin-propose(val):                                     ▷ binary consensus at p_i with val ∈ {0, 1}
31:   loop:                                               ▷ loop that starts with round r = 1
32:     (bv-broadcast(EST, r, val) → cvals)               ▷ reliable bcast if not done at l.15
33:     start-timer(r)                                    ▷ timeout increases with rounds
34:     if i = r mod n then                               ▷ coordinator rebroadcasts
35:       wait until (cvals = {w})                        ▷ cvals stores delivered values
36:       broadcast(COORD, r, w) → c                      ▷ coordinator broadcasts
37:     wait until (cvals ≠ ∅ ∧ timer expired)            ▷ wait enough time
38:     if c ∈ cvals then e ← {c} else e ← cvals          ▷ prioritize coord value
39:     broadcast(AUX, r, e) → bvals                      ▷ broadcast these values
40:     wait until ∃s ⊆ bvals where the two following conditions hold:
41:       • s contains contents received from at least n − f distinct nodes
42:       • ∀v ∈ s, v ∈ cvals                             ▷ every value in s is in cvals
43:     if s = {v} then                                   ▷ if there is only one value in s
44:       val ← v                                         ▷ adopt this singleton value
45:       if v = (r mod 2) and not decided yet then decide(v)   ▷ decide
46:     else val ← (r mod 2)                              ▷ otherwise, adopt the current parity bit
47:     if decided in round r − 2 then exit()             ▷ help others in two last rounds
48:     r ← r + 1                                         ▷ increment the round number
```

Each replica refines an estimate value, initially its input value to the consensus, across consecutive rounds until it decides (line 45). It invokes broadcast primitives that deliver some values into a dedicated variable pointed out by → at lines 32, 36 and 39. The bv-broadcast (line 32) is a reliable broadcast for binary values [70]. (It is optimised by piggybacking it for $r = 1$ with the verified-reliable-broadcast at line 15.) One replica per round acts as a coordinator by broadcasting its value c (line 36) that others prioritise (line 38) to help them converge to

the same decision. Hence, RBBC is leaderless with multiple coordinators. The binary values are then forwarded in AUX messages (line 39) and each replica waits to receive a sufficiently represented set of these AUX values (lines 40–42). If only one value is sufficiently represented (line 43) and if it corresponds to the parity of the round, then it is decided (line 45). Otherwise, *val* is set to the parity of the round and another round starts.

Complexity

In the worst case, each bin-propose decides within $O(f)$ rounds after the network stabilises and messages start being delivered in bounded time, which is optimal [222]. Hence, as the (multivalue) propose needs a constant additive factor of message delays, its time complexity is asymptotically optimal. Moreover, propose is resilience optimal as it tolerates any $f < n/3$ failures [14]. It has the same communication complexity $O(n^4)$ as PBFT [15] and DBFT [51] but decides up to n times more transactions than them in good executions: $O(n^3)$ bits exchanged per committed transaction batch. Recall that the communication complexity of PBFT is $O(n^4)$ bits because there are at most $f + 1$ view-change rounds, the message in each round contains the state received from the previous view-change rounds, which is $O(f)$ bits, and is broadcast by n nodes, leading to $(f + 1) \cdot O(f) \cdot (n - 1) \cdot n = O(n^4)$. Both PBFT and DBFT exhibit a time complexity of $O(f)$, making them both resilient and time optimal.

When comparing to blockchain-focused consensus, Tendermint [142] and HotStuff [149] exhibit a lower bit complexity than DBFT at $O(n^3)$ bit complexity. However, it should be noted that DBFT provides $\Omega(n)$ transactions to be decided.

4.5 Correctness analysis

To explain how RBBC implements a secure, fair and scalable blockchain, disallowing double-spending by implementing an RSM is demonstrated. Exclusively valid non-conflicting transactions are stored in reliable storage by RBBC (Theorem 8). Finally, the reason why throughput scales with the number of nodes is explained (Theorem 9) and it is shown that RBBC offers censorship resistance (Theorem 11). The proofs are deferred to the appendix (Appendix D).

Correctness

To avoid forks that could lead to double-spending, RBBC executes consensus instances in a totally ordered sequence and at the end of each instance the decided superblock orders all transactions it contains in the same order at all replicas. Note that to implement a blockchain system, RBBC offers stronger guarantees than a simple RSM, by for example discarding the incorrectly signed transactions eagerly (line 13) and the conflicting transactions lazily (lines 28).

Theorem 8 (Replicated State Machine)**.** *In RBBC, all correct replicas observe the same sequence of committed transactions, which are all valid and non-conflicting.*

Proof. In RBBC, all permissioned nodes run the consensus algorithm either because they receive messages from proposers or because they propose themselves (Section 4.4.1[❷]). Each node

starts by running a single instance of this consensus algorithm for the block at index 1 (after the Genesis block). A proposer can start a new consensus instance for a block at index $j > 1$ only after the consensus instance at index $j - 1$ has terminated. As Theorem 23 shows that consensus guarantees agreement there is a single block decided per index of the blockchain at correct permissioned nodes. The result is that blocks are totally ordered through their index number: whenever a block is decided, it is ordered after all previously decided blocks. Given that in each block the transactions do not conflict and are ordered through the same deterministic strategy employed by all correct permissioned nodes (lines 24–29), transactions are replicated by $f + 1$ correct permissioned nodes in the same total order. By examination of the code, all committed transactions are valid and non-conflicting because RBBC discards the incorrectly signed transactions eagerly (line 13) and the non-provisioned and conflicting transactions lazily (lines 28). □

Durability

To ensure durability [262], the remaining transactions are stored in a block in an append only log on disk. For recovery, each node keeps a write-ahead log containing the messages it has broadcast during at least the previous two committed multivalued consensus instances, older messages being garbage collected. To ensure transactions are verified and committed quickly, the UTXO table is stored in memory. To minimise the size of the UTXO table, transactions should consume all UTXOs for their account. After a crash, nodes can reconstruct their UTXO table by parsing their blocks from disk. Nodes that need to recover messages from older consensus instances simply collect $f + 1$ equal instances of the decided block.

Scalability

The scalability of RBBC is explained by showing that RBBC commits $\Omega(n)$ transactions per consensus instance in good executions where BFT algorithms (e.g., PBFT, DBFT, Tendermint, HotStuff) that do not solve SBC (Def. 7) commit $O(1)$ transactions. Note that leader-based consensus algorithms (PBFT, Tendermint, HotStuff) could have been modified for their leader to batch $\Omega(n)$ times more requests, however, this would have added to the leader load. More dramatically, the leader would have sent $\Omega(n)$ bytes to $n - 1$ nodes, which would have taken a quadratic amount of time in a WAN, where no Ethernet broadcast is available and the message authentication codes (MACs) optimisation of PBFT can thus not be used. Instead, RBBC commits $\Omega(n)$ transactions as explained below.

Theorem 9 (Scalable throughput)**.** *Let w be the number of transactions proposed to the Set Byzantine Consensus by each proposer and let $n = |P|$ be the number of proposers. Assume that proposals are all reliably delivered before the algorithm times out (line 17) but that all $\lceil \frac{n}{3} \rceil - 1$ Byzantine proposers do not propose any transaction. The Set Byzantine Consensus commits between $2w$ (as n tends to infinity) and $\Omega(n)$ distinct transactions.*

Proof. Note that correct requesters must send their transaction to $f + 1$ proposers to guarantee that it will be committed. As a result, the same transaction could appear in the proposal of

different proposers.

- Consider the worst case scenario where all the duplicated transactions of all requesters are proposed to the same consensus instance. As a result there are at most $A = w \cdot (\lfloor \frac{2n}{3} \rfloor + 1)$ transactions and each transaction is duplicated $B = \lfloor \frac{n}{3} \rfloor + 1$ times. Thus A/B transactions are committed such that $\lim_{n \to \infty} A/B = 2w$.
- Consider the best case scenario where each requester transaction is sent to 1 correct proposer and the f Byzantine proposers. As the Byzantine proposers do not propose anything, there are no duplicates, and each of the $\lfloor \frac{2n}{3} \rfloor + 1$ correct proposers propose w distinct transactions, leading to $w \cdot \lfloor \frac{2n}{3} \rfloor + 1 = \Omega(n)$ transactions committed per consensus instance.

It follows that the minimum number of transactions that can be committed per consensus instance is $2w$ as n tends to infinity and the maximum number is $\Omega(n)$. □

Deduplication

If a Byzantine client can closely approximate the age parameter and the timing of the system, then more transactions can be proposed twice and their duplicates may persist until the reconciliation phase, after which one of them is discarded (line 28) as they necessarily conflict. One way to reduce the ability of Byzantine to create duplicates is for each node to choose the age parameter for each transaction at random as a small constant. Another is to apply this randomisation only for clients that are duplicating transactions. Note that some recent efforts [220] were devoted to eliminate duplicated transactions during the consensus execution by adopting a more conservative approach where each transaction can only be proposed by a single proposer at a time. If the proposer of this particular transaction fails, then an epoch change happens and another one is selected. Although the benefit is to eliminate duplicates during the consensus execution, the drawback is to increase linearly this transaction latency as up to f epoch changes may be necessary before proposing it correctly.

We show that RBBC is censorship-resistant where $2f < |P| \leq n$ in that every transaction sent by a correct requester to only $f + 1$ proposers is eventually committed.

Lemma 10. *Every transaction sent by a correct requester is eventually proposed by a correct proposer.*

Proof. It is known by (Section 4.4.1[❶]) that a correct requester sends its transaction to $f + 1$ proposers. It follows that at least one correct proposer receives this transaction. By the *bounded-requesting-rate* assumption (Section 4.2), it is known that this correct proposer will add it to its mempool and thus propose (line 14) it to the Set Byzantine Consensus. □

Theorem 11 (Censorship-Resistance)**.** *Every transaction sent by a correct requester is eventually included in a superblock.*

Proof. By Lemma 10, it is known that the transaction tx of a correct requester is proposed by a correct proposer, say in the k^{th} proposal. If, during the current consensus instance, this k^{th}

Table 4.2: Parameters used in Red Belly Experiments.

notation	meaning	value	motivation
η	compute primary proposer	$0 \leq id < n$	avoid conflicting transactions to different proposers
β	block size	$1 \leq \beta \leq 10,000$	adjust block size impact on network delay
P	set of proposers	$f+1 \leq \|P\| \leq n$	fewer (resp. more) proposers commits faster (resp. more)
σ	transaction signature	ECDSA	require smaller key and signatures than RSA for similar security

proposal is included in the superblock, then tx is committed immediately yielding the result. So let us consider that the k^{th} proposal is not included, this can happen if the corresponding binary consensus returned 0 because $bitmask[k] = 0$. As the proposer is correct, its proposal will be eventually rb-delivered by all $|P| - f$ correct proposers. By the *bounded-requesting-rate* assumption and because tx is not committed, these $|P| - f$ proposers record tx in their mempool with a birth date corresponding to their local clock.

At each multivalue consensus instance, one of the correct permissioned nodes proposes transactions that get included in the agreed upon superblock and thus committed. This is due to the pigeonhole principle applied to the *bitmask* of $|P|$ indices among which $|P| - f > f$ correspond to the identifiers of correct nodes and $|P| - f$ indices map to 1s (line 17). There is no guarantee that in a given multivalue consensus instance any correct proposer proposes tx, but as transactions are proposed in decreasing order of their age, it is known that it will be proposed among the $(|P| \cdot T + 1)$ next transactions were T is the maximum number of transactions in the mempool of each of the up to $|P|$ correct proposers when it inserts tx in its mempool. Due to the same pigeonhole argument as before and because of the sequential-transaction-requests assumption (Section 4.2), transaction tx will be committed. □

The Censorship-Resistance defined here differs from other notations of blockchain liveness, validity or resilience by offering guarantees to a correct *requester* [144, 224, 234] and without requiring its transaction to be sent to all replicas [77, 171] It is important to note that censorship-resistance does not require all requesters to be correct, it simply ensures that if a requester follows the protocol to submit a valid transaction, then the transaction is guaranteed to be committed by the system. In contrast, a Byzantine requester not following the request protocol does not have this guarantee.

4.6 Experimental evaluation I: Red Belly experiments

This section shows that RBBC scales up to hundreds of Amazon EC2 VMs running consensus located on different continents and replicating the blockchain state to up to 1000 machines in 14 separate regions. To this end, performance comparisons of (1) *RBBC* with its verification and its consensus as depicted in Section 4.4, (2) *HBBFT* which corresponds to the original code of the HoneyBadgerBFT protocol as made available by its authors [144] and (3) *CONS1* that corresponds to a variant of RBBC with the classic 3-step leader-based BFT algorithm of PBFT taken from [219] with BFT-SMaRt optimisations [15, 106, 241] are presented.

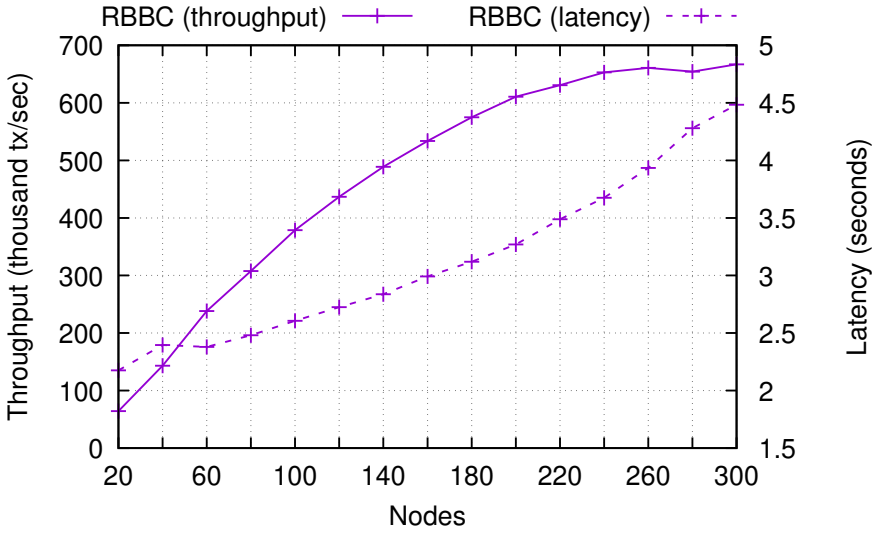


Figure 4.5: The performance (latency and throughput) of RBBC in a single datacenter.

Four types of experiments were run with parameters from Table 4.2: *(i)* with a varying fault tolerance and verification in a geodistributed environment (Section 4.6.1); *(ii)* with all three blockchains on low-end machines (Section 4.6.2); *(iii)* with up to 1000 replicas all updating their copy of all account balances (Section 4.6.3); and finally *(iv)* with Byzantine failures (Section 4.6.4).

Machine specifications

All blockchains were run on all the 14 Amazon datacenters that we had at our disposal at the time of the experiment: North Virginia, Ohio, North California, Oregon, Canada, Ireland, Frankfurt, London, Tokyo, Seoul, Singapore, Sydney, Mumbai, São Paulo. Each pair of datacenters is separated by a specific delay and bandwidth detailed in Section 4.6.3 with a visual representation detailed in Appendix C.

Three VM types were tested: (1) *high-end* c4.8xlarge instances with an Intel Xeon E5-2666 v3 processor of 18 hyperthreaded cores, 60 GiB RAM and 10 Gbps network performance when run in the same datacenter where storage is backed by Amazon's Elastic Block Store (EBS) with 4 Gbps dedicated throughput; (2) *mid-end* c4.4xlarge instances with an Intel Xeon E5-2666 v3 processors with 16 vCPUs and 30 GiB RAM with "high" network performance (as defined by Amazon), 2 Gbps EBS dedicated throughput; (3) *low-end* c4.xlarge instances with an Intel Xeon E5-2666 v3 processor of 4 vCPUs, 7.5 GiB RAM and "moderate" network performance, and 750 Mbps EBS dedicated throughput. To limit the bottleneck effect on the leader of PBFT, the leader was always placed in the most central (w.r.t. latency) region, Oregon. When not specified, proposals contain 10,000 transactions and f is set to $\lceil \frac{n}{3} \rceil - 1$.

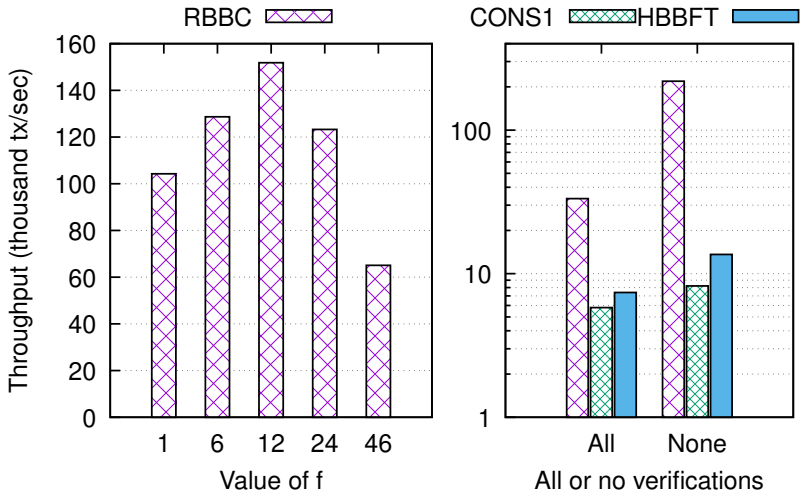


Figure 4.6: Impact of fault tolerance on RBBC and verification on 3 blockchains with $n = 140$ geodistributed machines.

Leader-based (CONS1) and randomised BFT (HBBFT)

CONS1 is the classic 3-step leader-based Byzantine consensus implementation similar to PBFT [15], the Tendermint consensus [106], and including the concurrency optimisations of BFT-SMaRt [241]. To reduce network consumption, CONS1 is implemented using digests in messages that follow the initial broadcast. Both CONS1 and HBBFT variants use a classic verification, as in traditional blockchain systems [9, 27], that takes place at every proposer upon delivery of the decided superblock from consensus. HBBFT uses a common coin [69] and reliable broadcast with erasure codes. The interest in comparison against HBBFT lies in the commonality of the use of reduction from multivalue consensus. Although HBBFT provides differing assumptions, the randomization was directly compared against by implementing RBBCs consensus in the same language. The results indicate that the randomized algorithm of HBBFT performs slower due to the coin tossing for randomization.

4.6.1 Peak scalability and leaderless fault tolerance

In a leaderless case, the first messages from $n - f$ replicas determine the performance, so a lower f can yield lower performance. RBBC is stress-tested in a single datacenter with up to 300 high-end VMs and a fixed fault tolerance to measure how fast it could go in a consortium setting. To this end, f is fixed to the largest possible fault tolerance with $n = 20$ nodes and increase the number of nodes from 20 to 300 permissioned nodes in steps of 20. The results, shown in Figure 4.5, indicate that the throughput scales to hundred of nodes with a practical latency: the throughput scales up to $n = 260$ nodes to reach 660,000 TPS while the latency remains lower than 4 seconds. At $n = 280$ throughput drops slightly. The impact of varying f on performance is discussed below.

Impact of fault tolerance without a leader

The performance was evaluated when running 10 high-end VMs in each of the 14 regions for a total of 140 machines. We varied f from the minimum to its maximum value ($46 < \frac{140}{3}$) with sharded verification as depicted in Figure 4.6 (left).

The peak throughput of 151,000 TPS is achieved with the fault-tolerance parameter $f = 12$. When $f \leq 6$, performance is limited by the $(f-1)^{th}$ slowest node as the consensus waits for a higher number of $n - f$ proposers. The peak throughput occurs while waiting for $n - f = 128$ nodes, probably because it avoids waiting for any node of the slowest region (Table C.1), São Paulo. When $f \geq 24$, performance tends to be limited and drops further as the fault tolerance f keeps increasing. We conjecture (and show below) that the $f + 1$ necessary cryptographic verifications induce a higher computational load as f increases. As mentioned in Section 4.1, alternative leaderless Byzantine consensus algorithms lack details [247] or have exponential complexity, which would be impractical [243].

Impact of verification sharding

To verify the conjecture that more verifications slow performance down, additional experiments were run, and the performance was measured with different numbers of verifications per transaction. As depicted in Figure 4.6 (right), all three blockchains were compared with all nodes verifying all transactions (all) and without any verification (no verification). The performance of all blockchains is higher without verification than with full verification. RBBC is the most affected, dropping from 219,000 TPS to 33,000 TPS while HBBFT and CONS1 throughputs drop less but from a lower peak. As will be shown, there are factors other than verification, like the use of a leader and erasure codes Section 4.6.2, that have a larger impact on these algorithms, yet this confirms that the previous conjecture was correct.

4.6.2 Scaling throughput up to hundreds of low-end machines

Experiments are now performed on up to 240 low-end VMs evenly spread on 5 datacenters in Europe (Ireland and Frankfurt) and the United States (Oregon, Northern California, and Ohio). Following up on the previous verification observations, the CPU usage was precisely measured with go `pprof` on microbenchmarks and confirmed that the workload could be CPU-bound. In particular, dedicating the 4 vCPUs of these low-end instances led to verify about 7800 serialised transactions per second with 97% of CPU time spent verifying signatures and 3% spent deserialising and updating the UTXO table.

RBBC vs. a leader-based BFT blockchain

Figure 4.7 shows the throughput and latency of RBBC with $f + 1$ proposers and CONS1 with different sizes of proposals. As CONS1 is limited to a single proposer (its leader) while RBBC supports multiple proposers, we tested whether CONS1 performance would be better

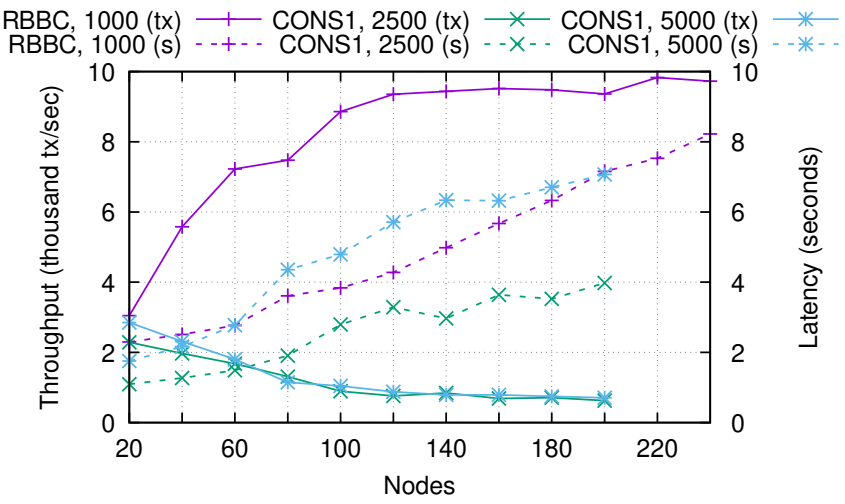


Figure 4.7: The performance of CONS1 with batching and RBBC with $f+1$ proposer nodes; the number following the algorithm name represents the number of transactions in the proposals; solid lines represent throughput, dashed lines represent latency.

with batching more transactions per proposal than RBBC. With proposal size of 1000, RBBC throughput increases from 3000 TPS to 9000 TPS because of the additional resources and proposals of the growing number of nodes. It flattens out around 10,000 TPS while latency increases from 2 to 8 seconds. By contrast, CONS1 throughput decreases as the number of nodes increases, despite larger proposals.

Latency vs. throughput at 100 nodes

To better understand the difference in performance of RBBC compared to the leader-based approach, Figure 4.8 depicts the evolution of the latency as a function of throughput at a reasonable number of consensus nodes, $n = 100$ and different proposal sizes of 1 to 5000. One can clearly see that the throughput of CONS1 reaches a limit of about 1100 TPS while RBBC approaches 14,000 TPS, which indicates a 12-fold speedup of RBBC over CONS1. CONS1 has a better minimum latency of 270 ms compared to 640 ms for RBBC for proposals of size 1 but CONS1 latency explodes rapidly (Note that HBBFT does not appear due to lower performance).

RBBC vs. a randomized BFT blockchain

Figure 4.9 depicts the performance of RBBC and HBBFT with n proposers, with proposal sizes of 100 and 1000 transactions. With a larger portion of proposers the throughput and latency of RBBC increases faster. With n proposers, the throughput peaks at 11,124 TPS and latency reaches 25,100 ms with 240 nodes, while with $f + 1$ proposers the throughput was lower and the latency was much lower. HBBFT performance degrades as the number of nodes increases: latency increases and throughput decreases (we omit latencies beyond $n = 100$ as they reach minutes). This is because each node broadcasts $n - 1$ erasure coded messages with as many distinct signatures to distinct nodes that must be echoed, yielding $\Omega(n^2)$ verifications per node.

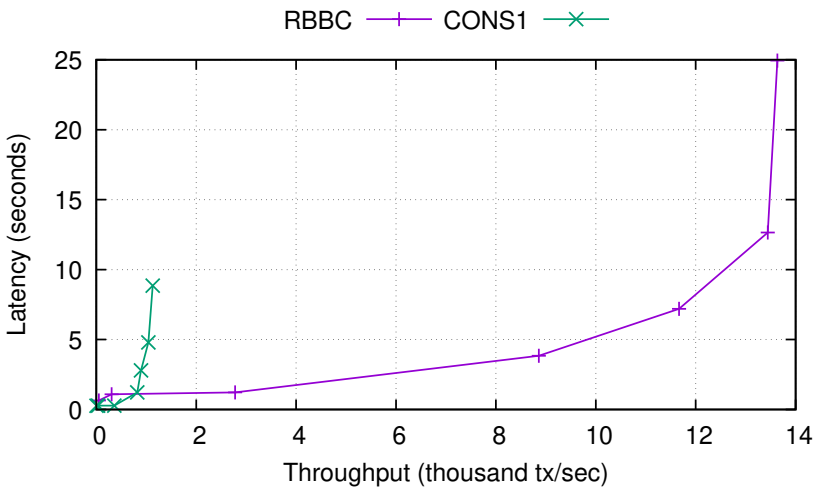


Figure 4.8: Comparing throughput and latency of CONS1 and RBBC with $f+1$ proposer nodes on 100 geodistributed nodes; each point represents the number of transactions in the proposals, either 10, 100, 1000, 2500, 5000 or 10000.

Reducing verification count helps scaling

As is known, our verification sharding reduces the number of nodes that verify each transaction from n in classic blockchains to between $f+1$ and $2f+1$ the exact number is ignored. To evaluate accurately the number of nodes verifying each transaction, the average number of times a transaction was verified for $f+1$ and n proposer nodes was also recorded. The results are shown in Figure 4.10 where it is observed that with $f+1$ proposers the number of verifications stays close to the optimal, while with n proposers the number of verifications remains around the middle of $f+1$ and $2f+1$. This is likely due to the increased load on the system causing verifications to occur in different orders at different nodes. This tends to confirm that verification sharding is important for scalability.

Comparing the blockchains

Figure 4.11 explored the effect of deciding the unions of proposals when running the blockchain. CONS1 has the lowest latency because in all executions the leader acts correctly, allowing it to terminate in only 3 message delays, where RBBC requires 4 message delays. Due to its inherent concurrency, RBBC offers the best latency/throughput tradeoff: at 1000 ms latency, RBBC offers 12,100 TPS whereas at 1750 ms latency, CONS1 offers only 5800 TPS. Note that CONS1 fast path evaluated here is known to come with a notoriously costly recovery path in case of failures [242, 243, 244, 149] was not experimented here. By contrast, RBBC does not suffer this slow path because it does not need a correct leader and its latency remains similar, even when faults occur (cf. Section 4.6.4). Finally, HBBFT has the worst performance because its consensus [69] is randomized and it uses erasure codes: each node spends over 200 ms to compute 1000 transactions for each of the 140 proposals of this experiment. Again this confirms

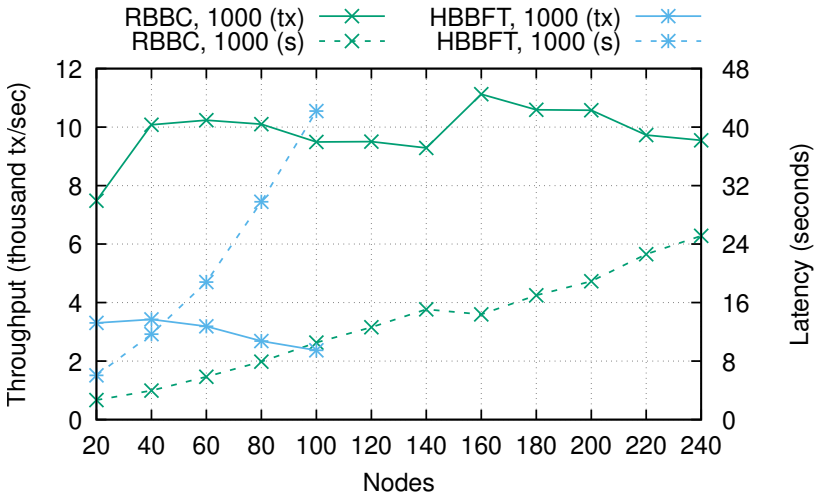


Figure 4.9: The performance of HBBFT and RBBC with n proposer nodes. The number following the algorithm name represents the number of transactions in the proposals; solid lines represent throughput, dashed lines represent latency.

