Securing the blockchain against IoT cyber attacks

Kadir Korkmaz

THÈSE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR
DE L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE MATHÉMATIQUES ET

INFORMATIQUE

Informatique

Par **Kadir KORKMAZ**

Protection de la Blockchain IoT Face aux Cyber Attaques

Sous la direction de : **Laurent RÉVEILLÈRE**

Soutenue le 24 Mars 2023

Membres du jury :

M. Laurent Réveillère	Professeur	Université de Bordeaux	Directeur de Thèse
M. Serge Chaumette	Professeur	Université de Bordeaux	Examinateur
M. Etienne Rivière	Professeur	Université catholique de Louvain	Rapporteur
M. François Taiani	Professeur	Université de Rennes 1	Rapporteur
Mme. Sonia Ben Mokhtar	Directrice de Recherche	LIRIS	Invitée
M. Joachim Bruneau-Queyreix	Assistant Professor	Université de Bordeaux	Invité

Protection de la Blockchain IoT Face aux Cyber Attaques

Résumé : La première blockchain, le bitcoin, a été proposée en 2008 comme un mécanisme permettant de mettre en œuvre une monnaie numérique sans parties de confiance. Avant le protocole Bitcoin, les monnaies numériques étaient mises en œuvre en utilisant des intermédiaires de confiance comme les banques. Malheureusement, en raison de choix de conception, le protocole Bitcoin présente plusieurs problèmes de performance inhérents : un faible débit et une latence élevée sont parmi les plus importants. Les chercheurs ont conçu des protocoles de blockchain pour surmonter les problèmes inhérents au protocole Bitcoin, mais aujourd'hui, le plus grand obstacle à l'adoption mondiale de la technologie blockchain est la mauvaise performance.

Dans cette thèse, nous visons à améliorer les performances des blockchains existantes en éliminant les goulots d'étranglement courants dans la conception des systèmes. À cette fin, nous étudions les goulots d'étranglement courants dans la conception des systèmes de blockchains existants et proposons des mécanismes pour les atténuer.

Notre contribution est double : tout d'abord, nous avons conçu ALDER, une construction générique pour améliorer les performances des blockchains basées sur les leaders. ALDER étend l'algorithme de consensus des blockchains existantes basées sur les leaders pour enrichir avec plusieurs leaders coopérant pour traiter les demandes des utilisateurs. La conception d'ALDER augmente le débit et la latence des blockchains existantes en éliminant les goulots d'étranglement au niveau du consensus et du réseau. Nous fournissons une évaluation détaillée d'ALDER à travers des expériences à grande échelle. Deuxièmement, nous avons fait une analyse approfondie du protocole IDA-Gossip : un protocole de diffusion de ragots basé sur des morceaux. IDA-Gossip est une primitive essentielle pour les blockchains afin d'éliminer le goulot d'étranglement provenant du mécanisme de diffusion des rumeurs de type store-and-forward, mais ses propriétés n'ont pas été étudiées en profondeur. Notre travail quantifie la performance d'IDA-Gossip avec différents paramètres de protocole sous fautes en utilisant des expériences et des simulations. Notre enquête vise à aider les chercheurs à comprendre les propriétés d'IDA-Gossip en révélant son comportement dans différentes conditions.

Keywords: Systèmes Distribués, Blockchain, Consensus, Réplication, Diffusion de Rumeurs

Securing Blockchains Against IoT Cyberattacks

Abstract: The first blockchain, Bitcoin, was proposed in 2008 as a mechanism to implement a digital currency without trusted parties. Before the Bitcoin protocol, digital currencies were implemented using trusted intermediaries like banks. Unfortunately, due to design choices, the Bitcoin protocol has several inherent performance problems: low throughput and high latency are among the most important. Researchers have been designing blockchain protocols to overcome the inherent problems of the Bitcoin protocol, but today the biggest obstacle to the global adoption of blockchain technology is poor performance.

In this thesis, we aim to improve the performance of existing blockchains by removing common bottlenecks in system designs. To this end, we investigate common bottlenecks in the system design of existing blockchains and propose mechanisms to mitigate them.

Our contribution is twofold: first, we designed ALDER, a generic construction to improve the performance of leader-based blockchains. ALDER extends the consensus algorithm of existing leader-based blockchains to enrich with multiple leaders cooperating to process user requests. The design of ALDER increases the throughput and latency of existing blockchains by eliminating bottlenecks at the consensus and network levels. We provide a detailed evaluation of ALDER through large-scale experiments. Second, we made an in-depth analysis of the IDA-Gossip protocol: a chunk-based gossip dissemination protocol. IDA-Gossip is an essential primitive for blockchains to remove the bottleneck stem from the store-and-forward gossip dissemination mechanism, but its properties have not been studied extensively. Our work quantifies the performance of IDA-Gossip with different protocol parameters under faults using experiments and simulations. Our investigation aims to help researchers understand the properties of IDA-Gossip by revealing its behavior under different conditions.

Keywords: Distributed Systems, Blockchain, Consensus, Replication, Gossip Dissemination

---

Unité de recherche
UMR 5800 Université, 33000 Bordeaux, France.

Contents

List of Figures v

List of Tables vii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Research challenges . . . 4
- 1.3 Research goals . . . 5
- 1.4 Roadmap of the thesis . . . 6

2 Background 7
- 2.1 Background on distributed systems . . . 7
  - 2.1.1 System models and assumptions . . . 8
  - 2.1.2 Cryptographic primitives for distributed systems . . . 9
  - 2.1.3 Communication in distributed systems . . . 13
  - 2.1.4 Consensus . . . 16
  - 2.1.5 Replicated state machines . . . 19
  - 2.1.6 Challenges of open distributed systems . . . 20
- 2.2 Background on blockchains . . . 21
  - 2.2.1 Bitcoin . . . 21
    - 2.2.1.1 System model . . . 22
    - 2.2.1.2 Value transfer . . . 22
    - 2.2.1.3 Blockchain and Proof-of-Work . . . 22
    - 2.2.1.4 Transaction model . . . 26
    - 2.2.1.5 Observations about the Bitcoin protocol . . . 28
    - 2.2.1.6 Problems of Bitcoin approach . . . 28
  - 2.2.2 Family of blockchains . . . 30
    - 2.2.2.1 Proof of Work based blockchains . . . 31
    - 2.2.2.2 Proof of Stake based blockchains . . . 33
    - 2.2.2.3 Committee based blockchains . . . 33
    - 2.2.2.4 Proof of X based blockchains . . . 35

2.2.2.5 Sharded blockchains . . . 37
2.2.2.6 Permissioned and permissionless blockchains . . 38
2.2.2.7 Leader-based and leaderless blockchains 39
2.3 Conclusions . . . 40

3 Related work **41**
3.1 Multiple leader approach to improve the performance . . . 41
3.1.1 Classical consensus algorithms with multiple leaders . . . 41
3.1.2 Blockchains with multiple leaders . . . 43
3.2 Efficient dissemination techniques to improve performance . . . 44
3.2.1 General purpose efficient dissemination techniques 44
3.2.2 Efficient dissemination techniques for blockchains 46
3.3 Conclusions . . . 48

4 Alder **49**
4.1 Introduction . . . 49
4.2 ALDER's foundations . . . 50
4.2.1 Bottlenecks in improving blockchain performances 50
4.2.2 ALDER: multiplexed blockchain consensus 51
4.3 Design of ALDER . . . 52
4.3.1 Transaction hash space partitioning . . . 53
4.3.2 Multiple leader election and bucket assignment 54
4.3.3 Multiplexed consensus and macroblocks 55
4.3.4 ALDER Security Analysis . . . 57
4.4 Case studies . . . 59
4.4.1 Applying ALDER to Bitcoin . . . 59
4.4.1.1 Overview of Bitcoin . . . 59
4.4.1.2 Bottlenecks of Bitcoin . . . 60
4.4.1.3 Bitcoin++ . . . 60
4.4.2 Applying ALDER to Algorand . . . 63
4.4.2.1 Overview of Algorand . . . 63
4.4.2.2 Bottlenecks of Algorand . . . 64
4.4.2.3 Algorand++ . . . 65
4.4.3 Applying ALDER to Rapidchain . . . 66
4.4.3.1 Overview of Rapidchain . . . 66
4.4.3.2 Bottlenecks of Rapidchain . . . 68
4.4.3.3 Rapidchain++ . . . 68
4.5 Evaluation . . . 69
4.5.1 Implementation and evaluation environment 69

4.5.2 Bitcoin++ performance evaluation . . . 70
4.5.2.1 Bitcoin++ results . . . 70
4.5.3 Algorand++ performance evaluation . . . 71
4.5.3.1 Algorand++ latency and throughput . . . 73
4.5.3.2 Algorand++ evaluation at scale . . . 74
4.5.4 Rapidchain++ performance evaluation . . . 76
4.5.4.1 Rapidchain++ results . . . 76
4.6 Conclusion . . . 78

5 In-depth analysis of the IDA-Gossip 79
5.1 Introduction . . . 79
5.2 The *IDA-Gossip* Protocol . . . 80
5.2.1 System model and assumptions . . . 81
5.2.2 The details of *IDA-Gossip* protocol . . . 81
5.2.3 On the adoption of *IDA-Gossip* . . . 82
5.3 Evaluation with Experiments . . . 84
5.3.1 Methodology . . . 84
5.3.1.1 Evaluation environment and implementation . . . 84
5.3.1.2 Experimental protocol . . . 85
5.3.1.3 Measured Metrics . . . 86
5.3.2 *IDA-Gossip* vs classic gossip dissemination . . . 86
5.3.3 *IDA-Gossip* with different chunk counts and message sizes 89
5.3.4 *IDA-Gossip* with Faults . . . 91
5.3.5 *IDA-Gossip* with different dissemination concurrency values 91
5.4 Evaluation with simulations . . . 95
5.4.1 Methodology . . . 95
5.4.1.1 Simulations . . . 95
5.4.1.2 Simulated strategies . . . 95
5.4.1.3 Measured metrics . . . 96
5.4.2 *IDA-Gossip* vs classic multi-chunk gossip dissemination . 96
5.5 Conclusion . . . 98

6 Conclusions and future work 99
6.1 Conclusions . . . 99
6.2 Future work . . . 101

A Publications and Awards 103
A.1 Published papers in international peer reviewed conferences . . . 103
A.2 Awards . . . 103

Bibliography **104**

List of Figures

2.1 Example outputs of SHA-256 hash function. 10
2.2 The construction of a Merkle tree. 12
2.3 Communication scheme of reliable broadcast 14
2.4 Communication scheme of gossip dissemination 15
2.5 Communication scheme of the Paxos protocol 17
2.6 Communication scheme of the PBFT protocol 18
2.7 Non-replicated vs replicated service 20
2.8 Digital token transfer in Bitcoin (Nakamoto, 2008) 23
2.9 The block structure of Bitcoin, and its transaction Merkle tree. . 24
2.10 The Bitcoin mining reward and the total Bitcoin amount by years. 25
2.11 A chain of blocks with forks 26
2.12 The structure of a Bitcoin transaction. 27
2.13 Bitcoin mining time CDF 29

4.1 Multiplexing a blockchain *BCP* using ALDER 53
4.2 ALDER's transaction buckets 54
4.3 ALDER's blockchain structure 57
4.4 The bottleneck of the Bitcoin protocol 60
4.5 The round structure of Bitcoin and Bitcoin++ protocols 61
4.6 The round structure of Algorand and Algorand++ protocols . . 64
4.7 The bottleneck of the Algorand protocol 65
4.8 The round structure of Rapidchain and Rapidchain++ protocols 67
4.9 The mining power utilization of Bitcoin and Bitcoin++ protocols 71
4.10 The performance comparison of Bitcoin and Bitcoin++ protocols 72
4.11 The round latency and throughput of Algorand++ with various concurrency levels 73
4.12 Throughput and latency degradation of Algorand and Algorand++ protocols at scale 75
4.13 Latency and throughput figures of Rapidchain and Rapidchain++ protocols with various concurrency levels and macroblock sizes . 77

5.1 The two phases of the *IDA-Gossip* protocol 83

5.2 Classic gossip dissemination compared to *IDA-Gossip* 88
5.3 The behavior of *IDA-Gossip* with different chunk counts and message sizes 90
5.4 The behavior of *IDA-Gossip* with different percentages of Byzantine nodes 92
5.5 The behavior of *IDA-Gossip* with different dissemination concurrency values 94
5.6 Comparison of *IDA-Gossip* with classic multi-chunk gossip under Byzantine behaviors 97

List of Tables

5.1 Parameter values of IDA-Gossip used in our experiments 86

1 Introduction

1.1 Motivation

With the evolution of Internet technology, we have witnessed the deployment of large-scale distributed systems (Mockapetris and Dunlap, 1988; Cohen, 2003; Dingledine, Mathewson, and Syverson, 2004) on the Internet to provide various services to users around the world. In 2008, blockchains have been added to these systems. Blockchains are large-scale distributed ledgers that order user requests without the help of a trusted authority. Blockchain technology is based on cryptography, consensus, and replicated state machine technologies.

The first blockchain, Bitcoin (Nakamoto, 2008), is proposed as an alternative to today's centralized baking system. Bitcoin implements a decentralized digital currency, also called Bitcoin, and a payment system in which there is no centralized authority to reconcile financial transactions. Bitcoin achieves these goals by solving the problem of consensus in an open setting where nodes can freely join or leave the system without any restriction.

The Bitcoin system consists of a set of nodes connected over the Internet. Nodes cooperate to agree on the payment history. Bitcoin is a peer-to-peer system where contributing nodes are not only service providers but also users of the system. Bitcoin eliminates the need for trusted parties by relying on cryptographic primitives and replication.

The core innovation of the Bitcoin protocol was a novel leader-based eventually consistent consensus algorithm. It is called as Proof-of-Work (PoW). In the Bitcoin system, nodes solve cryptographic puzzles to be elected as leaders, and elected leaders propose a block of transactions to be added to the distributed ledger. There is no efficient way to solve cryptographic puzzles; therefore, nodes use the brute-force approach to solve cryptographic puzzles. A leader shares the block with other nodes using gossip dissemination with the store-and-forward mechanism where nodes validate blocks using a predicate before appending them to the local ledger and sharing them with other nodes; therefore, nodes do not disseminate invalid blocks.

The following figures may help to understand the extent of Bitcoin's adoption. There are currently more than 15,000 Bitcoin nodes processing transactions on the Bitcoin network (*Bitnodes* 2023). There are approximately 20 million Bitcoin tokens in circulation. One Bitcoin is valued by the market at approximately 20,000 USD. On a daily basis, the Bitcoin system processes approximately 300,000 transactions (*Bitcoin Transactions Per Day* 2023), transferring a value of approximately 30 billion USD (*Cryptocurrency Prices, Charts And Market Capitalizations* 2023). Many developed countries legally recognize Bitcoin as a digital asset, and institutional investors are investing in Bitcoin (*Countries Where Bitcoin Is Legal and Illegal* 2023). Note that all figures are collected at the time of writing and may change over time.

After the success of Bitcoin, blockchain technology has found new applications in data management (Ekblaw and Azaria, 2016; Zhu et al., 2019; Truong et al., 2020; Zhaofeng et al., 2020), data verification (Han et al., 2018; Wei et al., 2020; Li et al., 2020), financial services, and IoT (Dorri, Kanhere, and Jurdak, 2017; Huh, Cho, and Kim, 2017; Reyna et al., 2018; Minoli and Occhiogrosso, 2018; Novo, 2018). Majority of these applications become feasible with the help of smart contracts (Luu et al., 2016b). Smart contracts are programs that are deployed on blockchains by users. The content of a smart contract is visible to any user; therefore, users can validate the properties of smart contracts. The logic of smart contracts is enforced by the system: when the conditions are fulfilled, the program runs and produces expected outputs. The use of smart contracts leads to the development of distributed applications (dApps), where all the backend logic resides on a blockchain.

Initially, smart contracts evolved from the Bitcoin's token transfer mechanism that make use of a simple programming language with limited capacity of expression. After Bitcoin, smart contracts evolved to support complex business logic with the use of Turing complete programming languages such as Solidity (Wohrer and Zdun, 2018). Solidity is a special purpose programming language designed for writing smart contracts. Later on, general purpose programming languages such as JavaScript, GO and Rust are also used to write smart contracts.

Another important application area of blockchain technology is non-fungible tokens, which has attracted a lot of attention from the public. Using non-fungible tokens, users can register any type of digital asset on a blockchain, and they can transfer the ownership of the registered assets.

Although Bitcoin is an elegant implementation of a digital currency that solves the fundamental obstacles in front of a trust-free decentralized system, it has many inherent problems. Three of the most significant problems with Bitcoin's design are high confirmation latency, low throughput, and high energy consumption. These inefficiencies are the biggest obstacles in front of the adoption of blockchain technology as a global payment system. To put into context, Bitcoin's transaction confirmation latency is around 600 seconds while classical centralized payment systems have latencies of around a second. Also, Visa processes 24000 transactions per second while the original Bitcoin protocol can process around 7 transactions per second (Bach, Mihaljevic, and Zagar, 2018). Finally, the global energy consumption of the Bitcoin network is comparable with the energy consumption of some countries (Küfeoglu and Özkuran, 2019) such as Denmark and Finland.

On the one hand, in the last decade, researchers have proposed many techniques to circumvent the shortcomings of the Bitcoin protocol without changing its consensus algorithm. On the other hand, many new blockchains have been proposed that differ significantly from the Bitcoin approach, such as Proof of Stake systems, sharded systems, and Proof of X-based systems, where the core component, the consensus algorithm, is replaced by novel techniques. Most of these propositions enrich the system model of Bitcoin to improve its effectiveness while aiming to maintain its desirable features such as scalability and decentralization. Therefore, many blockchain propositions come with its own system model, and each proposition finds a use case in the envisioned system model.

For practical reasons, most blockchain proposals, like Bitcoin, use a single leader and gossip dissemination with the store-and-forward mechanism. In the presence of a leader, consensus requires significantly less communication between nodes. For large distributed systems, gossip dissemination is the only viable option to share updates with other nodes because it does not require the knowledge of all nodes in the system. The store-and-forward mechanism is necessary to ensure that only correct messages are disseminated in the system.

In blockchain systems, the use of a single leader imposes a strict limit on achievable throughput and latency figures, as a single leader will have limited resources—bandwidth and CPU—to prepare a block and submit. The majority of blockchains rely on blocks of transactions to synchronize the state of the system, and the sizes of blocks are on the order of several megabytes: gossip dissemination using the store-and-forward mechanism results in high latency

in case of large messages as each message needs to be received before being forwarded.

After 15 years, blockchains continue to attract the attention of the research community and the public, and new applications are being found. However, performance issues remain the biggest obstacle to the global adoption of blockchain technology, as there is no single blockchain that meets the needs of all applications.

1.2 Research challenges

Blockchains are complex systems that consist of many planes/layers as described by Croman et al. (Croman et al., 2016). These planes are the network, consensus, storage, view, and side planes. Any inefficiency on one of these planes can limit the performance of a blockchain system. Improving the performance of a blockchain requires domain expertise in all the planes because changes introduced on a plane can affect the performance of other planes. Therefore, blockchain research can be considered an interdisciplinary study that requires expertise in many domains.

This work aims to improve the performance of existing blockchains, rather than proposing a new blockchain or an incremental improvement for a blockchain. This is a challenging task in itself, as it requires an extensive analysis phase to understand the properties of many blockchains and uncover common bottlenecks in the system designs. Although different Blockchain proposals has resemblance, they are very different in terms of assumptions and system models, making this analysis phase a challenging and time-consuming task.

After the analysis phase, it is necessary to quantify the impact of the detected bottleneck on the performance of a blockchain, which requires the deployment of several thousand blockchain nodes on many machines. The majority of blockchain proposals do not open source any code to be used for future research. Some of them have production-ready open source code, but production-ready implementations are not the ones used to generate experimental data during the research phase because they tend to consume excessive system resources and prioritize correctness over testability. Accessing the correct implementation and conducting initial experiments to uncover the performance problems was another challenge that needed to be addressed.

For the purpose of performance study, most of the time, it is necessary to implement the entire blockchain protocol from scratch because of the lack of proper source code. The implementation phase is a challenging and time-consuming task in itself, as some blockchains have complex consensus algorithms and complex communication schemes. Also, for the sake of brevity, blueprints provided in publications tend to include important details for discussion rather than all the necessary details for implementation. This makes the implementation phase even more challenging.

1.3 Research goals

Unlike other approaches proposing new blockchain protocols, in this work, we investigate the viability of generic approaches to improve existing blockchains without changing their system model and assumptions.

On the consensus level, we are aiming to remove the inefficiencies of leader-based blockchains by introducing novel techniques to multiplex consensus instances in a generic way. On the network level, we investigate efficient dissemination solutions for blockchains to disseminate blocks.

We list four sub-goals to achieve our main objectives. These sub-goals are as follows:

Research Goal 1: Quantifying the effect of bottlenecks caused by a single leader, and caused by gossip dissemination with the store-and-forward mechanism. This goal is necessary to understand the scale of performance problems that stem from listed bottlenecks.

Research Goal 2: Investigating possible solutions to the problems caused by a single leader and gossip dissemination with the store-forward mechanism.

Research Goal 3: Designing a generalized mechanism to remove listed bottlenecks not on one blockchain but on family blockchains without changing the system models, and assumptions. This is the core goal of our work, as we want to provide a design that is reusable to solve observed bottlenecks on different systems.

Research Goal 4: Quantify the impact of our solutions on representative examples of blockchains through large-scale experiments.

1.4 Roadmap of the thesis

In Chapter 2, we provide the background to blockchain technology by introducing necessary concepts such as consensus and state machine replication. In this chapter, we also focus on the Bitcoin protocol to highlight the limitations of existing blockchains built on top of it. Then, we present important blockchain propositions by classifying them according to the properties of consensus algorithms.

In Chapter 3, we present related work that makes use of multiple leaders in classical consensus algorithms and in blockchain consensus algorithms. Also, we present efficient gossip dissemination mechanisms.

In Chapter 4, we present our proposal, *Alder*, a generic construction for extending existing leader-based blockchains: *Alder* enriches existing leader-based blockchains with multiple leaders to remove bottlenecks caused by single-leader approach. In this chapter, we extensively study the properties of *Alder* and present our experimental evaluation deployed on Grid5000 with up to 100 high-end machines and 10,000 processes. We have applied Alder to three state-of-the-art leader-based blockchains and quantitatively studied the benefits of *Alder*.

In Chapter 5, we present our work investigating the properties and performance of the $IDA-Gossip$ protocol. $IDA-Gossip$ is an efficient gossip dissemination protocol that relies on chunk-based dissemination to reduce the dissemination latency. It is proposed with the *Rapidchain* protocol—a sharded blockchain system. $IDA-Gossip$ can be adopted by other blockchains, but its properties have not been studied in depth. Our work provides an in-depth analysis of $IDA-Gossip$ using experiments and simulations involving 4096 processes deployed on Grid5000. Our work is essential for understanding its behavior under faults and with different protocol parameters.

Finally, in Chapter 6, we conclude this thesis with a brief overview and suggestions for future research. In Appendices A, we list our publications and awards.

2 Background

> "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable"
>
> *Leslie Lamport*

In this chapter, we mention important concepts for distributed systems and blockchains: system models and assumptions, cryptographic primitives, communication schemes, consensus algorithms, and replicated state machines. We then describe the Bitcoin protocol and provide an overview of existing blockchains by classifying them according to different criteria to highlight differences and similarities in system designs.

2.1 Background on distributed systems

According to Tanenbaum and Steen, 2006: "A distributed system is a collection of independent computers that appears to its users as a single coherent system". In general, a distributed system can be characterized by the fact that the global state is distributed and that a common time base does not exist (Mattern, 1988).

We can classify distributed systems into two broad categories close and open distributed systems. In close systems, participants are known and identified using a mechanism such as public keys. New nodes cannot freely join close systems, and they need to be authorized by an authority. Open systems are those in which nodes can join or leave the system without any restriction, and the identities of nodes are not known in advance. Some blockchain systems such as Bitcoin are examples of open distributed systems in which a set of nodes work together to achieve a common goal.

2.1.1 System models and assumptions

System designers design distributed systems by considering a system model and a set of assumptions. Without knowing the system model and assumptions of a distributed system, we cannot infer its properties: they convey information about when and under what conditions the system will function and provide service.

The most important entities in a system model are processes and communication links because, in a distributed system, processes communicate by sending messages to each other over communication links.

Process model: It defines the properties of processes that construct the system. Behavior and timing of processes are among the most important traits. A Process can be considered an automaton that updates its internal state according to received messages and internal events. All activities of a process are considered as its local computation.

The behavior of a process might be correct or faulty. Correct processes always follow the protocol. On the other hand, faulty processes can be faulty in many ways. There exist different models defining the behavior of faulty processes, and among the most important of them are crash, and Byzantine faulty processes.

Crash faulty processes fail by stopping, and they do not send or receive any messages. Byzantine faulty processes might violate the protocol in any possible way: they might send arbitrary messages to other nodes or an attacker might coordinate them. Protocols that can function in the presence of crash faulty processes are called crash fault-tolerant, and Protocols that can function in the presence of Byzantine faulty processes are called Byzantine Fault Tolerant (BFT). The design of crash fault-tolerant protocols is relatively easy compared to BFT protocols because of the restricted faulty behavior.

Process timing models convey information regarding the speed of local computations. Synchronous and asynchronous processes are among the widely accepted process timing models. While synchronous processes finish local computations in a bounded time interval, local computation of asynchronous processes might take an arbitrarily long time.

Communication model: It defines the properties of communication channels that connect processes with each other. Channels can be characterized according to timing, delivery, and ordering guaranties.

According to timing, the most common channel models are synchronous, asynchronous, and partially synchronous channels. Messages sent over a synchronous channel are guaranteed to be received by the receivers in a known time interval. Messages send over asynchronous channels can be delayed by channels for an arbitrarily long time interval. Partially synchronous channels are synchronous channels that can behave asynchronously from time to time. In these channels, it is assumed that each asynchronous period is followed by a synchronous period.

According to delivery guarantees, the most common channel models are lossy and reliable channels. Lossy channels can drop messages sent by nodes. Reliable channels do not drop messages. Reliable channels can be implemented on top of lossy channels by making synchrony assumptions and employing acknowledgment messages and retry mechanisms.

While some channels may reorder messages arbitrarily, others can provide some ordering properties: One of the most common channels with ordering guarantees are First in First Out (FIFO) channels where messages sent by the sender are received in the send order.

Finally, channels can be authenticated where the integrity and authenticity of messages are protected against tamper. Authenticated channels are built on top of non-authenticated channels using cryptographic primitives.

Adversarial model: this model is used to define the capabilities of an adversary that attacks the considered system. In general, adversary attacks the system by coordinating Byzantine nodes. In the literature, there exist different variants of adversarial models.

The availability of unbreakable cryptographic primitives, random oracles, one-way functions, and public key infrastructures are among the most common assumptions made by system designers. We mention all of these primitives in subsequent sections.

2.1.2 Cryptographic primitives for distributed systems

The use of cryptographic primitives in distributed systems is ubiquitous: in particular, they are used to constructing authenticated channels by encrypting messages. Cryptographic primitives are also used to implement random oracles. Last but not least, they are used in blockchains to prove the ownership of digital tokens, as we will see in the following subsections. In this subsection, we will briefly mention the properties of basic cryptographic primitives.

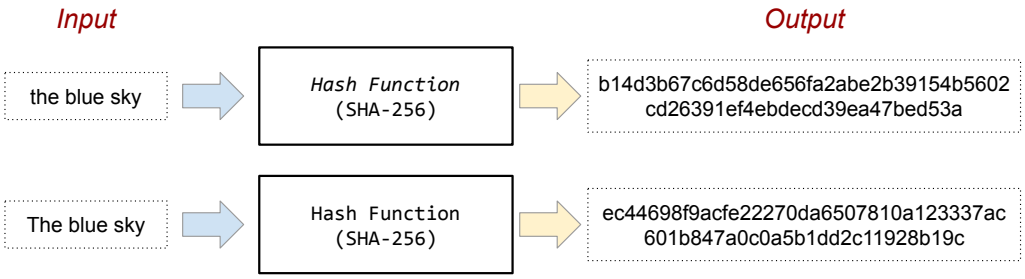

FIGURE 2.1: Example outputs of SHA-256 hash function: a small change in the input causes big change in the output of the hash function.

Hash functions: A hash function H is a function that accepts a bit string of arbitrary length l and maps it to a fixed length string t. A hash function is called collision free if it is computationally hard to find two strings x and y such that $H(x) = H(y)$. A hash function has pre-image resistance if it is difficult to find a message m with a given hash value of h. Finally, a hash function has second pre-image resistance, if it is difficult to find a message m_2 given message m_1 where $H(m_1) = H(m_2)$. Cryptographic hash functions are collision-free, pre-image, and second pre-image resistant hash functions: they are safe to use in cryptographic applications.

A cryptographic hash function maps an input value to an arbitrary value in the hash space. A small change in the input causes a random change in the output of a cryptographic hash function. We see an example of this in Figure 2.1: two hash values are calculated using SHA-256 (NIST, 2015) hash function, and a small change in the input causes a big change—which seems completely random—in the output.

Cryptographic hash functions are essential to ensure the data integrity of digital documents by using a fixed amount of storage space: by using a cryptographic hash function, one can calculate a fixed-length digest of a document, and one can use it as proof of the integrity of a document. Also, as we will see, they are important building blocks for digital signature schemes and symmetric key cryptography systems (Paar and Pelzl, 2009).

Cryptographic hash functions are practical alternatives to random oracles (Fischlin et al., 2010) where an oracle provides a random number according to provided input. For the same input, the oracle produces the same output. Cryptographic hash functions produce random output values according to the input. In a system, all parties can produce the same output value, if they call

the hash function with the same input value. This specific application has many use cases in blockchains to produce publicly known pseudo-random values.

Today, the most commonly used hash functions in distributed systems and blockchains are the SHA-2 (NIST, 2015) family of hash functions. There are non-cryptographic hash functions that can be used for other purposes such as to create message authentication codes (Aumasson and Bernstein, 2012).

Commitment scheme: Using a commitment scheme one can commit to a value by choosing a value from infinitely many values and keeping it secret and sharing the proof of commitment with others. Later on, one can reveal the committed value, and others can validate the correctness of the commitment by using the provided proof without any doubt. An example commitment scheme can be constructed using a cryptographic hash function H: a user chooses a value v and calculates the proof of commitment by calculating $H(v)$. Later on, the user can share the $H(v)$ with interested parties by keeping the v secret. When he revealed the v, others can validate the correctness of the commitment by just comparing the original commitment proof with $H(v)$.

Merkle Tree: A Merkle tree (Merkle, 1979) is a hash tree where leaf nodes are the hashes of data blocks calculated by using a hash function H, and inner nodes are calculated from the bottom up by concatenating the values of children nodes and hashing them using the hash function. Assume that an inner node v has two child nodes C_1 and C_2, and the value of v is equal to $H(C_1||C_2)$ where $||$ denotes concatenation operation. In Figure 2.2, we see an example Merkle tree that is calculated using 4 data blocks. A change in one of the data blocks changes the root of the Merkle tree. This feature of Merkle trees is used to protect data blocks against tampering attempts. Merkle trees were originally proposed to be used as a digital signature scheme. Also, they are extensively used as commitment schemes where one can commit on many values using a Merkle tree, and can calculate the proof of commitments using very little storage. Merkle trees are used in blockchains frequently as we will see in the next sections.

Symmetric key cryptography systems: These systems are used to encrypt strings of bits to create authenticated channels for communication or to store data securely. Symmetric key cryptography systems consist of two functions: encryption and decryption. The encryption function receives a bit string and a key, it produces a ciphertext. The decrypt function receives the ciphertext and a key, and it reproduces the original message. These systems are called symmetric because the encryption and decryption functions must be called with the same

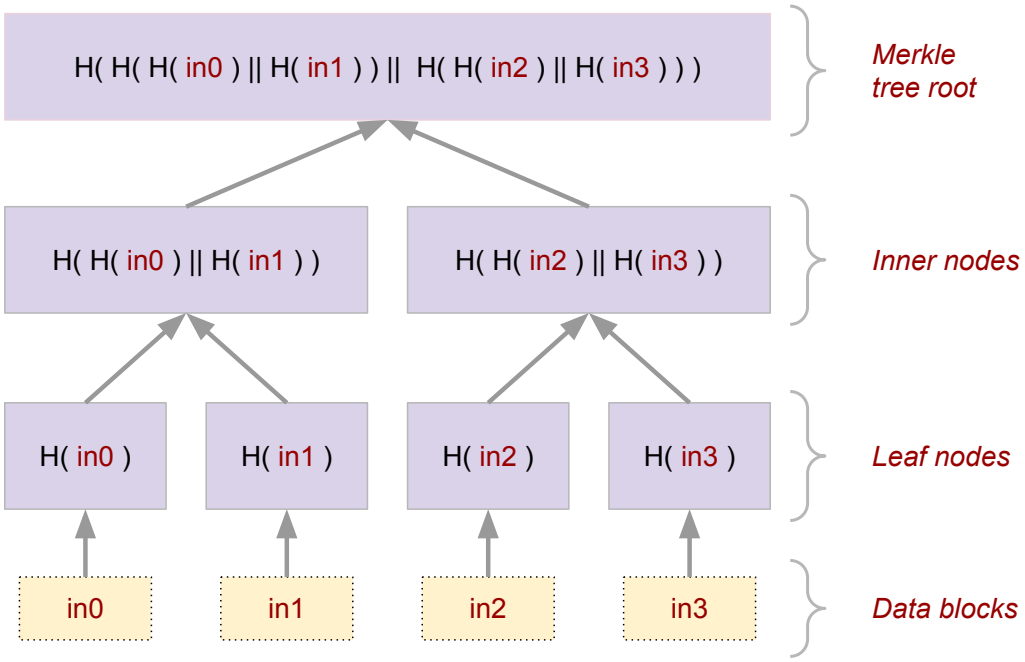

FIGURE 2.2: The construction of a Merkle tree: a change in any of the data blocks causes a change in the Merkle tree root.

key to reproduce the original message. Using a symmetric key cryptographic system, two parties can communicate securely over insecure channels, provided that they have previously agreed on a key. The security of the channel constructed depends on the specific protocol, the key length, and the randomness of the key. The main drawback of symmetric key cryptography systems is the requirement of an agreed key: two parties can meet in a location to agree on a secret key. This scheme does not scale well to communicate with many parties.

Public key cryptography: Public key cryptography is a method of encrypting and decrypting messages without a shared key. Public key cryptography systems consider two kinds of keys: public and private keys. Public keys are shared with other parties in the system, and secret keys are kept secret. Parties who want to use public key cryptography need to know each other public keys. A public key cryptography system consists of 3 functions: key exchange, encrypt, and decrypt functions. The key exchange function gets the secret key of the caller, and the public key of the other party that needs to be communicated, and it produces a shared key. Encrypt and decrypt functions work as described in symmetric key cryptography systems using the key produced by the key exchange function. This approach removes the need to have a shared key to construct an authenticated channel, but it is necessary to know the public

key of the other party to communicate.

Use of public key cryptography might require a Public Key Infrastructure (PKI) (Maurer, 1996) where public keys of all parties are registered in a trusted manner, and users can access them using PKI. This way, users can ensure the authenticity of public keys.

Digital signature scheme: An important application of public key cryptography is digital signatures. Digital signatures are used to sign digital documents for authentication purposes. A digital signature scheme has two functions: $sign$ and $verify$. $sign$ function needs to be called with a private key, and a bit string to sign. It returns a signature which is another fixed-length bit string. It is common practice to sign the hashes of the bit strings for efficiency and security reasons. The $verify$ function needs to be called with the public key of the signer, the bit string of the signed digital document, and a signature. It validates the signature and returns a boolean result.

Message Authentication Codes (MAC): Using MAC, the integrity and authenticity of messages can be validated without using digital signatures. HMAC (Krawczyk, Bellare, and Canetti, 1997) is one of the most common implementations of a MAC, and it is implemented using a secret key, and a hashing algorithm: for each message a MAC is calculated by hashing the message and the key together, and messages sent with its MAC so that receiver can validate the integrity and authenticity of a message. MAC computation can be significantly less CPU intensive compared to symmetric key cryptography systems and digital signature schemes.

2.1.3 Communication in distributed systems

Nodes in a distributed system communicate by message passing where a sender sends a message to a receiver: this is considered point-to-point communication because the communication happens between two parties. Apart from point-to-point communication, there are common interaction patterns that are used by many distributed systems such as broadcast, unicast, and multicast.

Among all, one of the most common communication patterns in distributed systems is broadcast where a sender wants to send a message to all nodes in the system. There are different variants of broadcast: best-effort broadcast, reliable broadcast, and uniform reliable broadcast. Each variant provides different delivery guarantees. Reliable broadcast is one of the most commonly used variants

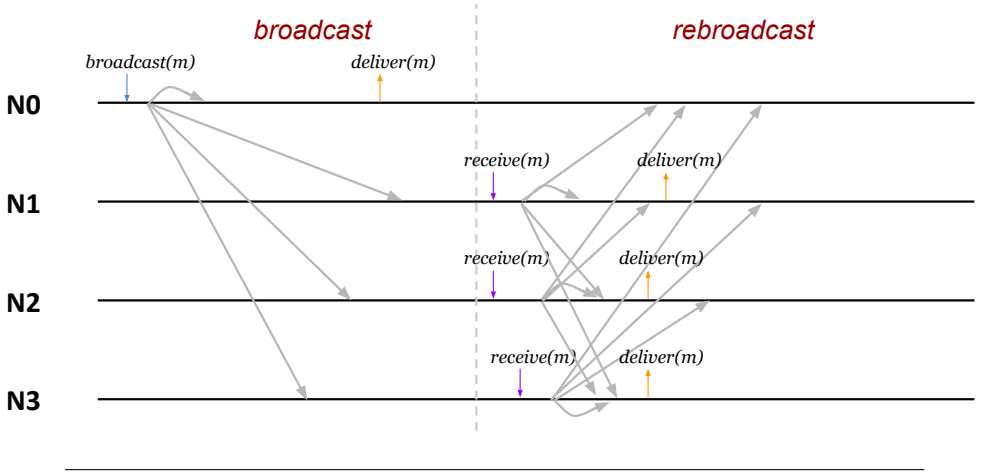

FIGURE 2.3: Communication scheme of reliable broadcast.

of broadcast in distributed systems: it guarantees that if a correct node delivers a message, all correct nodes eventually deliver the same message.

Figure 2.3 depicts the communication scheme of reliable broadcast: node $N0$ starts the broadcast with message m by sending the message to all the nodes in the system, and later on, it delivers the message. Upon receiving the message m for the first time, a node rebroadcasts the message, and, later on, delivers the message. Sending a message to all nodes in the system requires the knowledge of all nodes which might not be possible for all kinds of distributed systems. Also, this requirement limits the use of reliable broadcast to close systems.

Another important broadcast variant is atomic broadcast, which provides the same delivery guarantee as reliable broadcast, and it also provides a total order guarantee where nodes deliver messages in the same order. Atomic broadcast is equivalent to consensus (Milosevic, Hutle, and Schiper, 2011): solving one of them provides a solution to the other. As we will see in the next subsections, atomic broadcast and consensus are important building blocks of distributed systems.

Although reliable broadcast is an important primitive for distributed systems, it is not practical for large systems: as we see in Figure 2.3, each node needs to connect all other nodes in the system to broadcast the message, and the communication complexity is $O(N^2)$. For large open systems reliable broadcast is not an option because nodes can not know the members of the system. To circumvent all these problems, epidemic dissemination protocols are employed in large open systems as practical alternatives to reliable broadcast protocols. Unlike reliable broadcast, epidemic dissemination protocols provide probabilistic delivery guarantees.

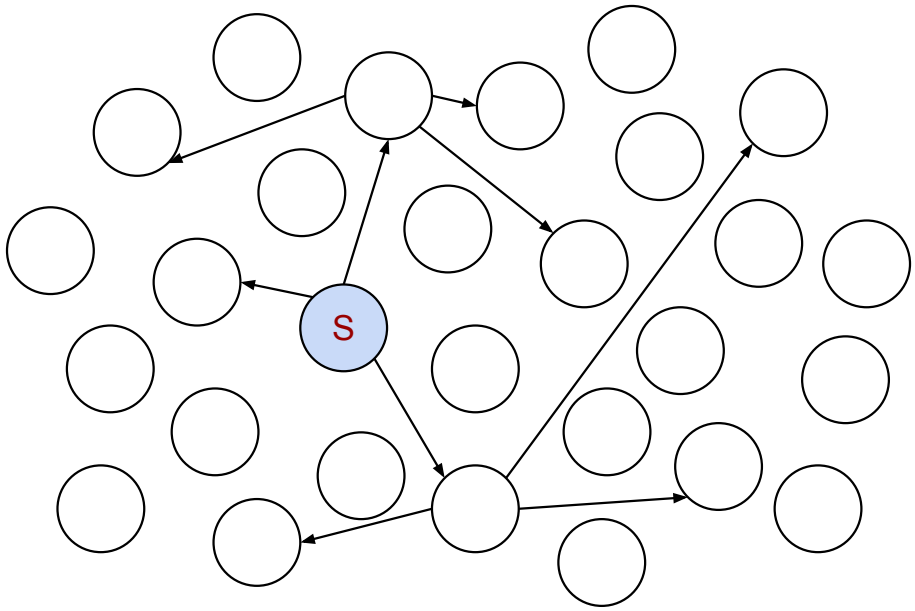

FIGURE 2.4: Communication scheme of gossip dissemination.

Epidemic dissemination protocols are proposed in the context of database replication (Demers et al., 1987), and later on, they are adopted as probabilistic alternatives of reliable broadcast protocols for large systems. Epidemic dissemination protocols are also known as gossip dissemination protocols. Using epidemic dissemination, messages are disseminated much like the dissemination of an epidemic disease: upon receiving a message for the first time, a node selects $fanout$ other nodes in the system and infects them by sending the message. $fanout$ is the most important parameter of an epidemic dissemination protocol, and it determines the redundancy of epidemic dissemination. Higher $fanout$ values increase the coverage—the percentage of nodes delivering the message—but also increase the resource consumption of nodes.

Figure 2.4 depicts the first few steps of dissemination of a message using gossip dissemination. The source node S starts the dissemination by sending the message to $fanout$ other nodes, and in this example, $fanout$ is 3. Upon delivering a message, each node forwards the message $fanout$ other nodes. Note that a node delivers a message only once, therefore a node forwards a message only once. Unlike reliable broadcast, a node needs to send the message to only a few other nodes, and it does not require knowing all the members of the system.

There exist several variants of gossip dissemination: push, pull, and hybrid gossip (Felber et al., 2011). In push gossip, upon receiving a message, a node forwards the message to a few other nodes. In the pull gossip, a node request

messages from a few other nodes sending requests. The hybrid method combines push and pulls mechanisms to increase the dissemination efficiency: in the first few steps push mechanism is used, and later on, the pull mechanism is used to increase the speed of dissemination, and decrease the bandwidth usage. Also, in some variants, nodes can communicate to select messages to exchange which increases the communication cost but decreases the bandwidth usage of dissemination.

Although gossip dissemination protocols are practical for large-scale systems, they have several inefficiencies. One of them is incurred high latency: when they are used to disseminating large messages using the store-and-forward mechanism where a message is received fully before being forwarded, gossip dissemination incurs a high latency. We mention efficient gossip dissemination protocols in Section 3.2.

2.1.4 Consensus

One of the central problems of distributed systems is consensus. The consensus problem is the problem of reaching an agreement among remote processes: according to the agreed result, all honest parties take the same action. As stated by Turek and Shasha, 1992: "Consensus is part of any distributed system that embodies coordinated activity—from the synchronization of clocks, to election of leaders, to the coordination of rocket firings". One of the most important applications of consensus is replicated state machines where replication is used to mask faulty nodes in the system. Instances of a replicated state machine use consensus to agree on the order of client requests, and each replica updates the state machine in the order of client requests. This way all replicas of the replicated state machine stay in a consistent state.

Consensus algorithms provide some properties, and these properties are generally categorized as safety, liveness, and availability properties. Specific properties may change according to the consensus algorithms and considered system models. Safety property guarantees that the algorithm never violates the safety conditions: unless the assumptions of the considered systems model are not violated, a correct consensus algorithm should not violate safety assumptions. Liveness property guarantees that something desirable eventually happens. Without liveness properties, a consensus algorithm does nothing can be considered correct because it does not violate the safety properties. The availability properties provide information about under which circumstances the system will be fully functional.

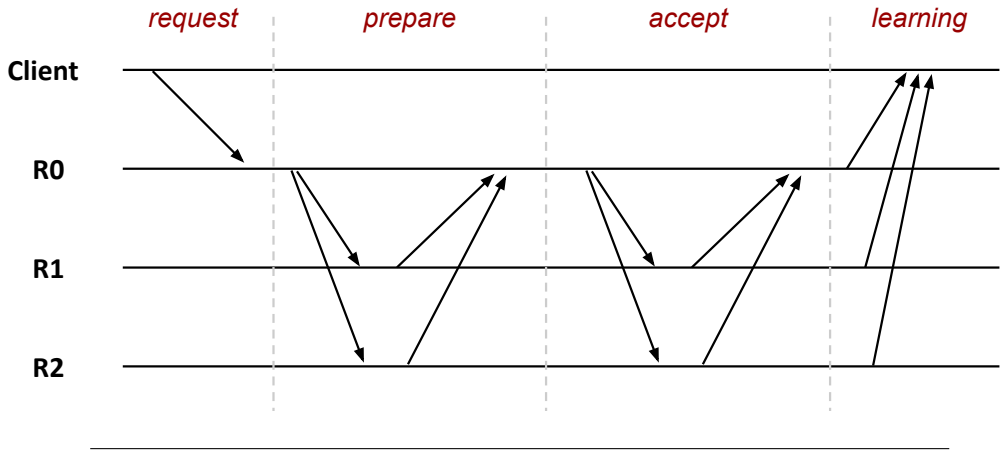

FIGURE 2.5: Communication scheme of the Paxos protocol.

One of the earliest and most well-known examples of a crash fault resilient consensus algorithm is Paxos (Lamport, 2001). Paxos solves the consensus problem in a partially synchronous setting: if the system behaves asynchronously, it does not violate any safety properties, and to provide liveness, it makes a synchrony assumption.

Paxos algorithm defines three different roles for nodes: proposer, acceptor, and learner. The proposer receives client requests and proposes values for acceptors. An acceptor accepts the requests coming from the proposer. Finally, learners learn the decided values. A Paxos system consists of $2F + 1$ nodes, and the protocol tolerates F crash faulty nodes. It means that a Paxos system with three nodes can tolerate up to 1 crash faulty node.

Paxos implements a 2 phase commit mechanism, and it makes use of a distinguished proposer, leader, to provide liveness. Figure 2.5 depicts the communication scheme in two phases of the Paxos protocol: in the first phase, a distinguished proposer gets client requests and selects a proposal number n, and sends a *prepare* request to the majority of acceptors. Upon receiving a *prepare* request with n, each acceptor sends a reply message to the proposer. In the second phase, if the proposer receives responses from the majority of acceptors, the proposer sends an *accept* request to each acceptor. Upon receiving an *accept* message, each acceptor accepts the request for the proposal number n. Without a distinguished leader, nodes can propose different values using the same proposal number forever which prevents nodes from agreeing on a value.

The Paxos algorithm is extended by Fast Paxos Lamport, 2006 to decrease the latency of learners. Later on, the Raft algorithm proposed by Ongaro and

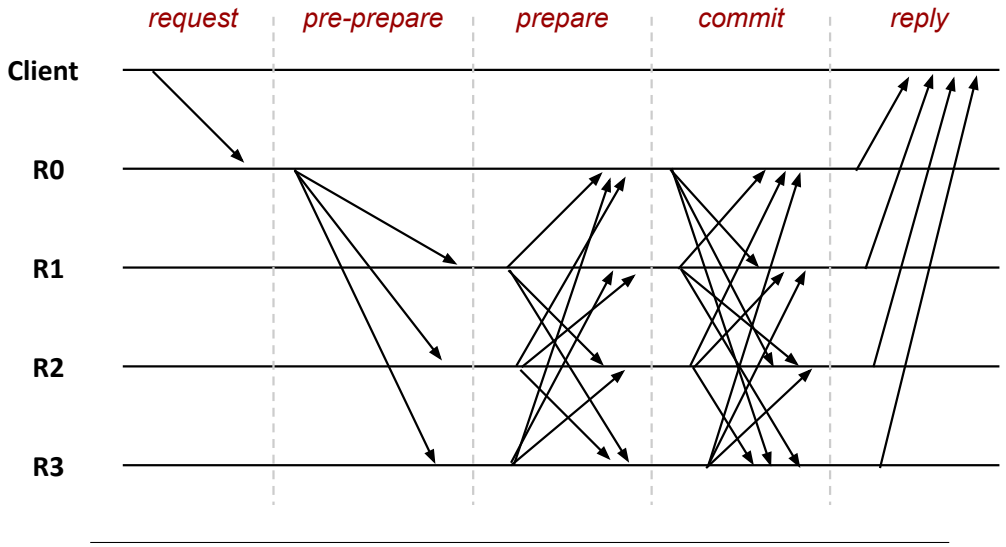

FIGURE 2.6: Communication scheme of the PBFT protocol.

Ousterhout, 2014 refines the Paxos algorithm: specifically, Raft redesigns the Paxos algorithm by targeting understandability, and ease of implementation.

There exist non-leader-based crash-resilient consensus algorithms that do not utilize a leader. One of them is Ben-Or (Ben-Or, 1983). It solves the binary consensus problem in which nodes agree on a binary value. Ben-Or employs randomization for achieving liveness.

One of the important Byzantine fault resilient consensus algorithms is Practical Byzantine Fault Tolerance (Castro, Liskov, et al., 1999) (PBFT). It is the earliest proven BFT consensus algorithm. PBFT implements a 3-Phase commit algorithm to mask the effect of Byzantine faulty nodes in the system. PBFT systems require $3F+1$ nodes where up to F can be Byzantine faulty. As Paxos, PBFT also uses a distinguished proposer, leader, to provide liveness.

A PBFT round consists of 3 phases: *preprepare*, *prepare*, and *committ*. The communication flow of the PBFT is seen in Figure 2.6. PBFT employs digital signatures, and MAC to protect the authenticity and integrity of messages: In the normal case, it employs MAC where every message send with its tag because they can be calculated faster than digital signatures. In view changes, PBFT uses digital signatures to protect the messages against tamper.

Other notable BFT consensus propositions are Zyzzya (Kotla et al., 2007), Prime (Amir et al., 2011), BFT-SMART (Bessani, Sousa, and Alchieri, 2014), 700BFT (Aublin et al., 2015), Hotstuff (Yin et al., 2019), SBFT (Golan Gueta et al., 2019).

Paxos and PBFT do not make timing assumption to provide safety guarantees: both protocols counts the cast votes to decide on a value. Both protocols make timing assumptions to handle faulty leaders: If a faulty leader does not respond for a long time, both protocol runs a view change protocol to replace a faulty leader. View change increases the complexity of both protocols significantly. Both protocols consider a closed system where the participants are known in advance. Finally, Paxos requires 2 rounds of all to all communication, and PBFT requires 3 rounds of all to all communication, and this makes them unpractical for large systems.

2.1.5 Replicated state machines

Replication is the essential technique to increase a state machine's availability, performance, and fault tolerance. By replicating a state machine, one can improve its availability and fault tolerance because requests can be processed by one of the correct replicas in case of a link or device failure. Furthermore, replication might provide a substantial performance increase— especially in the case of read-heavy state machines, because read requests can be processed by different replicas, and increases throughput and decrease latency. In this context, state machines are considered deterministic because after applying client requests in order, all replicas will be in the same state, and any of the replicas can serve concurrent requests. For example, in Figure 2.7, we see a state machine and its replicated version.

Although replicated state machines have many advantages, they have one crucial disadvantage: communication cost. Instances of a replicated state machine need to communicate to order client requests. Although implementing a replicated state machine is more complex than implementing a state machine, this complexity can be abstracted by using middleware as a black box. In Figure 2.7, we see how the replicas of the state machine use the ordering service. Although an ordering service does not need to use consensus, it is common to implement ordering service using consensus: ordering services implemented using consensus provide stronger fault resilience guarantees.