Table 4.3: Performance of RBBC with 1000 replicas spread in 14 datacenters.

#Replicas	#Requesters	Valid-tx/sec	Async write latency(ms)	Latency(ms)	Valid-tx/superblock	Invalid-tx/superblock
1000	8400	30684	238	3103	95407	378

the important CPU load induced by the signature verifications.

4.6.3 Evaluation with 1000 VMs

Before spawning 1000 VMs to confirm RBBC scalability, the variation of latencies and bandwidth between the 14 Amazon EC2 datacenters was measured (cf. Table C.1). The minimum latency is 11 ms between London and Ireland, whereas the maximum latency is 332 ms observed between Sydney and São Paulo. Bandwidth between Ohio and Singapore is measured at approximately 64.9 Mbits/s (with variance between 6.5 Mbits/s and 20.4 Mbits/s).

To avoid wasting bandwidth, the roles were segregated: all 1000 VMs act as servers, keeping a local copy of the balances of all accounts. On these replicas, 10 clients per 840 low-end machines (60 VMs in each of 14 datacenters) send transactions and 160 high-end machines (40 machines in each of the Ireland, London, Ohio and Oregon datacenters) decide upon each superblock. Each of the 8,400 clients start with 100 UTXOs of size 64 bytes each (for a state database of size 51.27 MiB) and each proposal contains up to 1000 transactions. The resulting performance is depicted in Table 4.3. Interestingly, the throughput is only around 30,000 TPS but this is not due to the low capacity of RBBC, but due to the difficulty of generating the workload. The replicas are located in 14 different datacenters and have to wait for owning a UTXO before they can request a transaction that consumes it. The asynchronous write latency measures the time a proposer acknowledges a transaction reception. Importantly, the

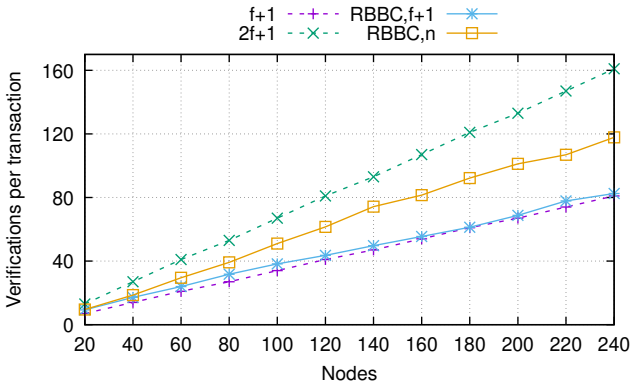
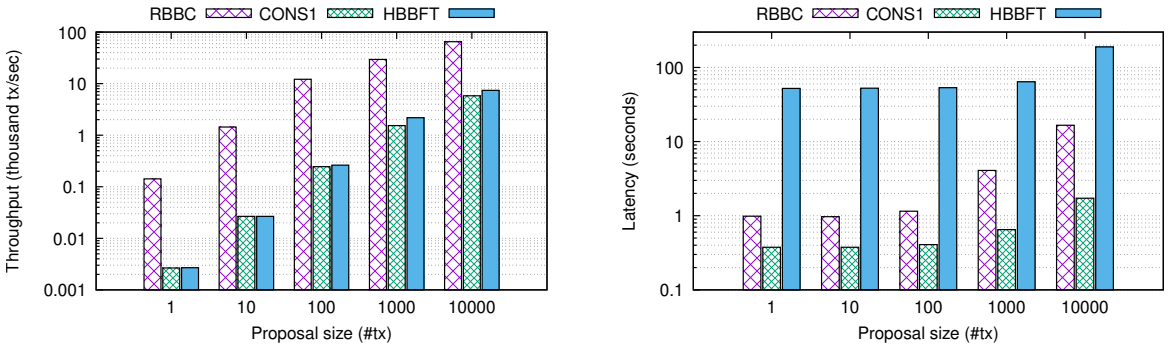


Figure 4.10: The number of times a transaction is verified in RBBC with proposal size of 100 transactions, with either $f+1$ or n proposer nodes; the dashed lines $f+1$ and $2f+1$ represent the minimum and maximum number of possible verifications.



(a) Throughput Comparison

(b) Latency Comparison

Figure 4.11: Throughput and latency comparison of the blockchain solutions with $n = 140$ and $f = 46$, and proposal sizes of $1, 10, 100, 1000$ and 10000 transactions.

transaction commit time (latency) remains about 3 seconds despite the large traffic.

4.6.4 Experiments under Byzantine attacks

RBBC is evaluated under 2 Byzantine attacks:

1) **Byz1** — The payload of the reliable broadcast message is altered such that no proposal is delivered for reliable broadcast instance led by faulty proposers. To this end, the payloads of the binary consensus are flipped. The goal of this behaviour is to reduce throughput and increase latency.

2) **Byz2** — The Byzantine proposers form a coalition in order to maximise the bandwidth cost of the reliable broadcast using the digests described in Section 4.4.2. As a result, for any reliable broadcast initiated by a Byzantine proposer, $f+1$ correct proposers will deliver the full message while the remaining f will only deliver the digest, meaning they

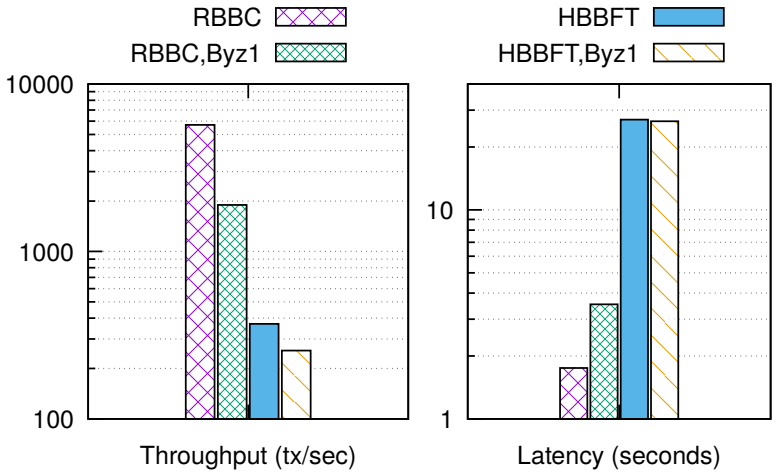


Figure 4.12: Comparing throughput and latency of RBBC and HBBFT, with normal and Byzantine behaviours on 100 geodistributed nodes; all n nodes are making proposals of 100 transactions.

will have to request the full message from $f + 1$ different proposers when receiving the ECHO messages.

Experiments are run with 100 low-end machines using the same 5 datacenters from US and Europe and with n proposers (as in Section 4.6.2). Figure 4.12 shows the impact of Byz1 on performance with n proposers and proposal sizes of 100. RBBC throughput drops from 5700 TPS to 1900 TPS due to having f less proposals being accepted (the proposals sent by Byzantine proposers are invalid) and to the increase in latency. The latency increases due to the extra rounds needed to be executed by the binary consensus to terminate with 0. The throughput of HBBFT drops from 350 to 256 TPS due to fewer proposals but the latency decreases because with less proposals erasure codes require less computation.

Byz2 is designed against the verified reliable broadcast of Section 4.4.2, to delay the delivery of the message to f of the correct proposers, and increasing the bandwidth used. HBBFT avoids this problem by using erasure codes, but has a higher bandwidth usage in the correct case. Figure 4.13 shows its impact on bandwidth usage and latency for RBBC and HBBFT with n proposers and proposal sizes of 100. The bandwidth usage of RBBC increases from 538 MB to 2622 MB per multivalued consensus instance compared to HBBFT, which uses 3600 MB in all cases. Furthermore, the latency of RBBC increases from 920 ms to 2300 ms.

Regarding corruptions, as it is impossible to solve consensus when $f \geq n/3$ [14], one must either replace the permissioned nodes [160] before $n/3$ collude, or implement an eventually consistent alternative of the service [255].

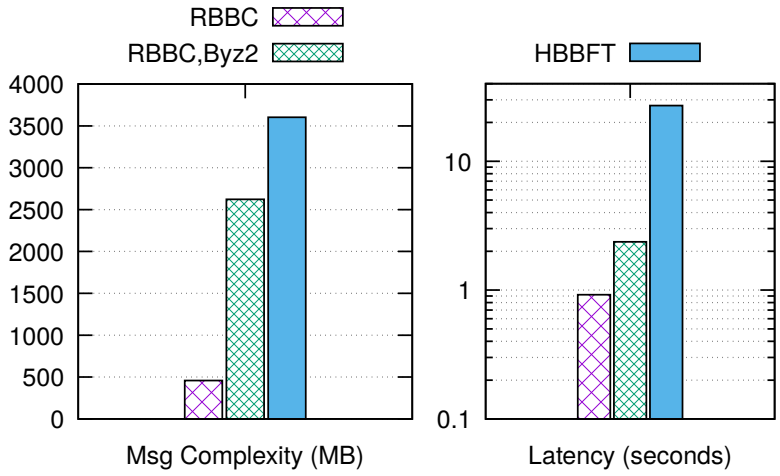


Figure 4.13: Comparing bandwidth usage and latency of RBBFT and HBBFT with normal and Byzantine behaviours on 100 geodistributed nodes.

4.6.5 Impact of remote requesters

To evaluate the impact of remote requesters, experiments were performed with RBBC and CONS1 using 125 mid-end instances: 100 server and 25 requester instances both evenly distributed across 5 datacenters in US and Europe (as depicted in Section 4.6.2).

The number of requesters is varied from 1,000 to 50,000. Each requester is initially assigned a random private key and a single UTXO with 100,000 coins written in the Genesis block. The requester then loops over the following two steps until the benchmark completes: *(i)* For each UTXO currently assigned to the requester a new transaction is created using that UTXO as input. For the transaction's output a UTXO is created using a randomly chosen account as the receiver with a value of 10 coins. Any change is included in a second UTXO sent back to the requester. Each transaction is then broadcast to the requester assigned proposers. *(ii)* The requester then repeatedly performs the request_utxos(*account*) operation until it receives at least one new UTXO and then returns to step (i). $\beta = 1000$ is selected for RBBC and the proposal size of 2500 for CONS1 as it noticeably maximises throughput. The experiments were run for 45 sec with a 15 sec warmup.

Table 4.4 depicts the average number of valid transactions committed per second (Valid-TPS), the average number of request_utxos(*account*) operations performed per second (Read-/sec), the ratio of the previous two values (R/W ratio), the average amount of time between committed blocks (Latency), the average number of valid transactions per block (Valid-tx/block) and the average number of invalid transactions per block (Invalid-tx/block). It is observed that the RBBC maximum throughput (14,450) is more than 3-fold the maximum throughput of CONS1 (4,064). RBBC also has the highest maximum latency (5,022) compared to CONS1 (625). This is due to RBBC better utilising resources and its reduced computation. Increasing

Table 4.4: Performance of RBBC and CONS1 with varying number of requesters.

	#Requesters	Valid-TPS	Read/sec	R/W ratio	Latency(ms)	Valid-tx/block	Invalid-tx/block
RBBC	1,000	5,359	2,143	0.4	870	4,648	0
	10,000	13,870	33,288	2.4	2,475	34,132	877
	20,000	12,664	31,660	2.5	5,022	63,607	3,033
	50,000	14,450	47,685	3.3	4,303	62,193	5,455
CONS1	1,000	3,759	1,127	0.3	401	1,513	0
	10,000	3,309	6,278	1.9	359	1,172	0
	20,000	4,064	10,566	2.6	488	1,981	0
	50,000	4,035	12,509	3.1	625	2,500	0

the number of clients beyond 1,000 (resp. 10,000) has little impact on CONS1 (resp. RBBC) throughput. This difference is due to CONS1 being limited by the single primary proposer. Furthermore in RBBC, increasing the number of clients increases the number of duplicate transactions occurring in blocks, due to transactions being committed by secondary proposers.

4.7 Experimental evaluation II: Caliper experiments

Further experimental analysis of the Red Belly Blockchain is presented by integrating into the Hyperledger Caliper [47] benchmark framework. It is compared against two other BFT-based blockchain systems, namely Hyperledger Burrow [169], a blockchain that comprises of Tendermint's consensus engine [106] plugged into Ethereum [27], and the BFT variant of Quorum [55] using IBFT [150, 152], a blockchain built upon foundations of Ethereum with a BFT consensus engine. RBBC performance is shown to increase with the workload and number of nodes. The results indicate that RBBC can handle a sending rate of almost one order of magnitude higher than the Quorum IBFT blockchain.

Experimental Settings The experiments conducted on AWS EC2 infrastructure, where both Caliper and blockchain nodes used c4.xLarge servers with an Intel Xeon E5-2666 v3 processor of 4 vCPUs at 2.9GHz, 7.5 GiB memory and "high" network bandwidth (as listed by AWS). All experiments were performed within a single AWS region, testing (i) a single Caliper instance performing requests to the blockchain network, or, (ii) 4 Caliper instances that perform requests and the results were aggregated. Throughput was measured as the number of transactions committed per second, where each transaction was successfully agreed into a superblock. Latency was measured as the time from the requester submitting the transaction to the time the transaction was updated and processed as part of the superblock.

4.7.1 Red Belly performance increases with workload

RBBC was evaluated with a single caliper client and 4 RBBC nodes to emulate a small private network. The sending rate of requests began at 50 transactions per second and increased to $2,000$ transactions per second, where no failed transactions were observed. Each experiment was repeated 5 times, with the average and deviation computed for each experiment batch.

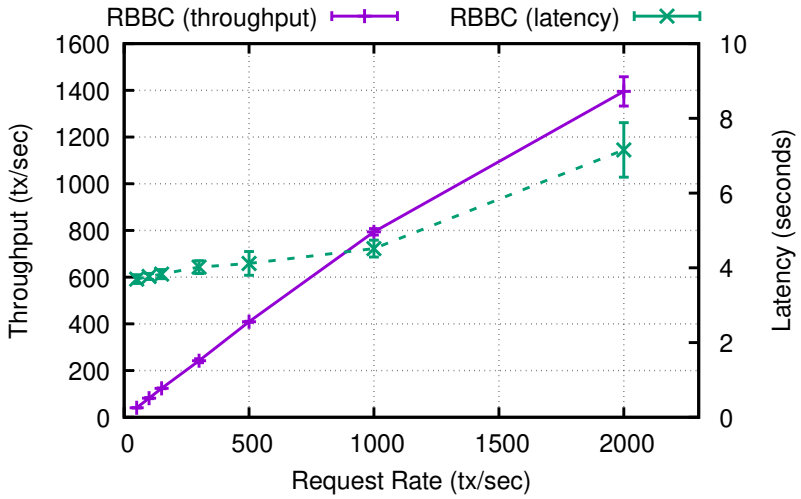


Figure 4.14: One Caliper Node with 4 RBBC nodes, increasing request rates.

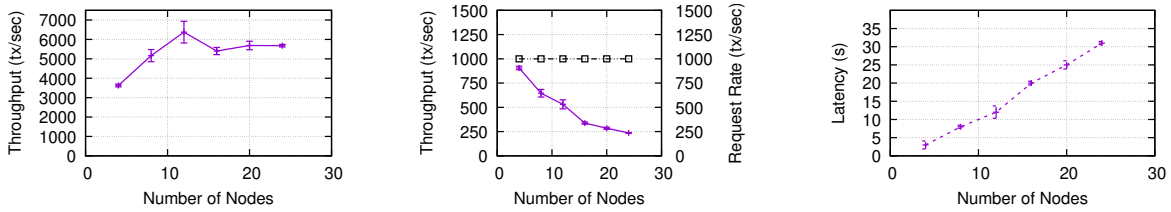
As depicted in Figure 4.14, RBBC achieved a maximum throughput of 1,395 TPS and a corresponding average latency of 7.2s when the request rate was 2,000 transactions per second. The fact that no transactions failed confirms that RBBC provides starvation freedom, in that every correctly submitted transaction gets eventually committed even when the system is under extreme load.

4.7.2 Comparing blockchain performance

At the time of implementation, Caliper had minimal support for distributed worker machines. To address this, scripts were utilised to configure and launch multiple Caliper instances across several machines to orchestrate a distributed benchmark in AWS. The script used in the experiment launches the specified number of blockchain nodes and caliper machines in AWS. It then configures Caliper with network parameters and then simultaneously launches the benchmark on multiple machines.

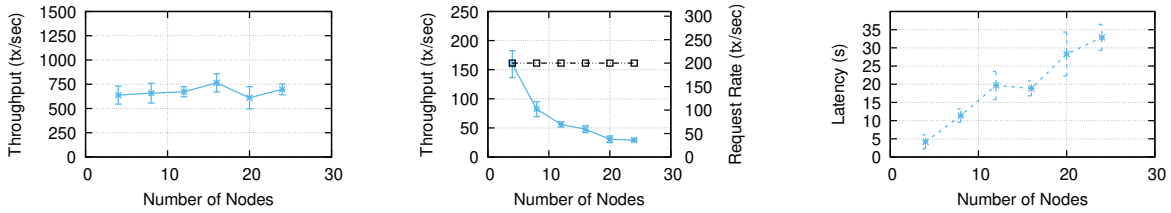
Per-blockchain ideal workload

Each unique system was able to sustain peak throughput at varying sending rates. The optimal sending rate for each blockchain was measured by varying sending rates and duration to calculate the most optimal combination by comparing peak throughput. Figure 4.14 highlights the relationship between sending rate and performance for the RBBC with a configuration of 4 consensus nodes and one caliper participant. However, the throughput may be limited by the capacity of (1) the benchmark client's sending and receiving rate, or (2) the connected blockchain node's capacity of receiving and processing transactions. This may not be indicative of the entire blockchain, hence the requirement for multiple benchmark clients.



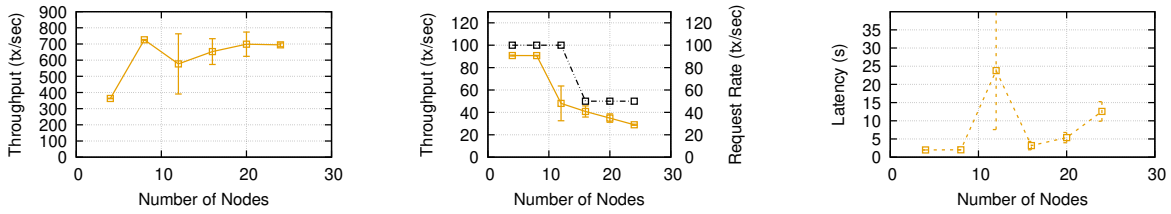
(a) Cumulative Throughput (b) Average Throughput Per Caliper Client (c) Latency

Figure 4.15: RBBC — Send Rate = 1000TPS per Caliper Machine.



(a) Cumulative Throughput (b) Average Throughput Per Caliper Client (c) Latency

Figure 4.16: Burrow — Send Rate = 200TPS per Caliper Machine.



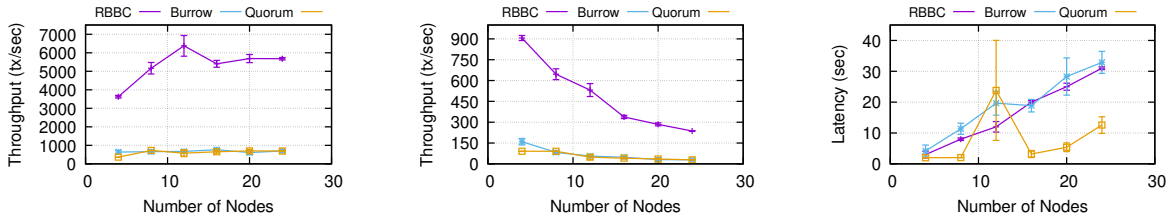
(a) Cumulative Throughput (b) Average Throughput Per Caliper Client (c) Latency

Figure 4.17: Quorum — Send Rate = 50–100TPS per Caliper Machine.

Performance variations

Figure 4.15, Figure 4.16 and Figure 4.17 show the overall results from experiments measured for each blockchain. The total throughput is a cumulative result aggregated for each Caliper client machine, showing the peak obtained throughput as a total measure. The average throughput per Caliper client is shown, with the sending rate attached on the secondary axis.

Figure 4.15 shows results achieved for RBBC with a sending rate of $1,000$ TPS. The throughput in Figure 4.15a increases as the number of nodes increases, but eventually plateaus to a constant throughput. The peak throughput of $6,375$ TPS is achieved with 12 RBBC nodes and a corresponding latency of 12s. Although the total throughput increases, the average throughput per node decreases from 906TPS to 237TPS as the number of nodes increases, as shown in



(a) Cumulative Throughput (b) Average Per Caliper Node (c) Latency

Figure 4.18: Comparison of RBBC, Burrow and Quorum.

Figure 4.15b.

Figure 4.16 shows results for Burrow with a sending rate of 200 TPS. Figure 4.16a shows the throughput remaining relatively constant with increasing node numbers. The peak throughput was achieved with 16 nodes at 765 TPS with a corresponding latency of 18.9s. Figure 4.16b reveals that the average throughput per node decreases to 29 TPS with the increasing number of nodes.

Figure 4.17 shows measurements for Quorum with an initial sending rate of 100 TPS per Caliper client. The initial send rate of 100 TPS per Caliper client was only valid for 4, 8 and 12 nodes, but was reduced to 50TPS as the blockchain system failed when the send rate was higher than this for more than 12 nodes, as shown in Figure 4.17b. As seen in Figure 4.17c the latency peaked to 24s with 12 nodes. The throughput peaked to 725 TPS at 8 nodes before dipping to 576 TPS and returning to 694 TPS.

Figure 4.18 displays a comparison between all three BFT blockchains side by side as evaluated through the Caliper framework. Figure 4.18a shows that the Red Belly Blockchain achieves a peak total throughput of $6,375$ TPS and an average throughput of 906 TPS per Caliper client. This is one order of magnitude higher than both Quorum and Burrow. The latency, as displayed in Figure 4.18c shows that the latency of RBBC and Burrow are relatively similar, whereas Quorum is lower.

4.8 Summary

In this chapter, the Red Belly Blockchain is presented, a non-forking, high performance blockchain capable of scaling to hundreds of distributed consensus participants without requiring synchrony. Its non-forking design and assumptions of partial-synchrony provides security against double-spending attacks by favouring consistency over availability, but is able to tolerate unknown message delays.

The foundational consensus at the heart of the Red Belly Blockchain, DBFT, solves the Set Byzantine Consensus problem, by unioning all proposed transaction sets into a superblock for

each index. This provides guarantees of censorship resistance for correct proposals to be eventually placed into the chain. Complementary to this, RBBC favours older transactions to achieve a level of fairness and prevent from censorship. The agreement of a committed superblock allows multiple proposals to be unified into a single result, where the number of proposals will grow with the number of participants, leading to scalability in the number of transactions committed per consensus completion.

Secondly, the verification sharding reduces overall computation. By distributing verification to primary and secondary verifiers, it reduces the waste of effort in re-verifying transactions by requiring $f + 1$ to $2f + 1$ verifiers to confirm a transaction's cryptographic validity.

Finally, theoretically and experimentally is is shown that the RBBC provides scalability. Experimental evaluation was performed through geodistributed experiments tailored to test consensus implementations as well as using a publicly available blockchain benchmark to evaluate the performance of the blockchain. The Red Belly Blockchain is able to scale to hundreds of consensus nodes and achieve $660,000$ TPS committed in a single datacenter.

Chapter 5

DIABLO: A Distributed Analytical Blockchain Benchmark for performance measurement

Our addition of the Red Belly (Chapter 4) to the growing plethora of available blockchains motivates the requirements for accurate performance measurement and fair comparisons between systems. Additionally, the success of blockchains has led to a rise of Decentralised Applications (DApps) which present unique request behaviours and environments. This raises the question of determining the most suitable blockchain for a given application, which requires points of reference, such as performance metrics, that can be used for accurate comparison. However, performance evaluations presented with the respective blockchains are frequently released through social channels attached to whitepapers or blog posts that are used as a medium for marketing [43, 44, 45, 46]. Commonly, these performance evaluations are obtained through carefully tailored environments using unspecified experimental settings, alongside presenting comparisons against blockchains of different models and assumptions that skew the interpretations to positively influence the presented blockchain. These are often paired with minimal implementation details, abstracting critical information that must be retrieved through the source code or further analysis, adding to the convolutions of comparisons shrouded by unknowns.

The experimental evaluation of the Red Belly Blockchain using Caliper (Section 4.7) prompted an investigation of pre-existing benchmark platforms that had the potential to provide information surrounding application workloads. Not only did this assist in further understanding intricacies of the RBBC implementation, it highlighted fundamental aspects hindering current benchmark frameworks. DIABLO, the *DIstributed Analytical BLOckchain* benchmark framework, is developed to offer a solution to shortcomings hindering the current benchmark frameworks. The core focus of DIABLO is simplicity, presenting a modular design that promotes integration of future blockchains and provides ease in defining new application and use case workloads.

Table 5.1: Blockchains Evaluated with DIABLO.

Chain	Consensus	Tolerance
Go-Ethereum (geth)	PoA Clique	Byzantine
OpenEthereum	PoA AuRA	Byzantine
Collachain	DBFT	Byzantine
Quorum	IBFT	Byzantine
Quorum	RAFT	Crash
HL Fabric	RAFT	Crash

Table 5.2: Decentralised applications evaluated with DIABLO.

App	Source	Feature
Microblogging	Twitter	Load Burst
Exchange	NASDAQ	Herd Behaviour
Supply Chain	Aviation	Demand Peaks
Constant	Synthetic	Tunable

To evaluate the effectiveness of DIABLO alongside measuring the performance of the design principles of the Red Belly Blockchain, we integrate support for a number of blockchains, shown in Table 5.1, and performed evaluations over 4 workloads as listed in Table 5.2.

In summary, we present the following contributions in this chapter;

(i) We provide insightful evaluation of current blockchain benchmarks, highlighting three major shortcomings that present difficulties when evaluating high performance systems.

(ii) We present DIABLO, a distributed blockchain benchmark framework building upon our findings and is designed with the foundations of distributed workloads and simplicity.

(iii) We analyse 6 blockchains with varying use-case scenarios, highlighting differences in their performance and showing empirically that tolerating Byzantine faults is more costly than exclusively tolerating crash faults.

Chapter Outline The remainder of the chapter is organised as follows. Section 5.1 presents our motivation by identifying shortcomings in available frameworks. Section 5.2 introduces DIABLO, highlighting the design and architecture that make it modular and extensible. Section 5.3 details the DIABLO configurations, the unique aspects that make dynamic workloads and blockchains easy to define. Section 5.4 presents experimental analysis performed with DIABLO. Section 5.5 and Section 5.6 reveal our observations during experimental evaluation. Section 5.7 presents a case study experiment of Aviation parts. Section 5.8 concludes and presents a summary of the chapter.

5.1 Drawbacks of current frameworks

There are a numerous solutions for benchmark frameworks, simulations and external measurements used for evaluating the performance of blockchains [47, 48, 50, 175] or underlying components such as BFT [49, 174, 173] or Virtual Machines [62], as discussed in Chapter 2. However, various drawbacks present difficulties when evaluating high performance blockchains.

5.1.1 The need for comparative blockchain benchmark frameworks

As of writing, in June 2021, there are over approximately 1090 unique blockchain systems available[1], but minimal information presented on their benefits in terms of performance, such as throughput, latency or scalability. Unfortunately, when this information is available [43, 44, 45, 46], it often results from tests run under isolated, tailored experimental conditions and are accompanied by information presented through whitepapers, which only provide a bird's eye view of implementation details.

This often requires researchers to dive into source code and implementation details of a blockchain to gain more knowledge of the intricacies of their implementations. Alternatively, researchers can approach evaluations of blockchains as black boxes, generating workloads that measure performance as an external tool. This result leads to benchmarks that have often focused on specified environments, such as Internet of Things (IoT) [263].

5.1.2 Shortcomings in current approaches

Interestingly, during the evaluation of high performance blockchains (Chapter 4.7) and assessment of available benchmarks (Chapter 2.3), three major drawbacks were identified that have plagued evaluations. Although we developed temporary resolutions during the previous experimental analysis, we failed to identify concrete solutions that would be easily-integrated to the design without heavy refactoring.

(1) Lack of distributed load generation

Blockchain benchmark frameworks have been designed in a centralised fashion, where a single client machine, potentially running concurrent client software, submits requests to the blockchain [48, 50]. As blockchains are distributed systems aiming to scale to large numbers of participating blockchain nodes and interacting clients, it is crucial to generate a workload that is sufficiently high in order to stress test these numerous blockchain nodes. Without provisioning enough physical client resources, there is a risk that an experiment may report misleading performance results, being limited by client resources rather than the true capacity of the blockchain system under test.

(2) Misleading overheads

For simplicity in deployments, and additional monitoring features, several blockchain systems provide deployment options that give quick virtualised, or containerised, installations which may limit their access to physical resources. For example, Hyperledger Fabric [172, 171] endorses the use of Docker containers for ease of deployment and available tooling[2]. Similarly, Caliper [47] also endorses the use of Docker containers for quick and easy deployment and networking for its workers, but also easy integration with Hyperledger blockchains. However,

[1] Data taken from https://coinmarketcap.com/coins/views/all/, Accessed 2021-06-01

[2] Available online from: https://openblockchain.readthedocs.io/en/latest/Setup/Network-setup/

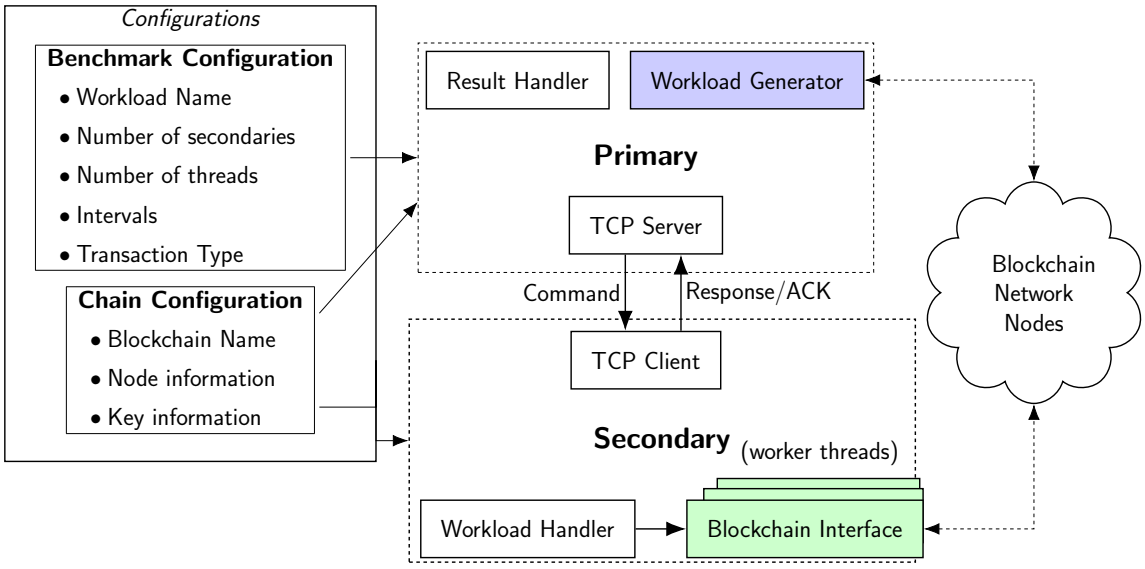


Figure 5.1: Overview of the Diablo architecture.

containers are known to induce unnecessary overheads [232]. Distributed synchronisation services, like Zookeeper [228], help with monitoring for systems and coordination, but these services are known to bottleneck at the leader network interface, hence misleading evaluation when the system reaches larger numbers. Thus, it is critical to limit the effects of these overheads to prevent associated slowdown from benchmark infrastructure, resulting in unjust conclusions of the evaluation.

(3) Cryptographic processing during timed periods

Blockchains require the use of cryptographic functions to guarantee authenticity of transactions, but were found to be particularly CPU-intensive in various occasions [7, 6]. Part of the CPU-intensive tasks are produced on the client side, during the creation and signing of transactions, where a benchmark must produce sufficiently many transactions to guarantee the intended workload. Some of the available benchmark frameworks perform these CPU-intensive tasks during the timed execution of their benchmark, which may bias the throughput measured and misrepresent the capacity of the blockchain. Instead, the computed measure would be limited by the benchmark client machine's throughput for signing and generating the transactions and measures of throughput would include the time taken to sign and form transactions.

5.2 Diablo design

With the growing popularity of Decentralised Applications (DApps), the focus is to provide accurate key measurements and comparisons of blockchains founded on their performance under various application-based workloads. To this end, Diablo, the *DIstributed Analytical BLOckchain* benchmark, was built with the foundations of distributed load generation to provide fair and accurate comparisons of blockchains under realistic, tunable workloads. Diablo provides

benchmark evaluations for blockchain deployments that mimic real-world application environments. Figure 5.1 presents an aerial view of the DIABLO design, highlighting the two core components, namely the *Primary* and the *Secondary* responsible for orchestrating and executing the benchmark. DIABLO improves upon the shortcomings identified in available benchmark frameworks, while offering simplicity in extensibility to add new blockchains or define new workloads.

5.2.1 Principles

To provide a usable benchmark framework capable of fair, accurate evaluations, alongside providing solutions to hindrances identified in other benchmark frameworks, the following design principles were adopted:

(i) **Distributed Load Generation.** Ensure that the benchmark framework is not bottlenecked by performance of the client machines. It should have multiple machines orchestrated to provide distributed load. To safeguard accurate measurement that represents realistic blockchain performance, requests are presented from a distributed set of machines, aiming for accuracy, but allowing variety in environmental setups.

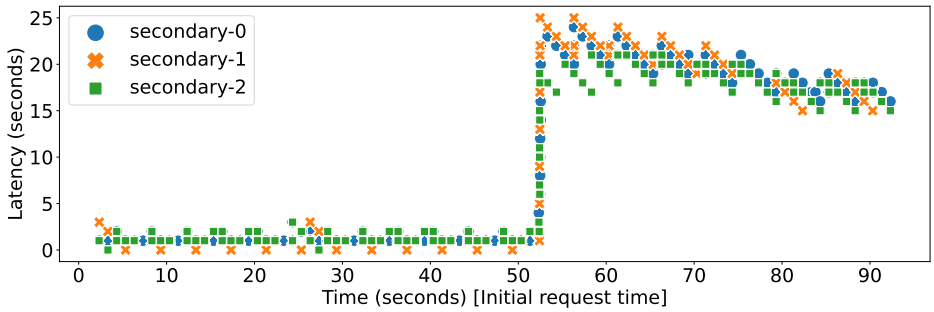
(ii) **Pregenerated Load.** Generate all information for the benchmark prior to interaction under a timed execution. To remove overheads of cryptographic processing during timed periods, all transaction signing and cryptographic processing should be done prior to the benchmark. A generation phase is presented such that all transactions and compute intensive tasks are performed before the timed execution has begun to limit any influence of machine capabilities on the calculations of throughput.

(iii) **Extensibility.** Provide ease of integration for future blockchain systems, enabling more input to fair comparison of systems. A simple, modular design is presented which allows new blockchains to be integrated and compared.

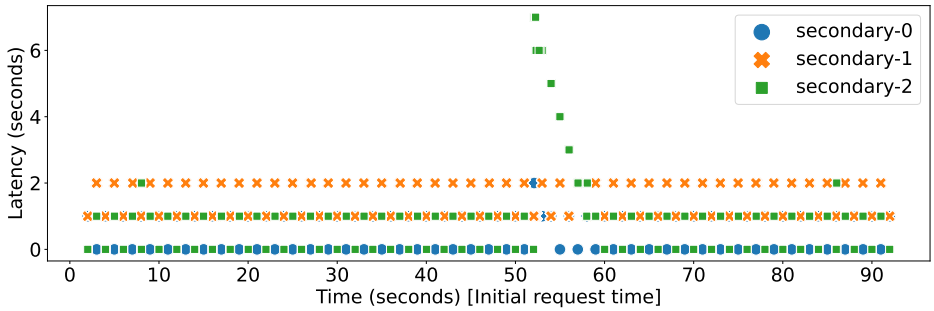
(iv) **Dynamic Load.** Generate variable request rates that reflect realistic behaviour patterns. Many blockchain benchmarks provide support for static request rates, or following a simple curve. The goal is to present load behaviour that mimics application request behaviour, so variability in the request over time and locality of load must be supported.

(v) **Easy Workload Definition.** Provide flexibility and ease of use to define new workloads, making it accessible for application designers and developers to accurately measure the suitability of different blockchains to their application.

(vi) **Simple Communication.** Limit requirements for misleading overheads, like containerisation or coordination services. Coordination is performed through simple message exchanges between the Primary and the Secondaries, removing any requirement for external tools.



(a) Geth latency per transaction grouped by secondary



(b) Collachain latency per transaction grouped by secondary

Figure 5.2: Diablo individual transaction latencies displayed over time for running the Twitter Experiment (Section 5.4). The x-axis displays the duration of the benchmark as a timeseries, corresponding to the initial request time of the transaction, and the y-axis denotes the latency in seconds. This data helps to identify slow-running machines, or, anomalous behaviour, such as Collachain having a slower secondary in this experiment run.

5.2.2 Diablo Primary

The Primary is the conductor machine of the benchmark, responsible for orchestrating timing and operations throughout the benchmark execution. Its main function is to generate and distribute workload data to the secondary machines, communicating by sending commands to start the benchmark and, upon completion, retrieve the results and aggregate the data. The Primary hosts the communication server to which all secondary machines connect, communicating over TCP by sending commands and retrieving responses. The responses indicate success or failure of the operation on the distributed set of secondary machines, while also providing user-friendly messages for identifying errors or debugging.

Workload generator

Each blockchain integration requires the implementation of a workload generator. The sole responsibility of the workload generator is to parse information from the configuration files, such as transaction metadata, total transactions, or functions to execute, generating workload data in a format usable by the blockchain system under test. The generation phase also provides an

initialisation step to deploy contract code on the chain if required. Alternatively, if the contract is already deployed, by specifying in the configuration, it will skip this step. Often this generator will interact directly with the blockchain system under test, requesting information that is used throughout the workload generation. Such information can be: chain identifying information, transaction parameters, account details, or any vital information that is required to generate and format the workload in a way that it can be submitted directly to the blockchain with minimal computational requirement during the benchmark.

The workload generator is specified per blockchain implementation, as it covers the unique aspects that form a request to the blockchain. The workload generator takes as input the configuration files, and will output workloads split for each thread and each secondary for them to execute.

Result handler

Once the DIABLO Secondary machines have signalled completion of the benchmark, the Primary will request to retrieve all the results for calculation and aggregation. At this point, all results are passed to the result handler on the Primary machine, which aggregates and formats output to usable information in a JSON format. The result handler not only provides a central location to collect, aggregate and perform calculations on the information, it also standardises the way information is represented across all blockchains, facilitating easy comparisons and visualisations. The obtained results not only indicate the performance of the overall system under consideration, but provide information from the perspective of each DIABLO secondary. Figure 5.2 presents an example of the granularity of data available, showing that each transaction request time can be seen individually and grouped by each thread or secondary. This fine-grained information about per-machine perspectives, such as transaction latency and throughput, is instrumental for identifying slow performing machines and their impact on the overall performance.

5.2.3 DIABLO Secondary

Secondaries are responsible for communicating directly with the blockchain under test using a *blockchain client interface*. Each secondary has the ability to have a number of worker threads running in parallel, allowing further concurrent sending of transactions from the source machine. Secondary machines individually connect to the DIABLO Primary, awaiting the appropriate commands and data to start the benchmark execution. Upon initialisation, each secondary connects to the primary and is allocated a unique identifier which not only distinguishes its results, but denotes which blockchain node it should connect to as the main blockchain node. Each of the Secondary's threads is a separate instance of the blockchain interface, connecting to the blockchain under test and interacting within its own work loop. It comprises of two major components, the *Workload Handler*, responsible for managing and directing the flow of the benchmark on each secondary, and the *Client Interface*, which is used to connect and interact with the blockchain.

```
type BlockchainInterface interface {
 // Provides the client with the list of all hosts
 Init(chainConfig *configs.ChainConfig)
 // Finishes up and performs any post-benchmark operations.
 Cleanup() results.Results
 // Start handles the starting aspects of the benchmark
 Start()
 // Handles the workload, converts the bytes transactions.
 // This parses the worker's workload into transactions
 ParseWorkload(workload workloadgenerators.WorkerThreadWorkload)
   ([][]interface{}, error)
 // Connects to one blockchain node as primary
 ConnectOne(id int) error
 // ConnectAll connects to all nodes given in the hosts
 ConnectAll(primaryID int) error
 // DeploySmartContract deploys the smart contract
 // Returns the address of the contract
 DeploySmartContract(tx interface{}) (interface{}, error)
 // SendRawTransactions sends the raw transaction bytes
 // It is safe to assume that these bytes will be formatted.
 SendRawTransaction(tx interface{}) error
 // Get Tx Done returns the number of transactions completed
 // This is already implemented with the GenericInterface
 GetTxDone() uint64
 // SetWindow sets the transaction window
 // Used for the throughput over time calculations.
 SetWindow(window int)
 // Close the connection to the blockchain node
 Close()
}
```

Figure 5.3: DIABLO client interface for generic blockchain usages.

Workload handler

The workload handler manages the distribution of workload information to worker threads running on the secondary machine and coordinates the execution of the benchmark on the machine. The handler is responsible for initialising worker threads, setting up interval schedules and operating the worker loops to perform the benchmark. After completion, the workload handler aggregates the results which will be sent to the DIABLO Primary upon request.

Client interface

For the sake of modularity, and similar to the *workload generator*, each blockchain integration requires the implementation of a generic blockchain interface, allowing DIABLO to support basic operations and interact with the blockchain system under test. The client interface is the key

```yaml
name: "sample benchmark config"
description: "A description"
secondaries: 2
threads: 1
bench:
  type: "simple"
  txs:
    0: 5
    3: 15
    10: 50
    15: 50
    16: 20
    25: 30
```

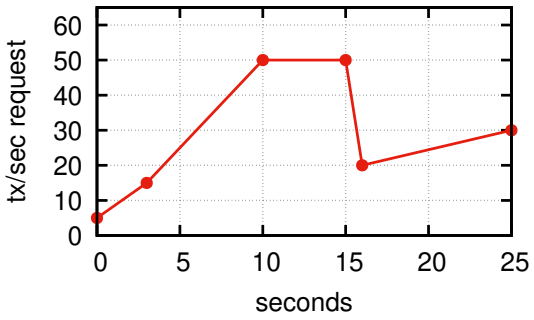


Figure 5.4: Sample benchmark configuration for simple transaction workload curve definition.

to modularity and ease of integration in Diablo. The interface is depicted in Figure 5.3. It integrates the vital functionality abstracted from the benchmark flow, making it possible to perform various actions, such as connecting to a node or sending transactions. The client interface specifies how the transaction lifestyle is executed, by submitting the request and waiting for the appropriate event to ensure the transaction has been committed. To provide a generic abstraction, but still gain critical metrics, the client interface will indicate when a transaction has been requested and committed, generating the results that will be passed to the workload handler and therefore forwarded to the Primary. The motivation behind this is the variety in blockchain transaction lifestyles and the definition of a committed transaction.

5.3 Diablo configurations

Core to the design of Diablo is the utilisation of configuration files to provide simplicity and flexibility when operating benchmarks. Defining a benchmark workload therefore consists of two fundamental aspects: (i) defining the workload pattern as request intensity and timing and (ii) defining the experimental information denoting the configuration of machines as access points to interact with the blockchain network. Thus, Diablo provides two configuration files to specify each of these characteristics, the *benchmark configuration* and the *chain configuration*, making it trivial to define what is being tested as well as trivial to understand result datasets.

5.3.1 Benchmark configuration

The benchmark configuration is fundamental in the operation of Diablo, defining the workload request pattern as well as all crucial aspects of how the benchmark is executed. It contains general information of the benchmark, such as name and description, as well as the number of Diablo secondaries and worker threads. The principle innovation lies in the definition of

the request pattern to emulate various application behaviours. To define a workload curve, the benchmark configuration has a timeseries representation of the transaction request intensity, showing the change in request intensity over time. As depicted in Figure 5.4, the transaction request curve is defined with per-second granularity, allowing a dynamic intensity of requests. This gives rise to importing real-world request curves obtained through timeseries datasets observed or captured in other systems, or, programmatically designed.

The benchmark configuration also details vital transaction information, containing the function calls, data types and values of what is passed into the function being invoked. The benchmark configuration currently supports three benchmark *types*:

1) **simple.** Denoting simple asset transfers, used to represent a coin transfer or a basic interaction on the chain for those chains who do not support scripts or smart contracts. It represents transactions transferring a "value" from one account to another. As shown in Figure 5.4, it requires minimal configuration.

2) **contract.** Indicates a contract-based workload that interacts with a smart contract with the functions provided. These workload types contain a number of functions that interact directly with the contract. The contract workload defines the functions that are to be called, as well as the parameters as input and the type of transaction. The transaction type is either a *write* transaction, where it invokes a function that will perform an update on state, or a *read* transaction, also referred to as a "request", that will call the blockchain node to obtain or operate on information stored in the state. An example of a contract configuration is presented for the experimental evaluation in Section 5.4.

3) **premade.** If further complexity is required, such as varied or pre-defined input of function calls or unbalanced load generation from secondaries, the benchmark configuration also specifies a *premade workload*. The premade workload references a preconfigured external file containing all transaction information and distribution of requests per secondary, allowing the work and intensity to be heterogeneous on each machine. This not only allows more fine-grained tuning of the workload, it facilitates the use of exact replicas of datasets running on DIABLO. The *premade data* is defined through a generic JSON structure passed into the workload generator, which in turn is formatted and parsed to useful information for the blockchain under test.

The abstract notation and flexibility in the benchmark configuration provides channels for extensibility, as the workload generator defined for each blockchain can convert the information in the configuration to usable transactions.

5.3.2 Chain configuration

The chain configuration comprises of two critical aspects: the blockchain information and the account information that denote how the benchmark will interact. Examples are depicted in Figure 5.5. The blockchain component defines the location of each blockchain node used in

```yaml
name: "ethereum"
window: 12
nodes:
  - 127.0.0.1:8545
  - 127.0.0.1:8546
  - 127.0.0.1:8547
keys:
  - address: "0x3fe512.."
    private: "0x4019ff.."
```

(a) Simple Chain Configuration where nodes denote blockchain nodes

```yaml
name: "fabric"
nodes:
  - 127.0.0.1:9051
  - 127.0.0.1:10051
  - 127.0.0.1:11051
extra:
  - label: "appUser"
    mspID: "Org1MSP"
    cert:
      ..
    key:
      ..
    localHost: "true"
    channelName: "mychannel"
```

(b) Hyperledger Fabric additional config

Figure 5.5: Example Diablo chain configurations.

the benchmark, specifying the address and port that will be used in the communication. It also defines the *window* for the blockchain, which denotes the period that transaction commits are expected, making fairer calculations of transactions per second. For example, the window for Ethereum public blockchain is 12 seconds expressing that a block will arrive every 12 seconds. Finally, Diablo assumes that a list of accounts are provided through the benchmark configuration, as depicted in Figure 5.5a and these accounts are funded to perform all required requests. The unique aspect of this configuration is that the modular design allows *extra data* to be defined in a way that can be interpreted by each blockchain integration, as demonstrated in Figure 5.5b. For example, if a specific blockchain requires additional key material or specific configuration requirements, it can be defined in the chain configuration and integrated through the related workload generator and client interface.

5.4 Experimental evaluation

A number of experiments with Diablo were performed to (1) evaluate the functionality of our benchmark, observing the quality of results returned, and (2) identify the suitability for blockchains to application-based workloads. The workloads range from datasets obtained through real-world applications to fully synthetic workloads, used to evaluate the performance of blockchains under varying conditions as well as evaluate the effectiveness of Diablo in executing these benchmarks. All experiments were performed with 5 repetitions and results aggregated and averaged to formulate the reported results.

5.4.1 Setup

A total of four unique workloads were performed, as depicted in Table 5.2, against a variety of blockchains shown in Table 5.1. Intentionally, blockchains of different guarantees were selected and classified in three groups: the Crash Fault Tolerant (CFT) blockchains that only tolerate crash failures where $n > 2f$; the Byzantine Fault Tolerant (BFT) ones that can tolerate $n > 3f$ Byzantine failures; and Proof-of-Authority (PoA) blockchains that can tolerate some Byzantine failures as long as all message delays take less than a known and bounded time [191] (i.e. these latter blockchains assume *synchrony*).

Cloud infrastructure

All experiments were performed on a local OpenStack [264] cluster, where all virtual machines are connected through a virtual network distributed over 4 host machines. The inbuilt "m1.xlarge" instance was utilised, having 8 vCPUs and 16GB Memory for distributed experiments. A case study was also conducted using *local* experiments through a local Docker swarm on a "c5.2xlarge" instance having 8 vCPUs, 16GB RAM and 700GB disk.

Blockchains

Three subsets of blockchains were selected to be evaluated. The setup tailored for each set of blockchains will now be described.

- **PoA Blockchains.** Geth's Clique [120] and OpenEthereum's AuRA [119] were chosen with a set of predefined validators selected to seal each block, one at a time in a round-robin style fashion. Since both algorithms claim Byzantine tolerance, the network consisted of four blockchain nodes, each participating as an elected sealer. The period between blocks was set to 1 second for optimal throughput, even though it is not recommended[3] for larger networks [265]. DIABLO interacts with these blockchains using a WebSocket JSON RPC[4], submitting transactions through a request, subscribing to new block events to confirm the existence of a transaction when receiving the block header information.
- **BFT Blockchains.** The selected blockchains include Quorum [55] running the Istanbul Byzantine Fault Tolerance (IBFT) [150] consensus, and Collachain, a blockchain running Ethereum with consensus and designs based on the Red Belly Blockchain. Both blockchain networks consisted of four blockchain nodes, to ensure that the threshold of $n > 3f$ was reached. Just as PoA blockchains, as these blockchains are both Ethereum adaptations, they are accessed using the WebSocket JSON RPC, subscribing to blocks to inform of the transaction commit status. In Quorum, the blocks are appended after a configured period (1 second), in which the elected blockchain nodes agree on the proposed block. Collachain, on the other hand, uses DBFT [51] and appends new blocks when necessary, after a timeout or block threshold had been reached.

[3] It is recommended to have larger block times to ensure block propagation to the entire network, but for maximal throughput on a small network, smaller blocks were chosen.

[4] Documentation Available Online: https://eth.wiki/json-rpc/API

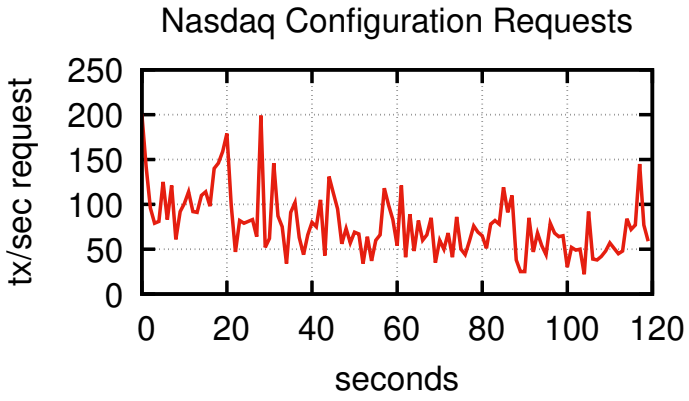


Figure 5.6: Visualisation of the NASDAQ workload request rate timeseries.

- **CFT Blockchains.** Both blockchains, Quorum and HyperLedger Fabric v2.2LTS [172, 171], were deployed using the RAFT [266] consensus algorithm. Quorum was deployed using a three blockchain node cluster to participate in the RAFT consensus, whereas Fabric was deployed with three orderers and three peers connected in a single channel across two organisations. Quorum's consensus is set to propose blocks after a configured time period (1 second), where Hyperledger Fabric appends blocks after a default timeout or threshold of transactions had been reached. Quorum supports the WebSocket API defined through the Ethereum node. Fabric's Golang API supports the functionality to submit transactions and observe the commit through a channel.

5.4.2 Workloads

Four workloads of varying behaviour were selected to investigate the suitability of blockchains. These workloads sources range from real-world application data to fully synthetic tuned data to provide key insights for varying application behaviours.

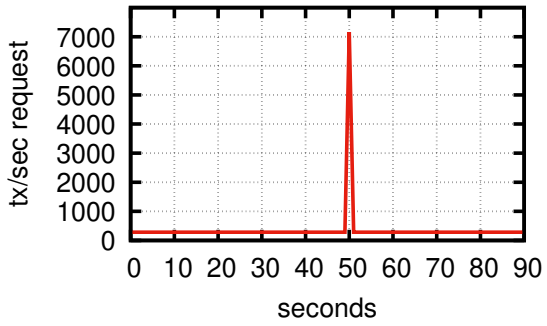
Real World: NASDAQ

Primary behaviour observed in current blockchains is the interaction and hosting of exchanges to transfer coins and tokens. Further blockchain integration looks to achieve performance offered by current centralised systems. A Decentralised Exchange (DEX) is an exchange deployed on the blockchain where all trades, listings and transfers occur through interaction with the smart contract by invoking the relevant functions. This workload featured information observed directly from the centralised NASDAQ exchange[5], following the request pattern obtained from the number of trades in a per-second granularity. Figure 5.6 depicts a visualisation of the request rate observed and used as input to the benchmark workload. The contract in use for the benchmark was a simplification heavily derived from the Maker simple market[6]. This

[5]The National Association of Securities Dealers Automated Quotations exchange.
[6]Sourced from: https://github.com/daifoundation/maker-otc

```yaml
bench:
  type: "contract"
  txs:
    0: 285
    49: 285
    50: 7159
    51: 285
    90: 285
contract:
  functions:
    - name: "tweet"
      ftype: "write"
      ratio: 100
      value: 0
      params:
        - type: "string"
          name: "data"
          value: "08ef..."
```

(a) DIABLO Twitter 5% Configuration



(b) Request Visualisation

```
modifier checklen(string memory data) {
  require(bytes(data).length <= maxlen,
      "tweet too large");
  _;
}
function tweet(string memory data)
  public checklen(data) {
    tweets[msg.sender] = data;
    emit NewTweet(msg.sender, data);
}
```

(c) Twitter Solidity Code

```
func (s *SmartContract) Tweet(
  ctx contractapi
  .TransactionContextInterface,
  message string, owner string) error {
    if len(message) > MaxLen {
     return fmt.Errorf(
          "tweet too large, \%v",
        len(message))
 }
 b := []byte(message)
 return ctx.GetStub().PutState(owner, b)
}
```

(d) Twitter Chaincode

Figure 5.7: Twitter workload represented in the DIABLO benchmark.

workload aimed to bring the behavioural pattern observed on a real-world centralised stock exchange to the decentralised exchanges available today.

Semi-Synthetic: Twitter

Microblogging, social media and instant messaging are increasingly popular applications both on-chain, for example Status.im[7] or AKASHA[8], and off chain including Twitter, WhatsApp and Instagram. While handling constant requests is attractive for most use-cases, unexpected bursts may occur throughput operation leading to unknown behaviour. Vital information about system stability and recovery can be obtained by simulating and experimenting such patterns. This workload was based from a Twitter "tweets per second" record that was set in 2013 during the airing of "Castle in the Sky" in Japan [267], where information was available online. This

[7] Available online: https://status.im/.
[8] Available online: https://akasha.org/about/.