The alternative way to implement highly available fault-resilient systems is the primary-backup approach (Budhiraja et al., 1993). In the primary-backup approach, a system consists of n nodes, and one of the nodes is assigned as the primary backup. All write requests are made to the primary, and primary forward requests to other nodes—followers. In case of a faulty primary, nodes assign one of the nodes as primary in a round-robin manner. The primary

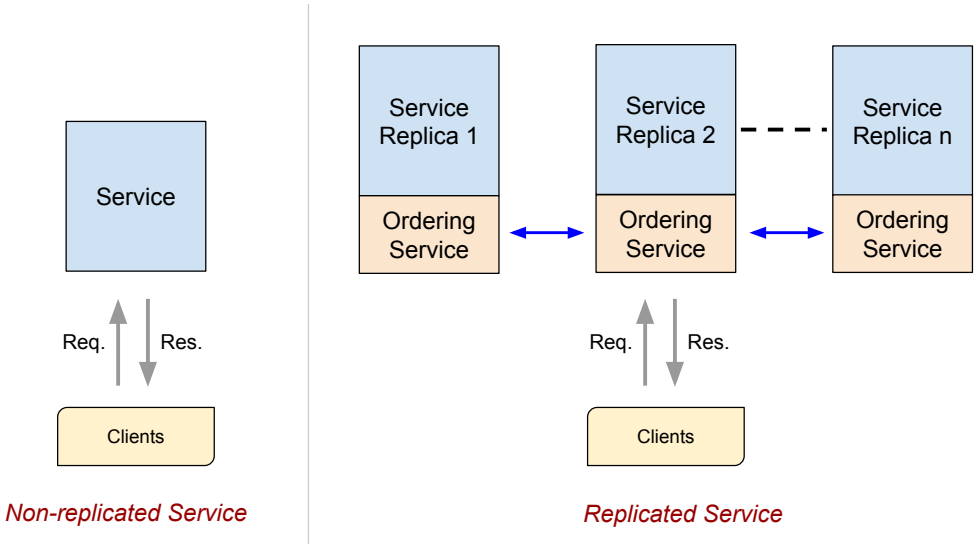

FIGURE 2.7: Comparison of a non-replicated service with a replicated service.

backup approach differs from the state machine replication approach regarding provided safety guarantees and considered system models.

2.1.6 Challenges of open distributed systems

Several problems need to be tackled by any practical open distributed systems. These are the Sybil problem, free riders' problem, and denial of service attacks.

The Sybil problem: In open systems, an attacker can create many fake identities to affect the decision or over-represent its presence in the system. This is known as the Sybil problem (Douceur, 2002). Many close systems do not consider the Sybil problem because a trusted party already authorizes identities. There is no known efficient way of handling Sybil's problem without trusted entities. Because of the Sybil problem, classical consensus algorithms can not function in open systems.

Free riders: Some nodes may try to benefit from an open system without contributing. This is known as the "free rider" problem (Feldman and Chuang, 2005). This problem mainly affects Peer to Peer file-sharing and Peer to Peer live-streaming networks because these types of systems function with the help of their contributor, and consuming without contributing decreases the performance of those systems. Closed systems do not consider this problem because the clients are pre-identified and can be evicted from the system in case of misbehavior. Several techniques exist to handle this problem using accountability

and incentive mechanisms (Li et al., 2006; Guerraoui et al., 2010; Mokhtar, Decouchant, and Quéma, 2014).

Denial of Service (DoS) attacks: Denial of Service attacks is particular types of attacks that try to temporarily or indefinitely disrupt a service temporarily or indefinitely (Park and Lee, 2001). A common way to conduct DoS attacks is by making many legitimate requests that can not be differentiated from requests coming from honest users. This attack can be handled in a closed system by excluding misbehaving nodes. However, it is particularly hard to handle in an open setting because an attacker can create new identities to continue to attack.

Any practical open distributed system needs to handle the Sybil problem, free rider problem, and DoS attacks.

2.2 Background on blockchains

In this section, we will provide the necessary background on blockchains. We start the discussion by studying the Bitcoin protocol because it is the first blockchain protocol, and many other blockchains follow the standards set by Bitcoin.

2.2.1 Bitcoin

The Bitcoin protocol succeeds in implementing a digital currency without the aid of trusted parties by solving two critical problems that have not been solved before. These problems are so-called double spending and coin distribution problems. The double spending problem stems from the nature of digital currencies or digital tokens: Digital tokens are bit stings, and the owner of a digital token can reuse the same token in more than one financial transaction by trivially copying the content. Physical tokens such as banknotes and coins are not concerned with this problem because, during a financial transaction, they are exchanged with goods or services; therefore, they can not be double-spent. The token distribution problem is the problem of distributing tokens initially to the interested parties.

Before the Bitcoin protocol, digital currencies were implemented with the help of trusted mediators such as banks and government institutions. All account balances were kept by trusted mediators, and all transactions were cleared and processed by them to handle disputes. The use of trusted mediators solves the

double spending problem because transacting parties trust the records kept by trustees. As of today, before the Bitcoin protocol, tokens (digital or physical currencies) were distributed by banks providing loans to interested parties.

2.2.1.1 System model

The Bitcoin system is an open system consisting of a set of nodes, and each node is connected to a few other nodes over the Internet. The resulting overlay network is known as the Bitcoin Network. Nodes of the Bitcoin network follow the Bitcoin protocol. The Bitcoin protocol makes use of cryptographic primitives and replication to remove trustees. The name of the digital currency implemented by the Bitcoin protocol is also Bitcoin.

The Bitcoin system keeps track of ownership of digital tokens; for that purpose, the Bitcoin protocol implements a distributed ledger: each node sustains a replica of the ledger, and user transactions are stored in the distributed ledger. The Bitcoin system does not keep any personal information of its users: users are identified pseudo-anonymously by public-private key pairs. The users of the Bitcoin system do not trust anyone except the protocol: users are expected to have a fully functional bitcoin node that follows the protocol.

2.2.1.2 Value transfer

In the Bitcoin system, users can create many accounts by creating public-private key pairs. Users of Bitcoin issue transactions locally to transfer digital tokens to others. A transaction contains the ID of the digital token (the hash of the token), the receiver's public key, and a signature as proof of ownership.

Figure 2.8 depicts the transfer of a Bitcoin (digital token) using cryptographic hash functions and signatures: the owner transfers the ownership of the token to another user by signing the hash of the transaction and public key of the next owner. Any node in the system can validate token ownership by validating the chain of transactions. Furthermore, the digital tokens of a user are tied with a private key: if a user loses its private key, it can not access tied digital tokens anymore.

2.2.1.3 Blockchain and Proof-of-Work

The Bitcoin protocol implements a distributed ledger to solve the double spending problem. The distributed ledger of Bitcoin consists of blocks of transactions,

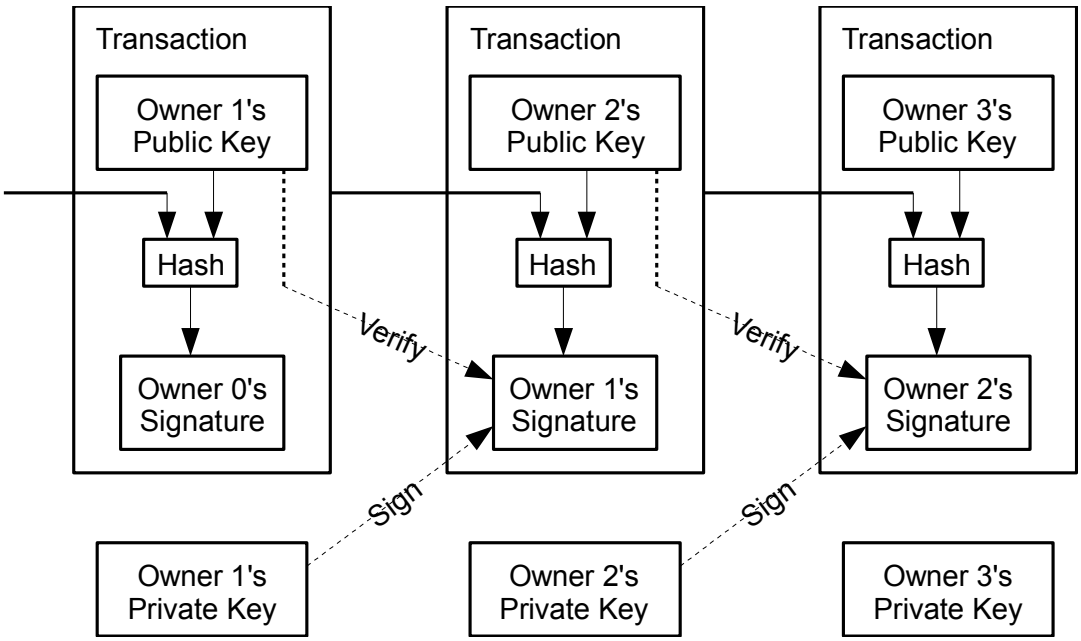

FIGURE 2.8: Transfer of a digital token in Bitcoin using digital signatures (Nakamoto, 2008).

and each block contains the previous block's hash. This construction creates a chain of blocks, and that is where the blockchain name is derived.

The first block of the blockchain is called the genesis block. The Bitcoin protocol defines the content of the genesis block. All nodes start the blockchain with the genesis block: as described in Section 2.1.5, the replicas of a replicated system should start from the same start. Each bock has an integer index which is known as a block height. As seen in Figure 2.11, the genesis has a height of 0, and the first block mines on top of it have a block height of 1, and so on.

The users of Bitcoin issue transactions locally, but to validate a transaction, they are required to register transactions into the distributed ledger. Validated transactions are stored in the blockchain, and each node of the Bitcoin system sustains a replica of the blockchain. For that purpose, nodes disseminate user transactions in the Bitcoin network using gossip dissemination. Upon receiving a transaction, for the first time, a node validates the transaction using the local copy of the blockchain, and if it is valid, stores it locally and forwards it to other nodes. Nodes do not share invalid transactions with other nodes that do spend or transfer unowned or spent digital tokens.

Bitcoin's gossip dissemination mechanism does not provide any ordering guarantees; therefore, nodes of the Bitcoin network receive disseminated transactions in different orders. Before appending transactions to the local ledger, nodes

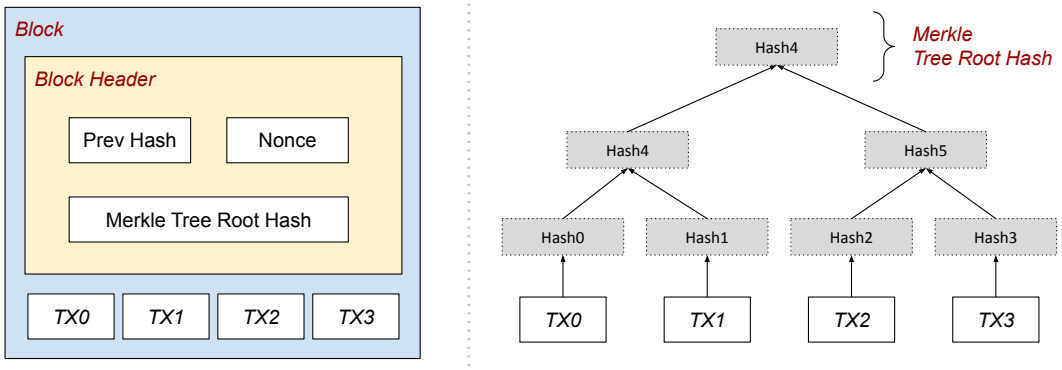

FIGURE 2.9: The block structure of Bitcoin, and its transaction Merkle tree.

need to agree on the order of transactions. For that purpose, the Bitcoin protocol implements a consensus algorithm. The name of the consensus algorithm is Proof-of-Work (PoW). It consists of two essential components: cryptographic puzzles and the longest chain rule.

Nodes of the Bitcoin network assemble received transactions into blocks of a limited size. A Block consists of two parts: block header and block body, as seen in Figure 2.9. The block body contains the transactions waiting to be appended to the distributed ledger. The block header consists of the hash of the previous block, a nonce field, and the Merkle root of the transactions. The Merkle root of transactions efficiently authenticates all transactions inside a block.

A valid block should contain valid transactions in the body, and the hash of its header should have a smaller value than a target hash value. Nodes try to find a valid block by hashing the block and updating the nonce field. This is the cryptographic puzzle mechanism of the Bitcoin protocol. There is no efficient algorithm to solve the cryptographic puzzles of Bitcoin, and nodes try to find a solution using a brute force approach. Upon finding a valid block, the node shares it with other nodes using gossip dissemination. Upon receiving a block, each node validates the content of the block, and if it is valid, it appends it to the local ledger.

The target hash value is known as the mining difficulty of the Bitcoin system: the smaller target values make finding a block harder, while greater target values make it easier. The system dynamically calculates the difficulty so that, on average, a block is found every 600 seconds. The original Bitcoin protocol limits the size of blocks to 1 MB. Therefore, on average, every 600 seconds, 1

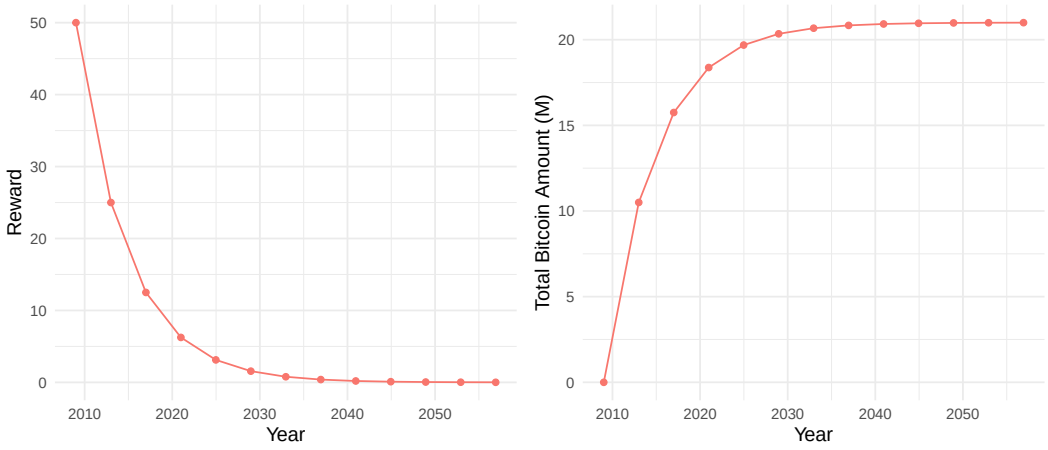

FIGURE 2.10: The Bitcoin mining reward and the total Bitcoin amount by years.

MB of data is appended to the distributed ledger. The cryptographic puzzle difficulty and block size are set to decrease the number of competing blocks for each time interval.

Solving cryptographic puzzles is CPU-intensive; hence, it requires energy consumption. The Bitcoin protocol incentives nodes by providing a block creation reward: a successful puzzle solver is rewarded with a fixed amount of Bitcoin. The first transaction of each block is called coin base transaction—a transaction without any input. The node sends the mining reward to a specific address or public key by issuing a coin base transaction. The reward amount is dynamically updated; in the beginning, it was 50 Bitcoins. The reward is halved every 210,000 blocks appended to the distributed ledger, and as of writing, it is 6.25 Bitcoin. The halving mechanism limits the total amount of Bitcoins to 21 million. Figure 2.10 shows the calculated change of Bitcoin mining rewards and the total amount of mined Bitcoins by years.

In the Bitcoin network, nodes solving cryptographic puzzles are named miners, and finding a valid block is called mining because of the analogy made to mining in a gold mine.

The block creation reward of the Bitcoin protocol effectively solves the coin-distribution problem: digital tokens are distributed over the nodes who contribute to the system, miners.

The blockchain of Bitcoin might fork if two or more valid blocks are mined for the next block height. In the case of a fork, a node accepts the first received block, and it tries to mine on top of it. A fork is resolved after a block is mined

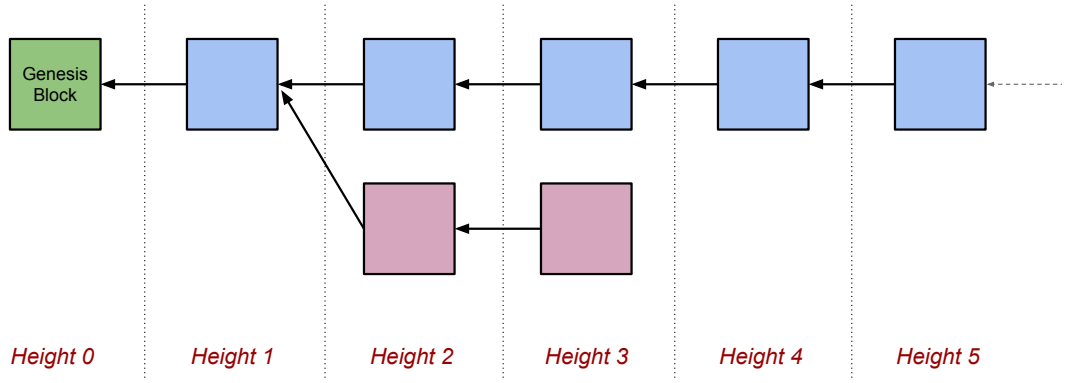

FIGURE 2.11: A chain of blocks with forks.

on top of one of the alternative chains, and nodes choose the longest chain in terms of invested computation power to mine on top. Hence, it is known as the longest-chain rule.

Figure 2.11 depicts an example chain of blocks with forks: the longest chain, the main chain, is depicted using the blue color, and pink blocks are considered stale blocks because they are not on the main chain, and transactions inside these blocks are not considered registered to the blockchain.

The cryptographic puzzles of Bitcoin protect the system against tampering attempts. Without cryptographic puzzles, a malicious party can try to recompute an alternative history of transactions by reordering transactions inside the blocks to double-spend. When the block's content is updated, the hash of the block changes, and it breaks the chain blocks. It is very easy to validate a broken chain in a blockchain. Therefore, a malicious party needs to solve all cryptographic puzzles starting from the block it wants to change; to succeed, it needs to make its chain longer than the honest chain. The Bitcoin protocol makes tamper attempts very costly—practically impossible—by using cryptographic puzzles.

2.2.1.4 Transaction model

Bitcoin protocol adopts the Unspent Transaction Output (UTXO) model. In this model, as seen in Figure 2.12, a transaction consumes the outputs of one or more previous transactions and produces new outputs. An exception to this rule is the coin base transaction: it is the first transaction of each block and produces outputs without consuming any. An output of a transaction can be consumed

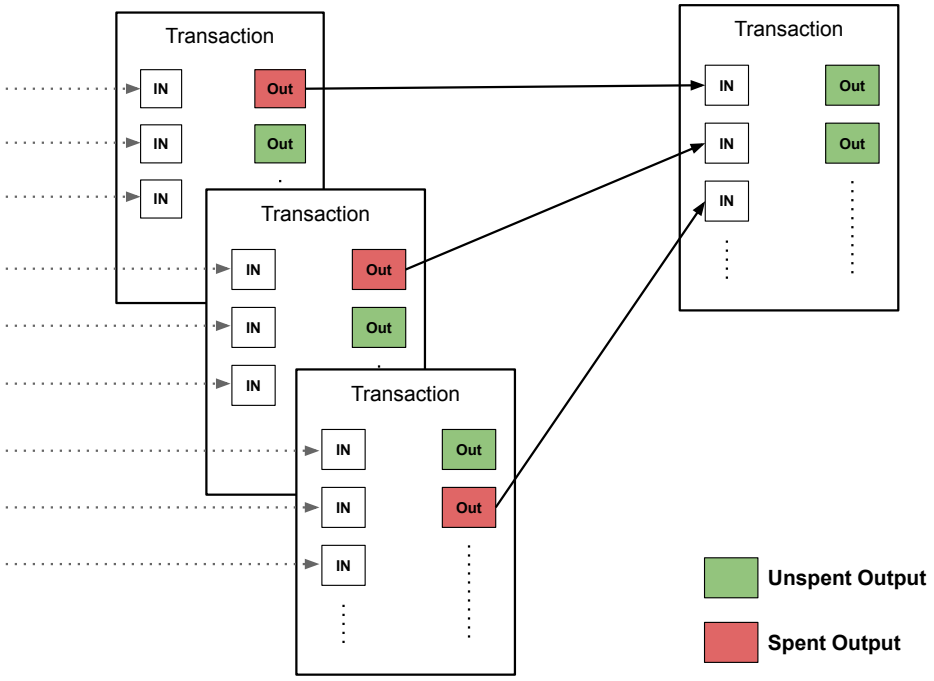

FIGURE 2.12: The structure of a Bitcoin transaction.

only once; trying to consume an already consumed output is considered double spending and is not allowed by the system.

In the Bitcoin system, transactions can pay a transaction fee to incentivize the miners. A transaction pays a fee by not outputting the total sum of its inputs. For example, assume that a transaction consumes 5 Bitcoins and transfers 4 Bitcoins to a receiver's account. If no output is specified for the remaining 1 Bitcoin, it is considered a transaction fee. In addition, the miner can transfer transaction fees in the coin base transaction to specific addresses. Therefore, a coin-base transaction's total outputs are the sum of the block creation reward and the paid transaction fees in a block.

The transaction model of Bitcoin allows nodes to validate new transactions efficiently. Nodes sustain the set of unspent transaction outputs, and inputs of new transactions are searched in this set. After appending a new block to the blockchain, nodes update the set by removing spent outputs and adding produced outputs.

2.2.1.5 Observations about the Bitcoin protocol

Unlike classical consensus algorithms mentioned in Section 2.1.4, the Bitcoin protocol solves the problem of consensus in an open setting where nodes can join or leave the system without any limitation. Solving consensus in an open setting requires resilience to Sybil and DoS attacks.

The Bitcoin protocol handles the Sybil problem using cryptographic puzzles: a node has a voting power proportional to its computation power; therefore, an attacker can not benefit from creating fake identities without investing in computation resources. Cryptographic puzzles of Bitcoin make Sybil attacks practically impossible because a successful Sybil attack requires investment in a substantial amount of computation resources. Also, cryptographic puzzles secure the blockchain against tampering attempts.

The Bitcoin protocol handles system-wide DoS attacks by authenticating all messages disseminated. In the Bitcoin protocol, two kinds of messages are disseminated in the system: transaction and block messages. Nodes validate these messages, and only valid protocol messages are disseminated. Therefore, although an attacker can target an individual node using a DoS attack, it can not target the whole system.

The Bitcoin system is protected against the free-riders: digital tokens are given to the nodes contributing to the system, and it is impossible to use the system without owning digital tokens.

Finally, Bitcoin's consensus algorithm can scale thousands of nodes because it does not use all to all communication to agree on blocks. The Bitcoin protocol pays the cost of scalability by providing a probabilistic consensus guarantee where the system provides eventual consistency. Because of forks, users need to wait for some time to make sure that transactions issued by them will stay on the ledger. The probability of excluding from the blockchain for a block exponentially decreases by mined blocks on top of that.

2.2.1.6 Problems of Bitcoin approach

The Bitcoin protocol is highly scalable: as of writing, there are more than 15,000 nodes in the network, and they agree on the order of user transactions. However, despite high scalability, It has several problems that stem from design decisions. These problems are high energy consumption, high latency, and low throughput.

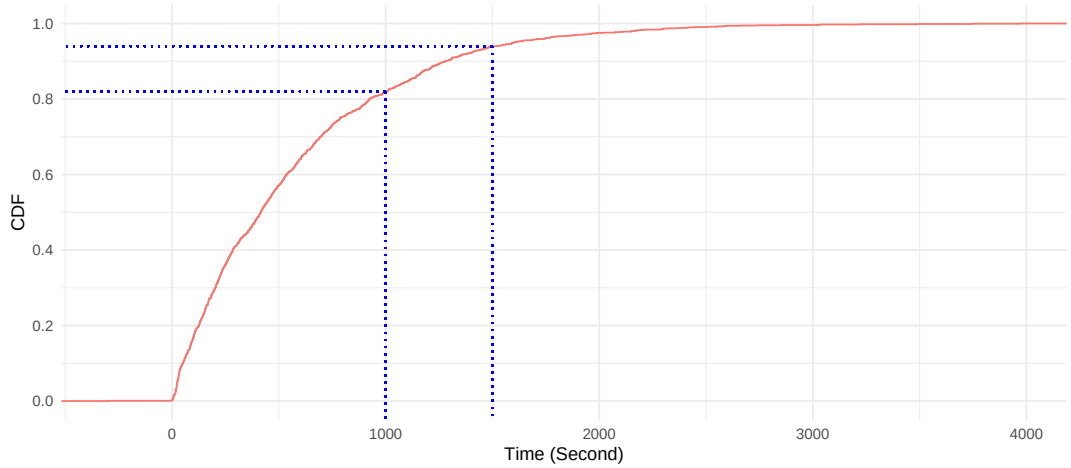

FIGURE 2.13: CDF of Bitcoin mining time calculated using the data of 1500 consecutive Bitcoin blocks.

High energy consumption: The difficulty of cryptographic puzzles increases with the increased computation power invested into the Bitcoin system. Because of this, the energy consumption of the Bitcoin system increases with the number of miners in the system. As a result, the Bitcoin system consumes a high amount of energy comparable to the energy consumption of some countries (Küfeoglu and Özkuran, 2019).

High latency: The Bitcoin system processes transactions with a high latency: if there are no competing transactions, a transaction needs to wait an average of 600 seconds to be appended to the ledger. In Figure 2.13, we see the Cumulative Distribution Function (CDF) of Bitcoin's block mining times: The CDF is calculated by using data of 1500 consecutive blocks of Bitcoin. As seen on the CDF, 18% of the time, it takes longer than 1000 seconds to mine a block, and 6% of the time, it takes longer than 1500 seconds to mine a block. These values are considerably higher than the expected value of 600 seconds.

In the presence of competing transactions waiting to be appended to the blockchain, the measured latency might be higher. As of writing, a bitcoin transaction that pays a transaction fee needs to wait, on average, 40 minutes to be appended to the blockchain (*Bitcoin Transactions Per Day* 2023).

Forks further alleviate the effect of transaction latency: when a fork is resolved, some transactions are removed from the ledger, and because of this risk, users of Bitcoin need to wait longer to make sure that a transaction will stay in the ledger. This is known as the confirmation latency, and the confirmation latency of Bitcoin is around an hour because the users of Bitcoin need to wait for at

least five more blocks after seeing their transactions in the ledger to have a small risk of transaction removal.

Low throughput: Bitcoin has a low transaction processing throughput because of the limited block size and block creation frequency. On average, the Bitcoin system processes 1 MB's of transactions every 600 seconds. Assuming a transaction size of 512 bytes, a block of 1 MB can contain up to 4096 transactions. This yields a transaction throughput of 7 transactions per second. This value is considerably lower than the peak throughput of Visa, which is 24,000 transactions per seconds (Bach, Mihaljevic, and Zagar, 2018).

Bitcoin has many other issues that are orthogonal to this work. Among the most important are fairness issues, centralization tendencies, the use of custom hardware (Application Specific Integrated Circuit), and problems related to the post-block reward era. The first three problems stem from the difficulty of cryptographic puzzles: Today, the difficulty of cryptographic puzzles is so high that an individual miner cannot mine a single bitcoin using standard equipment, which leads to two phenomena: the use of specialized equipment and the formation of mining pools. First, miners invest in specialized equipment to solve cryptographic puzzles. This raises the fairness issue, as it is difficult to contribute to the bitcoin system without investing in special equipment. Also, this equipment causes a waste of resources because they are too specific to be used for other purposes. In addition, individual miners combine their computing power to form mining pools. The nodes in a mining pool solve cryptographic puzzles together and share the block rewards. These mining pools are controlled by a trusted party, contrary to the Bitcoin protocol's nature. Finally, as we have described, bitcoin's mining reward is halved to 210,000 blocks, and, in the future, the number of miners is expected to decrease due to the low mining reward; this is a potential risk for Bitcoin because, without miners, the Bitcoin system can not process user transactions.

We have concluded the study of the Bitcoin protocol. After that, we are ready to study important blockchain propositions.

2.2.2 Family of blockchains

Researchers proposed numerous solutions and alternative designs to overcome the problems of Bitcoin. In this subsection, we will provide a broad overview of blockchain proposals. For that purpose, we have classified blockchains into

subfamilies. Our classification does not follow the traditional way of classifying blockchains into two broad categories as in other works: permissioned and permissionless. The reason is that we would like to provide a more detailed picture of the Blockchain research track. In our categorization, blockchains use a similar technique to handle similar problems considered in the same category. Our categories are not disjoint; therefore, a blockchain can be categorized more than once under different categories.

2.2.2.1 Proof of Work based blockchains

Although PoW has inherent energy consumption and performance issues, it provides a trust-free mechanism to handle the Sybil nodes in the system. Also, PoW-based blockchains are simpler in design than other blockchains, as the protocol does not need communication during the consensus phase. PoW-based blockchains use Bitcoin's Proof-of-Work (PoW) mechanism to decide on blocks to append to the ledger. Our classification distinguishes between the use of cryptographic puzzles and the use of the PoW consensus algorithm. Many blockchain propositions use cryptographic puzzles to elect leaders and committee members. However, most of them do not utilize PoW to decide on blocks. In this category, we only consider blockchains that rely on PoW to decide on blocks. One of the essential characteristics of these blockchains is eventual consistency: the consensus decision is not final, and with the increasing size of the blockchain, nodes agree on a prefix of the blockchain with a high probability. The most important member of this class of blockchains is Bitcoin, and we have already studied its properties in previous sections.

The Greedy Heaviest Observed Sub Tree (GHOST) (Sompolinsky and Zohar, 2013) is one of the earliest propositions that improves the PoW consensus algorithm to increase the transaction processing capacity of the Bitcoin system. Proposers of the GHOST protocol make an important observation: blocks not on the main chain can contribute to the consensus decision by voting on one of the forks. In light of this observation, the GHOST protocol proposes a new policy for selecting the main chain in the block tree to replace the longest chain rule of Bitcoin's protocol: the GHOST protocol chooses one of the forks with the heaviest subtree in terms of invested computation rooted at the fork by considering stale blocks. Therefore, stale blocks not on the main chain help decide on one of the forks. A modified version of the GHOST protocol was adopted by Ethereum (Wood, 2014), which is the second-largest blockchain after Bitcoin in terms of adoption. Although Ethereum recently moved to a

proof-of-stake consensus algorithm, the original Ethereum consensus algorithm targeted a cryptographic puzzle difficulty of about 12 seconds using the GHOST protocol. The short block confirmation time is the reason behind Ethereum's success. Also, the transaction processing cost of Ethereum is considerably lower compared to Bitcoin because of the level of the cryptographic puzzle difficulty. Monoxide (Wang and Wang, 2019), a sharded blockchain, also uses the GHOST protocol.

Bitcoin-NG (Eyal et al., 2015) targets improving the throughput and latency characteristics of the Bitcoin protocol without replacing PoW and decreasing the difficulty of cryptographic puzzles. In Bitcoin, a leader is elected by finding a solution to a cryptographic puzzle, and the leader has the right to submit a single block of transactions. Bitcoin-NG makes an important observation: leader election and transaction serialization can be separated in the Bitcoin protocol. Based on this observation, Bitcoin-NG separates leader election from transaction serialization by introducing two types of blocks: *key blocks* and *microblocks*. Key blocks are the blocks that contain solutions to cryptographic puzzles. A node becomes the leader after submitting a valid key block. Later, the leader can submit multiple microblocks that contain transactions until the election of a new leader. Microblocks do not contain solutions to cryptographic puzzles; therefore, submitting microblocks does not require energy consumption. The biggest downside of the Bitcoin-NG protocol is that forks frequently happen during the leader change, and Bitcoin-NG provides incentive mechanisms to handle these types of forks.

OHIE (Yu et al., 2020a) is a recent blockchain proposition that uses a modified version of PoW. OHIE envisions the use of many parallel chains. The number of parallel chains is k, and it is around 1000. OHIE sets the difficulty of cryptographic puzzles inversely proportional to the k value. Miners in OHIE mine for all chains using a Merkle commitment scheme: a block header references all previous blocks on k chains. At the end of the mining procedure, the miner is assigned one of the parallel chains, and it submits a block for that chain. OHIE envisions the use of very small blocks (around 20 KB) that disseminate in the network very fast. OHIE retains the longest chain rule of Bitcoin, but it has other rules to synchronize the growth of different chains, as different chains can grow at different paces.

FruitChain (Pass and Shi, 2017) is another novel approach that targets improving the Bitcoin protocol. FruitChain implements a fair blockchain consensus protocol in which honest parties are represented proportionally to the owned

computation power. To allow fair mining, FruitChain uses cryptographic puzzles with smaller difficulty values.

2.2.2.2 Proof of Stake based blockchains

The majority of Proof of Stake (PoS) blockchains can be considered committee-based on which the ownership of tokens is used to construct committees. However, only a few blockchains can be considered pure PoS, where proof of token ownership is used to select a leader to propose a block.

Leaders in PoS systems are assigned slots to propose blocks according to the amount of stake in the system. Because all parties know the stake distribution in advance, the most crucial challenge for this family of blockchains is producing an unbiased randomness source to use in the leader election. Failing to produce a true randomness source might endanger the safety of the blockchain system. However, randomness is not an issue for PoW-based blockchains because solving a cryptographic puzzle is itself a random process, and eventually, someone solves one puzzle and submits a block.

The most important example of this family is Ouroboros. Ouroboros is a family of blockchains that implements a provably secure PoS-based blockchain. Using PoS, Ouroboros eliminates the use of energy. In Ouroboros, nodes with the biggest stake in the system are assigned a slot to propose a block: Ouroboros implements a verifiable random number generator using the public ledger, and stakeholders are assigned slots using this generator.

2.2.2.3 Committee based blockchains

A big majority of blockchains use cryptographic puzzles, and ownership of tokens to construct committees of nodes with different responsibilities. It is hard to classify these blockchains as pure PoW or PoS because they do not rely on a single node, or leader, to propose a block. Some of these blockchains use classical consensus algorithms like Paxos and PBFT in committees to decide on blocks. Also, some of them implement new consensus algorithms similar to classical consensus algorithms. In both cases, these blockchains leverage the availability of distributed ledgers to circumvent the close membership requirement of classical consensus algorithms: specifically, using PoS and cryptographic puzzles they reconfigure the committees frequently to have consensus in an open setting in a way that committee membership always tracks the current stake or computation power distribution.

Byzcoin (Kogias et al., 2016) is a novel consensus protocol that relies on cryptographic puzzles to elect leaders. Unlike Bitcoin, the proposed block by the leader, the miner, is decided by a committee of nodes. The committee is constructed from the last n block miners, and n is a protocol parameter that is set at bootstrap time. In Byzcoin, the committee uses a PBFT instance to decide on the block proposed by the leader: the leader proposes blocks, and the last $n-1$ puzzle solver decides on the block using PBFT. Also, Byzcoin makes use of a tree-based communication scheme and a cosigning scheme to decide on blocks in a fast and efficient manner. As in Bitcoin-NG, the leader leads the instances of PBFT until a new leader is elected by solving a crypto puzzle. This way, Byzcoin effectively separates leader election from transaction serialization. When a new miner solves a new cryptographic puzzle, the oldest member of the previous committee is evicted, and the new miner leads the consensus instance until a new block is mined.

Algorand (Gilad et al., 2017) is one of the most known examples of committee-based blockchains, and it is one of the most scalable committee-based blockchains. Algorand constructs committees of nodes using cryptographic sortitions that make use of Verifiable Random Functions (VRF). Cryptographic sortitions of Algorand weight the nodes according to the owned stake in the system to protect the committees against Sybil attacks. Cryptographic sortitions produce proof of election that can be validated by any node in the system by relying on public information recorded in the ledger. Any protocol message of Algorand contains the proof of election produced by cryptographic sortitions. Using cryptographic sortitions, Algorand elects leaders and committee members. Leaders proposed blocks, and committee members votes for blocks to decide on blocks. Unlike other committee-based blockchains, committee members of Algorand are not known in advance, and this provides a strong Denial of Service (DoS) attack resilience to Algorand because an attacker can not target the committee members.

Rapidchain (Zamani et al., 2018) is a sharded blockchain and committee-based blockchain. Rapidchain creates a disjoint set of committees in which each committee is in charge of a shard of the blockchain. Rapidchain relies on cryptographic puzzles to protect the system against Sybil attacks. The time in Rapidchain consists of daylong epochs, and in each epoch nodes that want to join the system or want to stay in the system solve a cryptographic puzzle to attend the next epoch. In Rapidchain, cryptographic puzzles are not on the critical path of the consensus, and this removes many inefficiencies caused by

cryptographic puzzles like low throughput and high latency. We will provide further details about Rapidchain's committee mechanism in the sharded blockchains' subsection.

Snow White (Bentov, Pass, and Shi, 2016) is a committee-based blockchain that uses PoS to construct committees. Snow White divides time into epochs, and at the beginning of each epoch, a new committee is decided by the previous committee members. For that purpose, nodes rely on the blockchain's state by removing a suffix of it to obtain a stable prefix that is the same for all nodes with a high probability. Snow White assumes the availability of an unbiased leader election primitive to elect leaders in an epoch. Leaders are elected from the committee members therefore their identities are known in advance by all nodes in the system.

2.2.2.4 Proof of X based blockchains

PoW solves the Sybil problem by excessive energy consumption, devastating for the economy and ecology. PoS systems replace the cryptographic puzzles with ownership of a stake in the system, and this makes the rich richer because nodes with a high stake in the system can contribute more and benefit more from the system. The natural question is whether there are other mechanisms alternative to PoW and PoS. Proof of X-based blockchains propositions remove the PoW and PoS from the equation to provide energy-efficient and fair consensus algorithms.

Proof of Elapsed Time (PoET) is an alternative blockchain consensus algorithm that relies on Trusted Execution Environments (TEE). They are implemented in CPUs to provide confidentiality and integrity for the code running in TEE. PoET uses Intel's TEE called Intel Software Guard Extensions (Intel SGX). PoET replaces the cryptographic puzzles of PoW consensus with random sleep times, and the correctness of the behavior of a node is guaranteed using Intel SGX. For example, a PoET node creates a random lottery ticket that selects a future time to submit a block. When the time comes, the PoET node submits a block for the current block index if no block is received. Using this mechanism, PoET provides a consensus algorithm similar to PoW with an eventual consistency guarantee. Furthermore, in PoET, a node can only issue a single lottery ticket using a single CPU with TEE: this handles the Sybil problem because nodes need to own many CPUs with TEE to issue many lottery tickets. Although the original PoET protocol lacked a formal definition, later studies

filled this gap by providing a formal description and investigating the properties of PoET.

Proof of Retrievability for large files (Juels and Kaliski, 2007)(PoR) is an earlier proposition than Bitcoin. PoR is a cryptographic proof of knowledge designed for large files. PoR is designed to use in the context of semi-trusted digital data storage services: users of the storage service store data, and using PoR; they can make sure that the file is not deleted and not modified by the storage service. Specifically, a user (verifier) can verify that the storage service (prover) possesses a file or data object. PoR targets efficiency, and for that purpose, PoR requires that the prover touches a small percent of the large file to produce proof. For that purpose, PoR encrypts the large file and randomly embeds random valued encrypted check blocks inside the file. This way, the verifier can check the availability of random valued check blocks in specific positions. In case of file update or deletion, the prover can not provide the correct check blocks. As we will see later, the PoR approach was adopted by some blockchains.

Proof of Space (Dziembowski et al., 2015) is an alternative approach to replace the cryptographic puzzles of PoW with a different kind of puzzle that provides similar properties without energy usage: to make a proposition, a proposer needs to invest a considerable amount of effort as in PoW. Proof of Space targets the elimination of excessive energy use and special equipment use. Because disk space is a considerably cheap and abundant resource, Proof of Space implements puzzles that use disk space. Users of the Proof of Space can play two roles: prover and verifier. Prover P stores a fixed-size file F, and verifier V stores a small amount of information about the F. In this model, verifiers are service providers, and provers are clients or consumers of the service. When a prover wants to use the service, the verifier sends a challenge to prove that a prover stores the file. If the prover can answer the challenge with short proof, it can benefit from the service; otherwise, it can not.

Spacemint (Park et al., 2018) is a blockchain based on the Proof of Space consensus algorithm. Spacemint refines the Proof of Space idea to use in blockchains. In Spacemint, miners dedicate a hard disk space and initialize the hard disk space using a special function. Later, miners produce proofs of the election using the dedicated disk space. On average, it takes 30 seconds to produce an election proof. Using Proof of Space, Spacemint limits the number of miners to decrease forks. Spacemint does not use disk space for useful purposes; dedicated space contains data produced by a deterministic algorithm.

Permacoin (Miller et al., 2014) is a Proof of Retrievability (PoR) based blockchain consensus algorithm. In PoR, nodes provide proof that a node invests memory or disk to store a specific file. Permacoin aims to repurpose the mining resources of Bitcoin to achieve a more broadly useful goal: distributed storage service for archival data. Permacoin implements a highly distributed peer-to-peer file storage to store large, publicly valuable data archives. Permacoin assumes that file F is too big to be stored by a single node, and chunks of the file must be stored on multiple nodes. Permacoin also defines a lottery mechanism based on PoS where nodes produce a proof of election using PoR to propose a block.

2.2.2.5 Sharded blockchains

Sharded blockchains follow a different path to increase performance: they distribute nodes into smaller committees called shards, where each committee sustains a local blockchain. In a sharded blockchain, nodes need to have a consensus inside a shard, and blocks and transactions are only disseminated inside a shard; therefore, the sharding approach decreases the communication cost of consensus significantly.

Sharded blockchains can benefit from the classical BFT consensus algorithm with the help of small committee sizes and close shard membership. The performance of a sharded blockchain increases linearly with the number of shards because shards of a blockchain can process transactions in parallel. The downside of this approach is the increased complexity of the protocol because most sharded blockchains require a cross-shard transaction processing protocol that requires cross-shard communication. Also, these blockchains need subprotocols to map transactions onto specific shards to be processed and to ensure the health of shards in terms of correct and faulty node ratio. Furthermore, most sharded blockchains use cryptographic puzzles to construct shards: nodes need to solve cryptographic puzzles to join a shard. Finally, most of them use special committees assigned to conduct specific tasks, such as verifying solutions of cryptographic puzzles or assigning new nodes to shards.

Elastico (Luu et al., 2016a) is a sharded blockchain that creates committees of nodes and automatically changes the number of committees according to the number of nodes. Elastico uses a classical BFT consensus algorithm (PBFT) in a committee to order user transactions. Nodes of Elastico solve cryptographic puzzles to join or stay in the system. In Elastico, a special committee combines the consensus results of all other committees and shares them with all nodes

in the system. Finally, Elastico considers a directory committee that keeps track of nodes and their IP addresses so that nodes in the same committee can communicate directly by getting information from the directory committee.

One of the most notable sharded blockchains is Rapidchain (Zamani, Movahedi, and Raykova, 2018). Rapidchain constructs committees of nodes, and each committee sustains a blockchain. Rapidchain uses a synchronous consensus algorithm in a committee to decide on blocks. The number of committees of Rapidchain is static, and it does not change according to the number of nodes in the system. Rapidchain maps transactions into committees according to the transaction ID calculated by hashing, making the majority of transactions cross-shard that requires cross-shard communication. Rapidchain divides time into successive epochs, and in each epoch, nodes solve cryptographic puzzles to join the system or to stay in the system. A specific committee validates cryptographic puzzles to authorize nodes to join or stay in the system, and the same committee rotates and swaps the shard members to protect the shards' health. In Rapidchain, Nodes use Kademlia (Maymounkov and Mazieres, 2002) protocol to communicate with other shards.

Monoxide (Wang and Wang, 2019) is another sharded blockchain, and it names shards as zones. Each zone of monoxide process user transactions concurrently. In a zone, nodes use the PoW consensus algorithm. Each zone sustains a blockchain, and transactions and blocks are replicated inside a zone, but block headers are replicated system-wide. Like Rapidchain, Monoxide also benefits from the Kademlia protocol for cross-zone communication. To handle cross-shard transactions, Monoxide considers relay transaction, which transfers the value from the remote zone to the local zone. It is easy to process relay transactions locally in Monoxide because block headers are replicated globally.

2.2.2.6 Permissioned and permissionless blockchains

Blockchain propositions can be categorized into two broad categories permissioned or permissionless.

In permissioned blockchains, a trusted authority allows nodes to access the system; therefore, the identities of nodes are known in advance. As a result, permissioned blockchains can benefit from classical consensus algorithms to provide a stronger safety guarantee and fast confirmation time. Also, they do not need to handle Sybil and DoS attacks because they can easily exclude misbehaving nodes from the system.

The most important example of a permissioned blockchain is Hyperledger Fabric (Dhillon, Metcalf, and Hooper, 2017). Hyperledger Fabric assigns roles to nodes in the system: validators and orderers. Validators validate and execute the client requests, and orderers order them using a consensus algorithm. This scheme is known as execute-order-validate, where transactions are first executed, and outputs are ordered. In Hyperledger Fabric, orderers do not execute transactions and are unaware of the meaning of transactions. The consensus algorithm of Hyperledger is pluggable, which means that it is not hard coded, and the consensus algorithm can be chosen according to the requirements.

Permissionless blockchains do not have any trusted authority, and any node can join the system without restriction. Permissionless blockchains must handle the Sybil problem and possible DoS attacks using techniques listed in the previous section, such as PoW, PoS, or PoX. As we have already mentioned, some permissionless blockchains use classical consensus algorithms, but most of them use eventually-consistent consensus algorithms to scale the consensus. This family of blockchains generally prioritizes scalability over performance. Some of these blockchains might restrict when nodes can join the system: for example, in Rapidchain (Zamani et al., 2018), nodes can join the system at the beginning and epoch, and an epoch is a day-long time interval.

2.2.2.7 Leader-based and leaderless blockchains

We can classify blockchains into two broad categories: leader-based and leaderless. Leader-based blockchains are the ones that elect a distinct leader to propose blocks. All the mentioned blockchains up to now are leader-based.

On the contrary, leaderless blockchains do not elect leaders. One of the most notable of them is Avalanche (Rocket et al., 2019). Unlike blockchains, the Avalanche system constructs a Directed Acyclic Graph (DAG) of transactions. Therefore, in Avalanche, there is no concept of the block, and each transaction selects one or more parent transactions by containing a hash of them. Avalanche replaces the consensus with a random sampling of the network, and a node samples a set of nodes from the system to learn their status about a transaction. A node updates its behavior according to the behavior of other sampled nodes.

Another important leaderless consensus algorithm is Redbelly (Crain, Natoli, and Gramoli, 2021). It is a permissioned blockchain system, and it builds on top of Democratic BFT (DBFT) (Crain et al., 2018), which is a special consensus algorithm designed for blockchains, and it does not need a leader to terminate. In Redbelly, permissioned nodes order transactions using DBFT:

multiple nodes propose a small block, and a subset of them are agreed in the form of a superblock. A superblock is appended to the ledger.

2.3 Conclusions

In this chapter, we presented the fundamental concepts for understanding blockchains: we started the discussion with basic information about distributed systems and cryptographic primitives. Later, we focused on the consensus problem and state machine replication. We studied the properties of the Bitcoin protocol which is the first blockchain protocol, and understanding its properties is essential to understanding other blockchain proposals. We have listed the limitations of the Bitcoin approach. We classify the important blockchain proposals that aim to implement more efficient and effective blockchains, and mentioned their important characteristics.

3 Related work

3.1 Multiple leader approach to improve the performance

Most blockchain consensus algorithms use a leader: an elected leader proposes a block for a given block index, and other nodes try to decide on the block. This approach significantly reduces the communication cost of consensus because nodes are trying to decide on a single value.

The single-leader approach has several drawbacks. First, it causes an unbalanced communication and computation scheme because the leader must assign all requests a sequence number. Clients close to the leader might benefit from low latency, while others experience higher latency values. A malicious leader can slow down the consensus by submitting requests at the latest possible time to decrease the throughput. Also, a malicious leader can censor some clients by omitting their requests. Finally, resources available to the leader, such as CPU and network bandwidth, limits achievable throughput values.

In this section, we will investigate the state-of-the-art propositions that target the inefficiencies stemming from the single leader in classical and blockchain consensus algorithms.

3.1.1 Classical consensus algorithms with multiple leaders

Mencius (Mao, Junqueira, and Marzullo, 2008) is one of the earliest examples of multiple leader consensus algorithms. First, Mencius observes that an unbalanced communication scheme caused by the single leader in the Paxos algorithm might affect clients differently in a Wide Area Network (WAN). Later on, Mencius targets improving the WAN performance of the Paxos algorithm by extending Paxos with multiple leaders to provide throughput increase and to provide balanced communication cost, which is distributed evenly between all replicas. Mencius partitions the sequence numbers among replicas, and each replica becomes a leader for its sequence numbers by proposing client requests

using these sequence numbers. Assignments of sequence numbers to replicas are known in advance. Each replica can propose values for its sequence numbers: the downside of this approach is that replicas can not commit a request before committing all requests with lower sequence numbers. That might increase the latency of the system. To handle this case, Mencius introduces piggybacked SKIP messages that allow a replica with a low client load to skip its sequence numbers without incurring a high message latency.

BFT-Mencius (Milosevic, Biely, and Schiper, 2013) is another approach inspired by Mencius. BFT-Mencius builds on top of the PBFT protocol and benefits from multiple leaders. BFT-Mencius provides bound time latency guarantees for correct clients in the presence of Byzantine faulty replicas: a request submitted by a correct client is processed by the system in a bounded time interval. This property of the BFT-Mencius guarantees that a leader can not censor a client's requests forever. In the presence of Byzantine faulty replicas, the use of simple SKIP messages, as in Mencius, does not work because a faulty replica might send a SKIP message for a sequence number, and it can propose a value using the sequence number. To handle this case, BFT-Mencius developed a new abstraction called Abortable Timely Announced Broadcast (ATAB), and it implements ATAB using a PBFT-like protocol. BFT-Mencius partitions the sequence numbers among replicas, as in Mencius. It implements a consensus algorithm using ATAB primitive where nodes can SKIP their turns as in Mencius without violating the safety properties in the presence of Byzantine nodes.

Mir-BFT (Stathakopoulou, David, and Vukolic, 2019) is a recent proposition that builds on top of PBFT, and it targets improving the WAN performance of the PBFT protocol where the latency of messages is high, and also it handles request duplication attacks where a malicious client submits duplicate requests to decrease the performance of the system. As in previous propositions, Mir-BFT partitions the sequence numbers among replicas. Further, it partitions the hash space of the client requests into k disjoint bucket, and each replica is assigned an equal number of buckets. Replicas can propose client requests from their transactions bucket using assigned sequence numbers. To handle the censoring problem, Mir-BFT frequently rotates the bucket assignment of the replicas; therefore, a malicious replica can not delay the requests for a correct replica forever.

ISS (Stathakopoulou, Pavlovic, and Vukolić, 2022) protocol builds on top of the idea of Mir-BFT: it uses transaction buckets and bucket rotation techniques. However, unlike Mir-BFT, ISS provides a truly generic construct that can be

used to improve any leader-based BFT consensus algorithm with multiple leaders. ISS designs Sequenced Broadcast(SB), which is a Byzantine total order broadcast protocol. They prove that ISS can be implemented using a BFT consensus algorithm, and any target consensus algorithm can be used to implement SB. Later on, instances of SBs are multiplexed to implement a multiple-leader consensus algorithm.

OMADA (Eischer and Distler, 2019) enriches BFT consensus algorithms with multiple leaders: it is also a generic construct as ISS. Unlike previous approaches, it adopts a technique similar to sharding to enrich BFT consensus algorithms with multiple leaders. OMADA creates multiple consensus groups; each group runs an instance of the target consensus algorithm. Also, OMADA targets decreasing the cost of communication by keeping the size of consensus groups as small as possible. As in other propositions, OMADA partitions the sequence numbers among consensus groups, and each consensus group proposes values using assigned sequence numbers. Finally, a node of OMADA can take a role in more than one consensus group, which is a technique to efficiently utilize the machine's capacity: nodes with more resources can participate in multiple groups, while nodes with low resources can contribute only to a single group. In OMADA, clients select a consensus group and send requests to that group to be serialized. OMADA masks the Byzantine faults inside groups and uses a crash fault resilient consensus algorithm to mask unreachable groups during the transaction execution phase.

RCC (Gupta, Hellings, and Sadoghi, 2021) is another proposition that uses multiple leaders to utilize the system's available resources better and to remove the bottleneck caused by a leader. Like ISS and OMADA, RCC is also a general construct. In RCC, not all replicas need to behave as primary, and the number of primaries is a protocol parameter. The sequence numbers are distributed among primaries, and each primary concurrently proposes client requests using available sequence numbers. RCC is tested with PBFT protocol using a different number of primaries.