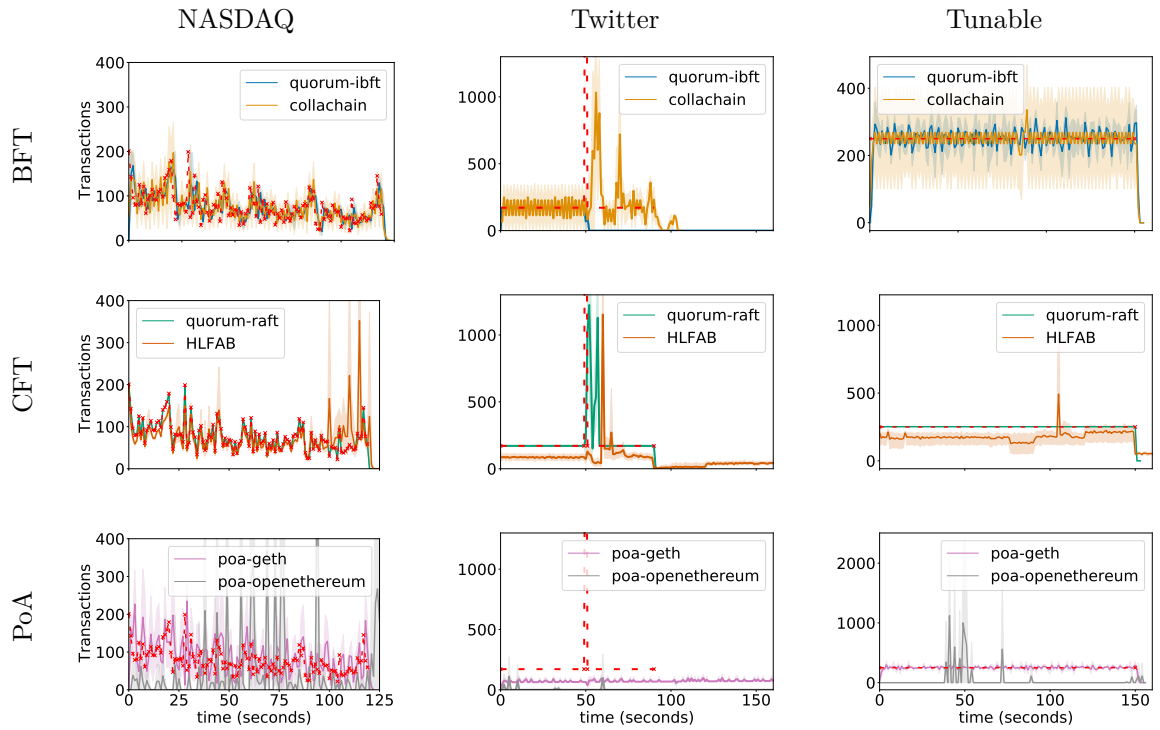


Figure 5.8: DIABLO throughput timeseries results, where the dashed red line represents the request rate.

workload features a constant stream of input with a one second peak 25 times greater than the average, providing an application workload where a burst of requests is sent. Initially this workload featured an average of 5,700 tweets per second, with a peak of 143,199 for one second. While DIABLO had no difficulty generating and handling the workload, it was necessary to make concessions due to hardware limitations, thus smaller percentages of the workload were performed in the experimental evaluation. Figure 5.7 displays the configuration and visual representation of the workload configuration used for the 5% workload. Figure 5.7c and Figure 5.7d provide code snippets of the smart contract code executed on the chain to facilitate the simulation of basic microblogging.

Synthetic: static request rates (Tunable)

Many applications integrated into businesses will contribute a factor of a consistent request rate of varying intensity. The fully synthetic workload was used as a benchmark to help identify the performance of blockchains under constant transaction per second for a defined period of time. This not only helped to identify characteristics of different blockchains, but also revealed information about the stability and performance of DIABLO. 5 iterations of each experiment were performed, calculating the average and variance for each experiment. The first experiment request rate started at 50 TPS for a duration of 150 seconds. Each experiment incremented the TPS by 25, eventually reaching 975 TPS as the final experiment.

5.4.3 Performance comparison

The overall performance comparison measures two axes: *throughput*, a measure of the number of transactions committed over time, and *latency*, measured as the time taken from the transaction request to it being acknowledged as committed in a block. The throughput over time is depicted as a timeseries for each application. Then, the latency of each system is described through a Cumulative Distribution Function (CDF) of these workloads.

Throughput

Figure 5.8 depicts the throughput of all blockchains — computed over a one-second sliding time window — as time elapses during the workload. Each column indicates the result of each application workload, *NASDAQ*, *Twitter* and *Tunable*, whereas the rows represent the groups of blockchains for direct visual comparison. The first observation is that all blockchains offer variable throughput while experiencing request behaviours of the exchange application, seemingly responding to the changes of request rates. The Microblogging workload, with a sudden burst of transactions, seems to have disruptive effects on all blockchains as they try to recover from the intensity of requests experienced. Finally, all blockchains show a steady throughput during the tunable workload, however, it can be seen that some blockchains experience periods of spiked throughput. In particular, Collachain exhibits a variable throughput as the proposal timeout and limit is reached and transactions are committed in batches, indicating opportunities to gain performance by tuning settings. The request rate in each of the workloads are represented by a dashed red line, followed closely by the throughput of each blockchain.

It is observed that in all applications, based on throughput alone, the results of Go-Ethereum's PoA implementation (Clique) outperforms that of OpenEthereum, indicating that Go-Ethereum performs generally better. This observation will be confirmed through the investigation of latency below. Interestingly, it is observed that some blockchains start losing requests under high load. This is the case of Quorum-IBFT under the Twitter workload. As the throughput drops to zero and it is confirmed that some of these requests were lost and returned errors. This is seemingly due to the contention induced by the burst of requests generated in the workload, which exceeds the capacity of Quorum-IBFT in this experimental setup. Quorum-RAFT and Hyperledger Fabric offer seemingly comparable results, even if Quorum-RAFT is at times faster. Later it is explained that this is a repercussion of the experimental settings and cannot be generalised for all settings.

Latency

Figure 5.9 depict the CDF of latencies experienced during the workload execution of the applications. As before, each column indicates the results for the application, whereas the rows group the blockchains into PoA, CFT and BFT. At first glance, it appears that there is large variation between transactions across most settings. This is due to the experimental settings and duration of the experiments. In particular, it is demonstrated that PoA blockchains have

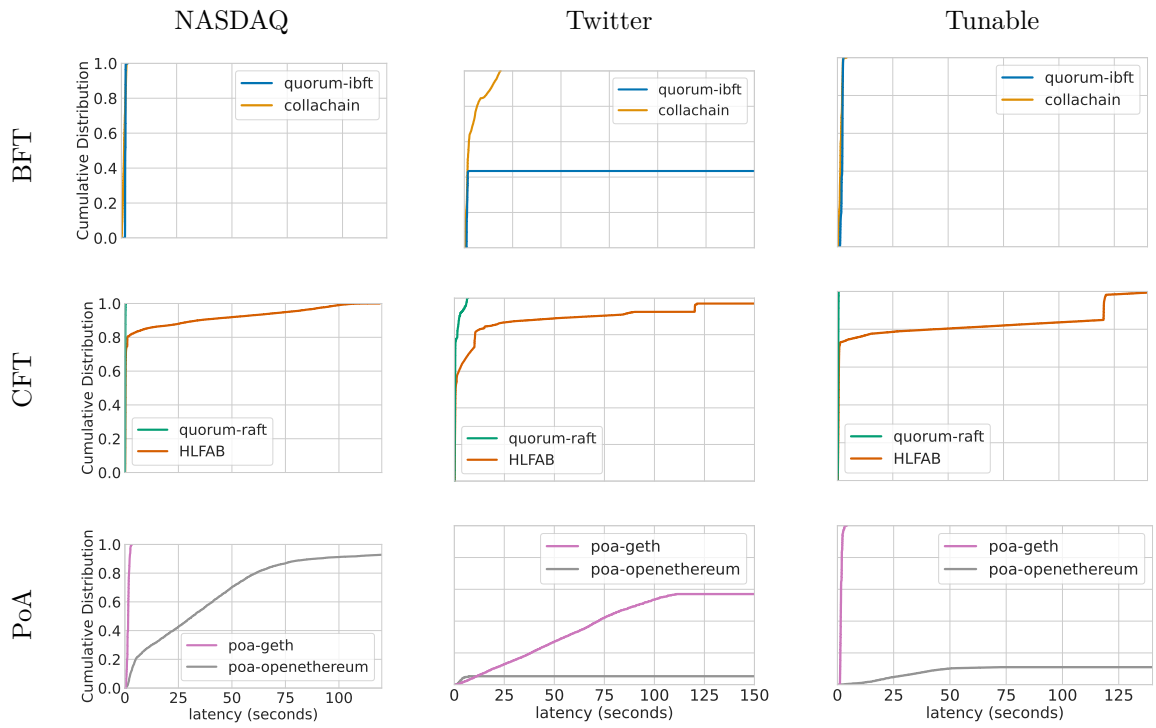


Figure 5.9: DIABLO cumulative distribution functions (CDFs) of the latency.

in general a larger latency than the CFT and BFT blockchains. OpenEthereum's AuRA and Quorum-IBFT experience request drops during workload executions, hence the reason why the latency probability never achieves 100%. In particular, the graphs displayed in the Twitter column indicate that OpenEthereum, Go-Ethereum and Quorum-IBFT all experience request losses due to high contention bursts, confirming the throughput drop experienced for Quorum-IBFT in Figure 5.8. In contrast, Collachain does not experience losses under bursts, however, the latency increases rapidly.

Overview

Through the experimental analysis, there are key takeaways for identifying suitable blockchains for application workloads. However, it is vital to understand deployment environments, as decisions of fault tolerance and networking will provide strict requirements to influence the final decision.

NASDAQ. The experimental results highlight that the throughput of CFT and BFT blockchains are responsive to exchange-like request patterns. However, the latency results highlight that Quorum-RAFT is the most suitable as it provides a latency CDF of sub 1 second, while displaying dynamic throughput matching request patterns. Similarly, both Quorum-IBFT and Collachain provide 1 second or lower latency, showing that if Byzantine faults are present then Collachain would be the most suitable blockchain. Geth Clique also displays comparable throughput and latency, but has a distinct shift in throughput which is evident in the slightly

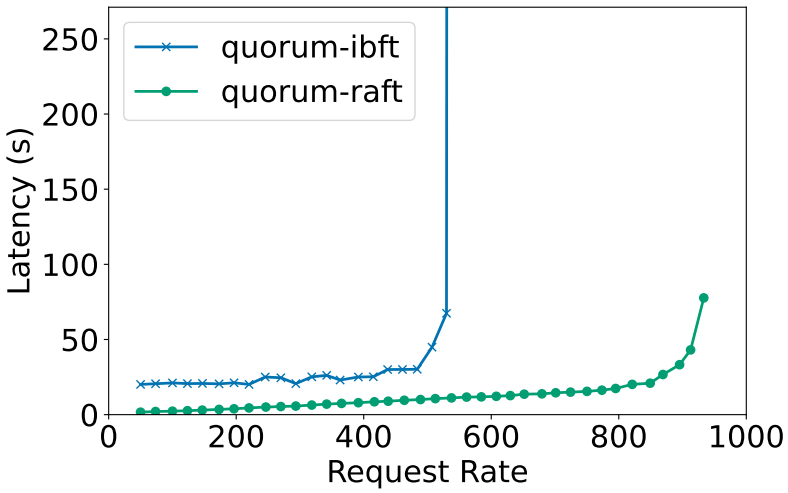


Figure 5.10: The BFT and CFT versions of Quorum.

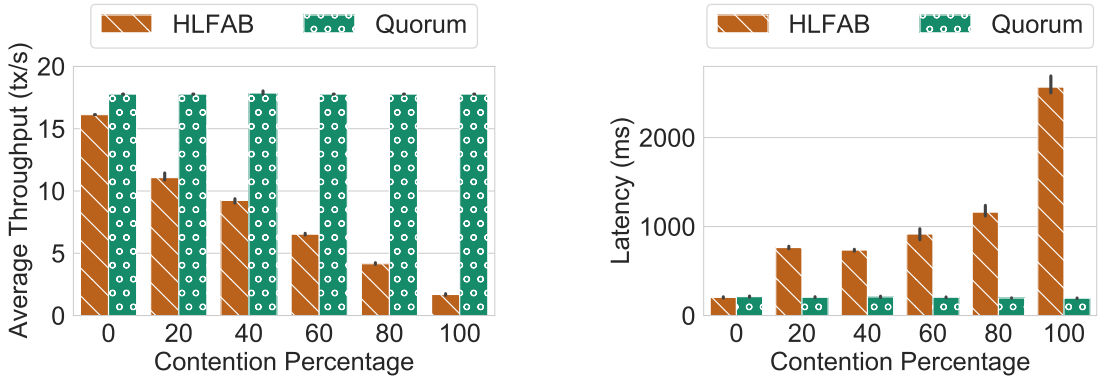
increased latency when compared to Collachain or Quorum.

Twitter. In this experiment, an anomalous burst of requests was experienced, which highlights a systems ability to recover from intense stress. The throughput results show that Quorum-IBFT struggles to recover, whereas Collachain recovers over time with lasting effects. For tolerating Byzantine faults, Collachain would be the most appropriate blockchain, as it presents throughput that handles the request rates, while also providing a sub 25 second latency when processing such large bursts. Quorum-RAFT presents similar results, showing quick recovery in the system but provides sub 10 second latency, making it an attractive candidate for behaviours of this sort.

Tunable. The Tunable workload provided a constant request rate of 250 transactions per second. This highlights a systems ability to maintain performance for ongoing transactions. Both Quorum integrations, IBFT and RAFT, offer similar performance, maintaining performance over the duration of the workload. Similarly, Collachain offers a maintained performance comparable to Quorum-IBFT but shows evidence of periods where threshold-based proposals were not met.

5.5 Observation: The impact of fault tolerance

In order to assess the inherent costs of tolerating Byzantine faults, the performance of Quorum was investigated. Quorum offers a unique perspective in which it provides an implementation of two consensus mechanisms on the same architecture, namely BFT consensus using Quorum-IBFT that tolerates Byzantine faults and CFT consensus using Quorum-RAFT that can only tolerate crash faults. This facilitates direct evaluation of the cost of tolerating Byzantine faults, as the differences in systems are restricted to the consensus, due to the foundation of Ethereum



(a) Throughput Results

(b) Latency Results

Figure 5.11: Throughput and Latency comparison of Hyperledger Fabric and Quorum-RAFT in workloads that vary contention percentages.

being used the same way across the varying consensus implementations. Not only does this provide valid constraints to evaluations, it allows for exploration into workloads where differences in Byzantine Fault Tolerance against Crash Fault Tolerance can be highlighted. Figure 5.10 shows the difference in performance between the BFT and CFT versions of Quorum, Quorum-IBFT and Quorum-RAFT. In particular, the graph indicates the evolution of latency as higher throughput is reached.

It can be seen that tolerating Byzantine faults requires additional mechanisms that impact the overall performance of the system. This performance difference is evident not only in lower throughput environments but has adverse effects at higher request rates as Quorum-IBFT was unable to handle requests and began failing. Therefore, it is critical to analyse the use case and environment when selecting a blockchain for application use, as tolerating Byzantine faults may lead to system instability with high request behaviour.

5.6 Exploring the impact of contention

Figure 5.11 depicts the throughput and latency of Hyperledger Fabric and Quorum-RAFT. The workload curve is a low constant send rate of 20 transactions per second with a duration of 30 seconds. The workload is purposefully very simple to showcase that transaction conflicts may arise even in low arrival rate situations. The contention percentage denotes the percentage of transactions which modify the same asset.

5.6.1 Drawbacks of speculative executions

As a permissioned blockchain, Hyperledger Fabric can provide parallel pre-ordered transaction execution on a subset of network nodes (subset defined by the endorsing policy). However,

a model that executes each transaction in parallel is inherently unable to detect transaction conflicts (e.g., concurrent modifications of a same asset) during execution.

As an example, suppose the system can support a maximal throughput of 1000 transactions per second with a maximum of 100 transactions per block. Given 20 transactions out of the 100 in the same block attempt to modify the same asset, only 1 will be executed and deemed "*valid*", resulting in 19 failed transactions. Subsequently, if the clients of the rejected transactions attempt to re-execute their transaction, it will add 19 more conflicting transactions to the following block. This cascading effect can eventually lead to more conflicting transactions being proposed as a re-execution of a failed outcome. Moreover, with cumulative re-execution, the number of aborted transactions grows linearly until it surpasses the throughput of the system. Thus, if clients re-execute invalid transactions as their default behaviour, it will effectively become an unintentional Denial-of-Service attack on the blockchain [268].

Although the idea of re-executing transactions was proposed to bypass this limitation, it appears that such solutions are not ready for production as they are not durable. FastFabric [232] improves over Hyperledger Fabric, achieving 20,000 transactions per second throughput. It features optimisations that replace the state database with a hash table, storing blocks in separate servers and separating the committing and endorsing peers on separate servers. This allows for parallel validation of the transaction headers as well as caching the unmarshalled blocks. XOX-Fabric [268] builds upon FastFabric by going a step further, introducing additional execution steps to enable correct performance even in the presence of contentious workloads. In a later paper, however, FastFabric was mentioned to feature "optimisations [that] are not practical for a production environment [...] there is no disk IO which is not suitable for a production setup" [269]. XOX seems to be a complicated solution to a specific use-case that may not be frequently encountered if designed carefully.

5.6.2 Benefits of pessimistic executions

Most blockchains, including Collachain, Ethereum, Quorum, execute transactions pessimistically, ordering transactions before executing them. The focus here is on Quorum-RAFT that is crash fault tolerant and can thus be compared directly with Hyperledger Fabric running RAFT. Quorum-RAFT, being a fork of Geth, implements an *order-execute*[9] pattern in its transaction flow. Therefore, even under fully contentious workloads, it is not susceptible to transaction conflicts as every node in the network will execute the transactions in the same order. Every blockchain node should thus end with the same results (as long as the smart contract is deterministic). As a result, the performance of these blockchains should not be affected by the example mentioned above. In particular, Figure 5.11 shows that the performance of Quorum-RAFT is stable in this workload, no matter the contention percentage.

[9]Order-execute occurs when transactions are ordered into a block prior to the entire execution.

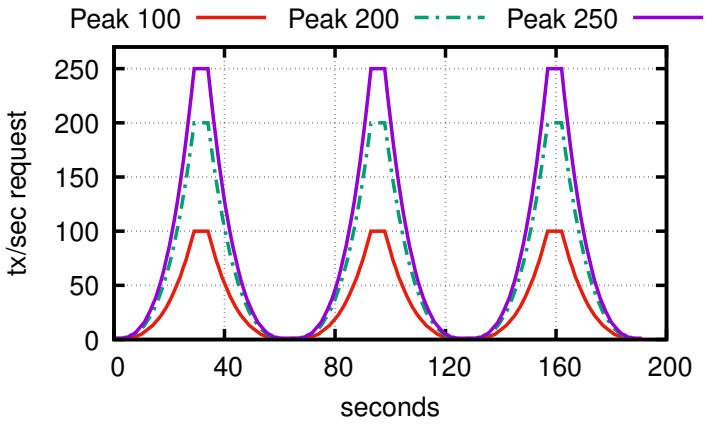


Figure 5.12: DIABLO Aviation workload requests visualisation of the Non-Homogeneous Poisson Process. The duration of three minutes presents three overall peaks in requests. The variation of intensity includes 100, 200 and 300 transactions per second.

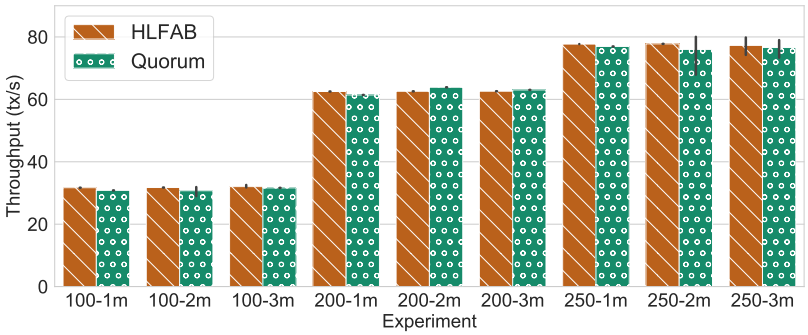
5.7 Case study: Aviation parts

Blockchains have attracted the attention of supply-chain industries, especially for sourcing and trading between vendors while tracking quality and handling. The Aviation workload was heavily influenced by the Fabric Honeywell Case Study [270], which depicts an online marketplace for aviation parts. This workload introduces an environment that requires both throughput and latency to be optimal.

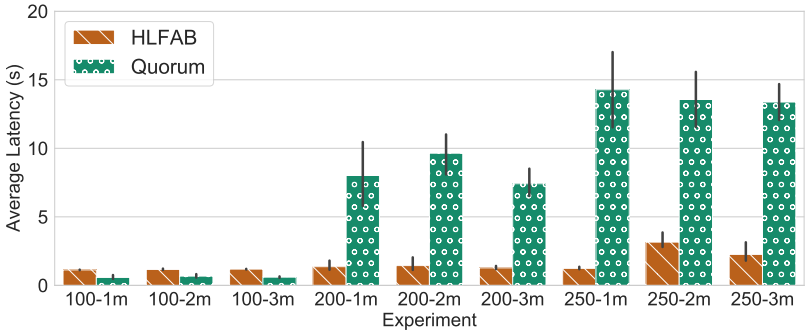
Since the aviation industry is heavily regulated, sales in the U.S. require certification from the U.S. Federal Aviation Administration and other agencies. Each part must be carefully documented with a complete history of its ownership, use, and repairs. Therefore, facilitating online purchasing of aircraft parts in an Amazon-style online store requires significant trust and processes in place to comply with regulations. This can be facilitated through a trusted ledger employing data integrity and anti-counterfeit measures, which enables the tracking of the entire life-cycle of a part, including its current and previous owners. Because aircraft pieces are large and expensive, deals tend to be made through purchase orders and not simple online or credit payments. For users to be confident in their purchase, each order is stored indefinitely on the ledger to provide traceability. The corresponding smart contract provides 3 core functions:

- CreatePart() which creates an aviation part and adds it to the ledger.
- QueryByOwner() which queries all parts owned by a certain owner. Depending on the smart contract language, this may not be easily implemented.
- TransferPart() which transfers an aviation part from owner to a new one by modifying the owner field of the part, adding a purchase order onto the ledger to provide traceability of the aviation part.

Due to the availability of the Honeywell data, a number of behaviour requests were synthesised following a Non-Homogeneous Poisson Process (NHPP) it is believed this best fits a



(a) Average Throughput



(b) Average Latency

Figure 5.13: Average throughput and latency of the aviation workloads. "100", "200" and "300" represent the peaks, "1m", "2m" and "3m" denote the duration and number of peaks.

behaviour pattern expected from a service of this type [271]. The assumption made here is that the intensity (mean arrival per second) of the Poisson process changes with a cubic rate as a function of time. The justification behind this assumption is that it mimics the very quick change in traffic between peak hour (e.g. day time just after work) and quiet hour (e.g. night time). Here it was scaled down to minutes for benchmarking purposes but the justification remains the same. It can be noted that the curve stays at its peak for 6 seconds before going down. Figure 5.12 shows a visual representation of the experiments performed for the Aviation workload.

These experiments were performed using a local Docker swarm using Hyperledger Fabric and Quorum-RAFT. In these experiments, both (i) the number of peaks and duration of the tests, and (ii) the intensity of the peaks were modified. This results in 9 different workloads, where the length (1, 2, or 3 minutes) and the peak intensity of the Poisson Process (100, 200, or 250) were combined to generate requests. The transactions invocations are made up of 50% CreatePart(), 25% QueryByOwner() and 25% TransferPart(). Each experiment was repeated three times and the results display the averages over the repetitions.

Figure 5.13 shows the average throughput and latency for each experiment. It was observed that the length of the workload had little impact on the average overall throughput and latency

Table 5.3: A comparison of benchmark frameworks. Caliper presents current work to distributed benchmarking requiring Zookeeper, whereas others work from a single source. However, DIABLO differs from Blockbench which provides a rigid design that supports microbenchmarking of individual components but requires significant investment in implementation for integration of new blockchains.

	ChainHammer [48]	Caliper [47]	Blockbench [50]	DIABLO
Blockchain extensibility	✗	✓	✓*	✓
Workload extensibility	✗	✓*†	✗	✓
Distributed clients	✗	✓‡	✗	✓
Pregenerated Load	✗	✗	✓	✓
Limited Coordination Overhead	✓	✗	✓	✓
Microbenchmark	✗	✗	✓	✗

* Requires significant implementation for integration of extended components

‡ Currently under development

† Limitations with workflow, expects contracts and implementation of rate limiters.

but intensity was the key factor, as expected. In the 100 request rate peak, Hyperledger Fabric achieves marginally higher throughput but experiences a higher latency. The majority of the transactions for both Hyperledger Fabric and Quorum-RAFT at this request rate were sub 1 second, as shown in Figure 5.13b, and sustained a consistent throughput and latency for the experimental reruns. In experiments with higher request rates, although Quorum provides similar throughput, the average latency drastically increases, whereas Hyperledger Fabric maintains almost constant latency.

These experimental results highlight the importance of experimental setups, as Hyperledger Fabric seems to outperform Quorum in a low latency network, such as a local Docker swarm, whereas experiments in Section 5.4 show varying results.

5.8 Summary

It is no surprise that benchmarking and performance evaluation has captured the interest of many, assisting in understanding the vast array of blockchains available today. Accurate performance measurement and evaluation is vital in the identification of advantages and challenges of different systems and their suitability to various use cases. Unfortunately, however, a majority of the results available to the public are released through non peer-reviewed channels, often through blogs, social posts or whitepapers where evaluations are tailored for optimal conditions with hidden details.

Although solutions exist [47, 48, 50, 175, 173] for comparing blockchains, they often experience drawbacks as a consequence of early design decisions. Three major concerns faced

by current implementations have been identified: (1) the lack of distributed load generation, resulting in misleading results due to client machine-level bottlenecks, (2) misleading overheads caused by various infrastructure integrations, and (3) cryptographic processing during timed periods, such as transaction signing, causing miscalculations in the measure of throughput.

In this chapter, we presented DIABLO, a distributed blockchain benchmark framework that solves hindrances that effect other available benchmark frameworks, as summarised in Table 5.3. An analysis was then performed on three groups of blockchains, namely (i) Crash Fault Tolerant (CFT), consisting of Quorum-RAFT [55] and Hyperledger Fabric [172, 171], (ii) Byzantine Fault Tolerant, consisting of Collachain and Quorum-IBFT [150] and (iii) Proof-of-Authority offered by Go-Ethereum's Clique [120] and OpenEthereum's Aura [119].

DIABLO offers an easily extensible benchmark with support for real-world application-level benchmark workloads. By adhering to a modular design, DIABLO presents a system where new blockchains can be easily integrated through implementing two interfaces: a client interface for interactions and basic metrics; and a workload generator to format and generate workload information. The ability to provide a distributed, dynamic workload allows imitation of application behaviour, facilitating comparisons of blockchains against workloads they will experience in deployment. This provides application developers with insight into how their applications will behave on different systems, while informing blockchain developers about bottlenecks in their designs. Future direction presents two opportunities, leading to the evaluation of new blockchain systems, as well as integrating more workloads to assist in the selection of blockchains under varying application environments. This will also entail future integrations of simulated faults, such as crash or Byzantine faults, to measure a systems performance under these respective scenarios.

A comparison of six blockchains under three different workloads and a case study was presented, providing throughput and latency comparisons across blockchains. It is observed that a burst of intensity will have lasting effects on most blockchains, as they recover from numerous connections and requests. Specifically, Go-Ethereum, Quorum-IBFT and OpenEthereum fail to recover correctly. It is also observed that, when comparing the CFT and BFT variants of Quorum, handling Byzantine faults leads to higher latency on average and may affect the total throughput, refining the notion that selecting an appropriate blockchain heavily relies on a thorough understanding of the requirements set out by the deployment environment.

Chapter 6

Conclusion

In this dissertation, we have presented our contributions to the global effort of achieving secure, high performance blockchains. The overlaying notion of this dissertation is identifying concepts and components that contribute to safety and performance, both relying on meticulous analysis. A number of weaknesses have been identified [28, 29, 30, 31, 32, 33, 34, 35, 36], targeting various components of the blockchain based on design choices and assumptions that prove challenging in practice. Identification of these weaknesses has paved the way for numerous blockchain variants, all aiming to improve on shortcomings or extend beyond the original specifications set out by Bitcoin. However, this added considerable complexity to understanding blockchains, as each variant offers new perspectives of components, assumptions and completely revitalised models that are tailored for specific environments. As these new blockchains are commonly released through blog posts, whitepapers or social channels, specific details remain hidden by abstractions or implications that are not clearly stated.