3.1.2 Blockchains with multiple leaders

The multiple-leader approach is less common in blockchain consensus algorithms than classical consensus algorithms: most blockchain propositions target large open systems, and it is relatively hard to distribute the sequence number among

contributing nodes in open systems. However, techniques described in the previous subsection can be used by blockchains that employ classical consensus algorithms in a closed setting.

One notable example of multiple leader blockchain consensus algorithms is OHIE (Yu et al., 2020b), which we briefly described in Section 2.2.2.1. OHIE builds on top of the Bitcoin protocol, and like Bitcoin, it uses cryptographic puzzles. Unlike Bitcoin, OHIE constructs parallel chains that each have a genesis block. The OHIE considers k parallel chains (where k is in the order of 1,000) and decreases the difficulty of cryptographic puzzles to elect k leaders for a unit time interval; therefore, there are k blocks submitted on average for every unit of time. Decreasing the difficulty of cryptographic puzzles causes an increase in forks: OHIE handles this by assigning each leader one of the parallel chains using the hash of the mined block. Because the hash of the block is not available before mining, a miner uses a Merkle tree to point all previous blocks on all chains: a Merkle tree is constructed using the hashes of blocks on k chain, and the root of the Merkle tree added into the block header as the hash of the previous block. Finally, OHIE considers blocks around 20 KBs: smaller blocks propagate faster in the network, and it causes lower block dissemination latencies. OHIE does not handle the problem of disjoint block submission. Therefore, different blocks can contain the same transaction, which could decrease the resource utilization in the system because the same transaction can be appended to the blockchain in different blocks.

3.2 Efficient dissemination techniques to improve performance

Numerous propositions try to remove inefficiencies of blockchains in the network plane; many are too specific and designed for a single blockchain. Also, there are propositions that implement efficient gossip dissemination protocols, potentially valuable for large-scale distributed systems, and blockchains can benefit from them. Because of this reason, in this section, we will focus on efficient gossip dissemination protocols.

3.2.1 General purpose efficient dissemination techniques

The most notable efficient gossip dissemination techniques are chunk-based gossip dissemination, where a message is chunked into multiple pieces, and individual chunks are disseminated. This technique eliminates the problems that stem

from store-and-forward gossip dissemination used in blockchains. Also, chunk-based gossip dissemination introduces the possibility of using erasure coding, where chunks are erasure-coded to protect against loss. This subsection will look at important examples of chunk-based gossip dissemination protocols.

One of the earliest examples of multi-chunk gossip dissemination is SplitStream (Castro et al., 2003). It aims for efficient dissemination of messages and fair distribution of the dissemination cost among contributing nodes. SplitStrem considers a structured peer-to-peer network where the communication pattern of processes is scheduled in advance to obtain optimum latency and communication cost. SplitStream chunks a large message into multiple pieces and constructs disjoint dissemination trees in a deterministic manner for each chunk. In a dissemination tree, only inner nodes disseminate the message. A node has different roles in different trees; therefore, each node contributes to disseminating messages in one of the trees. SplitStrem's evaluation considers a structured peer-to-peer network and does not consider Byzantine faulty nodes. Also, Splitstream does not consider parity use of parity chunks.

Sanghavi, Hajek, and Massoulie, 2007 investigate the cost of gossip dissemination in an unstructured setting where nodes randomly contact other nodes to send or receive messages. They propose a gossip dissemination protocol, INTERLEAVE, that relies on multi-chunk gossip dissemination. They compare classic gossip with multi-chunk gossip dissemination from a theoretical point of view. They also provide an analysis of the optimum gain that can be achieved from splitting a multi-chunk message compared to sending a single large message. Their analysis shows the benefit of multi-chunk gossip dissemination theoretically. Although they mention the possible benefits of using erasure coding in push-and-pull gossip dissemination, they do not investigate it. Later on, Lo Cigno, Russo, and Carra, 2008 investigate the performance of INTERLEAVE protocol by using simulations. They aim to quantify the properties of the INTERLEAVE protocol.

A critical use case for multi-chunk gossip dissemination is live-streaming. Multi-chunk gossip dissemination is indispensable for live-streaming because of the size of the messages. Bar gossip (Li et al., 2006), LiFtinG (Guerraoui et al., 2010), and AcTinG (Mokhtar, Decouchant, and Quéma, 2014) are protocols designed to handle rational nodes in streaming systems. Rational nodes do not want to contribute to disseminating messages to decrease resource consumption, and they are risk averse, meaning that if there is a risk of exclusion from the system, they will stick to the protocol. Although they consider authenticated messages,

none of these protocols use erasure coding or an efficient chunk authentication mechanism. Also, their fault model only considers rational behaviors that are a subset of Byzantine behaviors.

In the context of live-streaming, another important proposition is Gossip++ (Frey et al., 2010). Gossip++ uses push-and-pull gossip mechanisms together to implement a hybrid gossip protocol. It uses erasure coding to improve dissemination performance. In Gossip++, the source node chunks a large message into 100 chunks and adds five parity chunks. Gossip++ considers a few parity chunks, and the source node forwards each chunk multiple times. Gossip++ uses different $fanout$ values for source nodes and other nodes. These $fanout$ values are respectively 5 and 8. The evaluation of Gossip++ considers only 200 nodes, a considerably small system by today's standards. Finally, the evaluation considers only rational nodes.

3.2.2 Efficient dissemination techniques for blockchains

There exist different inefficiencies regarding the network plane of blockchains. The first is high latency caused by gossip dissemination with the store-and-forward mechanism. Many blockchains disseminate blocks using the store-and-forward mechanism where a node receives the entire block and validate it using a particular predicate: if the block is valid, it forwards it $fanout$ other nodes. This construction is necessary to protect the system again DoS attacks, but it increases the latency of block propagation, increasing the probability of forks on the blockchain. The second inefficiency stems from the dissemination of transactions; in most blockchains, transactions are disseminated twice in the network by gossip dissemination: once the issuer announces it and the second time when it is appended into a block. Again, this decreases the system's performance and increases the latency because of inefficient use of available bandwidth resources.

The earliest attempt to improve the performance of Bitcoin's block dissemination performance is Bitcoin Relay Network (Corallo, 2015): it is a set of trusted nodes deployed on the Bitcoin network, and they are connected. In the relay network, nodes disseminate blocks with minimum validation to decrease dissemination latency caused by the full block validation. Other nodes connected to the nodes of the relay network should validate blocks fully before disseminating. Later on, The Fast Internet Bitcoin Relay Engine (Corallo, 2016b)(FIBRE) is proposed by the same author, which targets improving the performance of the Bitcoin relay network further by replacing transport level protocol TCP with

UDP, and by using erasure coding so that missing network level packages will be recovered using erasure coding on the application level.

The problem that stems from the dissemination of transactions is attacked by several propositions that eliminate the dissemination of transactions twice. The most notable of them is *compactblocks* (Corallo, 2016a) that are proposed in the context of Bitcoin to decrease the block dissemination time and bandwidth usage of nodes. In the compact blocks, the block body does not contain the transactions themselves but the ID of transactions that are calculated using a hash function; This decreases the size of blocks. Furthermore, compact block proposition does not use cryptographically secure hash functions like SHA256 (NIST, 2015) that produces 32 bytes long message digest but uses SipHash (Aumasson and Bernstein, 2012) that produces 8 bytes long message digest. Finally, the compact block proposition envisions the drop of 2 most significant bytes from SipHash output to make transaction IDs 6 bytes long. Upon receiving a block, each node check transaction IDs inside the block, and if there are any missing transaction, a node requests them from other nodes by sending a request that contains the transaction ID.

Xtreme Thinblocks (Tschipper, 2016) is another proposition that targets the same goal with compact blocks. Like compact blocks, Xtreme Thinblocks replaces transactions inside the block body with transaction IDs, using eight bytes-long transaction IDs. Also, Xtreme Thinblocks make use of Bloom filters (Bloom, 1970) to decrease the message latency to request missing transactions inside a received block: node A requests an Xtreme Thinblock from another node B by sending a bloom filter that is constructed by using all known transaction but not appended to the blockchains. Upon receiving the bloom filter, node B sends the Xtreme Thinblocks with all missing transactions that are not known by node A. This method decreases the latency caused by missing transactions inside a block. Graphene (Ozisik et al., 2019) protocol further improves the mechanism by combining bloom filters and Invertible Bloom Lookup Table (Goodrich and Mitzenmacher, 2011)(IBLT): it implements a set reconciliation protocol using both data structure previously mentioned, and it makes use of short transaction IDs of 8 bytes as in previous propositions. Graphene decreases the size of blocks further compared to Xtreme Thinblocks because combined of used Bloom filters and IBLT.

3.3 Conclusions

In this chapter, we focused on related work targeting to improve the performance of existing distributed systems and blockchains by addressing bottlenecks in consensus and network layers. On the consensus layer, we focused on the multiple-leader approach, which addresses problems arising from the single-leader approach. On the network layer, we focused on proposals that improve the performance of message propagation.

4 Alder

4.1 Introduction

Leader-based blockchain consensus protocols are deployed at the heart of major blockchains (e.g., Bitcoin, Ethereum, the Hyperledger suite, Algorand). In these blockchains, leaders are frequently elected to propose blocks and carry out other protocol-specific tasks. *ALDER* is a general construction to enrich leader-based blockchain consensus protocols with multiple leaders to increase performance by removing bottlenecks in consensus and network layers. In this Chapter, we present *ALDER* and its evaluation.

ALDER multiplexes blockchain consensus protocols to append not one (large) but several (smaller) blocks to the blockchain. Specifically, ALDER lets multiple leaders propose candidate blocks while allowing the blockchain system to compose and agree on a *macroblock*, an ordered set of blocks, to extend the chain. ALDER elects leaders by leveraging candidate leaders not exploited in the original consensus protocol. Then, elected leaders independently propose concurrent blocks containing disjoint sets of transactions, the union of which constitutes a macroblock. Since ALDER builds on existing blockchain protocols, the resulting protocols inherit the safety and liveness properties of the parent protocol.

To assess the effectiveness of ALDER, we apply its principles to three major blockchains: RapidChain (Zamani, Movahedi, and Raykova, 2018), an efficient sharded blockchain, Algorand(Gilad et al., 2017), a scalable proof-of-stake blockchain, and Bitcoin, the iconic proof-of-work blockchain. Our experimental evaluation of *ALDER* consists of large-scale deployments involving up to 10,000 nodes deployed on up to 100 physical machines.

In the following Sections, we provide a detailed description of *ALDER* and its evaluation.

4.2 ALDER's foundations

ALDER assumes the availability of a permissionless leader-based blockchain protocol to build on. The available blockchain protocol can adopt the synchronous or eventually synchronous network model, and ALDER can be applied in both network models.

The blockchain protocol incorporates the election of leaders, the creation of blocks, the dissemination of blocks throughout the system, and a consensus algorithm that allows all nodes to agree on the block that effectively extends the latter chain. The blockchain protocol satisfies safety and liveness properties that have been translated into *common prefix*, *chain quality* and *chain growth* defined in previous work (Badertscher et al., 2018; Pass, Seeman, and Shelat, 2017; Pass and Shi, 2017). Abstracting from the specifics of consensus protocols employed in leader-based permissionless blockchains, we provide their informal definitions of these properties as follows:

- *Chain-quality*: the number of blocks contributed by the adversary is not too large, i.e. any (large enough) subset of an honest node's chain contains blocks from honest nodes.
- *Chain-growth*: the chain of any honest node grows at a steady rate, proportional to the number of time steps.
- *Common prefix*: the chains of all honest nodes must be identical, except for a few tail blocks that are not yet stabilized, i.e., if two honest nodes discard a sufficient number of blocks from their respective chains, they obtain the same prefix.

4.2.1 Bottlenecks in improving blockchain performances

The performance of a blockchain is generally quantified using two metrics: throughput, which is the amount of data that a blockchain protocol can append to the chain per unit of time, and latency, which is the duration before a generated block is added to the ledger. Effective ways to improve the performance of blockchain systems are to increase the frequency at which blocks are generated and increase the block size. While these solutions seem straightforward to implement, they raise several challenges in practice.

Increasing the block size can be counterproductive because nodes must receive the entire block before disseminating it to avoid spreading an invalid block. Thus, larger blocks would be added to the blockchain at a slower rate, which

in the optimistic scenario, would maintain a given level of throughput with increased latency.

Increasing the frequency of block generation also has drawbacks, for example, with proof-of-work or proof-of-stake based blockchain consensus protocols, whose probabilistic termination properties allow the existence of forks (e.g., Bitcoin or Ethereum). As the frequency of block generation increases, so does the number of forks appearing in the system, resulting in a larger share of generated blocks not appended to the ledger. Consequently, this lack of efficiency (in this example, mining power efficiency) make the envisioned performance improvement strategy far from optimal, with a significant proportion of blocks generated quickly and not used to improve blockchain throughput. At this point, the fundamental limits of disseminating a block in the network and adding it to the chain are reached, preventing any further performance gains for blockchain protocols with these strategies.

4.2.2 ALDER: multiplexed blockchain consensus

ALDER aims to improve the performance of existing leader-based blockchain protocols by circumventing the bottlenecks encountered when increasing the block size or the block generation frequency. ALDER consists in multiplexing the execution of a blockchain consensus protocol instance. We call multiplexing a consensus instance the process by which the nodes in the system agree to append a *macroblock*, i.e. a set of blocks totally ordered, to the blockchain instead of a single block per round. The blocks composing the resulting macroblock contain disjoint sets of transactions.

The construction of ALDER allows circumventing the bottlenecks described above by fine-tuning the operations of the multiplexed blockchain protocol: either by changing the size of the blocks within the macroblock (thereby increasing the size of the resulting macroblock) and keeping the same macroblock generation frequency or by increasing the macroblock generation frequency and keeping the same block size. With these new capabilities, nodes in the system can propose and append multiple small blocks per consensus round instead of a single large block. As a result, the dissemination of these blocks from different nodes in the system optimizes the consumption of network resources and increases the throughput of the blockchain system.

Finally, the multiplexed version of a blockchain model preserves its safety and liveness properties, namely chain-quality, chain-growth, and common prefix.

This is achieved by keeping the properties of the underlying consensus protocol unaltered.

4.3 Design of ALDER

In this section, we present the principles of ALDER that allow to multiplex a leader-based Blockchain Consensus Protocol (*BCP*) and increase its performance. We refer to *BCP++* for the resulting blockchain consensus protocol in the remainder. Figure 4.1 illustrates this transformation on a *BCP* in its canonical form that executes in time-based epochs or rounds: leader election, block proposal, block dissemination, and consensus. In the blockchain models that we consider, the leader election mechanism identifies a node (sometimes several, either by design principle or by side effect) that must propose a candidate block to the rest of the nodes in the system. Once the block(s) is(are) disseminated, the nodes execute a consensus algorithm to reach an agreement on a block to consider it a valid extension of the *BCP* blockchain.

As illustrated in Figure 4.1, ALDER extends *BCP* to *BCP++* which executes its four main components as follows: In each round, *BCP++* elects several leaders based on the election mechanism of the original *BCP*. *BCP++* partitions the transaction hash space into Cl disjoint regions called *buckets*, where Cl is the concurrency level of *BCP++*, which denotes the number of blocks composing the macroblocks. *BCP++* assigns a separate transaction bucket to each leader in a publicly verifiable way. Then, leaders propose blocks with disjoint sets of transactions using the assigned transaction bucket. Finally, *BCP++* runs a multiplexed version of the *BCP* consensus to produce a decision on an ordered composition of the proposed blocks, i.e., a *macroblock*. The nodes in the system wait for this decision before assembling the agreed-upon list of blocks and appending the locally constructed *macroblock* to the chain before moving on to the following protocol round.

To multiplex *BCP*, ALDER leverages three primitives: (1) transaction space partitioning, (2) multiple leader election and bucket assignment, and (3) multiplexed consensus and macroblocks. We now detail these primitives and how they relate to each other.

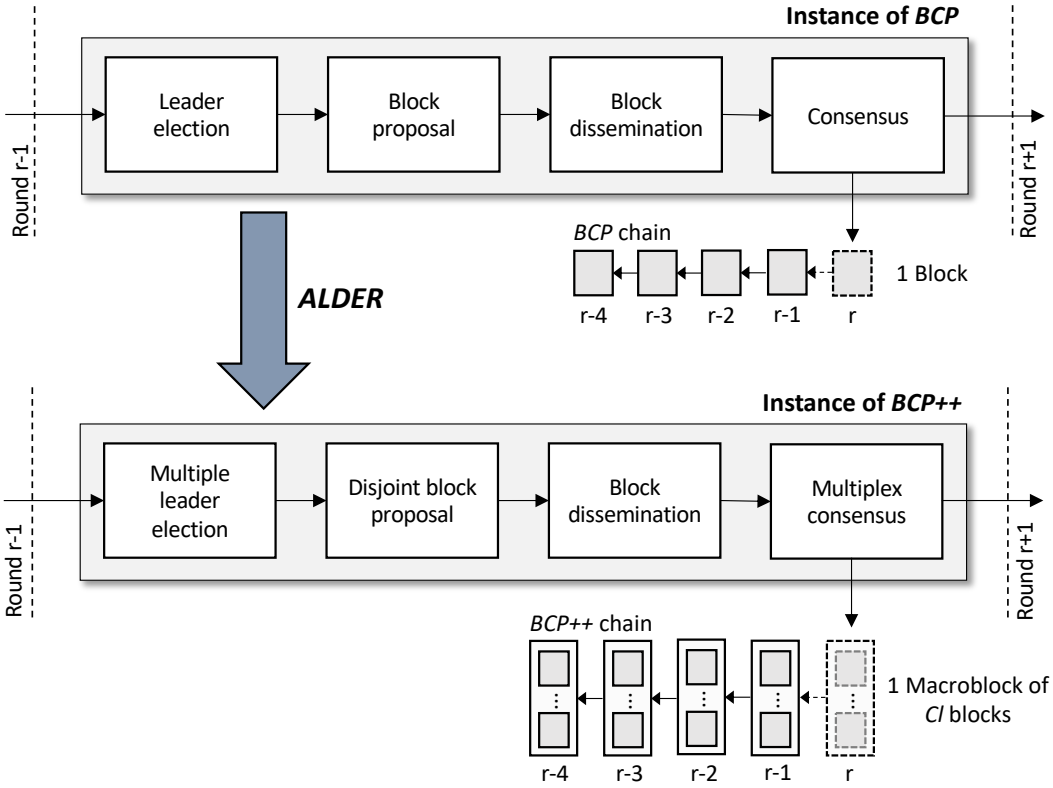

FIGURE 4.1: Multiplexing a blockchain *BCP* using ALDER

4.3.1 Transaction hash space partitioning

ALDER exploits the leader election mechanism of *BCP* to involve more leaders in the blockchain consensus protocol and increase its performance. Specifically, all elected leaders propose candidate blocks to build the next macroblock. However, appending multiple blocks of transactions in a given round poses the problem of duplicated transactions and duplication attacks. Indeed, in a simple approach, leaders create blocks with the transactions they possess. As a result, some blocks could include transactions that could appear in some of the blocks proposed by other leaders. This redundancy would reduce the throughput gain envisioned in our approach and increase the complexity of the transaction execution phase. Duplication of transactions also opens the doors to duplication attacks by an adversary controlling Byzantine nodes. When some of these nodes are elected, the adversary can wait to learn about the blocks proposed by the honest leaders and have Byzantine leaders propose blocks containing the same transactions, thus reducing the performance gain of *BCP++*. To cope with this problem, ALDER creates a transactions hash space, partitions it into regions called *buckets*, and forces each leader to propose blocks containing transactions from a unique bucket.

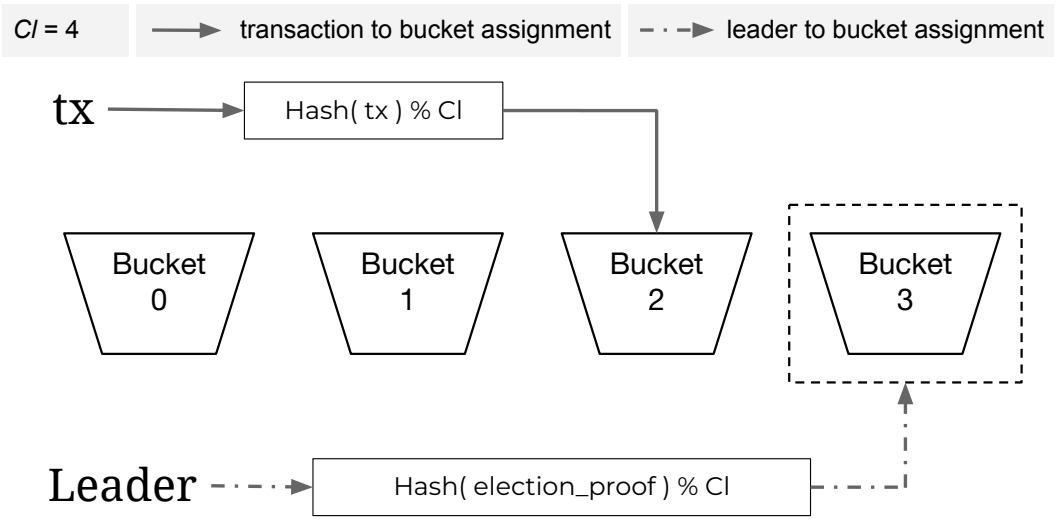

FIGURE 4.2: Mapping a transaction to a bucket, and a leader in a bucket with $Cl = 4$. The *election proof* is a blockchain consensus protocol-dependent value. Note that more than one leader could be assigned to the same transaction bucket.

ALDER partitions the transaction space into Cl non-intersecting buckets of equal size, where Cl is the concurrency level of $BCP{+}{+}$, set at its bootstrap. Figure 4.2 illustrates the mapping of a received transaction, tx, to a bucket, with four buckets available ($Cl = 4$). When receiving a transaction tx, a node first checks its validity before computing its image from a secure cryptographic hash function. Then, to determine the corresponding bucket number that will store the transaction, the modulo Cl operation is used against the hash image of the transaction. Nodes regularly remove transactions from their buckets as transactions appear in macroblocks.

4.3.2 Multiple leader election and bucket assignment

To avoid transaction duplications and duplication attacks, each leader is assigned a bucket of transactions in an unforgeable and publicly verifiable way during the leader election phase. In this way, the proposed blocks represent disjoint sets of transactions.

In the blockchain model we consider, the election of the leader of BCP already includes the pre-identification of several leader candidates for a given round. To elect multiple leaders in $BCP{+}{+}$ and have them propose blocks with disjoint transaction sets, ALDER relies on the leader election mechanism of BCP, and assigns a transaction bucket to each elected leader. ALDER must extend the

election of leaders of *BCP* and ensure that enough leaders are identified to prevent macroblocks from being partially filled, hence preventing the multiplexed blockchain protocol from reaching the envisioned performance gains.

In order to avoid transaction duplications and duplication attacks, ALDER publicly verifiably assigns a transaction bucket to each elected leader. To this end, ALDER leverages the protocol-dependent tamper-resilient *election proof* employed in *BCP* to prove the leader's legitimacy in conducting the task of creating and proposing a block. For example, the election proof of Bitcoin is the solution to the cryptographic puzzle set by the system. Regarding Algorand, the election proof results from a cryptographic sortition procedure evaluating a verifiable random function on publicly known variables. The bucket assignation results directly from the computation of the election proof modulo the number of buckets Cl. Figure 4.2 illustrates the bucket assignment to a leader, with $Cl = 4$. Each block proposed by a leader must exclusively contain transactions whose hashes fall within the bucket assigned to the leader. In this way, any node can validate the correct fabrication of a received block by checking its contents against the bucket number assigned to the block proposer. Using the described construction, leaders of ALDER submit disjoint blocks that can be aggregated to construct one macroblock.

Bucket assignment is straightforward when considering blockchain protocol where the election proof is known before the block creation. However, considering blockchain protocols such as PoW-based ones, the election proof is known after the node has committed to a block and found a solution to a cryptographic puzzle. In section 4.4, we provide examples of bucket assignment techniques with various blockchains.

4.3.3 Multiplexed consensus and macroblocks

Multiplexing consensus requires a decision on not one but a set of proposed blocks. Although multiplexing is highly implementation-dependent, we formalize the consensus results required to preserve the safety and liveness properties of the blockchain model we consider.

The decision takes the form of a vector of block hash values totally ordered by the bucket numbers from which each block originates (from 0 to $Cl - 1$). The vector can contain at most one value for each bucket number. Each node in the system listens for the blocks proposed by the leaders and locally builds a macroblock based on the consensus decision produced. A macroblock is a logical

composition of up to Cl blocks of size bs bytes. The resulting macroblock is of size $Cl \times bs$ bytes. Each block in the macroblock must contain the hash of the previously appended macroblock computed as the hash image of the vector of block hash values totally ordered by the bucket numbers. Figure 4.3 depicts an example of a chain of macroblocks.

A node proceeds through a series of validity and semantic checks regarding the block content and its creation process when receiving a block. Namely, the nodes verify the leader's election legitimacy carried out by the election proof delivered with the block; the validity of a bucket assignment to the leader; the validity of the transactions themselves; the transactions' membership to the claimed bucket; and the presence of the hash value referencing the previously appended macroblock. If any verification returns a negative result, the node discards the block. If the block passes all checks, it is stored in a global data structure until the end of the round, and is passed on to other nodes in the system. This semantic check is not limited to ALDER and can be extended with regard to other protocol-specific constraints on the block.

When a node receives all blocks for a given macroblock in a given round, it applies a global verification to assess double-spending attempts. Indeed, a malicious party might try to double-spend by appending transactions referencing the same unspent transaction outputs into different blocks within the same macroblock. Probabilistically, this can also occur without any malicious intent. In such a case, ALDER deterministically solves double-spending attempts by considering valid the first transaction listed in the total order of transactions inside the macroblock, and discarding the others. More specifically, invalid transactions are left unexecuted. ALDER does not require that each macroblock contain exactly Cl blocks, as shown in the figure. Indeed, the blockchain protocol model considered for BCP includes existing protocols that can produce empty blocks, such as Algorand (Gilad et al., 2017). In addition, consensus on some of the proposed blocks may not be reached.

In consensus protocols with probabilistic termination (such as PoW-based ones), there could be valid blocks generated by legitimate leaders competing for the same bucket number. In this case, ALDER requires an additional protocol-dependent deterministic consensus rule assigning a priority to competing blocks.

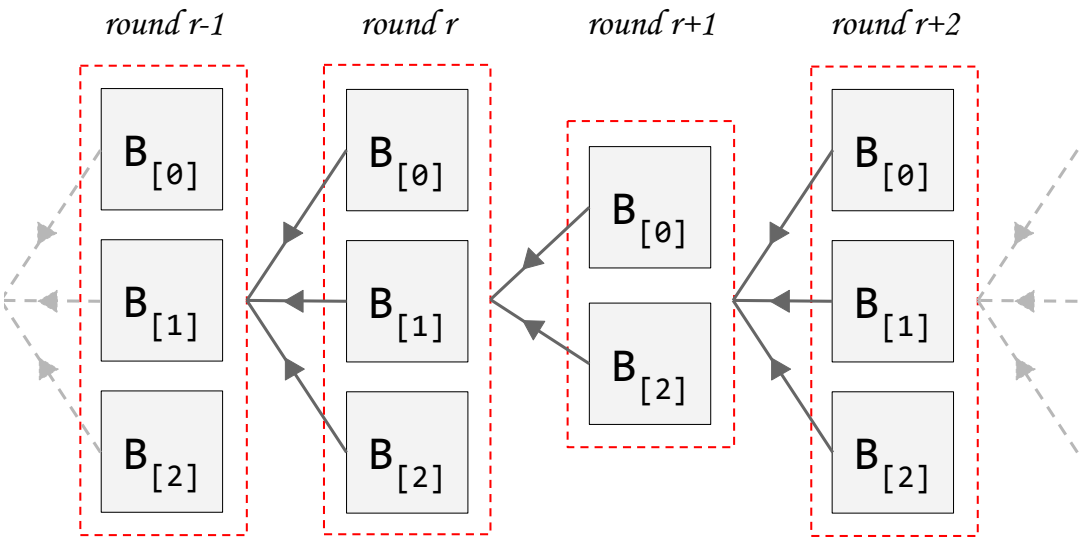

FIGURE 4.3: Example of a chain of macroblocks with a concurrency level Cl of 3. Dashed lines represent macroblocks, and solid squares indicate the composing blocks referencing the previous macroblock hash.

4.3.4 ALDER Security Analysis

In this section, we give a sketch of the security of ALDER. As shown in Figure 4.1, ALDER decomposes *BCP++* into four main components, namely multiple leader election, disjoint block proposal, block dissemination and multiplex consensus. In the following, we analyze each of these components in light of the vulnerabilities they may introduce.

Multiple Leader Election: ALDER extends the election mechanism of the original protocol to elect multiple leaders. In practice, ALDER does not replace the leader election algorithm with another algorithm. Instead, it leverages the existing leader election protocol, which often already elects multiple leader candidates (e.g. in Algorand and Bitcoin). If the protocol does not natively enable multiple leader candidates, ALDER extends the leader election algorithm (e.g. in RapidChain). In both cases, this step does not introduce vulnerabilities into the resulting blockchain beyond those of the underlying leader election protocol (in the same way as Bitcoin, Bitcoin++ will have forks and these will be fixed in a similar way).

Disjoint Block Proposal: Having multiple leaders proposing blocks may introduce vulnerabilities. To solve this problem, ALDER relies on a mechanism that allows candidate leaders to propose disjoint blocks. While correct leaders will faithfully follow the protocol, malicious leaders may very well propose blocks without respecting their assigned bucket. In this case, the proposed block will be rejected by the correct nodes, which will eventually check whether the leaders have built their block using the legitimate bucket.

Block Dissemination: ALDER uses the same dissemination protocol with the base consensus protocol to propagate blocks and other protocol-specific messages; Therefore, no new vulnerability is introduced by this component.

Multiplex Consensus: This part of ALDER is the most complex and could introduce vulnerabilities. Indeed, because the structure of ALDER blocks is more complex, additional verification steps must be performed by the correct nodes in order to validate a macroblock and execute its transactions. Specifically, a correct node that reconstructs a macroblock must perform the following verification. First, it verifies that all blocks composing the macroblock contain the hash of the same previous macroblock. Then, it verifies that all blocks were generated by legitimate leaders. For example, in the case of Bitcoin, the correct node checks the cryptographic puzzle solution. In the case of Algorand, it checks the proof of election resulting from the evaluation of verifiable random functions with the node's stake in the system. In the case of Rapidchain, it checks the proof of election resulting from a deterministic election procedure using a random epoch value and a round number value as input. In addition, a correct node verifies that the assignment of the bucket to the leader is valid. Finally, a correct node checks that all transactions within blocks are valid (with respect to UTXO -unspent transaction output- semantics and the buckets they originate from). In addition, each node checks for double-spending transactions in the blocks composing the macroblock. Where applicable, the total order established over the set of transactions resulting from the ordered set of blocks within the macroblock allows nodes to ignore transactions spending the same UTXO more than once. The above verification allows correct nodes to discard blocks submitted by malicious leaders and discard invalid transactions submitted by malicious clients.

4.4 Case studies

In this section, we show the application of ALDER's principles on three different permissionless blockchain consensus protocols, each representative of a different blockchain family. Bitcoin is the original PoW-based blockchain protocol described by Nakamoto (Nakamoto, 2008). Algorand is among the most scalable stake- and committee-based blockchain protocols achieving minimal transaction confirmation latency. Finally, Rapidchain illustrates one of the most throughput-efficient approaches for sharded blockchain protocols.

4.4.1 Applying ALDER to Bitcoin

Bitcoin is a Proof of Work (PoW) based blockchain that relies on cryptographic puzzles to elect leaders. We first explain how the Bitcoin protocol works and highlight its bottlenecks. Then, we describe Bitcoin++.

4.4.1.1 Overview of Bitcoin

The global course of a round in Bitcoin includes the following steps: (1) each node builds a block of transactions and the associated block header containing, among others, the hash of the previously appended block, a merkle root of the transaction set, and a nonce value of its choice; (2) Then, each node repeatedly hashes the block header by changing the considered nonce value until the resulting hash image (the election proof) falls below a threshold known as the difficulty level (a process referred to as *mining*). This difficulty is adjusted by the system so that a single solution to the cryptographic puzzle is found every 10 minutes on average; (3) When a node finds a solution to the puzzle, it disseminates its block to its neighbors via a gossip protocol; (4) Upon receiving a block, each node verifies the validity of the solution before appending the block to its blockchain; (5) If two different nodes each find a solution for the same cryptographic puzzle, two valid candidate blocks are competing to extend the same blockchain. In this case, nodes keep trying to solve the next puzzle by considering the first block they received, potentially extending the chain along two different paths, usually called forks. Bitcoin consensus relies on *the longest chain rule* to resolve forks by considering the largest amount of computation dedicated to a given fork, which results in the longest chain in terms of the number of blocks appended since the fork occurred. In Bitcoin, nodes submit blocks of 1 MB.

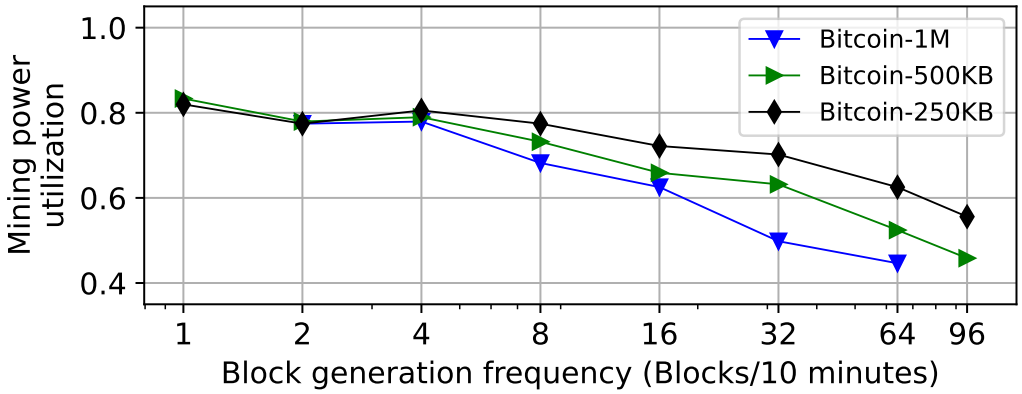

FIGURE 4.4: Bitcoin bottleneck

4.4.1.2 Bottlenecks of Bitcoin

The two main bottlenecks of Bitcoin limiting performance in terms of throughput or transaction latency are the small size of the blocks and the low frequency at which blocks are added to the chain (Eyal et al., 2016). Increasing the block size would result in a longer block dissemination time, consequently increasing the probability for forks to appear. Indeed, longer dissemination times increase the probability that a node will solve a puzzle before being informed that another node has already done so. However, when forks are solved, some blocks are left outside the main chain, reducing the mining power utilization ratio that we define as the ratio between the amount of data appended to the blockchain and the amount of data generated by the nodes. Therefore, the actual throughput is much less than the theoretical one where no fork exists. Figure 4.4 illustrates this. Reducing the cryptographic puzzle difficulty (hence increasing the block generation frequency) leads to more than half of the generated blocks being discarded.

4.4.1.3 Bitcoin++

Bitcoin++ allows multiple nodes to independently propose blocks containing disjoint sets of transactions and grow the chain by appending a subset of the proposed blocks in each round. The course of a Bitcoin++ round is described in Figure 4.5.

Macroblocks: Macroblocks in Bitcoin++ are composed of exactly Cl blocks. Indeed, as the consensus termination of Bitcoin is probabilistic, nodes cannot be sure that a given block is final and will never be removed from the blockchain.

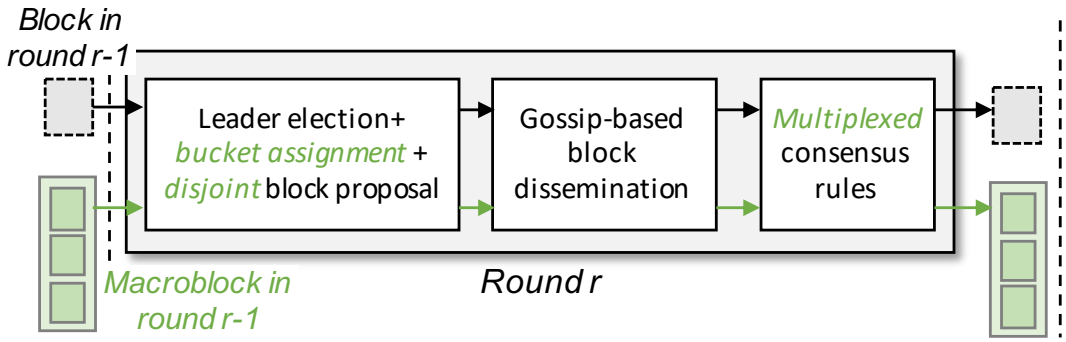

FIGURE 4.5: Round structures of Bitcoin (black texts and arrows) and Bitcoin++ (in green).

For this reason, nodes in Bitcoin++ only consider that they can proceed mining for the next round only when they can build a macroblock fully with Cl blocks.

Multiple leader election and bucket assignment: Similar to its parent, Bitcoin++ relies on cryptographic puzzles to elect multiple leaders. To do so within the same time interval (10 minutes on average), ALDER requires decreasing the cryptographic puzzle's difficulty. More precisely, to obtain Cl leaders at the same time interval as Bitcoin, the difficulty is divided by Cl.

Bitcoin belongs to this family of blockchain protocols where nodes have to commit to the content of a block before the election process succeeds. As ALDER requires the bucket assignment to be unforgeable and verifiable by leveraging the election proof from the election, the bucket assignment in Bitcoin++ must not let nodes create blocks and find valid puzzle solutions, to then assign them buckets for which blocks have not been generated. Consequently, each mining node needs to commit to the set of Cl disjoint blocks, respectively including transactions from the Cl disjoint buckets, one of which will be assigned once the cryptographic puzzle is solved. To that end, Bitcoin++ modifies the cryptographic puzzle definition of Bitcoin. Instead of mining a block header containing the merkle root of the transaction set of a block, each node mines a header that includes the Merkle root of the Merkle tree, taking as leaves the Merkle roots of the blocks to which the node is committed. When the node finds a solution to this puzzle, the resulting hash value (the election proof) is employed to identify the bucket number assigned to the elected leader and the block that should be gossiped to the system. Then, the node gossips altogether the appropriate block along with the election proof and the Merkle path necessary to validate the block content.

Because Cl leaders are expected every 10 minutes on average, and because the bucket assignment process may assign one bucket to more than one leader, the time required to obtain Cl blocks, one for each bucket, could be greater than 10 minutes. This could inhibit the performance gain envisioned by ALDER. Bitcoin++ addresses this issue by introducing a mechanism that enables collecting blocks that may compete for the same bucket number. When building a block, each node incorporates an array *sibling* in the header that defines the puzzle. This field is a Cl-long array containing the hashes of blocks proposed by other nodes in the same round. When the node receives a valid block b for a given bucket while mining, the node updates the puzzle by inserting b's hash image into the Cl^{th} position of the sibling array. Suppose the node solves the puzzle and finds an election proof pointing to a bucket already in use in the sibling array. In that case, Bitcoin++ deterministically assigns the next available bucket in the sibling array.

Multiplexed consensus: Each node waits to collect Cl valid blocks originating from Cl different buckets. Then, the node constructs a macroblock, conducts the series of checks described in section 4.3.3, and appends it to its chain before proceeding to the next round.

Although forks are defined in Bitcoin as the results of two candidate blocks competing to extend the same common prefix, forks in Bitcoin++ can occur when two different valid blocks compete for the same bucket number in the same macroblock. In this case, two valid candidate macroblocks can extend the chain. To narrow down the fork space probability, Bitcoin++ employs an additional consensus rule with lower priority than the original longest-chain rule: when receiving several blocks for the same bucket, nodes prioritize blocks based on their hash values (highest wins) to help nodes decide on which one to consider. We distinguish three cases by differentiating when is learned the existence of other candidate blocks competing for the same bucket. Either the node learns that another valid block exists for a bucket already filled in the macroblock for which the node is currently mining. In this case, the node selects the block with the highest hash value, updates its sibling array accordingly, and continues mining. Or the node learns about another valid block in the lastly created macroblock. In this case, the node locally builds the second macroblock, selects the macroblock which contains the block with the highest hash value, and continues mining. In the last case, we consider the node that learns about another valid block for a macroblock located farther than the lastly created macroblock. In this situation, the node selects and mines on top of the

chain of macroblocks containing the highest amount of computational power, i.e. applying the longest chain rule of Bitcoin.

4.4.2 Applying ALDER to Algorand

Algorand (Gilad et al., 2017) is among the most scalable PoS-based permissionless blockchain. In the following, we explain its protocol, highlight its bottlenecks, and describe Algorand++.

4.4.2.1 Overview of Algorand

Figure 4.6 depicts the main course of an Algorand round. First (❶), each node executes a cryptographic sortition that produces an *election proof* and a verifiable *priority* value used in order to determine whether the node belongs to some of the committees responsible for conducting the following steps: the block proposal, the reduction, and the multiple steps of the binary agreement of Algorand's Byzantine agreement $BA\star$. This cryptographic sortition elects nodes at random based on their weights (i.e. their currency stake in the system) in a publicly verifiable, and non-interactive way using a verifiable random function (Micali, Rabin, and Vadhan, 1999) (VRF). The sortition is designed to elect an expected number of $\tau_{proposer}$ block proposers and to assign each selected node a priority, along with its proof. This sortition protects nodes against an adversary aiming at learning the identity of committee nodes and forging targeted attacks. In addition, committees are different for each step of the protocol to prevent targeted attacks on committee members once they send a message.

Once elected as a member of the proposal committee (❷), a node builds a block before sending it along with the priority and election proof value to its neighbors (❸), disseminating these messages via an efficient gossip protocol. To reduce unnecessary communications(❹), and because only one of the proposed blocks will be appended to the chain, each node disseminates blocks based on the priority of the block proposer, ignoring blocks with lower associated priorities. The Byzantine agreement procedure $BA\star$ from Algorand reduces (❺) the problem of agreeing on one among many block hashes to agreeing on one selected block hash or a default empty block hash. Nodes operate this reduction in precisely two steps, and then reach consensus on one of these two values via a binary agreement called $BinaryBA\star$ (❻). Nodes wait a certain amount of time to receive priority messages and blocks (respectively 10 seconds and 1 minute, empirically set by the authors (Gilad et al., 2017)). If a node does not receive a block within this delay, it proceeds to the protocol step considering an empty

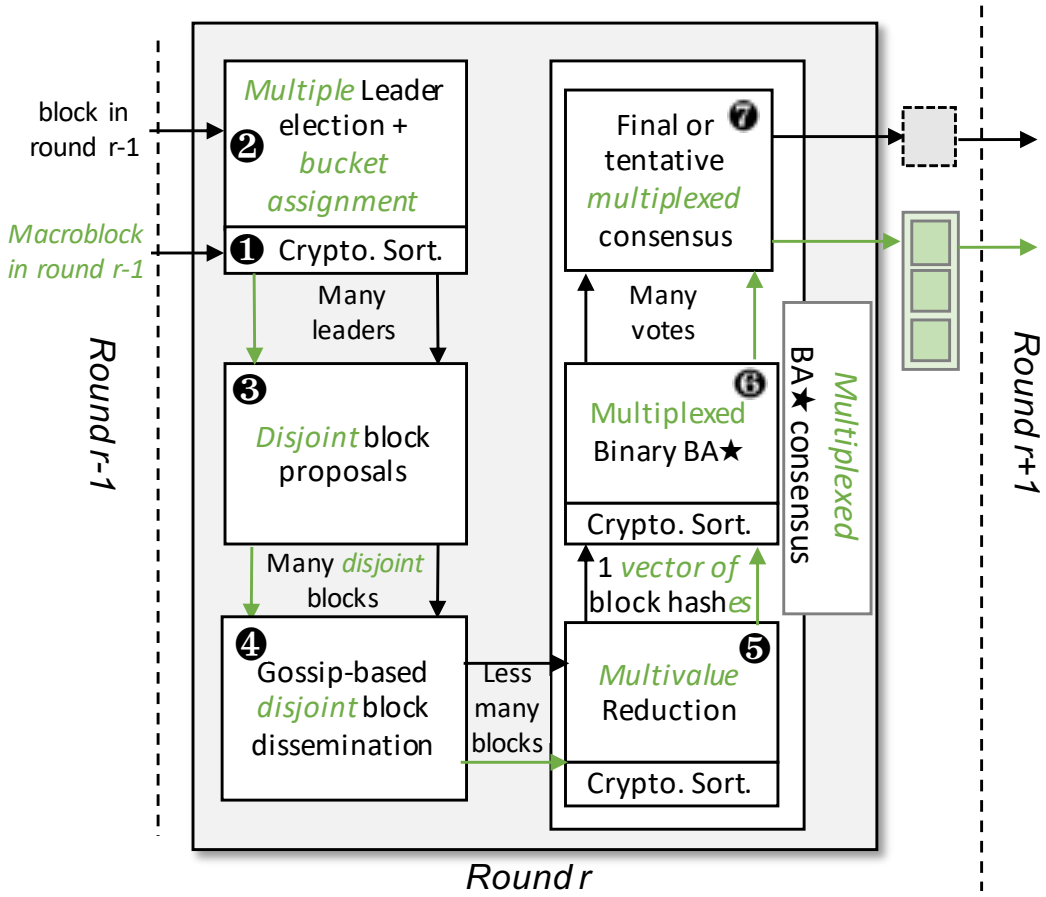

FIGURE 4.6: Round structures of Algorand (black texts and arrows) and Algorand++ (in green).

block. Finally (❼), every node counts votes cast during the $BA\star$ phase to learn about the outcome of the agreement procedure, reaching final consensus.

4.4.2.2 Bottlenecks of Algorand

Algorand has a low transaction confirmation latency of the order of a minute. Despite the impressive performance of Algorand (125-fold throughput improvement compared with Bitcoin), it still suffers from performance limitations. In particular, its performance drops dramatically with large block sizes. Indeed, the time of gossiping blocks in the network largely dominates the duration required for the Algorand to reach consensus with $BA\star$ ($\sim$15 seconds on average). This long gossip is a significant limitation to increasing throughput, as shown in our preliminary experiment depicted in Figure 4.7. Increasing the block size only increases the confirmation latency and keeps the throughput at its highest level in the best case.

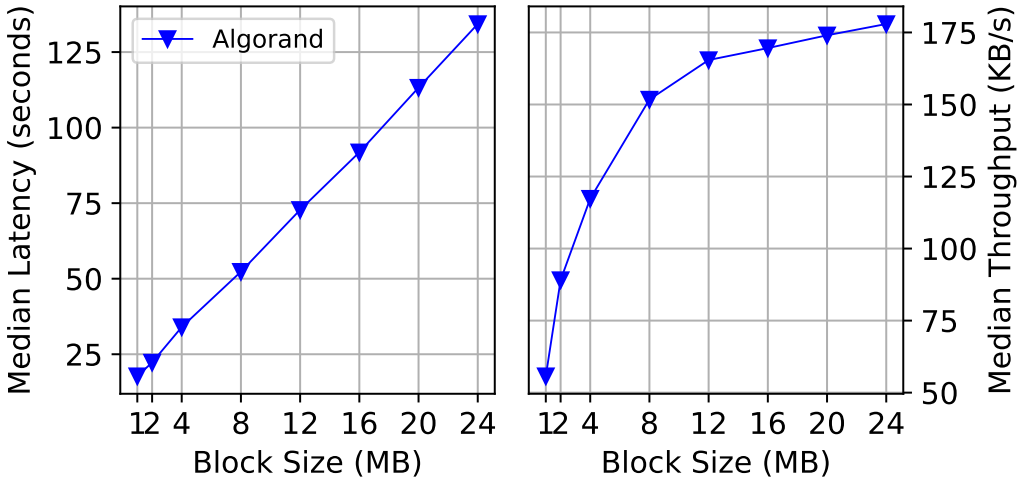

FIGURE 4.7: Algorand bottleneck

4.4.2.3 Algorand++

Algorand++ allows multiple independently proposed blocks containing disjoint sets of transactions to be appended to the ledger in a given round in the form of a macroblock. Figure 4.6 depicts the course of round.

Multiple leader election and bucket assignment: Electing multiple leaders is already part of Algorand. Assigning buckets in an unforgeable and publicly verifiable way is performed by directly applying the Cl modulo operation over the election proof generated by the cryptographic sortition. To avoid bucket being unassigned, Algorand++ must ensure a sufficiently high number of nodes elected as block proposers. A too low $\tau_{proposer}$ value could lead to buckets being frequently unassigned, which would not only hinder the envisioned throughput gains but also lead to transactions with specific hashes being ignored for some period. To devise an appropriate value for $\tau_{proposer}$, we rely on the uniform distribution of the probability $(\frac{1}{Cl})$ of bucket assignment to a leader, directly derived from the hash function employed in the sortition. In other words, each node has an equal chance of being assigned one bucket over another; a bucket is assigned to at least one proposer with the following probability $1-(1-\frac{1}{Cl})^{\tau_{proposer}}$; and all buckets are assigned to at least one proposer with probability $p=\sum_{i=0}^{Cl}(-1)^{Cl-i}\binom{Cl}{i}\left(\frac{i}{Cl}\right)^{\tau_{proposer}}$. We set $\tau_{proposer}$ so that $p=0.95$.

Disjoint block proposal and dissemination: The block proposal step of Algorand++ is very similar to that of Algorand, except that nodes can be assigned the same transaction bucket as other nodes. Because only one proposed block

per bucket number will be appended to the chain, Algorand++ extends the original gossip-based dissemination protocol to reduce unnecessary communications. Similarly to Algorand, each block proposer employs an additional priority value derived by hashing the VRF hash output concatenated with publicly verifiable information of the node's stake in the system and the assigned bucket number. This priority value is then used during block dissemination to discard blocks originating from the same bucket but with lower priority.

Multiplexed consensus and macroblocks: The multiplexed $BA\star$ agreement protocol takes as input a set of hashes from blocks originating from different buckets. It produces a vector of block hashes composing the macroblock to be appended at the end of the round. In the multiplexed reduction step, committee members reduce the problem of agreeing on a set of proposed blocks to agreeing either on a *vector* of Cl disjoint block hashes ordered by their bucket number (some of which could be default empty block hashes as in Algorand), or on a vector consisting only of empty block hashes. Then, the multiplexed binary Byzantine agreement over the two possible vectors is executed. Once reached, nodes gather the blocks Algorand++ has agreed on, proceed through the checks described in section 4.3.3, and build the *macroblock* corresponding to the consensus decision before appending it and continuing to grow the chain. The final consensus decision could be a vector containing the hashes of some empty blocks. Hence Algorand++ grows a chain of macroblocks of possibly different sizes.

4.4.3 Applying ALDER to Rapidchain

Rapidchain is among the most efficient sharded permissionless blockchain protocol. Rapidchain splits the system into k non-intersecting committees of m nodes. Each committee is in charge of maintaining and growing a specific shard of the blockchain. Also, Rapidchain uses an efficient gossip dissemination mechanism, $IDA - Gossip$, to improve the dissemination performance, and we investigate the properties of this mechanism in Chapter 5. In the following section, we explain its protocol, highlight its bottlenecks and describe Rapidchain++.

4.4.3.1 Overview of Rapidchain

Figure 4.8 depicts the course of a round. Rapidchain proceeds by successive epochs consisting of several rounds of consensus and a reconfiguration phase.

First, a leader is deterministically and verifiably elected in each committee by leveraging a publicly known epoch randomness value and the current round

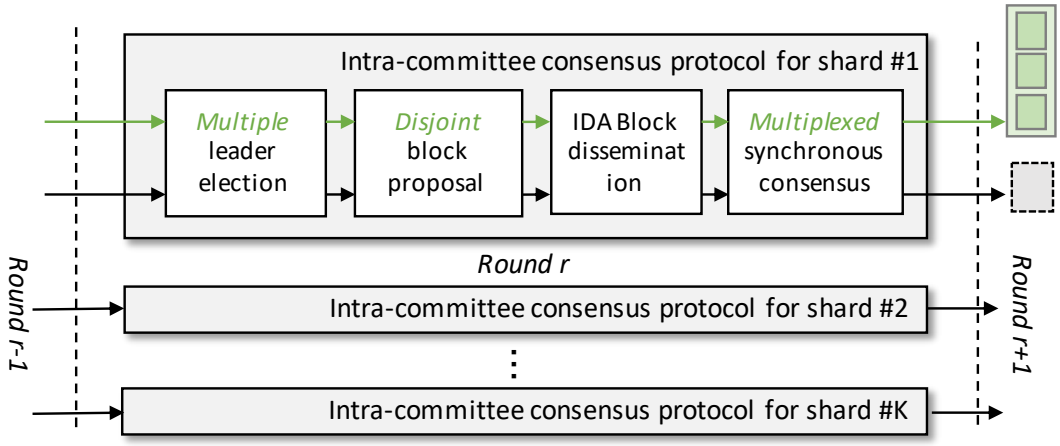

FIGURE 4.8: Round structures of Rapidchain (black texts and arrows) and Rapidchain++ (in green).

number. Then, the leader builds a block and gossips it to the committee using an efficient gossip dissemination protocol, inspired by IDA(Alon et al., 2003), designed to fasten the propagation of large blocks. Finally, the committee reaches consensus on the proposed block via an intra-committee synchronous algorithm.