The core research objective was to *design a secure, high performance blockchain*, which can be broken into two streams — secure implies correctness and safety against known weaknesses and attacks, whereas performance implies qualities such as throughput, latency and scalability requiring both theoretical and experimental evaluation and comparison. This dissertation presented fundamental contributions to achieve the objective of a secure, high performance blockchain demonstrated through the *Red Belly Blockchain*. First, we evaluated the assumptions of blockchains, presenting the *Blockchain Anomaly* that indicates a weakness in traditional probabilistic blockchain consensus. We developed the *Balance Attack*, a novel attack that highlights the dangers of probabilistic consensus with transient forks allowing an adversary to double-spend when abusing network connectivity amongst subgroups of nodes, demonstrating experimentally and theoretically the feasibility of such an attack. This motivated our design and implementation of a secure blockchain, the *Red Belly Blockchain*, that provides safety against Byzantine faults and favours consistency over availability, mitigating the risks of the Balance Attack and Blockchain Anomaly. Finally, to evaluate the performance of the Red Belly Blockchain design, we performed experimental evaluation and developed a framework, DIABLO, for fair, comparative analysis of blockchains to facilitate insightful evaluations of performance under ranging experimental conditions tailored for varying application behaviours.

However, a supplemental backstory of this dissertation, the expeditions and the road to "El Dorado", hints at an integral finding that the journey for secure, high performance blockchains will forever be on-going, searching for the blockchain that provides perfect security and overwhelming high performance under all conditions, even with the use of ancillary components such as smart contract executions.

6.1 Outcome of research objectives

Our study presents three key research objectives forming one core objective. These three key objectives were, in turn, split into several sub-objectives that formulate the process to achieve the core objective. The outcomes and the contributions that were facilitated through these research objectives will now be presented.

6.1.1 Objective 1: Investigate weaknesses of blockchain consensus

Objective 1 focused on identifying shortcomings of blockchains, an integral evaluation to use as motivation as to why secure, high performance blockchains are required. The objective was split into two sub-objectives, namely:

1.1: *To analyse and assess the assumptions of blockchains in the context of their deployment environments.*

1.2: *To investigate the strength of consensus in an adversarial scenario.*

Sub-Objective 1.1. Analysis of assumptions

Our analysis began by evaluating whitepapers and documentation surrounding the mainstream blockchains, at the time concentrating on Bitcoin and Ethereum, evaluating the assumptions stated, or lack thereof. We turned our focus to the consensus, as previous results had indicated weaknesses surrounding the probabilistic guarantees [77, 79, 29, 30, 31, 32, 33, 34, 210]. Many of these analyses provide insight into network-related assumptions and the effect of delaying critical messages.

Blockchain Anomaly Our evaluation of blockchain assumptions and environmental conditions led to the investigation of implicit synchrony assumptions that are misleading for users that have minimal understanding. We presented the Blockchain Anomaly, demonstrating anomalous behaviour that occurs as a result of a chain reorganisation based on transient forks during periods of large message delay. It is emphasised that users should understand the guarantees presented by the blockchain and familiarise themselves with functionality and techniques before integrating it with critical operations.

These findings highlight that there are weakness present in the implicit synchrony assumptions and probabilistic guarantees adopted by blockchains, specifically targeting blockchains where transient forks may cause unknown chain reorganisations.

Sub-Objective 1.2. Investigation through adversarial scenario

In our previous sub-objective, we highlighted the effects of probabilistic consensus with fork resolution, leading to unwanted results during periods of message delays and chain reorganisations.

The Balance Attack Based on our findings, we developed an attack, where the goal is an adversary being able to successfully double-spend with high probability. To this end, we presented our contribution, The Balance Attack, depicting a scenario where an adversary can double-spend by manipulating the network based on the assumptions. If the adversary can successfully segregate the network into subgroups, then influence the fork choice rule, they can double-spend with high probability.

By demonstrating the attack, both theoretically and experimentally, we highlight the impact of probabilistic synchronous consensus that succumbs to forks that can be reorganised. Although this attack may prove infeasible on the large scale with the Internet, large state actors or extremely anomalous events, such as country-wide connection issues, may provide vectors to increase feasibility of performing the attack.

Our conclusion is that safety under these conditions is imperative for critical infrastructure and business consortiums that may look to use the blockchain as a settlement platform or integrated into their underlying framework. We proposed a solution using an unforkable blockchain, favouring consistency over availability in the presence of a network segregation providing a more practical model.

6.1.2 Objective 2: Design and implement a secure, high performance blockchain

Based on our findings in Objective 1, we tailored a design that provides safety under partial synchrony, while also offering high performance. To achieve this, the research objective was divided into two sub-objectives:

2.1: *To assess and evaluate current blockchain designs, with a focus on designing a secure, high performance blockchain.*

2.2: *To design and develop a blockchain providing high performance that is highly resilient, especially in adversarial scenarios.*

2.3: *To analyse and evaluate, experimentally, the performance of the implementation and competing designs.*

Sub-Objective 2.1. Assessment and evaluation of blockchain designs

Our findings in Objective 1 highlight the importance of practical assumptions. However, designing a blockchain proves difficult due to the complexity hidden in the underlying components. Identification of appropriate components was integral to provide a system that not only is safe under these adversarial conditions, but yields potential for high performance.

Deconstruction of Blockchains This prompted us to view the blockchain with a new perspective, seeing it as a construction of components, each with varying assumptions and requirements that is passes to the others. We proposed the Deconstruction of Blockchains; a dissection of the blockchain into three key components, namely Membership Selection, Blockchain Consensus, and Structure, which provides a categorisation of blockchain proposals assisting in identifying suitable components for the design while reducing the complexity to better understand the landscape. By categorising proposals, we were able to formulate suitable components to use that match our requirements and guarantees.

The work of DBFT by Crain et al. [51] formed the foundation of the design, as it proposed a unique view of consensus that was extremely fitting to the blockchain as it was leaderless and efficient with properties that matched expectations. We also selected to follow the design of a single, canonical chain proposed by Bitcoin as the core structure to (1) alleviate complexity, and (2) to provide the consensus with a strict requirement that only one block per index was required from the consensus, leading to the foundation of forming "superblocks". Complex structures, such as DAG proposals, often required high latency to read state, which did not match performance requirements set in our goals.

Sub-Objective 2.2. Design and development of a secure, high performance blockchain

We used our investigation of components to form the foundation of the design, where DBFT [51] provided the fundamental core due to its unique properties fitting our desired requirements. Our primary focus dealt with the environment where a smaller subset of nodes would be tasked with consensus, whereas the majority of the network interact by submitting transactions. We formulated an initial design, however, to perform experimental analysis and understand the feasibility in practice, an implementation would be required.

Red Belly Blockchain We proposed The Red Belly Blockchain, an unforkable blockchain that provides safety under the conditions presented in our previous findings. The implementation amalgamates the designs of consensus and structure to define a proof-of-concept and working system with the ability to produce blocks across a distributed set of participants, and process transactions. The key design aspects provide censorship-resistance by combining proposed sets of transactions into a union, forming a superblock, while also lowering the global CPU wastage through verification sharding. These not only provide performance improvements, as shown in experimental evaluations, but also offer scalability on the order of hundreds

of consensus participants.

Sub-Objective 2.3. Analysis and experimental evaluation of implementation compared with competitors

To evaluate the effectiveness of our design, we performed experimental analysis in a range of environments (Section 4.6 and Section 4.7), with an aim to understand the performance bottlenecks introduced through design choices. Initially, comparisons were made between two consensus implementations, an implementation of PBFT [15] as well as HoneyBadger [144], showing empirically that our design provided performance improvements to the blockchain and was able to outperform the comparisons. This experimental analysis also provided results showing that our design is scalable, in which the performance grows with the number of nodes, and that we can achieve a peak throughput of 660,000 transactions per second in an optimal environment. We then implemented an integration of the Red Belly Blockchain in the Hyperledger Caliper benchmark, providing us with comparative benchmarking to evaluate performance. We showed that the Red Belly Blockchain can handle requests almost an order of magnitude higher than some other Byzantine Fault Tolerant blockchains, Quorum-IBFT [55] and Burrow [169].

Our experimental analysis highlighted that our design provides high performance, demonstrating that the Red Belly Blockchain performs almost three times higher than the closest competitor. We conclude that the design of the Red Belly Blockchain provides safety and high performance, as shown both theoretically and experimentally, fulfilling our objective to design a secure, high performance blockchain.

6.1.3 Objective 3: Compare and evaluate available blockchains to investigate suitability to applications

Growing interest in the blockchain resulted in the development of number of decentralised applications. Research Objective 3 was formulated to investigate the performance of various blockchains under transaction behaviour experienced through application use. To investigate, we split the objective into three sub-objectives:

3.1: *To design, develop and evaluate a benchmark framework to measure blockchain performance under real world application workloads, in particular focusing on fair evaluations.*

3.2: *To evaluate, experimentally, the performance of blockchains through investigation of real world application workloads.*

Sub-Objective 3.1 Design, development and evaluation of a benchmark framework

Our experimental analysis of the Red Belly Blockchain consisted of two evaluations, one from a self-developed set of experiments, and another from an integration on a platform. Whilst evaluating available benchmark frameworks, we identified and resolved a number of shortcomings

that presented limitations in the ability to benchmark high performance systems, which could have led to misinterpretations and false conclusions being drawn.

Overall, we identified three key hindrances affecting available benchmark frameworks; (1) Lack of distributed load generation, since a vast majority consist of requests originating from a single source, limited by the resources of that machine rather than the blockchain network under test, (2) Misleading overheads such as infrastructure or containerisation affecting the ability to accurately capture and measure performance, and (3) Cryptographic processing during time periods, where transaction generation and signing would be included as part of the benchmark, leading to throughput being affected by the performance of the benchmark client's ability to sign.

We concluded that the frameworks available today would be ill-suited for real world application-based workloads, as they lack the flexibility to provide dynamic workloads that capture performance metrics on a fair ground. It is imperative to ensure that benchmarking blockchains capture performance to form a fair basis of comparison, mitigating any limitations induced by the benchmarking infrastructure. This prompted a need for a benchmark framework that could accurately capture application-based workloads, providing dynamic request behaviour while distributing the workload.

Diablo To this end, we present our benchmark framework, Diablo, building upon the shortcomings identified in other frameworks while presenting application-based workloads. Diablo provides the capabilities of distributed load generation as well as dynamic transaction request rates, aiming to fully emulate the environment experienced by blockchains during application usage. While providing built-in workloads and integration, its modular design allows DApp designers and blockchain developers to easily integrate and test their ideas.

Sub-Objective 3.2. Evaluation of blockchains in application workloads

Our benchmark framework provided us with the ability to evaluate blockchains under various application conditions, using both synthetic and real-world behaviours. A comparison of three groups of blockchains was presented: (i) Proof-of-Authority, where we test Go-Ethereum's Clique [120] and OpenEthereum's Aura [119], designed for smaller networks or testnets where delegated authorities generate and propose blocks, (ii) Crash Fault Tolerant blockchains which include Quorum [55] and Hyperledger Fabric both running RAFT consensus [266], and (iii) Byzantine Fault Tolerant blockchains which includes Quorum running Istanbul BFT [150] and Collachain running DBFT [51] and features from the Red Belly Blockchain (Chapter 4).

Our experimental analysis provided insight into three application workloads and a case study, highlighting that large bursts of requests have lasting effects as the systems try to recover. We identified the issues of contention and how speculative executions may suffer when performing workloads that follow conflicting writes. We also observed that handling Byzantine faults may

provide security in the system, but has an impact in latency due to the added complexity of the messaging for the consensus to reach agreement.

We present our benchmark framework, DIABLO, that offers the ability to provide a comparative analysis across blockchains under various application workloads. We conclude by presenting that handling Byzantine faults may provide security at the cost of latency for messaging, and that speculative execution may prove problematic in workloads of high contention. We showed that Go-Ethereum's Clique performs, on average, better than OpenEthereum under all workloads. We highlighted that Quorum's RAFT provides similar throughput to that of Hyperledger Fabric, but lower latency in distributed environments, whereas Hyperledger Fabric is influenced by low latency environments. We also showed that Collachain performs similar to Quorum's IBFT integration, but can handle and recover from high bursts of requests, whereas Quorum's IBFT struggles. We wish to highlight that selection of an appropriate blockchain requires the users to understand the requirements of security and in-depth design of the deployment environment to best provide input to narrow down the selection.

6.2 Future direction

The work presented in this dissertation opens several channels for future research. As blockchain technology matures, with new proposals emerging frequently, there is a global effort towards secure, high performance blockchains. Yet, a perfect solution in all environments against all attacks does not exist. Future work entails research into how these performance and security guarantees can be presented in larger deployments whilst considering various behaviours of malicious actors, such as collusion or large factions. These efforts assist with further blockchain adoption in various financial fields with an objective of increased transparency and less reliance on third party intervention.

Investigation of future weaknesses New proposals provide solutions against related attacks, however, a number of assumptions are placed on network and user behaviour. As various proposals grow in popularity and deployment, such as Proof-of-Stake being developed and deployed in Ethereum at a large scale [87], it is imperative to investigate how new vectors present themselves and must be analysed to ensure security. The Blockchain Anomaly hints at assumptions that may overlook the impact on a user if they are unaware of the guarantees and requirements of the system. Such findings are critical in the operational safety of blockchains if they are to ever bridge the gap into critical infrastructure and widespread use. Potential for weakness is heightened when considering behaviour of nodes and the presence of collusion amongst adversaries.

Performance improvement in various deployments Our contribution of the Red Belly Blockchain hints at the potential improvements available to blockchain technology. As consensus is only a component of the blockchain, other components may yield better performance with further investigation. Various emerging deployment environments, such as low-powered

devices, mobile and IoT devices that interact with the blockchain have distinctive requirements placing restrictions on the overall operation of the blockchain. Future research direction would be optimising components for these environments to allow high scalability while offering high performance and security.

Performance measurement DIABLO provides a benchmark framework to openly extend with workloads and blockchain integrations. Future direction involves the integration of more blockchains, such as Algorand [42] or Diem [40], to provide more data points for application designers. Similarly, additional workloads will provide blockchain developers with key performance indicators to better understand where their proposals may be suited and assist in identifying any shortcomings or bottlenecks in their designs. Future direction will also entail the integration of simulated malicious behaviour, such as crashes or Byzantine behaviour to measure performance under faults.

Notations

Balance Attack

Ψ	total mining power of the system (in million hashes per second, MH/s)
d	difficulty of the crypto-puzzle (in million hashes, MH)
ρ	fraction of the mining power owned by the malicious miner (in percent, %)
κ	the number of communication subgraphs
S	a blockchain system (e.g. Ethereum, Bitcoin)
m	the number of block confirmations to commit
T	a set of known transactions
b	a block consisting of a subset of all known transactions
B	the set of blocks
tx	a transaction
R	the set of "pointers", linking blocks to their respective parent
τ	disruption time of communication between subgraphs (in seconds, s)
μ_c	mean of the number of blocks mined by each subgraph during τ
μ_m	mean of the number of blocks mined by the attacker during time τ
Δ	the maximum difference of mined blocks for the two subgraphs
f	the upper bound on the number of faulty nodes present in the system
n	the total number of nodes in the system
N	the set of nodes in the system
pow	A mapping from a node in N to its mining power in $\mathbb{R}$

Red Belly

η	the deterministic selection function to select primary proposer for tx
a	an account
b	a block consisting of a subset of all known transactions
B	the set of blocks
β	the size of a block
f	the upper bound on the number of faulty nodes present in the system
n	the total number of nodes in the system
N	the set of nodes in the system
V	the set of verifiers
P	the set of proposers
Q	a set of correct proposers
w	a set of transactions proposed to the Set Byzantine Consensus
tx	a transaction
T	the set of known transactions
σ	transaction signature

Glossary

Double-Spending A phenomenon that occurs during a fork, where the same coin is able to be used twice in conflicting transactions, where an adversary is able to obtain more goods for the price of one coin.. 34, 36

Membership Selection The process of selecting a subset of nodes to participate in critical system tasks, such as Proof-of-Work selecting one node to propose a block.. 17

Mempool The "memory pool" contains a collection of transactions stored on a node that have yet to be included in a block.. 73

Nakamoto's Consensus Nakamoto's consensus depicted in the Bitcoin whitepaper states that in the event of a fork, the chain to be selected is the one with the highest number of blocks, or the longest chain.. 3, 21

Permissionless The environment in which there is an unknown, dynamic set of nodes able to join and leave at any time.. 3

Smart Contract Code that is run on the blockchain depicting conditions for payment, or the execution of functions that lead to a conditional transfer of assets.. 4, 11

Sybil Attack First introduced in [24], depicts an attack where an adversary spoofs multiple identities to gain a disproportionate amount of influence in the system.. 17

Testnet A blockchain network that is primarily used for testing and often has no financial values. It reflects characteristics of the main network, but often with modified parameters for consensus or difficulty. Many of these test networks are also used to test protocol upgrades or consensus changes to ensure correctness.. 19

Trusted Execution Environment (TEE) Secure space of trusted hardware that guarantees the code run in the enclave will follow pre-defined policies and algorithms. Introduced through Intel's SGX [272], AMD's PSP [273] and heavily influenced by ARM's TrustZone [274].. 19

Uncle Block A block not included in the main canonical chain, but whose parent is included.. 10

Unforkable A blockchain system that is designed in a way that it does not experience transient forks as a result of probabilistic guarantees or network partitions. It may experience hard forks as a result of protocol upgrades, if designed, but will not be transient as a result of proposals being added on the same index on a canonical chain. In short, no two honest nodes disagree because of the consensus.. 15

List of Frequent Acronyms

BFT Byzantine Fault Tolerant. 2, 20, 23, 112

CFT Crash Fault Tolerant. 2, 112

DApps Decentralised Applications. 11, 101, 104

DBFT Democratic Byzantine Fault Tolerance. 67, 68

DEX Decentralised Exchange. 113

DLT Distributed Ledger Technology. 9

DPoS Delegated Proof-of-Stake. 18

FLP Impossibility Fischer, Lynch, Paterson Impossibility. 3

GST Global Stabilisation Time. 3

HBBFT Honey Badger's BFT Consensus. 71

IBFT Istanbul Byzantine Fault Tolerance. 25, 68, 112

IoT Internet of Things. 20, 103

PoA Proof-of-Authority. 19

PoS Proof-of-Stake. 18

PoW Proof-of-Work. 3, 4, 17

RBBC Red Belly Blockchain. 7, 67

RSM Replicated State Machine. 71, 74, 179

SMR State Machine Replication. 2, 67

TPS Transactions Per Second. 29

Bibliography

[1] Christopher Natoli, Jiangshan Yu, Vincent Gramoli, and Paulo Jorge Esteves Veríssimo. "Deconstructing Blockchains: A Comprehensive Survey on Consensus, Membership and Structure". In: *CoRR* abs/1908.08316 (2019). arXiv: 1908.08316. URL: http://arxiv.org/abs/1908.08316.

[2] Christopher Natoli and Vincent Gramoli. "The Blockchain Anomaly". In: *15th IEEE International Symposium on Network Computing and Applications, NCA 2016, Cambridge, Boston, MA, USA, October 31 - November 2, 2016*. Ed. by Alessandro Pellegrini, Aris Gkoulalas-Divanis, Pierangelo di Sanzo, and Dimiter R. Avresky. IEEE Computer Society, 2016, pp. 310–317. DOI: 10.1109/NCA.2016.7778635. URL: https://doi.org/10.1109/NCA.2016.7778635.

[3] Christopher Natoli and Vincent Gramoli. "The Balance Attack or Why Forkable Blockchains are Ill-Suited for Consortium". In: *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017, Denver, CO, USA, June 26-29, 2017*. IEEE Computer Society, 2017, pp. 579–590. DOI: 10.1109/DSN.2017.44. URL: https://doi.org/10.1109/DSN.2017.44.

[4] Christopher Natoli and Vincent Gramoli. "The Balance Attack Against Proof-Of-Work Blockchains: The R3 Testbed as an Example". In: *CoRR* abs/1612.09426 (2016). arXiv: 1612.09426. URL: http://arxiv.org/abs/1612.09426.

[5] Gary Shapiro, Christopher Natoli, and Vincent Gramoli. "The Performance of Byzantine Fault Tolerant Blockchains". In: *19th IEEE International Symposium on Network Computing and Applications, NCA 2020, Cambridge, MA, USA, November 24-27, 2020*. IEEE, 2020, pp. 1–8. DOI: 10.1109/NCA51143.2020.9306742. URL: https://doi.org/10.1109/NCA51143.2020.9306742.

[6] Tyler Crain, Christopher Natoli, and Vincent Gramoli. "Red Belly: A Secure, Fair and Scalable Open Blockchain". In: *2021 IEEE Symposium on Security and Privacy, SP May 24-27, 2021*. IEEE Computer Society, 2021.

[7] Tyler Crain, Christopher Natoli, and Vincent Gramoli. "Evaluating the Red Belly Blockchain". In: *CoRR* abs/1812.11747 (2018). arXiv: 1812.11747. URL: http://arxiv.org/abs/1812.11747.

[8] Harold Benoit, Vincent Gramoli, Rachid Guerraoui, and Christopher Natoli. “Diablo: A Distributed Analytical Blockchain Benchmark Framework Focusing on Real-World Workloads”. In: (2021), p. 13. URL: http://infoscience.epfl.ch/record/285731.

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

[10] Leslie Lamport. “Password Authentification with Insecure Communication”. In: *Commun. ACM* 24.11 (1981), pp. 770–772. DOI: 10.1145/358790.358797. URL: http://doi.acm.org/10.1145/358790.358797.

[11] Fred B. Schneider. “Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial”. In: *ACM Comput. Surv.* 22.4 (1990), pp. 299–319. DOI: 10.1145/98163.98167. URL: https://doi.org/10.1145/98163.98167.

[12] 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. DOI: 10.1145/322186.322188. URL: http://doi.acm.org/10.1145/322186.322188.

[13] Flaviu Cristian. “Understanding Fault-Tolerant Distributed Systems”. In: *Commun. ACM* 34.2 (1991), pp. 56–78. DOI: 10.1145/102792.102801. URL: https://doi.org/10.1145/102792.102801.

[14] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. “The Byzantine Generals Problem”. In: *ACM Trans. Program. Lang. Syst.* 4.3 (1982), pp. 382–401. DOI: 10.1145/357172.357176. URL: http://doi.acm.org/10.1145/357172.357176.

[15] Miguel Castro and Barbara Liskov. “Practical byzantine fault tolerance and proactive recovery”. In: *ACM Trans. Comput. Syst.* 20.4 (2002), pp. 398–461. DOI: 10.1145/571637.571640. URL: https://doi.org/10.1145/571637.571640.

[16] Miguel Castro and Barbara Liskov. “Practical Byzantine Fault Tolerance”. In: *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*. Ed. by Margo I. Seltzer and Paul J. Leach. USENIX Association, 1999, pp. 173–186. URL: https://dl.acm.org/citation.cfm?id=296824.

[17] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. “Impossibility of Distributed Consensus with One Faulty Process”. In: *J. ACM* 32.2 (1985), pp. 374–382. DOI: 10.1145/3149.214121. URL: https://doi.org/10.1145/3149.214121.

[18] James Aspnes and Maurice Herlihy. “Fast Randomized Consensus Using Shared Memory”. In: *J. Algorithms* 11.3 (1990), pp. 441–461. DOI: 10.1016/0196-6774(90)90021-6. URL: https://doi.org/10.1016/0196-6774(90)90021-6.

[19] Michael O. Rabin. “Randomized Byzantine Generals”. In: *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*. IEEE Computer Society, 1983, pp. 403–409. DOI: 10.1109/SFCS.1983.48. URL: https://doi.org/10.1109/SFCS.1983.48.

[20] Michael Ben-Or. "Fast Asynchronous Byzantine Agreement (Extended Abstract)". In: *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing, Minaki, Ontario, Canada, August 5-7, 1985*. Ed. by Michael A. Malcolm and H. Raymond Strong. ACM, 1985, pp. 149–151. DOI: 10.1145/323596.323609. URL: https://doi.org/10.1145/323596.323609.

[21] Danny Dolev, Cynthia Dwork, and Larry J. Stockmeyer. "On the minimal synchronism needed for distributed consensus". In: *J. ACM* 34.1 (1987), pp. 77–97. DOI: 10.1145/7531.7533. URL: https://doi.org/10.1145/7531.7533.

[22] Gabriel Bracha. "Asynchronous Byzantine Agreement Protocols". In: *Inf. Comput.* 75.2 (1987), pp. 130–143. DOI: 10.1016/0890-5401(87)90054-X. URL: https://doi.org/10.1016/0890-5401(87)90054-X.

[23] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. "Consensus in the presence of partial synchrony". In: *J. ACM* 35.2 (1988), pp. 288–323. DOI: 10.1145/42282.42283. URL: http://doi.acm.org/10.1145/42282.42283.

[24] John R. Douceur. "The Sybil Attack". In: *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*. Ed. by Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron. Vol. 2429. Lecture Notes in Computer Science. Springer, 2002, pp. 251–260. DOI: 10.1007/3-540-45748-8_24. URL: https://doi.org/10.1007/3-540-45748-8_24.

[25] Cynthia Dwork and Moni Naor. "Pricing via Processing or Combatting Junk Mail". In: *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Springer, 1992, pp. 139–147. DOI: 10.1007/3-540-48071-4_10. URL: https://doi.org/10.1007/3-540-48071-4_10.

[26] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2014, pp. 459–474. DOI: 10.1109/SP.2014.36. URL: https://doi.org/10.1109/SP.2014.36.

[27] Gavin Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger Petersburg version 3e2c089 – 2020-09-05*. [Online] Available: https://ethereum.github.io/yellowpaper/paper.pdf. 2014.

[28] Christian Decker, Jochen Seidel, and Roger Wattenhofer. "Bitcoin meets strong consistency". In: *Proceedings of the 17th International Conference on Distributed Computing and Networking, Singapore, January 4-7, 2016*. ACM, 2016, 13:1–13:10. DOI: 10.1145/2833312.2833321. URL: https://doi.org/10.1145/2833312.2833321.

[29] Ghassan Karame, Elli Androulaki, and Srdjan Capkun. "Double-spending fast payments in bitcoin". In: *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*. Ed. by Ting Yu, George Danezis,

and Virgil D. Gligor. ACM, 2012, pp. 906–917. DOI: 10.1145/2382196.2382292. URL: https://doi.org/10.1145/2382196.2382292.

[30] Johannes Göbel, Holger Paul Keeler, Anthony E. Krzesinski, and Peter G. Taylor. "Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay". In: *Perform. Evaluation* 104 (2016), pp. 23–41. DOI: 10.1016/j.peva.2016.07.001. URL: https://doi.org/10.1016/j.peva.2016.07.001.

[31] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. "Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack". In: *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*. IEEE, 2016, pp. 305–320. DOI: 10.1109/EuroSP.2016.32. URL: https://doi.org/10.1109/EuroSP.2016.32.

[32] Hal Finney. *Finney's attack*. [Online] Available: https://bitcointalk.org/index.php?topic=3441.msg48384#msg48384. 2011.

[33] vector76. *The vector76 attack*. [Online] Available: https://bitcointalk.org/index.php?topic=36788.msg463391#msg463391. 2011.

[34] Meni Rosenfeld. "Analysis of Hashrate-Based Double Spending". In: *CoRR* abs/1402.2009 (2014). arXiv: 1402.2009. URL: http://arxiv.org/abs/1402.2009.

[35] 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*. Ed. by Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff. Vol. 9604. Lecture Notes in Computer Science. Springer, 2016, pp. 19–26. DOI: 10.1007/978-3-662-53357-4_2. URL: https://doi.org/10.1007/978-3-662-53357-4_2.

[36] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. "Eclipse Attacks on Bitcoin's Peer-to-Peer Network". In: *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*. Ed. by Jaeyeon Jung and Thorsten Holz. USENIX Association, 2015, pp. 129–144. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman.

[37] Monero. *About Monero*. [Online] Available: https://www.getmonero.org/resources/about/. 2014.

[38] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. "Zerocoin: Anonymous Distributed E-Cash from Bitcoin". In: *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. IEEE Computer Society, 2013, pp. 397–411. DOI: 10.1109/SP.2013.34. URL: https://doi.org/10.1109/SP.2013.34.

[39] Bitcoin Cash. *Bitcoin Cash*. [Online] Available: https://www.bitcoincash.org/.

[40] Diem Foundation. *Diem: Whitepaper*. [Online] Available: https://diem.com/en-US/white-paper/. 2020.

[41] Timo Hanke, Mahnush Movahedi, and Dominic Williams. "DFINITY Technology Overview Series, Consensus System". In: *CoRR* abs/1805.04548 (2018). arXiv: 1805.04548. URL: http://arxiv.org/abs/1805.04548.

[42] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. "Algorand: Scaling Byzantine Agreements for Cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 2017, pp. 51–68. DOI: 10.1145/3132747.3132757. URL: https://doi.org/10.1145/3132747.3132757.

[43] BusinessInsider. *FLETA Blockchain Platform Launched Public Testnet with with Peak 15,000 TPS*. [Online] Available: https://markets.businessinsider.com/news/stocks/fleta-blockchain-platform-launched-public-testnet-with-peak-15-000-tps-1028207175. 2019.

[44] unita. *QtumX Reaches 10,000 TPS in Benchmark Tests*. retrieved from https://blog.qtum.org/qtumx-reaches-10-000-tps-in-benchmark-tests-cee6452166fd. 2019.

[45] Stephen O'Neal. *Who Scales Best? Inside Blockchains' Ongoing Transactions-Per-Second Race*. retrieved from https://cointelegraph.com/news/who-scales-it-best-inside-blockchains-ongoing-transactions-per-second-race. 2019.

[46] Marco Cavicchioli. *EoS in the lead for TPS thanks to the new testnet*. retrieved from https://en.cryptonomist.ch/2020/02/12/eos-tps-testnet/. 2020.

[47] Hyperledger Foundation. *Hyperledger Caliper*. [Online] Available: https://hyperledger.github.io/caliper/. 2018.

[48] Andreas Kreuger. *ChainHammer*. [Online] Available: https://github.com/drandreaskrueger/chainhammer. 2018.

[49] Divya Gupta, Lucas Perronne, and Sara Bouchenak. "BFT-Bench: A Framework to Evaluate BFT Protocols". In: *Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering, ICPE 2016, Delft, The Netherlands, March 12-16, 2016*. Ed. by Alberto Avritzer, Alexandru Iosup, Xiaoyun Zhu, and Steffen Becker. ACM, 2016, pp. 109–112. DOI: 10.1145/2851553.2858667. URL: https://doi.org/10.1145/2851553.2858667.

[50] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. "BLOCKBENCH: A Framework for Analyzing Private Blockchains". In: *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. Ed. by Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu. ACM, 2017, pp. 1085–1100. DOI: 10.1145/3035918.3064033. URL: https://doi.org/10.1145/3035918.3064033.

[51] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. "DBFT: Efficient Leaderless Byzantine Consensus and its Application to Blockchains". In: *17th IEEE International Symposium on Network Computing and Applications, NCA 2018, Cambridge,*

MA, USA, November 1-3, 2018. IEEE, 2018, pp. 1–8. DOI: 10.1109/NCA.2018.8548057. URL: https://doi.org/10.1109/NCA.2018.8548057.

[52] EoS. *EoS.io Technical White Paper v2*. [Online] Available: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md. 2018.

[53] Serguei Popov. *The Tangle*. [Online] Available https://iota.org/IOTA_Whitepaper.pdf. 2017.

[54] Colin LeMahieu. *RaiBlocks: A Feeless Distributed Cryptcurrency Network*. [Online] Available: https://raiblocks.net/media/RaiBlocks_Whitepaper__English.pdf. 2017 (original 2014).

[55] JP Morgan Chanse. *Quorum Whitepaper*. [Online] https://github.com/ConsenSys/quorum/blob/master/docs/Quorum%20Whitepaper%20v0.2.pdf. 2018.

[56] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. "SoK: Layer-Two Blockchain Protocols". In: *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*. Ed. by Joseph Bonneau and Nadia Heninger. Vol. 12059. Lecture Notes in Computer Science. Springer, 2020, pp. 201–226. DOI: 10.1007/978-3-030-51280-4_12. URL: https://doi.org/10.1007/978-3-030-51280-4_12.

[57] Christian Cachin and Marko Vukolic. "Blockchain Consensus Protocols in the Wild". In: *CoRR* abs/1707.01873 (2017). arXiv: 1707.01873. URL: http://arxiv.org/abs/1707.01873.

[58] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. "SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies". In: *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, pp. 104–121. DOI: 10.1109/SP.2015.14. URL: https://doi.org/10.1109/SP.2015.14.

[59] Ethereum. *Ethereum Whitepaper*. [Online] Available: https://ethereum.org/en/whitepaper. 2014.

[60] Hyperledger. *An Introduction to Hyperledger*. [Online] Available: https://www.hyperledger.org/wp-content/uploads/2018/08/HL_Whitepaper_IntroductiontoHyperledger.pdf. 2018.

[61] Zachary Amsden, Ramnik Arora, Shehar Bano, Mathieu Baudet, Sam Blackshear, Abhay Bothra, Christian Catalini George Cabrera, Konstantinos Chalkias, Evan Cheng, Avery Ching, Andrey Chursin, Gerardo Di Giacomo George Danezis, David L. Dill, Hui Ding, Nick Doudchenko, Victor Gao, Zhenhuan Gao, Michael Gorven François Garillot, Philip Hayes, J. Mark Hou, Yuxuan Hu, Kevin Hurley, Kevin Lewi, Chunqi Li, Zekun Li, Sonia Margulis Dahlia Malkhi, Ben Maurer, Payman Mohassel, Ladi de Naurois, Valeria Nikolaenko, Todd Nowacki, Dmitri Perelman Oleksandr Orlov, Alistair Pott, Brett Proctor, Shaz Qadeer, Rain, Dario Russi, Bryan Schwab, Alberto Sonnino Stephane

Sezer, Herman Venter, Lei Wei, Nils Wernerfelt, Brandon Williams, Qinfan Wu, Xifan Yan, Tim Zakian, and Runtian Zhou. *The Libra Blockchain*. [Online] Available: https://developers.libra.org/docs/assets/papers/the-libra-blockchain/2020-05-26.pdf. 2020.

[62] Kirk Baird, Seongho Jeong, Yeonsoo Kim, Bernd Burgstaller, and Bernhard Scholz. "The Economics of Smart Contracts". In: *CoRR* abs/1910.11143 (2019). arXiv: 1910.11143. URL: http://arxiv.org/abs/1910.11143.

[63] Jeffrey Wilcke. *The Ethereum network is currently undergoing a DOS attack*. [Online] Available: https://blog.ethereum.org/2016/09/22/ethereum-network-currently-undergoing-dos-attack/ (Accessed: 2017-11-13). 2016.

[64] Vitalik Buterin. *On Public and Private Blockchains*. [Online] Available: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/. 2015.

[65] Marko Vukolic. "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication". In: *Open Problems in Network Security - IFIP WG 11.4 International Workshop, iNetSec 2015, Zurich, Switzerland, October 29, 2015, Revised Selected Papers*. Ed. by Jan Camenisch and Dogan Kesdogan. Vol. 9591. Lecture Notes in Computer Science. Springer, 2015, pp. 112–125. DOI: 10.1007/978-3-319-39028-4_9. URL: https://doi.org/10.1007/978-3-319-39028-4_9.

[66] *ZCash Basics*. [Online] Available: https://zcash.readthedocs.io/en/latest/rtd_pages/basics.html. 2016.

[67] Daria Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. *ZCash Protocol Specification*. [Online] Available: https://github.com/zcash/zips/blob/master/protocol/protocol.pdf. 2021.

[68] Monero, Koe, Kurt M. Alonso, and Sarang Noether. *Zero to Monero: Second Addition*. [Online] Available: https://web.getmonero.org/library/Zero-to-Monero-2-0-0.pdf. 2020.

[69] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. "Signature-free Asynchronous Byzantine Consensus with $T < N/3$ and $O(N^2)$ Messages". In: *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*. PODC '14. Paris, France: ACM, 2014, pp. 2–9. ISBN: 978-1-4503-2944-6. DOI: 10.1145/2611462.2611468. URL: http://doi.acm.org/10.1145/2611462.2611468.

[70] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. "Signature-Free Asynchronous Binary Byzantine Consensus with t < n/3, O(n2) Messages, and O(1) Expected Time". In: *J. ACM* 62.4 (2015), 31:1–31:21. DOI: 10.1145/2785953. URL: https://doi.org/10.1145/2785953.

[71] Marko Vukolic. *Quorum Systems: With Applications to Storage and Consensus*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012. DOI: 10.2200/S00402ED1V01Y201202DCT009. URL: https://doi.org/10.2200/S00402ED1V01Y201202DCT009.

[72] Jean-Philippe Martin and Lorenzo Alvisi. “Fast Byzantine Consensus”. In: *IEEE Trans. Dependable Secur. Comput.* 3.3 (2006), pp. 202–215. DOI: 10.1109/TDSC.2006.35. URL: https://doi.org/10.1109/TDSC.2006.35.

[73] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems - concepts and designs (3. ed.)* International computer science series. Addison-Wesley-Longman, 2002. ISBN: 978-0-201-61918-8.

[74] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. “Secure and Efficient Asynchronous Broadcast Protocols”. In: *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Springer, 2001, pp. 524–541. DOI: 10.1007/3-540-44647-8_31. URL: https://doi.org/10.1007/3-540-44647-8_31.

[75] Seth Gilbert and Nancy A. Lynch. “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services”. In: *SIGACT News* 33.2 (2002), pp. 51–59. DOI: 10.1145/564585.564601. URL: https://doi.org/10.1145/564585.564601.

[76] Eric A. Brewer. “Towards robust distributed systems (abstract)”. In: *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 16-19, 2000, Portland, Oregon, USA*. Ed. by Gil Neiger. ACM, 2000, p. 7. DOI: 10.1145/343477.343502. URL: https://doi.org/10.1145/343477.343502.

[77] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. “The Bitcoin Backbone Protocol: Analysis and Applications”. In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. Lecture Notes in Computer Science. Springer, 2015, pp. 281–310. DOI: 10.1007/978-3-662-46803-6_10. URL: https://doi.org/10.1007/978-3-662-46803-6_10.

[78] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. “The Bitcoin Backbone Protocol with Chains of Variable Difficulty”. In: *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. Lecture Notes in Computer Science. Springer, 2017, pp. 291–323. DOI: 10.1007/978-3-319-63688-7_10. URL: https://doi.org/10.1007/978-3-319-63688-7_10.

[79] Christian Decker and Roger Wattenhofer. “Information propagation in the Bitcoin network”. In: *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013, Trento, Italy, September 9-11, 2013, Proceedings*. IEEE, 2013, pp. 1–10. DOI: 10.1109/P2P.2013.6688704. URL: https://doi.org/10.1109/P2P.2013.6688704.

[80] Ethereum Classic. *Ethereum Classic*. [Online] Available: https://ethereumclassic.org/knowledge/what-is-etc.

[81] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. *Enabling Blockchain Innovations with Pegged Sidechains*. [Online] Availablle: https://blockstream.com/sidechains.pdf. 2014.

[82] Alberto Garoffolo and Robert Viglione. "Sidechains: Decoupled Consensus Between Chains". In: *CoRR* abs/1812.05441 (2018). arXiv: 1812.05441. URL: http://arxiv.org/abs/1812.05441.

[83] Sandra Johnson, Peter Robinson, and John Brainard. "Sidechains and interoperability". In: *CoRR* abs/1903.04077 (2019). arXiv: 1903.04077. URL: http://arxiv.org/abs/1903.04077.

[84] Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. "Proof-of-Stake Sidechains". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 139–156. DOI: 10.1109/SP.2019.00040. URL: https://doi.org/10.1109/SP.2019.00040.

[85] Aggelos Kiayias and Dionysis Zindros. "Proof-of-Work Sidechains". In: *Financial Cryptography and Data Security - FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*. Ed. by Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala. Vol. 11599. Lecture Notes in Computer Science. Springer, 2019, pp. 21–34. DOI: 10.1007/978-3-030-43725-1_3. URL: https://doi.org/10.1007/978-3-030-43725-1_3.

[86] Joseph Poon and Vitalik Buterin. *Plasma: Scalable Autonomous Smart Contracts*. [Online] Available https://plasma.io/plasma.pdf. 2017.

[87] Ethereum. *Ethereum 2.0 Specifications*. [Online] Available: https://github.com/ethereum/eth2.0-specs. 2020.

[88] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. "Parallel Chains: Improving Throughput and Latency of Blockchain Protocols via Parallel Composition". In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 1119. URL: https://eprint.iacr.org/2018/1119.

[89] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding". In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 583–598. DOI: 10.1109/SP.2018.000-5. URL: https://doi.org/10.1109/SP.2018.000-5.

[90] Richard Peebles and Eric G. Manning. "A Computer Architecture for Large (Distributed) Data Bases". In: *Proceedings of the International Conference on Very Large Data Bases, September 22-24, 1975, Framingham, Massachusetts, USA*. Ed. by Douglas S. Kerr. ACM, 1975, pp. 405–427. DOI: 10.1145/1282480.1282511. URL: https://doi.org/10.1145/1282480.1282511.

[91] David J. DeWitt, Robert H. Gerber, Goetz Graefe, Michael L. Heytens, Krishna B. Kumar, and M. Muralikrishna. "GAMMA - A High Performance Dataflow Database Machine". In: *VLDB'86 Twelfth International Conference on Very Large Data Bases, August 25-28, 1986, Kyoto, Japan, Proceedings*. Ed. by Wesley W. Chu, Georges Gardarin, Setsuo Ohsuga, and Yahiko Kambayashi. Morgan Kaufmann, 1986, pp. 228–237. URL: http://www.vldb.org/conf/1986/P228.PDF.

[92] Andrew Birrell, Roy Levin, Roger M. Needham, and Michael D. Schroeder. "Grapevine: An Exercise in Distributed Computing". In: *Commun. ACM* 25.4 (1982), pp. 260–274. DOI: 10.1145/358468.358487. URL: https://doi.org/10.1145/358468.358487.

[93] Sang Hyuk Son. "Replicated Data Management in Distributed Database Systems". In: *SIGMOD Rec.* 17.4 (1988), pp. 62–69. DOI: 10.1145/61733.61738. URL: https://doi.org/10.1145/61733.61738.

[94] Wei Dai. *bmoney*. [Online] Available: http://www.weidai.com/bmoney.txt. 1998.

[95] Adam Back. *Hashcash - A denial of service counter-measure*. Tech. rep. [Online] Available: http://www.hashcash.org/papers/hashcash.pdf. 2002.

[96] Martín Abadi, Michael Burrows, Mark S. Manasse, and Ted Wobber. "Moderately hard, memory-bound functions". In: *ACM Trans. Internet Techn.* 5.2 (2005), pp. 299–327. DOI: 10.1145/1064340.1064341. URL: https://doi.org/10.1145/1064340.1064341.

[97] Cynthia Dwork, Andrew V. Goldberg, and Moni Naor. "On Memory-Bound Functions for Fighting Spam". In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 426–444. DOI: 10.1007/978-3-540-45146-4_25. URL: https://doi.org/10.1007/978-3-540-45146-4_25.

[98] Alex Biryukov and Dmitry Khovratovich. "Equihash: Asymmetric Proof-of-Work Based on the Generalized Birthday Problem". In: *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016. URL: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/equihash-asymmetric-proof-of-work-based-generalized-birthday-problem.pdf.

[99] Jiangshan Yu, David Kozhaya, Jeremie Decouchant, and Paulo Jorge Esteves Veríssimo. "RepuCoin: Your Reputation Is Your Power". In: *IEEE Trans. Computers* 68.8 (2019), pp. 1225–1237. DOI: 10.1109/TC.2019.2900648. URL: https://doi.org/10.1109/TC.2019.2900648.

[100] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. "On the Security and Performance of Proof of Work Blockchains". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl, Stefan Katzen-

beisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM, 2016, pp. 3–16. DOI: 10.1145/2976749.2978341. URL: https://doi.org/10.1145/2976749.2978341.

[101] QuantumMechanic. *Bitcoin Forum - Proof of Stake instead of Proof of Work*. [Online] Available: https://bitcointalk.org/index.php?topic=27787.0. 2011.

[102] Sunny King and Scott Nadal. *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*. [Online] Available: https://decred.org/research/king2012.pdf. 2012.

[103] Iain Stewart. *Proof of Burn - a potential alternative to proof of work and proof of stake*. [Online] Available: https://bitcointalk.org/index.php?topic=131139.msg1404195. 2012.

[104] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. "Proof-of-Burn". In: *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*. Ed. by Joseph Bonneau and Nadia Heninger. Vol. 12059. Lecture Notes in Computer Science. Springer, 2020, pp. 523–540. DOI: 10.1007/978-3-030-51280-4_28. URL: https://doi.org/10.1007/978-3-030-51280-4_28.

[105] P4Titan. *SlimCoin: A Peer-to-Peer Crypto-Currency with Proof-of-Burn*. [Online] Available: https://slimcoin.info/whitepaperSLM.pdf. 2014.

[106] Jae Kwon. *Tendermint: Consensus without mining*. [Online] Available: http://tendermint.com/docs/tendermint_v04.pdf. 2014.

[107] Ethan Buchman. "Tendermint: Byzantine fault tolerance in the age of blockchains". PhD thesis. 2016.

[108] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol". In: *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. Lecture Notes in Computer Science. Springer, 2017, pp. 357–388. DOI: 10.1007/978-3-319-63688-7_12. URL: https://doi.org/10.1007/978-3-319-63688-7_12.

[109] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. "Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain". In: *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. Lecture Notes in Computer Science. Springer, 2018, pp. 66–98. DOI: 10.1007/978-3-319-78375-8_3. URL: https://doi.org/10.1007/978-3-319-78375-8_3.

[110] Bitshares. *Bitshares DPoS*. [Online] Available: https://bitshares.org/technology/delegated-proof-of-stake-consensus/. 2013-2017.

[111] bytemaster and bitsharestalk.org. *Transactions as Proof-of-Stake & The End of Mining*. [Online] Available: https://bitsharestalk.org/index.php?topic=1138.msg13602#msg13602A. 2013.

[112] Lisk.io. *Delegated Proof of Stake*. [Online] Available: https://lisk.io/academy/blockchain-basics/how-does-blockchain-work/delegated-proof-of-stake. 2018.

[113] Antonina Begicheva and Alexey Kofman. *Fair Block Delay Distribution In Proof-of-Stake*. [Online] Available: https://forum.waves.tech/uploads/default/original/2X/7/7397a4cb5fa77d659a7b7ecc9188dd0a4fe0decc.pdf. 2018.

[114] Ari Juels and Burton S. Kaliski Jr. "Pors: proofs of retrievability for large files". In: *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*. Ed. by Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson. ACM, 2007, pp. 584–597. DOI: 10.1145/1315245.1315317. URL: https://doi.org/10.1145/1315245.1315317.

[115] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. "Permacoin: Repurposing Bitcoin Work for Data Preservation". In: *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2014, pp. 475–490. DOI: 10.1109/SP.2014.37. URL: https://doi.org/10.1109/SP.2014.37.

[116] Sunoo Park, Albert Kwon, Georg Fuchsbauer, Peter Gazi, Joël Alwen, and Krzysztof Pietrzak. "SpaceMint: A Cryptocurrency Based on Proofs of Space". In: *Financial Cryptography and Data Security - 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26 - March 2, 2018, Revised Selected Papers*. Ed. by Sarah Meiklejohn and Kazue Sako. Vol. 10957. Lecture Notes in Computer Science. Springer, 2018, pp. 480–499. DOI: 10.1007/978-3-662-58387-6_26. URL: https://doi.org/10.1007/978-3-662-58387-6_26.

[117] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. "Proofs of Space". In: *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*. Ed. by Rosario Gennaro and Matthew Robshaw. Vol. 9216. Lecture Notes in Computer Science. Springer, 2015, pp. 585–605. DOI: 10.1007/978-3-662-48000-7_29. URL: https://doi.org/10.1007/978-3-662-48000-7_29.

[118] Yiannis Psaras and David Dias. "The InterPlanetary File System and the Filecoin Network". In: *50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020 - Supplemental Volume*. IEEE, 2020, p. 80. DOI: 10.1109/DSN-S50200.2020.00043. URL: https://doi.org/10.1109/DSN-S50200.2020.00043.

[119] Parity. *Proof-of-Authority Chains*. [Online] Available: https://openethereum.github.io/Proof-of-Authority-Chains. 2017.

[120] karalabe and Ethereum. *Clique PoA protocol & Rinkeby PoA testnet*. [Online] Available: https://eips.ethereum.org/EIPS/eip-225. 2017.

[121] Intel and HyperLedger. *Hyperledger Sawtooth PoET Documentation*. [Online] Available: https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html. 2017.

[122] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. "On Security Analysis of Proof-of-Elapsed-Time (PoET)". In: *Stabilization, Safety, and Security of Distributed Systems - 19th International Symposium, SSS 2017, Boston, MA, USA, November 5-8, 2017, Proceedings*. Ed. by Paul G. Spirakis and Philippas Tsigas. Vol. 10616. Lecture Notes in Computer Science. Springer, 2017, pp. 282–297. DOI: 10.1007/978-3-319-69084-1_19. URL: https://doi.org/10.1007/978-3-319-69084-1_19.

[123] Mitar Milutinovic, Warren He, Howard Wu, and Maxinder Kanwal. "Proof of Luck: an Efficient Blockchain Consensus Protocol". In: *Proceedings of the 1st Workshop on System Software for Trusted Execution, SysTEX@Middleware 2016, Trento, Italy, December 12, 2016*. ACM, 2016, 2:1–2:6. DOI: 10.1145/3007788.3007790. URL: https://doi.org/10.1145/3007788.3007790.

[124] Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert van Renesse. "REM: Resource-Efficient Mining for Blockchains". In: *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*. Ed. by Engin Kirda and Thomas Ristenpart. USENIX Association, 2017, pp. 1427–1444. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/zhang.

[125] Ana Reyna, Cristian Martín, Jaime Chen, Enrique Soler, and Manuel Díaz. "On blockchain and its integration with IoT. Challenges and opportunities". In: *Future Gener. Comput. Syst.* 88 (2018), pp. 173–190. DOI: 10.1016/j.future.2018.05.046. URL: https://doi.org/10.1016/j.future.2018.05.046.

[126] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. "Towards an Optimized BlockChain for IoT". In: *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation, IoTDI 2017, Pittsburgh, PA, USA, April 18-21, 2017*. Ed. by Tarek F. Abdelzaher, P. R. Kumar, Alejandro P. Buchmann, and Chenyang Lu. ACM, 2017, pp. 173–178. DOI: 10.1145/3054977.3055003. URL: https://doi.org/10.1145/3054977.3055003.

[127] Nir Kshetri. "Can Blockchain Strengthen the Internet of Things?" In: *IT Prof.* 19.4 (2017), pp. 68–72. DOI: 10.1109/MITP.2017.3051335. URL: https://doi.org/10.1109/MITP.2017.3051335.

[128] Michele Amoretti, Giacomo Brambilla, Francesco Medioli, and Francesco Zanichelli. "Blockchain-Based Proof of Location". In: *2018 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS Companion 2018, Lisbon, Por-*

tugal, July 16-20, 2018. IEEE, 2018, pp. 146–153. DOI: 10.1109/QRS-C.2018.00038. URL: https://doi.org/10.1109/QRS-C.2018.00038.

[129] Wei Wu, Erwu Liu, Xinglin Gong, and Rui Wang. "Blockchain Based Zero-Knowledge Proof of Location in IoT". In: *2020 IEEE International Conference on Communications, ICC 2020, Dublin, Ireland, June 7-11, 2020*. IEEE, 2020, pp. 1–7. DOI: 10.1109/ICC40277.2020.9149366. URL: https://doi.org/10.1109/ICC40277.2020.9149366.

[130] Zhichao Zhu and Guohong Cao. "Toward Privacy Preserving and Collusion Resistance in a Location Proof Updating System". In: *IEEE Trans. Mob. Comput.* 12.1 (2013), pp. 51–64. DOI: 10.1109/TMC.2011.237. URL: https://doi.org/10.1109/TMC.2011.237.

[131] Platin. *Platin Whitepaper*. [Online] Available: https://platin.io/assets/whitepaper/Platin_Whitepaper_v3.03.pdf. 2019.

[132] Werner Vogels. "Eventually consistent". In: *Commun. ACM* 52.1 (2009), pp. 40–44. DOI: 10.1145/1435417.1435432. URL: https://doi.org/10.1145/1435417.1435432.

[133] Rafael Pass, Lior Seeman, and Abhi Shelat. "Analysis of the Blockchain Protocol in Asynchronous Networks". In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10211. Lecture Notes in Computer Science. 2017, pp. 643–673. DOI: 10.1007/978-3-319-56614-6_22. URL: https://doi.org/10.1007/978-3-319-56614-6_22.

[134] Yonatan Sompolinsky and Aviv Zohar. "Secure High-Rate Transaction Processing in Bitcoin". In: *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*. Ed. by Rainer Böhme and Tatsuaki Okamoto. Vol. 8975. Lecture Notes in Computer Science. Springer, 2015, pp. 507–527. DOI: 10.1007/978-3-662-47854-7_32. URL: https://doi.org/10.1007/978-3-662-47854-7_32.

[135] Vitalik Buterin. *A CBC Casper Tutorial*. [Online] Available: https://vitalik.ca/general/2018/12/05/cbc_casper.html. 2018.

[136] Dahlia Malkhi and Michael K. Reiter. "Byzantine Quorum Systems". In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*. Ed. by Frank Thomson Leighton and Peter W. Shor. ACM, 1997, pp. 569–578. DOI: 10.1145/258533.258650. URL: https://doi.org/10.1145/258533.258650.

[137] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. "Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing". In: *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. Ed. by Thorsten Holz and Stefan Savage. USENIX Association, 2016, pp. 279–296. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kogias.

[138] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. "Solida: A Blockchain Protocol Based on Reconfigurable Byzantine Consensus". In: *21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal, December 18-20, 2017*. Ed. by James Aspnes, Alysson Bessani, Pascal Felber, and João Leitão. Vol. 95. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 25:1–25:19. DOI: 10.4230/LIPIcs.OPODIS.2017.25. URL: https://doi.org/10.4230/LIPIcs.OPODIS.2017.25.

[139] Michael K. Reiter. "A secure group membership protocol". In: *1994 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, May 16-18, 1994*. IEEE Computer Society, 1994, pp. 176–189. DOI: 10.1109/RISP.1994.296582. URL: https://doi.org/10.1109/RISP.1994.296582.

[140] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. "Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning". In: *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 2016, pp. 526–545. DOI: 10.1109/SP.2016.38. URL: https://doi.org/10.1109/SP.2016.38.

[141] Ethan Buchman, Jae Kwon, and Zarko Milosevic. "The latest gossip on BFT consensus". In: *CoRR* abs/1807.04938 (2018). arXiv: 1807.04938. URL: http://arxiv.org/abs/1807.04938.

[142] Yackolley Amoussou-Guenou, Antonella Del Pozzo, Maria Potop-Butucaru, and Sara Tucci Piergiovanni. "Dissecting Tendermint". In: *Networked Systems - 7th International Conference, NETYS 2019, Marrakech, Morocco, June 19-21, 2019, Revised Selected Papers*. Ed. by Mohamed Faouzi Atig and Alexander A. Schwarzmann. Vol. 11704. Lecture Notes in Computer Science. Springer, 2019, pp. 166–182. DOI: 10.1007/978-3-030-31277-0_11. URL: https://doi.org/10.1007/978-3-030-31277-0_11.

[143] Silvio Micali. "ALGORAND: The Efficient and Democratic Ledger". In: *CoRR* abs/1607.01341 (2016). arXiv: 1607.01341. URL: http://arxiv.org/abs/1607.01341.

[144] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. "The Honey Badger of BFT Protocols". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM, 2016, pp. 31–42. DOI: 10.1145/2976749.2978399. URL: https://doi.org/10.1145/2976749.2978399.