At the end of each epoch, a reconfiguration phase led by a specific reference committee C_R that allows nodes to join or remain in existing committees. C_R agrees on, and appends to each shard, a reference block that includes the list of all active nodes and their assigned committees for that epoch. To join or remain in existing committees, nodes need to solve a cryptographic puzzle that C_R defines. To do this, C_R runs a distributed random generation protocol to produce an unbiased random value called epoch randomness incorporated into the puzzle. Then, nodes wishing to participate in committees must solve the puzzle within 10 minutes and send the solution to C_R. C_R validates the received solutions, assigns each member to a sharding committee, and informs the other committees by publishing a configuration block.

Transactions sent by external users are deterministically assigned to specific shards based on their hash images. Nodes receiving transactions forward them to their destination committee using a routing protocol inspired by Kademlia (Maymounkov and Mazieres, 2002). Committee members batch several transactions into a block and run an intra-committee consensus to append it to their shard. Since transactions are stored into disjoint shards held by different committees, committee members need to communicate with the corresponding input committees to ensure that the input transactions exist in their shards.

In order to append a block to a shard, Rapidchain uses an intra-committee consensus protocol based on the synchronous protocol of Ren et al. (Ren et al., 2017), resilient to $f = 1/2$ of the committee size being faulty. The protocol executes the following four steps in which all communications are signed by the sender's private key to provide authentication and integrity: (1) The leader submits a *propose* message containing the block header, (2) all nodes receiving this message send an *echo* message containing the same block header to the entire committee, (3) If a honest node sees another version of the block header in one of the *echo* messages it received, then it knows that the leader has equivocated and therefore gossips a *pending* message containing an empty block hash, (4) Finally, if an honest node receives $mf + 1$ echo messages of the same and only block header (m is the committee size and f is the fraction of honest nodes), it decides on this block header by submitting an *accept* message including the $mf + 1$ received echo messages.

4.4.3.2 Bottlenecks of Rapidchain

Rapidchain uses an optimal intra-committee synchronous consensus algorithm to achieve very high throughput via a novel gossiping protocol for large blocks, and a provably-secure reconfiguration mechanism to ensure robustness. Using an efficient cross-shard transaction verification technique, Rapidchain prevents transactions from being gossiped across the network. We chose to apply ALDER on Rapidchain to assess the impact on performance when multiplexing blockchain protocols that do not suffer from clearly identified bottlenecks.

4.4.3.3 Rapidchain++

We apply ALDER's principles to the consensus protocol executed by each committee. For the duration of an epoch, the Rapidchain++ closely resembles a permissioned consensus protocol. Figure 4.8 depicts the course of a Rapidchain++ round: First, Rapidchain++ uses the deterministic leader election mechanism of Rapidchain to create an ordered set of Cl leaders from the committee membership. To do so, nodes locally create a leader set by sorting the committee members with a deterministic sortition algorithm relying on the epoch randomness and the current round number. Then, each leader builds and gossips blocks containing disjoint sets of transactions and including the hash of the previous macroblock. Finally, the intra-committee synchronous consensus protocol is executed to decide on a vector of block headers. Because each committee is already responsible for a region of the entire transaction hash space,

each Rapidchain++'s committee splits the designated region into Cl buckets of equal size.

Bucket assignment and disjoint block proposal: Elected leaders are deterministically assigned buckets based on their order in the leader set. An elected leader uses the assigned transaction bucket to build a block and disseminates it with the IDA gossip protocol. Thanks to the publicly verifiable properties of the leader election and bucket assignment mechanisms, any node can verify the correctness of the proposed blocks. Then each elected leader in a given committee starts its instance of the Rapidchain consensus protocol by submitting a propose message containing a single block header.

Multiplexed intra-committee consensus: Rapidchain++ extends the synchronous consensus algorithm of Rapidchain to decide on a vector of block headers. Specifically, *echo* and *accept* votes contain a vector of block headers. Each committee member awaits up to Cl *propose* votes until the end of the synchronous round timeout, and then submits *echo* votes composed of the received block headers to the rest of the committee. If a node receives more than one version of a block header from a specific leader and bucket, it submits a *pending* message including an empty block header. When receiving $mf+1$ echo votes for the same vector of block headers, a committee member sends an *accept* message for this vector and proceeds to the next round.

4.5 Evaluation

This section evaluates the extent of ALDER's ability to improve the performances of the three blockchains we considered. We first present our implementation and evaluation environment (Section 4.5.1) before presenting the performance of Bitcoin++, Algorand++, and Rapidchain++ in Sections 4.5.2, 4.5.3, and 4.5.4 respectively.

4.5.1 Implementation and evaluation environment

We implemented all baseline protocols and their multiplexed versions using Golang. The experiments presented in this section were carried out using the Grid'5000 (Bolze et al., 2006) testbed. We used powerful physical machines with 18 cores, 96 GB of memory, and a 25 Gbps network connectivity link. In all experiments, we emulate wide-area network conditions as in major blockchain propositions (Gilad et al., 2017; Zamani, Movahedi, and Raykova, 2018): we cap the bandwidth for each process to 20 Mbps and add a one-way latency

of 50 milliseconds to each communication link between processes using traffic control. We set the number of peers with which each node communicates when disseminating blocks to 8 for Bitcoin and Algorand, and 16 for Rapidchain as recommended by the authors. In addition, we relied on a custom registry service to bootstrap the system: at startup, a node registers itself to the registry service and receives a list of available nodes. Each node has access to a pre-initialized transaction pool to populate block payloads.

4.5.2 Bitcoin++ performance evaluation

Bitcoin++ reduces the difficulty of cryptographic puzzles inversely proportional to Cl value to elect Cl leaders on average every 10 minutes. We conducted experiments with 20 machines and 1000 nodes to evaluate the performance of Bitcoin++ and compare them to the ones of Bitcoin with similar cryptographic puzzle difficulty, i.e. with similar block generation frequencies. To this end, we measure mining power utilization of both Bitcoin and Bitcoin++ defined as the ratio between the amount of data appended to the blockchain and the amount of data generated by the system in the form of blocks. We also measure both protocols' throughput and block/macroblock latencey (time interval). In our experiments, we vary the concurrency level of Bitcoin++ from 1 to 96, i.e. generating a block from once every 600 seconds to once every $\frac{600}{96} = 6.25$ seconds.

We considered blocks of size 1 MB, 500 KB, and 250 KB, therefore the correspoding macroblock sizes for Bitcoin++ experiments are $Cl*1$ MB, $Cl*500$ KB, and $Cl*250$ KB respectively. Bitcoin++ aims to improve the original protocol's throughput by increasing the number of blocks in a macroblock, hence increasing the size of the macroblock appended to the chain. Each experiment consists of a given puzzle difficulty, a given block size, and a given protocol (Bitcoin or Bitcoin++). Each experiment runs until 150 same macroblocks are appended to the chain of each node.

4.5.2.1 Bitcoin++ results

Figure 4.10 shows median throughput and mean block interval time figures for both protocols. The configuration with 1 MB block size and $Cl = 1$ represents the original Bitcoin protocol providing a throughput of 1.6KB/s with a mean latency of 600 seconds. In all experiments, Bitcoin++ provides greater throughput than Bitcoin with reduced difficulty. For the 1 MB block size experiments with $Cl = 64$, Bitcoin++ provides 91 KB/s, a x2.25 throughput improvement

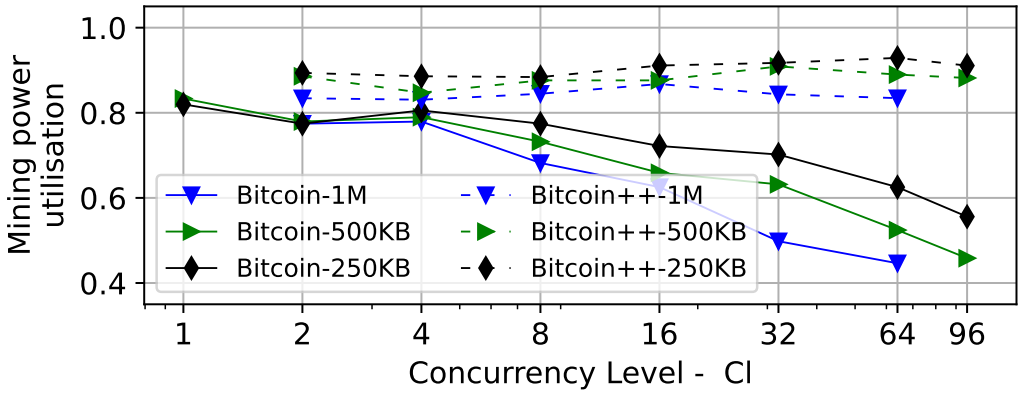

FIGURE 4.9: Mining power utilization. Higher Cl means high block generation frequency and lower difficulty.

compared to Bitcoin with the associated reduced difficulty (i.e. the same block generation frequency), and a x57 improvement to the original Bitcoin protocol. Regarding the experiments with the 500 KB and 250 KB block sizes, we observe similar trends. To better understand the origins of such throughput gains, we correlate these results with the mining power utilization ratio that remains steady between 0.9 and 0.95 when the concurrency level increases with Bitcoin++, while dropping to 0.44 with Bitcoin. This shows that Bitcoin++ can effectively collect the generated blocks at high block generation frequency while Bitcoin can hardly do so. Indeed, in Bitcoin with 1 MB blocks and $Cl = 64$, more than 56% of the generated blocks are not included in the blockchain shared by all nodes at the end of the experiment. This measure also helps understand the extent of fork occurrences in each protocol. The corollary of these results is visible on the mean macroblock block interval that remains at 600-650 seconds for all Bitcoin++ configurations while dropping to approximately 10 seconds for Bitcoin with $Cl = 64$. Indeed, Bitcoin can append blocks quickly, but forks occur very frequently with higher block frequency. Bitcoin++ does not suffer from this problem because macroblocks are added at a slow rate (once per 10 minutes on average), which leaves enough time for any forks to be resolved.

4.5.3 Algorand++ performance evaluation

We conducted two sets of experiments to evaluate the performance improvements of Algorand++ compared to Algorand. In the first set of experiments, we compare the performance of both protocols using 1,000 nodes on 10 machines. In the second set of experiments described in Section 4.5.3.2, we deployed up

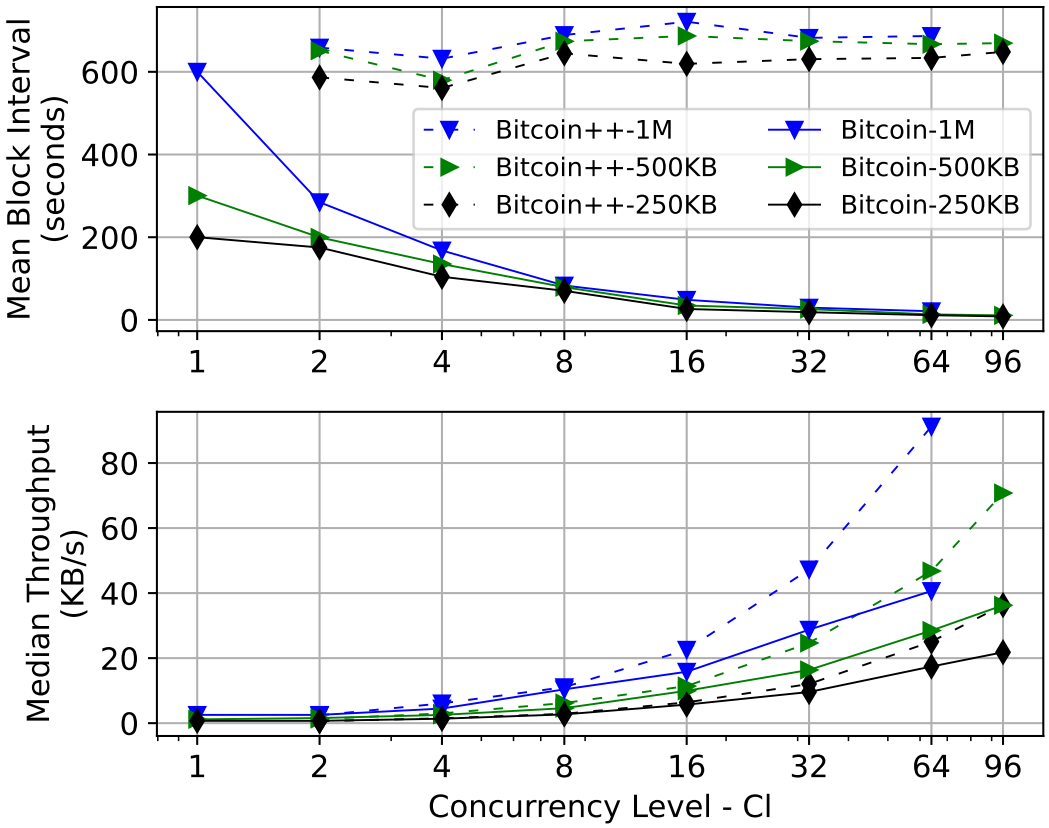

FIGURE 4.10: Bitcoin and Bitcoin++ performance comparison (throughput and latency). Higher Cl means high block generation frequency and lower difficulty.

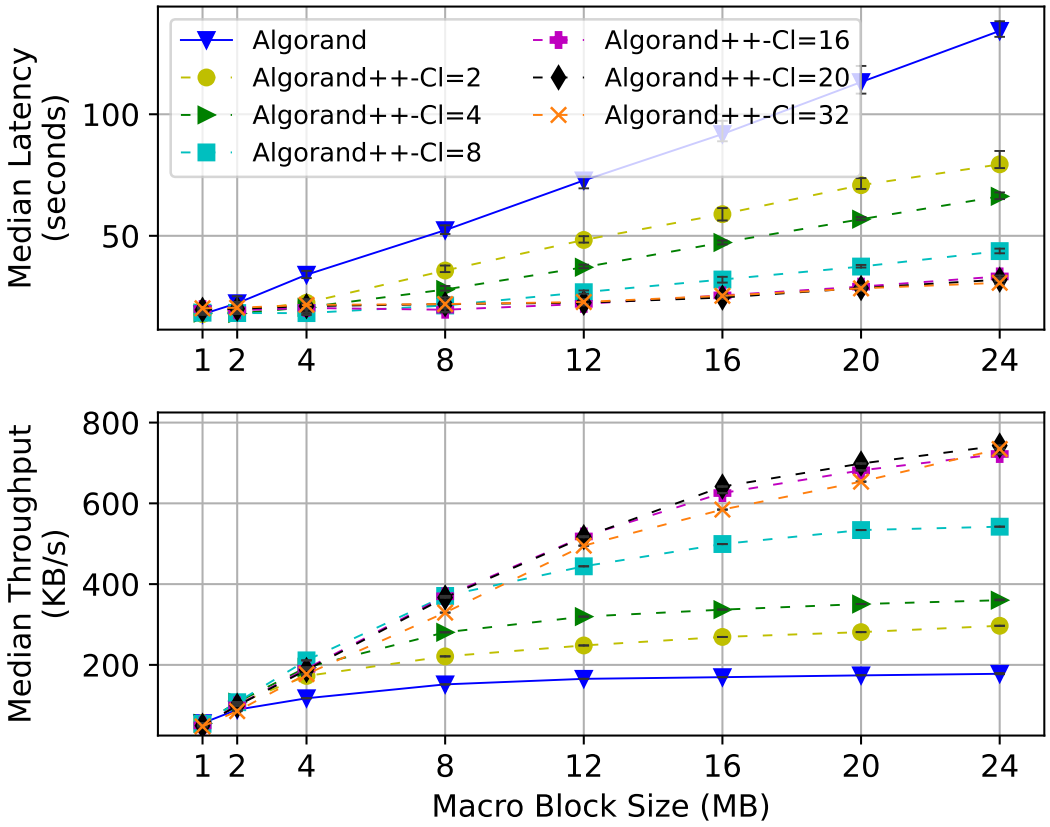

FIGURE 4.11: Round latency and throughput of Algorand++ with various concurrency levels (Cl)

to 10,000 nodes on 100 machines to compare the scalability characteristics of both protocols.

4.5.3.1 Algorand++ latency and throughput

To evaluate the performance of Algorand and Algorand++, we varied the macroblock size from 1 MB to 24 MB, and the concurrency level of Algorand++ from 1 to 32. Elected leaders build blocks of size the expected macroblock size divided by Cl. For Algorand we varied the size of blocks appended to the chain to match the size of macroblocks appended to Algorand++ chain. Each experiment lasts 150 rounds.

We measure the round latency of the two protocols, indicating the time it takes for a block to be appended to the chain. Latency is the duration between the time a block is proposed and the time all nodes observe this block in the blockchain. The results are depicted in Figure 4.11. They show that the round latency of Algorand increases rapidly as the size of the appended blocks becomes larger, reaching a latency of 134 seconds for blocks of 24 MB. Furthermore, we

observe that the round latencies of Algorand++ largely outperform the ones of Algorand for macroblock sizes larger or equal to 2 MB. For instance, for the block size 4 MB, Algorand++ reduces the round latency of Algorand by a factor of 0.66 with Cl=2, and 0.53 with Cl=8. The gap between Algorand++ and Algorand becomes more considerable with larger block sizes. Indeed, for 24 MB blocks, we observe round latency reductions between the two protocols by 0.56 for $Cl = 2$ and 0.22 with $Cl = 32$ in favor of Algorand++.

In addition to round latency, we evaluate the effective throughput of Algorand++ compared to Algorand. Since some macroblocks may contain fewer blocks than expected, we cannot derive throughput directly from the round latency and the macroblock size. We define effective throughput as the amount of data appended to the blockchain per second. As shown in this Figure 4.11, Algorand++ outperforms Algorand in all configurations by reaching up to 743 KB/s with Cl=20 and 24 MB blocks. This specific configuration exposes a x4.17 throughput improvement over Algorand, a x6.35 throughput improvement over Algorand with the 4 MB block size configuration that offers the same round latency, and a x13 throughput improvement over the original Algorand configuration with 1 MB blocks. Our results also highlight the limits of multiplexing Algorand instances. Indeed, we reach the maximum throughput with $Cl = 20$, and increasing the concurrency level does not improve performance further.

4.5.3.2 Algorand++ evaluation at scale

To evaluate how Algorand++ and Algorand scale, we vary the number of machines in the testbed from 10 to 100 with 100 nodes per machine, allowing us to emulate up to 10,000 nodes. We use 1 MB blocks for Algorand and 20 MB blocks and $Cl = 20$ for Algorand++. We then measure the throughput degradation and round latency increase as a function of the number of nodes in the system compared to the baseline with 1,000 nodes. Regarding throughput, results depicted in Figure 4.12 show that similarly to Algorand, Algorand++ suffers a throughput degradation when the number of nodes increases in the system. Nevertheless, the mechanisms brought by ALDER to Algorand do not degrade its scalability. It improves it when the number of nodes increases as illustrated by the configuration with 10,000 nodes where a 6% lower degradation difference is observed for Algorand++ compared to Algorand. Results related to the latency increase with respect to the number of nodes in the system are depicted in Figure 4.12. We observe that the gap in terms of round duration increase becomes greater between the two protocols in favor of Algorand++. This

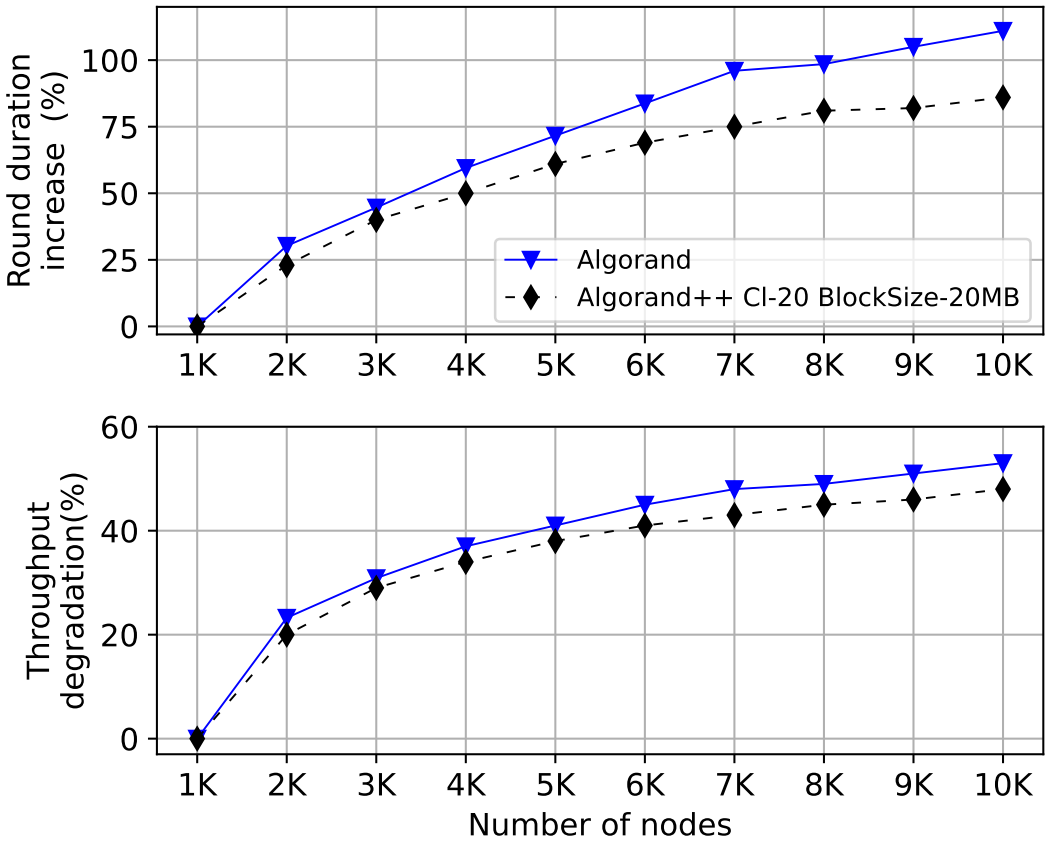

FIGURE 4.12: Throughput and latency degradation of Algorand and Algorand++ protocols at scale

illustrates the fact that Algorand++ exhibits better scalability than Algorand.

4.5.4 Rapidchain++ performance evaluation

We conducted a set of experiments to evaluate the performance of Rapidchain++ compared to Rapidchain in terms of round latency and throughput. Because the throughput of Rapidchain is the sum of the throughputs of each shard, we have considered a single shard deployment scenario and investigated the improvement possible by ALDER on one shard. In the original paper, Rapidchain considers a committee size of up to 250 nodes. We applied this configuration and deployed 250 nodes on 10 machines. In our experiments, we vary the macroblock size from 2 to 16 MB, and for each macroblock size, we vary the concurrency level Cl from 1 to 16. Similar to Algorand, elected leaders build blocks of size the expected macroblock size divided by Cl. Each experiment lasted 50 rounds, and we measured the median throughput and latency values as observed by individual nodes.

4.5.4.1 Rapidchain++ results

Figure 4.13 shows the measured round latency for Rapidchain++ with various concurrency levels. We observe that for all macroblock sizes, the round latency provided by Rapidchain++ is 10% lower than the provided round latency of Rapidchain. All Rapidchain++ instances provide similar latency values. The highest obtained throughput with Rapidchain is 262 KB/s with the 16-MB block size. Instead, Rapidchain++'s highest throughput is 292 KB/s obtained with macroblock size of 16 MB and with a concurrency level $Cl = 4$. For all block sizes, Rapidchain++ provides a throughput increase of approximately 10% compared with Rapidchain.

In our experiments, we can observe that Rapidchain++ provides latency and throughput figures similar to the ones of Rapidchain, exhibiting that ALDER does not degrade the performance of blockchains without bottlenecks. This result stems from the fact that Rapidchain uses $IDA - Gossip$: a chunk-based dissemination protocol that alleviates the latency incurred by the store-validate-forward mechanism. In the next chapter, we will investigate the properties of $IDA - Gossip$ in detail.

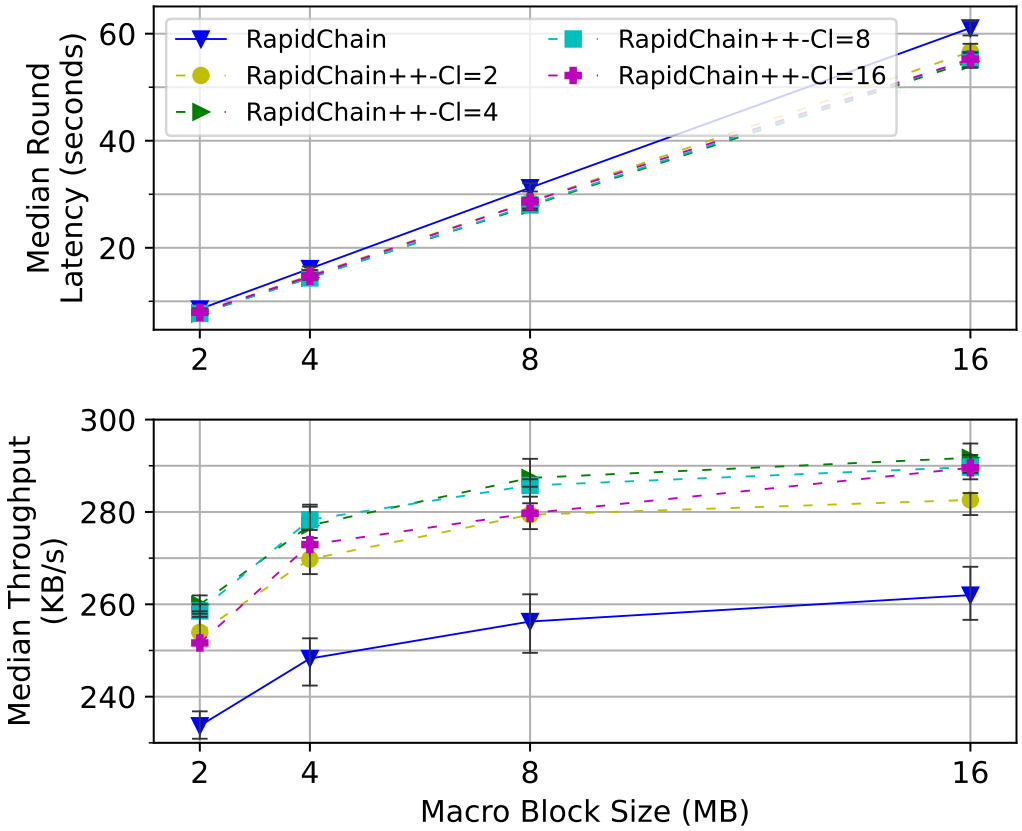

FIGURE 4.13: Latency and throughput figures of Rapidchain and Rapidchain++ protocols with various concurrency levels and macroblock sizes

4.6 Conclusion

We presented ALDER, a general construction to improve the performance of leade-based leader-based blockchain consensus protocols by multiplexing consensus instances: ALDER removes bottlenecks on the consensus and network layers by using multiple leaders.