[145] Joonsang Baek and Yuliang Zheng. "Simple and efficient threshold cryptosystem from the Gap Diffie-Hellman group". In: *Proceedings of the Global Telecommunications Conference, 2003. GLOBECOM '03, San Francisco, CA, USA, 1-5 December 2003*. IEEE, 2003, pp. 1491–1495. DOI: 10.1109/GLOCOM.2003.1258486. URL: https://doi.org/10.1109/GLOCOM.2003.1258486.

[146] Pierre Tholoniat and Vincent Gramoli. "Certifying Blockchain Byzantine Fault Tolerance". In: *CoRR* abs/1909.07453 (2019). arXiv: 1909.07453. URL: http://arxiv.org/abs/1909.07453.

[147] Libra BFT Team. *State Machine Replication in the Libra Blockchain*. [Online] Available: https://developers.diem.com/papers/diem-consensus-state-machine-replication-in-the-diem-blockchain/2020-05-26.pdf. 2020.

[148] Vlad Zamfir, Nate Rush, Aditya Asganokar, and Georgios Piliouras. *Introducing the "Minimal CBC Casper" Family of Consensus*. [Online] Available: https://github.com/cbc-casper/cbc-casper-paper/blob/master/cbc-casper-paper-draft.pdf. 2018.

[149] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. "HotStuff: BFT Consensus with Linearity and Responsiveness". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*. Ed. by Peter Robinson and Faith Ellen. ACM, 2019, pp. 347–356. DOI: 10.1145/3293611.3331591. URL: https://doi.org/10.1145/3293611.3331591.

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

[151] Roberto Saltini. "IBFT Liveness Analysis". In: *IEEE International Conference on Blockchain, Blockchain 2019, Atlanta, GA, USA, July 14-17, 2019*. IEEE, 2019, pp. 245–252. DOI: 10.1109/Blockchain.2019.00039. URL: https://doi.org/10.1109/Blockchain.2019.00039.

[152] Roberto Saltini. "Correctness Analysis of IBFT". In: *CoRR* abs/1901.07160 (2019). arXiv: 1901.07160. URL: http://arxiv.org/abs/1901.07160.

[153] 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: *CoRR* abs/1909.10194 (2019). arXiv: 1909.10194. URL: http://arxiv.org/abs/1909.10194.

[154] 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 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM, 2018, pp. 931–948. DOI: 10.1145/3243734.3243853. URL: https://doi.org/10.1145/3243734.3243853.

[155] Ittai Abraham, Srinivas Devadas, Kartik Nayak, and Ling Ren. "Brief Announcement: Practical Synchronous Byzantine Consensus". In: *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*. Ed. by Andréa W. Richa. Vol. 91. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 41:1–41:4. DOI: 10.4230/LIPIcs.DISC.2017.41. URL: https://doi.org/10.4230/LIPIcs.DISC.2017.41.

[156] Ling Ren, Kartik Nayak, Ittai Abraham, and Srinivas Devadas. "Practical Synchronous Byzantine Consensus". In: *CoRR* abs/1704.02397 (2017). arXiv: 1704.02397. URL: http://arxiv.org/abs/1704.02397.

[157] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. "Scalable and Probabilistic Leaderless BFT Consensus through Metastability". In: *CoRR* abs/1906.08936 (2019). arXiv: 1906.08936. URL: http://arxiv.org/abs/1906.08936.

[158] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. "Bitcoin-NG: A Scalable Blockchain Protocol". In: *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016*. Ed. by Katerina J. Argyraki and Rebecca Isaacs. USENIX Association, 2016, pp. 45–59. URL: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eyal.

[159] Guillaume Vizier and Vincent Gramoli. "ComChain: Bridging the Gap Between Public and Consortium Blockchains". In: *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), iThings/GreenCom/CPSCom/SmartData 2018, Halifax, NS, Canada, July 30 - August 3, 2018*. IEEE, 2018, pp. 1469–1474. DOI: 10.1109/Cybermatics_2018.2018.00249. URL: https://doi.org/10.1109/Cybermatics_2018.2018.00249.

[160] Guillaume Vizier and Vincent Gramoli. "ComChain: A blockchain with Byzantine fault-tolerant reconfiguration". In: *Concurr. Comput. Pract. Exp.* 32.12 (2020). DOI: 10.1002/cpe.5494. URL: https://doi.org/10.1002/cpe.5494.

[161] Rafael Pass and Elaine Shi. "FruitChains: A Fair Blockchain". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*. Ed. by Elad Michael Schiller and Alexander A. Schwarzmann. ACM, 2017, pp. 315–324. DOI: 10.1145/3087801.3087809. URL: https://doi.org/10.1145/3087801.3087809.

[162] Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia C. Fanti, and Pramod Viswanath. "Prism: Deconstructing the Blockchain to Approach Physical Limits". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM, 2019, pp. 585–602. DOI: 10.1145/3319535.3363213. URL: https://doi.org/10.1145/3319535.3363213.

[163] Leemon Baird. *The Swirlds hashgraph consensus algorithm: Fair, fast, Byzantine fault tolerance*. [Online] Available https://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf. 2016.

[164] Rachid Guerraoui, Michal Kapalka, and Jan Vitek. "STMBench7: a benchmark for software transactional memory". In: *Proceedings of the 2007 EuroSys Conference, Lisbon, Portugal, March 21-23, 2007*. Ed. by Paulo Ferreira, Thomas R. Gross, and Luís Veiga.

ACM, 2007, pp. 315–324. DOI: 10.1145/1272996.1273029. URL: https://doi.org/10.1145/1272996.1273029.

[165] Vincent Gramoli. "More than you ever wanted to know about synchronization: synchrobench, measuring the impact of the synchronization on concurrent algorithms". In: *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2015, San Francisco, CA, USA, February 7-11, 2015*. Ed. by Albert Cohen and David Grove. ACM, 2015, pp. 1–10. DOI: 10.1145/2688500.2688501. URL: https://doi.org/10.1145/2688500.2688501.

[166] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. "Benchmarking cloud serving systems with YCSB". In: *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010*. Ed. by Joseph M. Hellerstein, Surajit Chaudhuri, and Mendel Rosenblum. ACM, 2010, pp. 143–154. DOI: 10.1145/1807128.1807152. URL: https://doi.org/10.1145/1807128.1807152.

[167] Transaction Processing Performance Council. *TPC-C an On-Line Transaction Processing Benchmark*. [Online] Available: http://tpc.org/tpcc/. 1992.

[168] Transaction Processing Performance Council. *TPC-C an On-Line Transaction Processing Benchmark*. [Online] Available: http://tpc.org/tpce/. 2007.

[169] Hyperledger Foundation. *Hyperledger Burrow*. [Online] Available: https://www.hyperledger.org/use/hyperledger-burrow. 2018.

[170] Hyperledger Foundation. *Hyperledger Besu*. [Online] Available: https://github.com/hyperledger/besu. 2019.

[171] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolic, Sharon Weed Cocco, and Jason Yellick. "Hyperledger fabric: a distributed operating system for permissioned blockchains". In: *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*. Ed. by Rui Oliveira, Pascal Felber, and Y. Charlie Hu. ACM, 2018, 30:1–30:15. DOI: 10.1145/3190508.3190538. URL: https://doi.org/10.1145/3190508.3190538.

[172] Hyperledger Foundation. *Hyperledger Fabric*. [Online] Available: https://www.hyperledger.org/use/fabric. 2017.

[173] Shehar Bano, Alberto Sonnino, Andrey Chursin, Dmitri Perelman, and Dahlia Malkhi. "Twins: White-Glove Approach for BFT Testing". In: *CoRR* abs/2004.10617 (2020). arXiv: 2004.10617. URL: https://arxiv.org/abs/2004.10617.

[174] Divya Gupta, Lucas Perronne, and Sara Bouchenak. "BFT-Bench: Towards a Practical Evaluation of Robustness and Effectiveness of BFT Protocols". In: *Distributed Applications and Interoperable Systems - 16th IFIP WG 6.1 International Conference, DAIS 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*. Ed. by Márk Jelasity and Evangelia Kalyvianaki. Vol. 9687. Lecture Notes in Computer Science. Springer, 2016, pp. 115–128. DOI: 10.1007/978-3-319-39577-7_10. URL: https://doi.org/10.1007/978-3-319-39577-7_10.

[175] Maher Alharby and Aad van Moorsel. "BlockSim: A Simulation Framework for Blockchain Systems". In: *SIGMETRICS Perform. Evaluation Rev.* 46.3 (2018), pp. 135–138. DOI: 10.1145/3308897.3308956. URL: https://doi.org/10.1145/3308897.3308956.

[176] Ittay Eyal and Emin Gün Sirer. "Majority Is Not Enough: Bitcoin Mining Is Vulnerable". In: *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*. Ed. by Nicolas Christin and Reihaneh Safavi-Naini. Vol. 8437. Lecture Notes in Computer Science. Springer, 2014, pp. 436–454. DOI: 10.1007/978-3-662-45472-5_28. URL: https://doi.org/10.1007/978-3-662-45472-5_28.

[177] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. "Optimal Selfish Mining Strategies in Bitcoin". In: *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*. Ed. by Jens Grossklags and Bart Preneel. Vol. 9603. Lecture Notes in Computer Science. Springer, 2016, pp. 515–532. DOI: 10.1007/978-3-662-54970-4_30. URL: https://doi.org/10.1007/978-3-662-54970-4_30.

[178] Samiran Bag, Sushmita Ruj, and Kouichi Sakurai. "Bitcoin Block Withholding Attack: Analysis and Mitigation". In: *IEEE Trans. Inf. Forensics Secur.* 12.8 (2017), pp. 1967–1978. DOI: 10.1109/TIFS.2016.2623588. URL: https://doi.org/10.1109/TIFS.2016.2623588.

[179] Yakov Rekhter, Tony Li, and Susan Hares. *A Border Gateway Protocol 4 (BGP-4)*. Tech. rep. 4271. RFC Editor, 2006. URL: https://www.rfc-editor.org/rfc/rfc1654.txt.

[180] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies". In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 375–392. DOI: 10.1109/SP.2017.29. URL: https://doi.org/10.1109/SP.2017.29.

[181] He Yan, Ricardo Oliveira, Kevin Burnett, Dave Matthews, Lixia Zhang, and Dan Massey. "BGPmon: A real-time, scalable, extensible monitoring system". In: *2009 Cybersecurity Applications & Technology Conference for Homeland Security*. IEEE. 2009, pp. 212–223.

[182] Maria Apostolaki, Gian Marti, Jan Müller, and Laurent Vanbever. "SABRE: Protecting Bitcoin against Routing Attacks". In: *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The

Internet Society, 2019. URL: https://www.ndss-symposium.org/ndss-paper/sabre-protecting-bitcoin-against-routing-attacks/.

[183] *FIBRE - the Fast Internet Bitcoin Relay Engine*. 2019. URL: http://bitcoinfibre.org.

[184] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. "Erlay: Efficient Transaction Relay for Bitcoin". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM, 2019, pp. 817–831. DOI: 10.1145/3319535.3354237. URL: https://doi.org/10.1145/3319535.3354237.

[185] Muoi Tran, Inho Choi, Gi Jun Moon, Anh V. Vu, and Min Suk Kang. "A Stealthier Partitioning Attack against Bitcoin Peer-to-Peer Network". In: *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 894–909. DOI: 10.1109/SP40000.2020.00027. URL: https://doi.org/10.1109/SP40000.2020.00027.

[186] Karl Wüst and Arthur Gervais. *Ethereum eclipse attacks*. [Online] Available: https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/121310/eth-49728-01.pdf. 2016.

[187] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. "Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network". In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 236. URL: http://eprint.iacr.org/2018/236.

[188] Sebastian A. Henningsen, Daniel Teunis, Martin Florian, and Björn Scheuermann. "Eclipsing Ethereum Peers with False Friends". In: *2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 2019, pp. 300–309. DOI: 10.1109/EuroSPW.2019.00040. URL: https://doi.org/10.1109/EuroSPW.2019.00040.

[189] Alex Morcos, Cory Fields, dexX7, fsb4000, Gavin Andresen, Gregory Maxwell, Ivan Pustogarov, Jonas Nick, Jonas Schnelli, Matt Corallo, mrbandrews, Pieter Wuille, Ruben de Vries, Suhas Daftuar, and Wladimir J. van der Laan. *Bitcoin Core version 0.10.1 released*. [Online] Available: https://bitcoin.org/en/release/v0.10.1. 2015.

[190] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. "Impact of Man-In-The-Middle Attacks on Ethereum". In: *37th IEEE Symposium on Reliable Distributed Systems, SRDS 2018, Salvador, Brazil, October 2-5, 2018*. IEEE Computer Society, 2018, pp. 11–20. DOI: 10.1109/SRDS.2018.00012. URL: https://doi.org/10.1109/SRDS.2018.00012.

[191] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. "The Attack of the Clones Against Proof-of-Authority". In: *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020. URL: https://www.ndss-symposium.org/ndss-paper/the-attack-of-the-clones-against-proof-of-authority/.

[192] Arthur Gervais, Hubert Ritzdorf, Ghassan O. Karame, and Srdjan Capkun. "Tampering with the Delivery of Blocks and Transactions in Bitcoin". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM, 2015, pp. 692–705. DOI: 10.1145/2810103.2813655. URL: https://doi.org/10.1145/2810103.2813655.

[193] Ethereum. *Problems: Ethereum Wiki*. [Online] Available: https://eth.wiki/en/faqs/problems. 2020.

[194] Vitalik Buterin. *Discouragement Attacks*. [Online] Available: https://github.com/ethereum/research/blob/master/papers/discouragement/discouragement.pdf. 2017.

[195] Vitalik Buterin. *The Problem of Censorship*. [Online] Available: https://blog.ethereum.org/2015/06/06/the-problem-of-censorship/. 2015.

[196] NXT. [Online] Available: https://nxtdocs.jelurida.com/Nxt_Whitepaper. 2016.

[197] Ethereum. *Proof-of-Stake FAQs*. [Online] Available: https://eth.wiki/en/concepts/proof-of-stake-faqs. 2020.

[198] Peter Gazi, Aggelos Kiayias, and Alexander Russell. "Stake-Bleeding Attacks on Proof-of-Stake Blockchains". In: *Crypto Valley Conference on Blockchain Technology, CVCBT 2018, Zug, Switzerland, June 20-22, 2018*. IEEE, 2018, pp. 85–92. DOI: 10.1109/CVCBT.2018.00015. URL: https://doi.org/10.1109/CVCBT.2018.00015.

[199] Jon Choi. *Ethereum Casper 101*. [Online] Available: https://medium.com/@jonchoi/ethereum-casper-101-7a851a4f1eb0. 2017.

[200] Vitalik Buterin. *Long Range Attacks: The Serious Problem with Adaptive Proof-of-Work*. [Online] Available: https://blog.ethereum.org/2014/05/15/long-range-attacks-the-serious-problem-with-adaptive-proof-of-work/. 2014.

[201] Sanket Kanjalkar, Joseph Kuo, Yunqi Li, and Andrew Miller. "Short Paper: I Can't Believe It's Not Stake! Resource Exhaustion Attacks on PoS". In: *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*. Ed. by Ian Goldberg and Tyler Moore. Vol. 11598. Lecture Notes in Computer Science. Springer, 2019, pp. 62–69. DOI: 10.1007/978-3-030-32101-7_4. URL: https://doi.org/10.1007/978-3-030-32101-7_4.

[202] Joachim Neu, Ertem Nusret Tas, and David Tse. "Ebb-and-Flow Protocols: A Resolution of the Availability-Finality Dilemma". In: *2021 IEEE Symposium on Security and Privacy, SP 2021, May 24-27, 2021*. IEEE, 2021.

[203] Ryuya Nakamura. *Analysis of bouncing attack on FFG*. [Online] Available: https://ethresear.ch/t/analysis-of-bouncing-attack-on-ffg/6113. 2019.

[204] Vitalik Buterin. *Bitwise LMD GHOST*. [Online] Available: https://ethresear.ch/t/bitwise-lmd-ghost/4749. 2019.

[205] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. "Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability". In: *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 910–927. DOI: 10.1109/SP40000.2020.00040. URL: https://doi.org/10.1109/SP40000.2020.00040.

[206] Yunhao Zhang, Srinath T. V. Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. "Byzantine Ordered Consensus without Byzantine Oligarchy". In: *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 2020, pp. 633–649. URL: https://www.usenix.org/conference/osdi20/presentation/zhang-yunhao.

[207] Danny Dolev and Andrew Chi-Chih Yao. "On the security of public key protocols". In: *IEEE Trans. Inf. Theory* 29.2 (1983), pp. 198–207. DOI: 10.1109/TIT.1983.1056650. URL: https://doi.org/10.1109/TIT.1983.1056650.

[208] Sergi Delgado-Segura, Surya Bakshi, Cristina Pérez-Solà, James Litton, Andrew Pachulski, Andrew Miller, and Bobby Bhattacharjee. "TxProbe: Discovering Bitcoin's Network Topology Using Orphan Transactions". In: *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*. Ed. by Ian Goldberg and Tyler Moore. Vol. 11598. Lecture Notes in Computer Science. Springer, 2019, pp. 550–566. DOI: 10.1007/978-3-030-32101-7_32. URL: https://doi.org/10.1007/978-3-030-32101-7_32.

[209] Pat Litke and Joe Stewart. *BGP Hijacking for Cryptocurrency Profit*. [Online Available: https://www.secureworks.com/research/bgp-hijacking-for-cryptocurrency-profit. 2014.

[210] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. "Hijacking Bitcoin: Large-scale Network Attacks on Cryptocurrencies". In: *CoRR* abs/1605.07524 (2016). arXiv: 1605.07524. URL: http://arxiv.org/abs/1605.07524.

[211] Muoi Tran, Akshaye Shenoi, and Min Suk Kang. "On the Routing-Aware Peering against Network-Eclipse Attacks in Bitcoin". In: *30th {USENIX} Security Symposium ({USENIX} Security 21)*. 2021.

[212] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. ISBN: 0-521-47465-5. DOI: 10.1017/cbo9780511814075. URL: https://doi.org/10.1017/cbo9780511814075.

[213] Vivek Ramachandran and Sukumar Nandi. "Detecting ARP Spoofing: An Active Technique". In: *Information Systems Security, First International Conference, ICISS 2005, Kolkata, India, December 19-21, 2005, Proceedings*. Ed. by Sushil Jajodia and Chandan Mazumdar. Vol. 3803. Lecture Notes in Computer Science. Springer, 2005, pp. 239–250. DOI: 10.1007/11593980_18. URL: https://doi.org/10.1007/11593980_18.

[214] Pete Rizzo. *R3 Publishes Vitalik Buterin Report on Ethereum for Banks*. [Online] Available: https://www.coindesk.com/r3-ethereum-report-banks/. 2016.

[215] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. "A Secure Sharding Protocol For Open Blockchains". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM, 2016, pp. 17–30. DOI: 10.1145/2976749.2978389. URL: https://doi.org/10.1145/2976749.2978389.

[216] NS-2. *NS-2 User Information - nsnam*. http://nsnam.sourceforge.net/wiki/index.php/User_Information. Accessed: 2018-03-20. 2011.

[217] Ethereum. *Geth - ethereum/go-ethereum*. [Online] Available: https://github.com/ethereum/go-ethereum/wiki/geth. 2015.

[218] Vitalik Buterin and Virgil Griffith. "Casper the Friendly Finality Gadget". In: *CoRR* abs/1710.09437 (2017). arXiv: 1710.09437. URL: http://arxiv.org/abs/1710.09437.

[219] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. "On Scaling Decentralized Blockchains - (A Position Paper)". In: *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*. Ed. by Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff. Vol. 9604. Lecture Notes in Computer Science. Springer, 2016, pp. 106–125. DOI: 10.1007/978-3-662-53357-4_8. URL: https://doi.org/10.1007/978-3-662-53357-4_8.

[220] Chrysoula Stathakopoulou, Tudor David, and Marko Vukolic. "Mir-BFT: High-Throughput BFT for Blockchains". In: *CoRR* abs/1906.05552 (2019). arXiv: 1906.05552. URL: http://arxiv.org/abs/1906.05552.

[221] Marta Lokhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafal Malinowsky, and Jed McCaleb. "Fast and secure global payments with Stellar". In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*. Ed. by Tim Brecht and Carey Williamson. ACM, 2019, pp. 80–96. DOI: 10.1145/3341301.3359636. URL: https://doi.org/10.1145/3341301.3359636.

[222] Michael J. Fischer and Nancy A. Lynch. "A Lower Bound for the Time to Assure Interactive Consistency". In: *Inf. Process. Lett.* 14.4 (1982), pp. 183–186. DOI: 10.1016/0020-0190(82)90033-3. URL: https://doi.org/10.1016/0020-0190(82)90033-3.

[223] Nathalie Bertrand, Vincent Gramoli, Igor Konnov, Marijana Lazic, Pierre Tholoniat, and Josef Widder. *Compositional Verification of Byzantine Consensus*. Tech. rep. 03158911. HAL, 2021.

[224] Benjamin Y. Chan and Elaine Shi. "Streamlet: Textbook Streamlined Blockchains". In: *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*. ACM, 2020, pp. 1–11. DOI: 10.1145/3419614.3423256. URL: https://doi.org/10.1145/3419614.3423256.

[225] Maurice Herlihy and Mark Moir. "Enhancing Accountability and Trust in Distributed Ledgers". In: *CoRR* abs/1606.07490 (2016). arXiv: 1606.07490. URL: http://arxiv.org/abs/1606.07490.

[226] Michael Spain, Sean Foley, and Vincent Gramoli. "The Impact of Ethereum Throughput and Fees on Transaction Latency During ICOs". In: *International Conference on Blockchain Economics, Security and Protocols, Tokenomics 2019, May 6-7, 2019, Paris, France*. Ed. by Vincent Danos, Maurice Herlihy, Maria Potop-Butucaru, Julien Prat, and Sara Tucci Piergiovanni. Vol. 71. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 9:1–9:15. DOI: 10.4230/OASIcs.Tokenomics.2019.9. URL: https://doi.org/10.4230/OASIcs.Tokenomics.2019.9.

[227] João Sousa, Alysson Bessani, and Marko Vukolic. "A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform". In: *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018, Luxembourg City, Luxembourg, June 25-28, 2018*. IEEE Computer Society, 2018, pp. 51–58. DOI: 10.1109/DSN.2018.00018. URL: https://doi.org/10.1109/DSN.2018.00018.

[228] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. "ZooKeeper: Wait-free Coordination for Internet-scale Systems". In: *2010 USENIX Annual Technical Conference, Boston, MA, USA, June 23-25, 2010*. Ed. by Paul Barham and Timothy Roscoe. USENIX Association, 2010. URL: https://www.usenix.org/conference/usenix-atc-10/zookeeper-wait-free-coordination-internet-scale-systems.

[229] Vincent Gramoli, Len Bass, Alan D. Fekete, and Daniel W. Sun. "Rollup: Non-Disruptive Rolling Upgrade with Fast Consensus-Based Dynamic Reconfigurations". In: *IEEE Trans. Parallel Distributed Syst.* 27.9 (2016), pp. 2711–2724. DOI: 10.1109/TPDS.2015.2499772. URL: https://doi.org/10.1109/TPDS.2015.2499772.

[230] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. "Asynchronous Secure Computations with Optimal Resilience (Extended Abstract)". In: *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, California, USA, August 14-17, 1994*. Ed. by James H. Anderson, David Peleg, and Elizabeth Borowsky. ACM, 1994, pp. 183–192. DOI: 10.1145/197917.198088. URL: https://doi.org/10.1145/197917.198088.

[231] Nuno Ferreira Neves, Miguel Correia, and Paulo Veríssimo. "Solving Vector Consensus with a Wormhole". In: *IEEE Trans. Parallel Distributed Syst.* 16.12 (2005), pp. 1120–1131. DOI: 10.1109/TPDS.2005.153. URL: https://doi.org/10.1109/TPDS.2005.153.

[232] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. "FastFabric: Scaling hyperledger fabric to 20 000 transactions per second". In: *Int. J. Netw. Manag.* 30.5 (2020). DOI: 10.1002/nem.2099. URL: https://doi.org/10.1002/nem.2099.

[233] Sisi Duan, Michael K. Reiter, and Haibin Zhang. "BEAT: Asynchronous BFT Made Practical". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM, 2018, pp. 2028–2041. DOI: 10.1145/3243734.3243812. URL: https://doi.org/10.1145/3243734.3243812.

[234] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. "Dumbo: Faster Asynchronous BFT Protocols". In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. ACM, 2020, pp. 803–818. DOI: 10.1145/3372297.3417262. URL: https://doi.org/10.1145/3372297.3417262.

[235] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. "Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited". In: *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*. Ed. by Yuval Emek and Christian Cachin. ACM, 2020, pp. 129–138. DOI: 10.1145/3382734.3405707. URL: https://doi.org/10.1145/3382734.3405707.

[236] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. "Zyzzyva: speculative byzantine fault tolerance". In: *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*. Ed. by Thomas C. Bressoud and M. Frans Kaashoek. ACM, 2007, pp. 45–58. DOI: 10.1145/1294261.1294267. URL: https://doi.org/10.1145/1294261.1294267.

[237] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. "Attested append-only memory: making adversaries stick to their word". In: *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*. Ed. by Thomas C. Bressoud and M. Frans Kaashoek. ACM, 2007, pp. 189–204. DOI: 10.1145/1294261.1294280. URL: https://doi.org/10.1145/1294261.1294280.

[238] Giuliana Veronese, Miguel Correia, Alysson Bessani, Lau Cheuk Lung, and Paulo Verissimo. *Minimal Byzantine Fault Tolerance*. Tech. rep. TR-2009-15. DI-FCUL, 2009.

[239] Yanhua Mao, Flavio P. Junqueira, and Keith Marzullo. "Towards low latency state machine replication for uncivil wide-area networks". In: *In Workshop on Hot Topics in System Dependability*. 2009.

[240] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knezevic, Vivien Quéma, and Marko Vukolic. "The Next 700 BFT Protocols". In: *ACM Trans. Comput. Syst.* 32.4 (2015), 12:1–12:45. DOI: 10.1145/2658994. URL: https://doi.org/10.1145/2658994.

[241] Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. "State Machine Replication for the Masses with BFT-SMART". In: *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*. IEEE Computer Society, 2014, pp. 355–362. DOI: 10.1109/DSN.2014.43. URL: https://doi.org/10.1109/DSN.2014.43.

[242] Allen Clement, Edmund L. Wong, Lorenzo Alvisi, Michael Dahlin, and Mirco Marchetti. "Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults". In: *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA*. Ed. by Jennifer Rexford and Emin Gün Sirer. USENIX Association, 2009, pp. 153–168. URL: http://www.usenix.org/events/nsdi09/tech/full_papers/clement/clement.pdf.

[243] Fatemeh Borran and André Schiper. "A Leader-Free Byzantine Consensus Algorithm". In: *Distributed Computing and Networking, 11th International Conference, ICDCN 2010, Kolkata, India, January 3-6, 2010. Proceedings*. Ed. by Krishna Kant, Sriram V. Pemmaraju, Krishna M. Sivalingam, and Jie Wu. Vol. 5935. Lecture Notes in Computer Science. Springer, 2010, pp. 67–78. DOI: 10.1007/978-3-642-11322-2_11. URL: https://doi.org/10.1007/978-3-642-11322-2_11.

[244] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. "RBFT: Redundant Byzantine Fault Tolerance". In: *IEEE 33rd International Conference on Distributed Computing Systems, ICDCS 2013, 8-11 July, 2013, Philadelphia, Pennsylvania, USA*. IEEE Computer Society, 2013, pp. 297–306. DOI: 10.1109/ICDCS.2013.53. URL: https://doi.org/10.1109/ICDCS.2013.53.

[245] Yackolley Amoussou-Guenou, Antonella Del Pozzo, Maria Potop-Butucaru, and Sara Tucci Piergiovanni. "Correctness of Tendermint-Core Blockchains". In: *22nd International Conference on Principles of Distributed Systems, OPODIS 2018, December 17-19, 2018, Hong Kong, China*. Ed. by Jiannong Cao, Faith Ellen, Luis Rodrigues, and Bernardo Ferreira. Vol. 125. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 16:1–16:16. DOI: 10.4230/LIPIcs.OPODIS.2018.16. URL: https://doi.org/10.4230/LIPIcs.OPODIS.2018.16.

[246] Guy Golan-Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K. Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. "SBFT: A Scalable and Decentralized Trust Infrastructure". In: *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Portland, OR, USA, June 24-27, 2019*. IEEE, 2019, pp. 568–580. DOI: 10.1109/DSN.2019.00063. URL: https://doi.org/10.1109/DSN.2019.00063.

[247] Leslie Lamport. "Brief Announcement: Leaderless Byzantine Paxos". In: *Distributed Computing - 25th International Symposium, DISC 2011, Rome, Italy, September 20-22, 2011. Proceedings*. Ed. by David Peleg. Vol. 6950. Lecture Notes in Computer Science. Springer, 2011, pp. 141–142. DOI: 10.1007/978-3-642-24100-0_10. URL: https://doi.org/10.1007/978-3-642-24100-0_10.

[248] Iulian Moraru, David G. Andersen, and Michael Kaminsky. "There is more consensus in Egalitarian parliaments". In: *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013*. Ed. by Michael Kaminsky and Mike Dahlin. ACM, 2013, pp. 358–372. DOI: 10.1145/2517349.2517350. URL: https://doi.org/10.1145/2517349.2517350.

[249] Zarko Milosevic, Martin Biely, and André Schiper. "Bounded Delay in Byzantine-Tolerant State Machine Replication". In: *IEEE 32nd Symposium on Reliable Distributed Systems, SRDS 2013, Braga, Portugal, 1-3 October 2013*. IEEE Computer Society, 2013, pp. 61–70. DOI: 10.1109/SRDS.2013.15. URL: https://doi.org/10.1109/SRDS.2013.15.

[250] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. "The Consensus Number of a Cryptocurrency". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*. Ed. by Peter Robinson and Faith Ellen. ACM, 2019, pp. 307–316. DOI: 10.1145/3293611.3331589. URL: https://doi.org/10.1145/3293611.3331589.

[251] Ittai Abraham, Guy Gueta, Dahlia Malkhi, Lorenzo Alvisi, Ramakrishna Kotla, and Jean-Philippe Martin. "Revisiting Fast Practical Byzantine Fault Tolerance". In: *CoRR* abs/1712.01367 (2017). arXiv: 1712.01367. URL: http://arxiv.org/abs/1712.01367.

[252] Pierre Sutra. "On the correctness of Egalitarian Paxos". In: *Inf. Process. Lett.* 156 (2020), p. 105901. DOI: 10.1016/j.ipl.2019.105901. URL: https://doi.org/10.1016/j.ipl.2019.105901.

[253] Musab A. Alturki, Jing Chen, Victor Luchangco, Brandon M. Moore, Karl Palmskog, Lucas Peña, and Grigore Rosu. "Towards a Verified Model of the Algorand Consensus Protocol in Coq". In: *Formal Methods. FM 2019 International Workshops - Porto, Portugal, October 7-11, 2019, Revised Selected Papers, Part I*. Ed. by Emil Sekerinski, Nelma Moreira, José N. Oliveira, Daniel Ratiu, Riccardo Guidotti, Marie Farrell, Matt Luckcuck, Diego Marmsoler, José Campos, Troy Astarte, Laure Gonnord, Antonio Cerone, Luis Couto, Brijesh Dongol, Martin Kutrib, Pedro Monteiro, and David Delmas. Vol. 12232. Lecture Notes in Computer Science. Springer, 2019, pp. 362–367. DOI: 10.1007/978-3-030-54994-7_27. URL: https://doi.org/10.1007/978-3-030-54994-7_27.

[254] Giuliano Losa and Mike Dodds. "On the Formal Verification of the Stellar Consensus Protocol". In: *2nd Workshop on Formal Methods for Blockchains, FMBC@CAV 2020, July 20-21, 2020, Los Angeles, California, USA (Virtual Conference)*. Ed. by Bruno Bernardo and Diego Marmsoler. Vol. 84. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 9:1–9:9. DOI: 10.4230/OASIcs.FMBC.2020.9. URL: https://doi.org/10.4230/OASIcs.FMBC.2020.9.

[255] Alejandro Ranchal-Pedrosa and Vincent Gramoli. "Blockchain Is Dead, Long Live Blockchain! Accountable State Machine Replication for Longlasting Blockchain". In: *CoRR* abs/2007.10541 (2020). arXiv: 2007.10541. URL: https://arxiv.org/abs/2007.10541.

[256] Gavin Wood. [Online] Available: https://polkadot.network/PolkaDotPaper.pdf. 2016.

[257] Jae Kwon and Ethan Buchman. [Online] Available: https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md. 2019.

[258] Maurice Herlihy and Jeannette M. Wing. "Linearizability: A Correctness Condition for Concurrent Objects". In: *ACM Trans. Program. Lang. Syst.* 12.3 (1990), pp. 463–492. DOI: 10.1145/78969.78972. URL: https://doi.org/10.1145/78969.78972.

[259] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. "Asymptotically Optimal Validated Asynchronous Byzantine Agreement". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*. Ed. by Peter Robinson and Faith Ellen. ACM, 2019, pp. 337–346. DOI: 10.1145/3293611.3331612. URL: https://doi.org/10.1145/3293611.3331612.

[260] Assia Doudou and André Schiper. "Muteness Detectors for Consensus with Byzantine Processes". In: *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*. Ed. by Brian A. Coan and Yehuda Afek. ACM, 1998, p. 315. DOI: 10.1145/277697.277772. URL: https://doi.org/10.1145/277697.277772.

[261] Christian Cachin, Daniel Collins, Tyler Crain, and Vincent Gramoli. "Anonymity Preserving Byzantine Vector Consensus". In: *Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part I*. Ed. by Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider. Vol. 12308. Lecture Notes in Computer Science. Springer, 2020, pp. 133–152. DOI: 10.1007/978-3-030-58951-6_7. URL: https://doi.org/10.1007/978-3-030-58951-6_7.

[262] Alysson Bessani, Eduardo Alchieri, João Sousa, André Oliveira, and Fernando Pedone. "From Byzantine Replication to Blockchain: Consensus is Only the Beginning". In: *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020*. IEEE, 2020, pp. 424–436. DOI: 10.1109/DSN48063.2020.00057. URL: https://doi.org/10.1109/DSN48063.2020.00057.

[263] Runchao Han, Gary Shapiro, Vincent Gramoli, and Xiwei Xu. "On the performance of distributed ledgers for Internet of Things". In: *Internet Things* 10 (2020), p. 100087. DOI: 10.1016/j.iot.2019.100087. URL: https://doi.org/10.1016/j.iot.2019.100087.

[264] Stanley Lima and Álvaro Rocha. "A View of OpenStack: Toward an Open-Source Solution for Cloud". In: *Recent Advances in Information Systems and Technologies - Volume 1 [WorldCIST'17, Porto Santo Island, Madeira, Portugal, April 11-13, 2017]*. Ed. by Álvaro Rocha, Ana Maria R. Correia, Hojjat Adeli, Luís Paulo Reis, and Sandra Costanzo. Vol. 569. Advances in Intelligent Systems and Computing. Springer, 2017, pp. 481–491. DOI: 10.1007/978-3-319-56535-4_49. URL: https://doi.org/10.1007/978-3-319-56535-4_49.

[265] Vitalik Buterin. *Toward a 12-second Block Time*. [Online] Available: https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/. 2014.

[266] Diego Ongaro and John K. Ousterhout. "In Search of an Understandable Consensus Algorithm". In: *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014*. Ed. by Garth Gibson and Nickolai Zeldovich. USENIX Association, 2014, pp. 305–319. URL: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro.

[267] Raffi Krikorian. *New Tweets per second record, and how!* [Online] Available: https://blog.twitter.com/engineering/en_us/a/2013/new-tweets-per-second-record-and-how.html. 2013.

[268] Christian Gorenflo, Lukasz Golab, and Srinivasan Keshav. "XOX Fabric: A hybrid approach to blockchain transaction execution". In: *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2020, Toronto, ON, Canada, May 2-6, 2020*. IEEE, 2020, pp. 1–9. DOI: 10.1109/ICBC48266.2020.9169478. URL: https://doi.org/10.1109/ICBC48266.2020.9169478.

[269] Parth Thakkar and Senthil Nathan. "Scaling Hyperledger Fabric Using Pipelined Execution and Sparse Peers". In: *CoRR* abs/2003.05113 (2020). arXiv: 2003.05113. URL: https://arxiv.org/abs/2003.05113.

[270] Hyperledger Foundation and Honeywell. *Case Study:Honeywell Aerospace creates online parts marketplace with Hyperledger Fabric*. [Online] Available: https://www.hyperledger.org/learn/publications/honeywell-case-study. 2020.

[271] Song-Hee Kim and Ward Whitt. "Choosing arrival process models for service systems: Tests of a nonhomogeneous Poisson process". In: *Naval Research Logistics (NRL)* 61.1 (2014), pp. 66–90.

[272] Intel. *Intel Software Guard Extensions*. [Online] Available: https://software.intel.com/en-us/sgx. 2014.

[273] AMD. *AMD Secure Technology*. [Online] Available: https://www.amd.com/en/technologies/security. 2013.

[274] ARM. *Building a Secure System using TrustZone Technology*. [Online] Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf. 2009.

Appendices

Appendix A

Blockchain Deconstruction

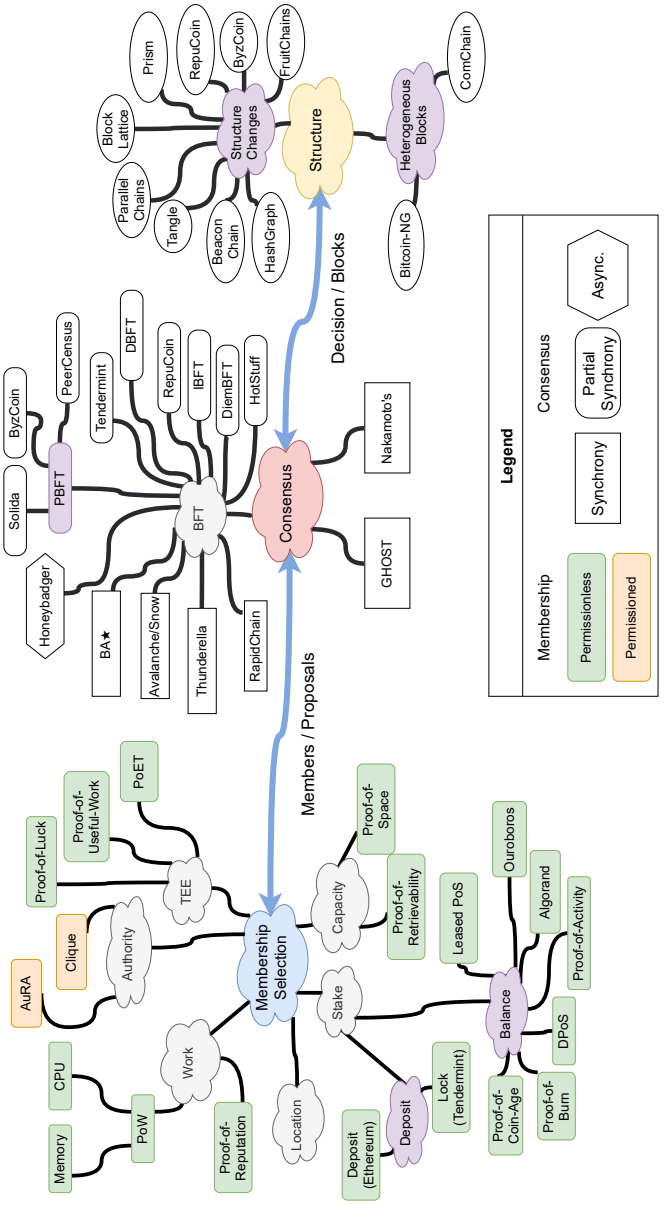


Figure A.1: Overview of the blockchain deconstruction and component interaction.

Appendix B

Balance Attack: Analysis for the General Case of κ

As we show below, if the attacker can delay communications for long enough between κ subgraphs of the communication graph where $\kappa \geq 2$ then the attacker does not need a large portion ρ of the mining power for the blockchain system to be block oblivious. We consider an attacker that can delay messages of the communication graph G between κ subgraphs $G_1, ..., G_k$, each graph owning the same portion $\frac{1-\rho}{\kappa}$ of the total mining power.

Similarly to Section 3.4, we consider that for $0 < i \leq \kappa$, each of the subgraph G_i mine blocks during delay τ. During that time, each G_i performs a series of $n = \frac{1-\rho}{\kappa}\Psi\tau$ independent and identically distributed Bernoulli trials that returns one in case of success with probability $p = \frac{1}{d}$ and 0 otherwise. Let the sum of these outcomes be the random variable X_i $(0 < i \leq \kappa)$ with a binomial distribution and mean:

$$\mu_c = np = \frac{(1-\rho)\Psi\tau}{\kappa d}. \tag{B.1}$$

Similarly, the mean of the number of blocks mined by the malicious miner during time τ is

$$\mu_m = \frac{\rho\Psi\tau}{d}. \tag{B.2}$$

We are interested in measuring the probability that an attacker can select a candidate blockchain among the existing ones and make it adopted by the system. Note that this is easier than rewriting the history with the attacker personal blocks but is sufficient to double-spend by simply making sure the initial transaction that spend the coins is only part of blockchains candidate that will be discarded.

The attack relies on minimizing the difference in mined blocks between any pair of communication subgraphs. Hence, let us denote Δ the difference of the number of blocks mined on any two subgraphs:

$$\Delta = \max_{\forall 0<i,j\leq\kappa} (|X_i - X_j|).$$

To this end, we bound the difference between the number of blocks mined in two subgraphs so that the attacker can mine more than the difference.

Fact 12 (Bernoulli's inequality)**.** $1 + n\Psi \leq (1 + \Psi)^n$ *for* $n \geq 1$ *and* $\Psi \geq -1$.

Theorem 13. *An attacker decomposing the communication graph into k subgraphs upper-bounds their difference* Δ *in mined blocks by* $2\delta\mu_c$ *with probability* $\Pr[\Delta < 2\delta\mu_c] > 1 - 2\kappa e^{-\frac{\delta^2}{3}\mu_c}$ *where* $\delta > 0$.

Proof. This difference is less than $2\delta\mu_c$ with probability larger than the probability that all random variables are within a $\pm\delta$ multiplying factor from the mean, we have:

$$\Pr[\Delta < 2\delta\mu_c] \geq \Pi_{i=1}^{\kappa} \Pr\left[|X_i - \mu_c| < \delta\mu_c\right]. \tag{B.3}$$

The probability that the numbers of blocks mined by each subgraph are within a $\pm\delta$ factor from their mean, is bound for $0 < \delta < 1$ and $0 < i \leq \kappa$ by Chernoff bounds [212]:

$$\begin{cases} \Pr\left[X_i \geq (1+\delta)\mu_c\right] & \leq e^{-\frac{\delta^2}{3}\mu_c}, \\ \Pr\left[X_i \leq (1-\delta)\mu_c\right] & \leq e^{-\frac{\delta^2}{2}\mu_c}. \end{cases}$$

Thus, we have

$$\begin{aligned} \Pr[|X_i - \mu_c| \geq \delta\mu_c] &\leq 2e^{-\frac{\delta^2}{3}\mu_c}, \\ \Pr[|X_i - \mu_c| < \delta\mu_c] &> 1 - 2e^{-\frac{\delta^2}{3}\mu_c}. \end{aligned}$$

and with Eq. B.3 we obtain:

$$\Pr[\Delta < 2\delta\mu_c] > \left(1 - 2e^{-\frac{\delta^2}{3}\mu_c}\right)^{\kappa}. \tag{B.4}$$

As $\mu_c \geq 0$, we have that:

$$\begin{aligned} -\frac{\delta^2}{3}\mu_c &\leq 0 \\ -e^{\frac{-\delta^2}{3}\mu_c} &\geq -1 \end{aligned}$$

and as $k \geq 1$ we can apply the Bernoulli inequality to Eq. B.4.

Hence, Eq. B.4 becomes:

$$\Pr[\Delta < 2\delta\mu_c] > 1 - 2\kappa e^{-\frac{\delta^2}{3}\mu_c}. \tag{B.5}$$

□

The next theorem bounds the number of blocks the attacker needs to mine to force the system to adopt the candidate blockchain of its choice.

Theorem 14. *If the attacker delays the links of* E_0 *while the miners on each of the k subgraphs mine for*

$$\tau \geq \frac{3kd\log(\frac{2\kappa}{\varepsilon})}{\delta^2(1-\rho)\Psi}$$

seconds, then with probability $1 - \varepsilon$ *the difference between the number of blocks mined on the two subgraphs is lower than* $\frac{2\delta(1-\rho)\Psi\tau}{\kappa d}$.

Proof. The proof relies on upper bounding $2\kappa e^{-\frac{\delta^2}{3}\mu_c}$ by ε:

$$\begin{aligned} 2\kappa e^{-\frac{\delta^2}{3}\mu_c} &\leq \varepsilon, \\ \mu_c &\geq \frac{3\log(\frac{2\kappa}{\varepsilon})}{\delta^2}. \end{aligned}$$

By replacing μ_c by the expression of Eq. B.1, we obtain:

$$\tau \geq \frac{3\kappa d\log(\frac{2\kappa}{\varepsilon})}{\delta^2(1-\rho)\Psi}.$$

□

Let us now bound the probability that the number of blocks mined on k subgraphs is always lower than the expectation of the number of blocks mined by the malicious node.

Theorem 15. *If* $\delta = \frac{\mu_m - 1}{2\mu_c}$ *then*

$$\Pr[\Delta < \mu_m] > 1 - 2\kappa e^{-\frac{(\mu_m-1)^2}{12\mu_c}}.$$

Proof. The proof follows from replacing δ by $\frac{\mu_m-1}{2\mu_c}$ in Eq. B.5. □

Appendix C

Red Belly Blockchain Heatmap

	Tokyo	Seoul	Mum.	Singa.	Syd.	Cana.	Frank.	Ireland	Lond.	São P.	N.Virg	Ohio	N.Cal.	Oregon
Tokyo	0	551	129	240	161	106	74	66.4	59	55.4	90.1	96.2	129	132
Seoul	33	0	137	157	141	91.5	54	60.8	54.7	56.6	84.2	114	84.2	116
Mumbai	133	164	0	121	67	90.9	176	178	145	46.7	81.9	80.5	69.1	64.2
Singapore	69	100	67	0	90.9	83.2	90.7	86.1	90.4	40.8	59.5	64.9	80.5	77.3
Sydney	106	135	235	170	0	77.1	61.3	53.8	51.2	40.2	74.9	99.7	135	119
Canada	166	185	196	220	225	0	166	250	164	159	808	760	205	168
Frankfurt	244	275	112	178	292	102	0	477	823	92.9	222	220	144	85.7
Ireland	226	246	122	188	286	78	25	0	829	114	185	183	104	117
London	255	284	111	179	281	90	15	12	0	107	190	195	107	85.5
São Paulo	271	293	302	328	332	125	210	184	192	0	131	124	77.7	81.7
N. Virginia	162	209	182	238	205	15	89	85	76	122	0	827	232	186
Ohio	169	199	193	227	196	25	99	91	87	131	13	0	428	219
N. California	120	150	262	178	148	76	148	142	138	182	64	52	0	681
Oregon	105	135	235	163	162	66	164	141	158	183	76	71	22	0

Table C.1: Heatmap of the bandwidth (Mbps) in the top right triangle and latency (ms) in the bottom left triangle between 14 AWS regions

Appendix D

Red Belly Blockchain: Proofs

D.1 Proofs of Correctness

D.1.1 Proofs of a Scalable Censorship-Resistant RSM

In this section, we show that RBBC is a censorship-resistant Replicated State Machine (RSM) that totally orders correct transactions and we give bounds on its theoretical throughput.

Proof that RBBC implements an RSM

We show that our Verified Reliable Broadcast ensures the properties of the Reliable Broadcast for valid values and discards invalid values, then we prove that our consensus protocol solves the Set Byzantine Consensus problem before showing that RBBC implements an RSM.

Theorem 16. *The Verified Reliable Broadcast (lines 1–13) ensures the properties of the Reliable Broadcast for all valid values and does not deliver an invalid value at any correct proposer.*

Proof. We proceed by showing the properties of Reliable Broadcast for a valid value: if a valid value is delivered then it was broadcast (validity), a correct proposer delivers at most one value from any given proposer (unicity), if a correct proposer broadcasts a valid value v then v is delivered at all correct proposers (termination1) and if a correct proposer delivers a valid value v, then all correct proposers deliver v (termination2). It is easy to ensure validity and unicity, so let us focus on termination1 and termination2. Consider that a valid value v is broadcast by some proposer p_i. There are two cases to consider, either p_i is correct or Byzantine.

1. **Proposer p_i is correct.** Proposer p_i broadcasts INIT to all proposers, hence each proposer broadcasts ECHO to all, and all correct proposers eventually receive ECHO messages with v from $n - f$ distinct correct proposers. These correct proposers, say Q, are thus ready to start verifying value v. As there are $f + 1$ primary verifiers and t secondary verifiers, there are up to $2f + 1$ verifiers, say Q', that will verify value v if not stopped at line 10. If $f + 1$ verifiers broadcast READYmessages with the same verification outcome *verif*, then we know that all correct proposers will then retransmit READYwith this *verif*, which will guarantee that all correct proposers will receive $n - f$ messages $\langle \text{READY}, verif, h(v), p_i \rangle$ and will thus deliver v (line 13). It thus remains to show that $f + 1$ verifiers will broadcast

READYmessages with the same verification outcome *verif*. Note that $|Q \cap Q'| \geq f + 1$, which means that among the correct proposers Q, $f + 1$ of them verify v and obtain the same verification outcome *verif* that they broadcast.

2. **Proposer p_i is Byzantine.** First, if proposer p_i broadcasts INIT successfully to all $n - f$ correct proposers, in which case, they all broadcast ECHO with value v to all proposers and the case is identical to case (1), where all correct proposers deliver v (line 13). In the case where proposer p_i does not broadcasts INIT to $f+1$ correct proposers because it broadcasts to less proposers, then not enough proposers will receive INIT for ECHO to be received by sufficiently many proposers at line 5 and v will not be delivered. Third, if proposer p_i broadcasts INIT to $f + 1 \leq \ell < n - f$ proposers, then it depends on the behaviors of the other Byzantine proposers, if sufficiently many of them send ECHO messages to $f + 1$ verifiers or if they help verifying correctly, then v will be delivered at all correct proposers, otherwise, it will not be delivered at any correct proposer.

To show that no invalid values can be delivered at any correct proposer, consider that v is invalid so there cannot be $f + 1$ distinct verifiers whose *verif* is identifying v as valid. As a result, if line 9 is enabled with $f + 1$ identical messages $\langle \text{READY}, verif, h(v), p_i \rangle$ from distinct proposers then we know that *verif* is necessarily identifying v as invalid and the precondition to deliver v at line 13 will not be satisfied. □

Lemma 17. *In the Byzantine Binary Consensus (lines 31–48), if at the beginning of a round r, all correct proposers have the same estimate val, they never change their estimate value thereafter.*

Proof. Let us assume that all correct processes (which are at least $n - f > f + 1$) have the same estimate *val* when they start round r. Hence, they all bv-broadcast the same message EST(val) either at line 15 or within the Reliable Broadcast at line 32. It follows from the properties of the Reliable Broadcast [22] and bv-broadcast [70] that each correct process p_i is such that $cvals_i = \{v\}$ at line 38, and consequently can broadcast only ECHO($\{v\}$) at line 39. Considering any correct process p_i, it then follows from the predicate of line 42 (s_i contains only v), the predicate of line 43 (s_i is a singleton), and the assignment of line 45, that val_i keeps the value v. □

The next lemma states that if the value s in the same round of two correct replicas are singletons then they are identical.

Lemma 18. *Let p_i and p_j be two correct proposers. In the Byzantine Binary Consensus (lines 31–48), if $s_i = \{v\}$ and $s_j = \{z\}$ in the same round, then $v = w$.*

Proof. Let p_i be a correct proposer such that $s_i = \{v\}$. It follows from line 42 that p_i received the same message AUX($\{v\}$) from $(n - f)$ different processes, i.e., from at least $(n - 2f)$ different correct processes. As $n - 2f \geq f + 1$, this means that p_i received the message AUX($\{v\}$) from a set Q_i including at least $(f + 1)$ different correct proposers.

Let p_j be a correct proposer such that $s_j = \{z\}$. Hence, p_j received AUX($\{z\}$) from a set Q_j of at least $(n - f)$ different proposers. As $(n - f) + (f + 1) > n$, it follows that $Q_i \cap Q_j \neq \varnothing$. Let $p_k \in Q_i \cap Q_j$. As $p_k \in Q_i$, it is a correct proposer. Hence, at line 39, p_k sent the same AUX message to p_i and p_j, and we consequently have $v = z$. □

Lemma 19 (Binary Consensus Validity)**.** *In the Byzantine Binary Consensus (lines 31–48), the value decided by a correct proposer was proposed by a correct proposer.*

Proof. Let us consider the round $r = 1$. Due to the property of the bv-broadcast executed at line 15 or piggybacked within the Reliable Broadcast at line 32, it follows that the sets $cvals_i$ contains only values proposed by correct proposers. Consequently, the correct proposers broadcast, at line 39 AUX messages containing sets with values proposed only by correct proposers. It then follows from the predicate of line 42 ($s_i \subseteq cvals_i$), and the reliable and bv-broadcast properties, that the set s_i of each correct proposer contains only values proposed by correct proposers. Hence, the assignment of val_i (be it at line 44 or 46) provides it with a value proposed by a correct proposer. The same reasoning applies to rounds $r = 2$, $r = 3$, etc., which concludes the proof of the lemma. □

Lemma 20 (Binary Consensus Agreement)**.** *In the Byzantine Binary Consensus (lines 31–48), no two correct replicas decide different values.*

Proof. Let r be the first round during which a correct proposer decides, let p_i be a correct proposer that decides in round r (line 45), and let v be the value it decides. Hence, we have $s_i^r = \{v\}$ where $v = (r \bmod 2)$.

If another correct replica p_j decides during round r, we have $s_j^r = \{z\}$, and, due to Lemma 18, we have $z = v$. Hence, all correct proposers that decide in round r, decide v. Moreover, each correct proposer that decides in round r has previously assigned $v = (r \bmod 2)$ to its local estimate s_i.

Let p_j be a correct proposer that does not decide in round r. As $s_i^r = \{v\}$, and p_j does not decide in round r, it follows from Lemma 18 that we cannot have $s_j^r = \{1-v\}$, and consequently $s_j^r = \{0, 1\}$. Hence, in round r, p_j executes line 44, where it assigns the value $(r \bmod 2) = v$ to its local estimate val_j.

It follows that all correct proposers start round $(r + 1)$ with the same local estimate $v = r \bmod 2$. Due to Lemma 17, they keep this estimate value forever. Hence, no different value can be decided in a future round by a correct proposer that has not decided during round r, which concludes the proof of the lemma. □

Lemma 21. *During the RBBC consensus (lines 14–23) execution, at least one binary consensus instance decides* 1*.*

Proof. By Theorem 16, we know that all correct proposers eventually populate their proposal array with at least one common value. Due to the reduction, all correct proposers will thus have input 1 for the corresponding binary consensus instance. By the validity (Lemma 19) and termination [51] properties of the binary consensus, the decided value for this binary consensus instance has to be 1. □

Lemma 22. *If a Byzantine Binary Consensus instance at index i decides 1, then the Verified Reliable Broadcast (lines 1–13) at index i reliably delivers a value at a correct proposer.*

Proof. A correct proposer does not propose 1 to a binary consensus instance at index i without reliably delivering a proposal at index i. The result follows from Theorem 16 and the validity of the binary consensus (Lemma 19). □

Theorem 23 (Set Byzantine Consensus)**.** *The RBBC consensus (lines 14–23) solves the Set Byzantine Consensus.*

Proof. By Lemma 21, at one index, a binary consensus instance terminates with 1. By Lemma 20, we know that all correct proposers have set 1 to the same indices of their *bitmask*. For each of these indices k there is a proposal $props[k]$ that will be delivered at every correct proposer by Lemma 22. As a result, all correct proposers invoke function reconciliate with the same argument at line 23. By examination of the code at lines 24–29, all correct proposers thus put in their *superblock* the same subset of valid and non-conflicting transactions hence guaranteeing all properties of the SBC problem (Section 3.1). □