To assess the improvements brought by ALDER, we applied it to three major blockchains: Algorand, Rapidchain, and Bitcoin. We presented how ALDER's principles applied to these blockchains and evaluated the performance of the resulting blockchains. Our evaluation, involving up to 10,000 nodes deployed on 100 physical machines, shows that ALDER provides progressive performance gains when a bottleneck exists and conservative performance when no bottleneck is identified.

5 In-depth analysis of the IDA-Gossip

5.1 Introduction

With advances in hardware and software technologies, the size of distributed systems is increasing. Today, there are many distributed systems that contain several thousand nodes distributed around the world and connected via the Internet (Gao et al., 2019; Simion and Pura, 2019; Park et al., 2019). An important characteristic of these systems is a high churn rate which is a high change in the set of participating nodes due to joins, graceful leaves, and failures (Godfrey, Shenker, and Stoica, 2006). Due to the large size and high churn rate, communication primitives such as reliable broadcast are not considered practical for today's large-scale distributed systems. In this context, gossip dissemination protocols fill an important gap by providing an efficient group communication primitive with probabilistic guarantees.

Gossip dissemination protocols are originally proposed in the context of database replication (Demers et al., 1987), and later on, they are adopted as probabilistic alternatives of reliable broadcast primitives. In a distributed system that depends on gossip dissemination, messages are spread in a manner very similar to epidemic diffusion: upon receiving a new message, a node becomes infected, and it infects fanout nodes by sending the message. Fanout is an important parameter of gossip dissemination protocol that controls the redundancy of dissemination. A message infects a node only once. Gossip dissemination protocols are considered practical alternatives to broadcast primitives for large-scale distributed systems because each node communicates with a few other nodes to disseminate a message.

Although gossip dissemination mechanisms are practical, they are not the most efficient ones because each message needs to be forwarded fanout times to provide a probabilistic dissemination guarantee. The higher fanout values cause a high resource consumption but provide high delivery guarantees. Also, gossip

dissemination of large messages incurs a high latency with store-and-forward mechanism where each message needs to be received completely and stored before being forwarded. For many open systems in which nodes can join or leave the system without any restriction, store-and-forward is the only viable option to protect the system against Denial of Service (DoS) (Lau et al., 2000) attacks: a correct node needs to validate each message before forwarding it to its neighbors, therefore; it should wait to deliver the full message before forwarding.

There are many gossip dissemination protocols, that aim to improve the efficiency of dissemination. *IDA-Gossip* (Zamani, Movahedi, and Raykova, 2018) is one of them. It is a gossip dissemination protocol proposed in the context of blockchains to disseminate large messages like blocks of transactions fast. Specifically, it is designed to disseminate large messages efficiently by removing the incurred high latency caused by the store-and-forward mechanism. *IDA-Gossip* combines Information Dispersal Algorithms(IDA) (Alon et al., 2003) and gossip dissemination protocols to circumvent the limitations of classic gossip protocols. *IDA-Gossip* makes use of message chunking, erasure coding, and Merkle hash trees. Using chunk-based message dissemination where a big message is chunked into smaller chunks, *IDA-Gossip* alleviates the high latency caused by the store-and-forward mechanism. By using an erasure coding mechanism, *IDA-Gossip* protects chunks against loss. Finally, by using Merkle hash trees, it provides an efficient chunk authentication scheme. *IDA-Gossip* is a promising building block for many distributed systems that needs to disseminate large messages—likes blocks of transactions (Nakamoto, 2008). Although the description of IDA-Gossip is straightforward, its properties are not investigated in depth under different conditions. We believe that understanding its properties might help the adoption of it.

This Chapter is dedicated to our investigation of $IDA-Gossip$ protocol. In the following sections: (1) we provide a formal description of the IDA-gossip protocol, (2) we conduct a thorough evaluation of IDA-gossip through experiments and simulations, (3) we identify the limitations of IDA-gossip and explain when this protocol is a good candidate to replace classic gossip alternatives in distributed systems.

5.2 The *IDA-Gossip* Protocol

In this section, first, we provide our system model, and later we provide the details of *IDA-Gossip* protocol. Finally, we communicate our analysis on the

adoption of *IDA-Gossip* protocol.

5.2.1 System model and assumptions

We consider a static system composed of N nodes, with a fraction $f < 1/2$ of Byzantine nodes, and a fraction $1 - f$ of honest nodes. Each node is connected to a set of d peers selected uniformly at random. A node only communicates with its peers. Nodes communicate over reliable synchronous channels in which messages are neither lost nor duplicated. Any message sent by one node is received by another within a known time interval. Correct nodes follow the protocol while Byzantine nodes may deviate from it in any possible way. A source node s is selected among the correct nodes whose objective is to disseminate a message M to all nodes in the system.

We assume the presence of an adversary controlling all Byzantine nodes to prevent correct nodes from delivering messages. Byzantine nodes can either drop the messages they receive or alter the content of the message they receive and forward. We assume the availability of cryptographic hash functions and Public-key cryptography mechanisms. Any message sent by nodes is authenticated using signatures. Finally, the adversary cannot break the cryptographic primitives. The source nodes are selected among the correct nodes, and the identity of source nodes are known in advance by other nodes: therefore, Byzantine nodes can flood the system with irrelevant messages.

5.2.2 The details of *IDA-Gossip* protocol

In the *IDA-Gossip* protocol, each node selects d neighbors to communicate. κ denotes the number of chunks of a message M. ϕ is a security parameter of the protocol that takes values between 0 and 1, and it controls the ratio of data chunks and parity chunks. ϕ is calculated theoretically to protect the source node against up to 10 faulty sampled neighbors out of d neighbors. The source node chunks a large message M into $(1 - \phi)\kappa$ equal sized chunks, $C_1, C_2, C_3, ..., C_{(1-\phi)\kappa}$. Later on, the source node calculates $\phi\kappa$ additional parity chunks using an erasure-coding scheme—Reed-Solomon erasure coding (Reed and Solomon, 1960). In total, the source node produces κ chunks.

The use of chunks in *IDA-Gossip* introduces the problem of chunk authentication: without an efficient authentication mechanism, it would be costly to authenticate chunks. *IDA-Gossip* makes use of Merkle hash trees to authenticate chunks of a message M. The source calculates a Merkle tree which is a

binary tree. The leaves of the Merkle tree consist of the hashes of chunks in order— $H(C_1), H(C_2), H(C_3), ..., H(C_\kappa)$, and inner nodes are calculated from bottom to up by concatenating and hashing the values of children's. The source node calculates a Merkle proof for each chunk, it disseminate each chunk with its Merkle proof. Any node can authenticate a received chunk using the attached Merkle proof.

The authors of RapidChain have shown that we can derive a threshold ζ for ϕ, such that the probability of having a proportion of corrupted nodes greater than ζ in the neighbor set of the source node is at most 0.1. Using a hypergeometric distribution, under the assumption $f < 1/2$ and $d = 16$, setting $\zeta = \phi = 0.63$ guarantees that a message is not delivered to all honest nodes with probability 0.1. Therefore, although at the end of the first phase none of the source node's neighbors can reconstruct the original message, the system is in a state where at least $(1 - \phi)\kappa$ chunks are possessed by honest nodes with high probability.

For analysis purposes, one can divide *IDA-Gossip* protocol into two distinct phases, as shown in Fig. 5.1: in the first phase, the source node chunks a message into smaller chunks, calculates parity chunks, and sends a subset of chunks to its peers. In the second phase, the peers of the source disseminate messages using classic gossip dissemination on behalf of the source node. Redundancy is the key element to protect the system against faulty nodes and message loss. Unlike classic gossip dissemination protocols, In the first phase of *IDA-Gossip*, the redundancy comes from parity chunks calculated using erasure coding. In the second phase of *IDA-Gossip* redundancy comes from forwarding message fanout times.

The original evaluation of *IDA-Gossip* conducted in RapidChain (Zamani, Movahedi, and Raykova, 2018) considers following parameter values: messages of size 2MB, $d = 16$, $\phi = 0.63$, and $\kappa = 128$.

5.2.3 On the adoption of *IDA-Gossip*

Because *IDA-Gossip* disseminates chunks of the message instead of the message itself, a node cannot verify the content of a received chunk according to the upper-layer protocol. This is a crucial property that should be taken into consideration for the adoption of *IDA-Gossip*, e.g., in open systems where the identity and number of nodes are not publicly known, and where the source node is not always expected to behave correctly (for example, permissionless blockchains systems, etc.). Indeed, an adversary in control of the source node

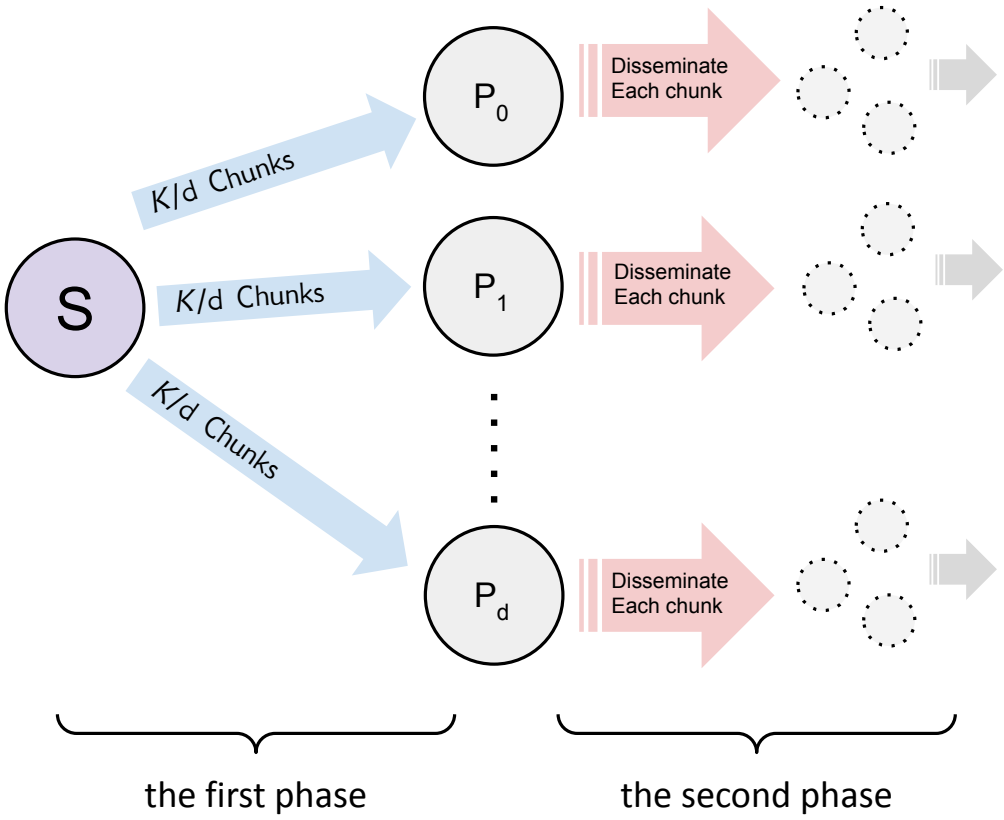

FIGURE 5.1: The two phases of the *IDA-Gossip* protocol: in the first phase, the source node sends κ/d distinct chunks to its peers, and in the second phase, its peers disseminate chunks using classic gossip.

could flood the whole system by disseminating irrelevant messages. Further, a node could behave in a faulty manner and disseminate an invalid message. In such scenarios, faulty behaviors would remain hidden until a correct node obtains enough chunks to reconstruct the original message. Although this issue is not specifically related to *IDA-Gossip* but to chunk-based gossip more generally, it highlights the need to develop solutions that allow verification of the content of each chunk as it is disseminated, in relation to the upper-layer protocol employing *IDA-Gossip*. Also, it is possible to mitigate this kind of behavior by employing accountability mechanisms as in (Mokhtar, Decouchant, and Quéma, 2014; Guerraoui et al., 2010; Li et al., 2006).

Conversely, *IDA-Gossip* represents a very good candidate for permissioned systems or semi-permissionless systems where a subset of nodes is identified to conduct a specific task such as in sharded blockchains where the sharding committees are elected (*e.g.*, RapidChain(Zamani, Movahedi, and Raykova, 2018), Omniledger (Kokoris-Kogias et al., 2018), Elastico (Luu et al., 2016a)), or in permissioned blockchains such as Hyperledger Fabric (Androulaki et al., 2018).

5.3 Evaluation with Experiments

In this section, the performance and limitations of *IDA-Gossip* are evaluated in a controlled testbed. We aim to address the following questions: (1) How does *IDA-Gossip* perform compared to classic gossip dissemination? (2) How does *IDA-Gossip* perform under different conditions such as varying message size, number of chunks, etc.? (3) Because *IDA-Gossip* employs erasure coding that comes with redundancy in the transmitted data, what is its impact on the utilization of network bandwidth and the performance of the dissemination compared to classical gossip techniques? (4) How are *IDA-Gossip*'s performances affected under faults?

5.3.1 Methodology

5.3.1.1 Evaluation environment and implementation

The experiments presented in this section are conducted on the Grid'5000 (Balouek et al., 2013) platform. We consider 32 physical machines with 18 cores, 96 GB of memory, and connected with high-speed links-25 Gbps. In each experiment, we deploy 4096 *IDA-Gossip* nodes (128 nodes per machine) to emulate a large-scale distributed system. The use of 4096 nodes reflects approximately the order of magnitude of today's open system size: Tor network being composed of ∼8,000

relays, and Bitcoin of ∼15,000 nodes. We emulate wide-area network conditions by capping the bandwidth of each node to 20 Mbps and adding a one-way latency of 15 milliseconds to each communication link between processes using cgroups and traffic control subsystem of Linux.

We implemented a prototype of *IDA-Gossip* in Golang, consisting of 3000 lines of code. Our implementation chunks a large message into $(1 - \phi\kappa)$ chunks, and adds $\phi\kappa$ parity chunks using a Reed-Solomon erasure coding library[1]. We implemented a coordination service, and it is used by each node as a rendezvous point at bootstrap.

5.3.1.2 Experimental protocol

An experiment starts with a fresh deployment of an *IDA-Gossip* system with 4096 nodes hosted on 32 machines. Upon start, a node registers its IP address and port number to the coordination service and waits for the registration of other nodes. Later on, the node retrieves the node list from the coordination service. The node list contains the IP addresses and port numbers of all nodes in the system. Each node samples 16 peers uniformly at random using the node list, and it establishes connections to sampled nodes. Each node accepts up to 125 incoming connections. To start dissemination, we select a source node uniformly at random. The source node disseminates a message using *IDA-Gossip*. Other nodes forward each delivered chunk 8 times, therefore we have used the fanout value of 8. An experiment terminates when all nodes have forwarded the chunks that they have delivered. To guarantee this condition, we use a conservative timeout value that we empirically determined in preliminary experiments. At the end of each experiment, we tear down the network and collect measured statistics from nodes. Each experiment is run 30 times to consolidate the measured metrics.

An experimental setup consists of a set of fixed parameters: the number of chunks, the message size, the proportion of Byzantine nodes, and the sequential or concurrent dissemination of chunks from a node to its neighbors—dissemination concurrency. Table 5.1 lists the data chunks and parity chunks used in our experiments. These values are calculated according to the target number of chunks, κ, and by setting ϕ to 0.63 to tolerate up to 10 faulty neighbors out of the 16 neighbors of the source node.

[1] https://github.com/klauspost/reedsolomon

#Chunk = κ	#Data Chunks	#Parity Chunks	d	κ/d
256	96	160	16	16
128	48	80	16	8
64	24	40	16	4
32	12	20	16	2
16	6	10	16	1

TABLE 5.1: Parameter values of IDA-Gossip used in our experiments

In our experiments, we vary the size of messages between 1 and 36 MB, the chunk count between 16 and 256, and the proportion of Byzantine nodes between 0 and 40%. Finally, we vary the concurrent dissemination of chunks—dissemination concurrency—from 1 (sequential) to 128 (high degree of concurrence).

5.3.1.3 Measured Metrics

We measure the following four metrics: 1) first chunk latency, 2) latency, 3) amount of uploaded data, and 4) coverage.

The *first chunk latency* is the time needed for any node in the system to deliver a chunk from the disseminated message. This information is a precursor of how early a node starts contributing to the dissemination process. In classical gossip protocols that employ the store-and-forward mechanism, the majority of nodes do not contribute to the dissemination until the last rounds of dissemination. An efficient dissemination mechanism should employ all available resources at the earliest possible time. Hence this metric is a sound performance indicator for gossip protocols disseminating large messages.

Latency is the time needed to deliver enough chunks to reconstruct the original message disseminated by the source node. *Uploaded data* is the amount of data uploaded by a node. It is directly proportional to the fanout value, and the message size. Uploaded data is a metric that measures the redundancy of gossip dissemination, helping us to quantify the bandwidth usage overhead caused by the use of erasure coding techniques in *IDA-Gossip*. Finally, *coverage* is the percentage of nodes that successfully deliver the initial message.

5.3.2 *IDA-Gossip* vs classic gossip dissemination

We compare *IDA-Gossip* with *classic gossip* dissemination where a message is disseminated in its entirety (without chunking) using the store-and-forward

mechanism. More specifically, we aim to understand the performance difference between these two protocols when considering message sizes between 1 and 36 MB. We conducted a set of experiments by varying the message size. In this set of experiments, we have considered a fault-free environment.

For *IDA-Gossip*, we consider the use of $\kappa = 128$ chunks as originally set by the designers of RapidChain. Our classic gossip instances use a fanout value of 8—as *IDA-Gossip* instances. Also, classic gossip instances use a single-threaded message dissemination mechanism in which a node sends chunks one peer at a time. In our preliminary experiments, we have observed that in case of limited bandwidth and large message sizes this is the best strategy that produces minimal message dissemination latency for classic gossip instances. Fig. 5.2 plots quartiles (first, second, and third quartiles) of the first chunk latency, latency, and uploaded data. Coverage is not represented as both protocols always reach 100% of the nodes.

The first chunk latency of classic gossip dissemination is equal to its latency because the whole message consists of a single chunk. When we compare both protocols in terms of first chunk latency, *IDA-Gossip* provides excellent results for all message sizes: with 36 MB messages, the first chunk is delivered and starts being disseminated to other nodes in the system within 11.5 seconds. Therefore, *IDA-Gossip* employs system resources at a much earlier time compared to classic gossip.

With all message sizes, *IDA-Gossip* provides lower latency compared to classic gossip for all message sizes. With messages size 1, 2, 16, and 36 MB: *IDA-Gossip* provides latency values of 2.7, 6.0, 43.8, and 98.9 seconds while classic gossip instance provides 7.1, 13.3, 93.7, and 216.7 seconds respectively. This highlights that *IDA-Gossip* outperforms the classic gossip technique that we have considered.

In terms of network usage, *IDA-Gossip* and classic gossip provide similar performance: As depicted in Fig. 5.2, although the uploaded data per node values are very close to each other for specific message size, they are not the same, and the difference is less than 0.5% for all configurations. The small difference in uploaded data statistics stems from two facts: (1) *IDA-Gossip* makes use of Merkle hash trees to authenticate messages, and they have an overhead of $O(\log(\kappa))$ storage. (2), individual chunks come with a small overhead of metadata storage which is $O(1)$ because each chunk needs to be disseminated with some metadata. In this comparison in the case of *IDA-Gossip*, we did

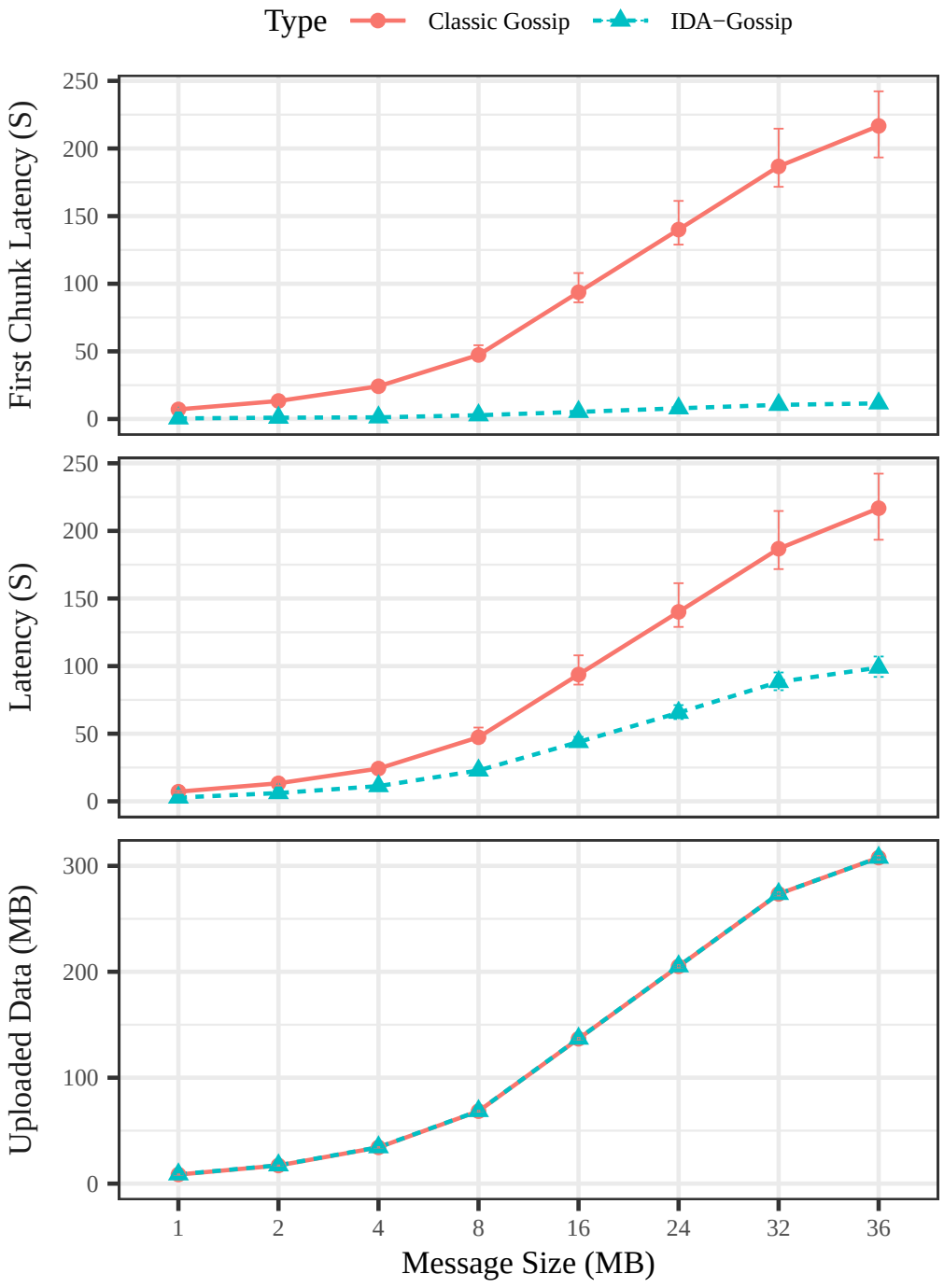

FIGURE 5.2: Classic gossip dissemination compared to *IDA-Gossip* with 128 chunks.

not observe a significant overhead caused by parity chunks because an *IDA-Gossip* node forwards messages until reconstructing the original message for that purpose an *IDA-Gossip* node needs to deliver $(1-\phi\kappa)$ chunks as we stated previously.

5.3.3 *IDA-Gossip* with different chunk counts and message sizes

We investigate the effect of chunk count and message size on the performance of *IDA-Gossip*. In this experiment, we considered *IDA-Gossip* instances with 16, and 256 chunk counts. Also, we considered message sizes between 1 and 36 MB. Fig. 5.3 depicts the collected statistics from experiments. First of all, the first chunk latency, latency, and uploaded data measurements for all instances of *IDA-Gossip* is increasing with the increase in message size. This is an expected result because nodes have limited bandwidth to disseminate messages, and increased message size causes an increase in latencies.

When we consider a specific message size, the first chunk latency of an *IDA-Gossip* instance with a high number of chunk counts is always lower—better—compared to another *IDA-Gossip* instance with a smaller chunk count. This is another expected result because an increased chunk count results in a smaller chunk size. Smaller chunks are disseminated faster in the network.

We made an interesting observation, *IDA-Gossip* with 32 chunks always provided the lowest latency measurements for all message sizes. The reason for this is that in this set of experiments we have used a dissemination concurrency value of 8 because each node forwards a message 8 times in parallel. This results in the best performance for *IDA-Gossip* with 32 chunks. For higher chunk counts, one needs to increase dissemination concurrency value otherwise the effect of latency added to channels—to emulate WAN conditions—will be overemphasized on the measured latency metric. We study the effect of dissemination concurrency value on the performance of *IDA-Gossip* in subsection 5.3.5

As seen in Fig. 5.3, the amount of uploaded data per node is increasing when the message size increases, as expected. However, for a given message size, all instances of *IDA-Gossip* nodes upload a similar amount of data. We note a minor difference lower than 1% between instances of *IDA-Gossip* with 16 chunks and 256 chunks due to the size of the Merkle proofs. This implies that a higher chunk count incurs a slightly higher bandwidth cost.

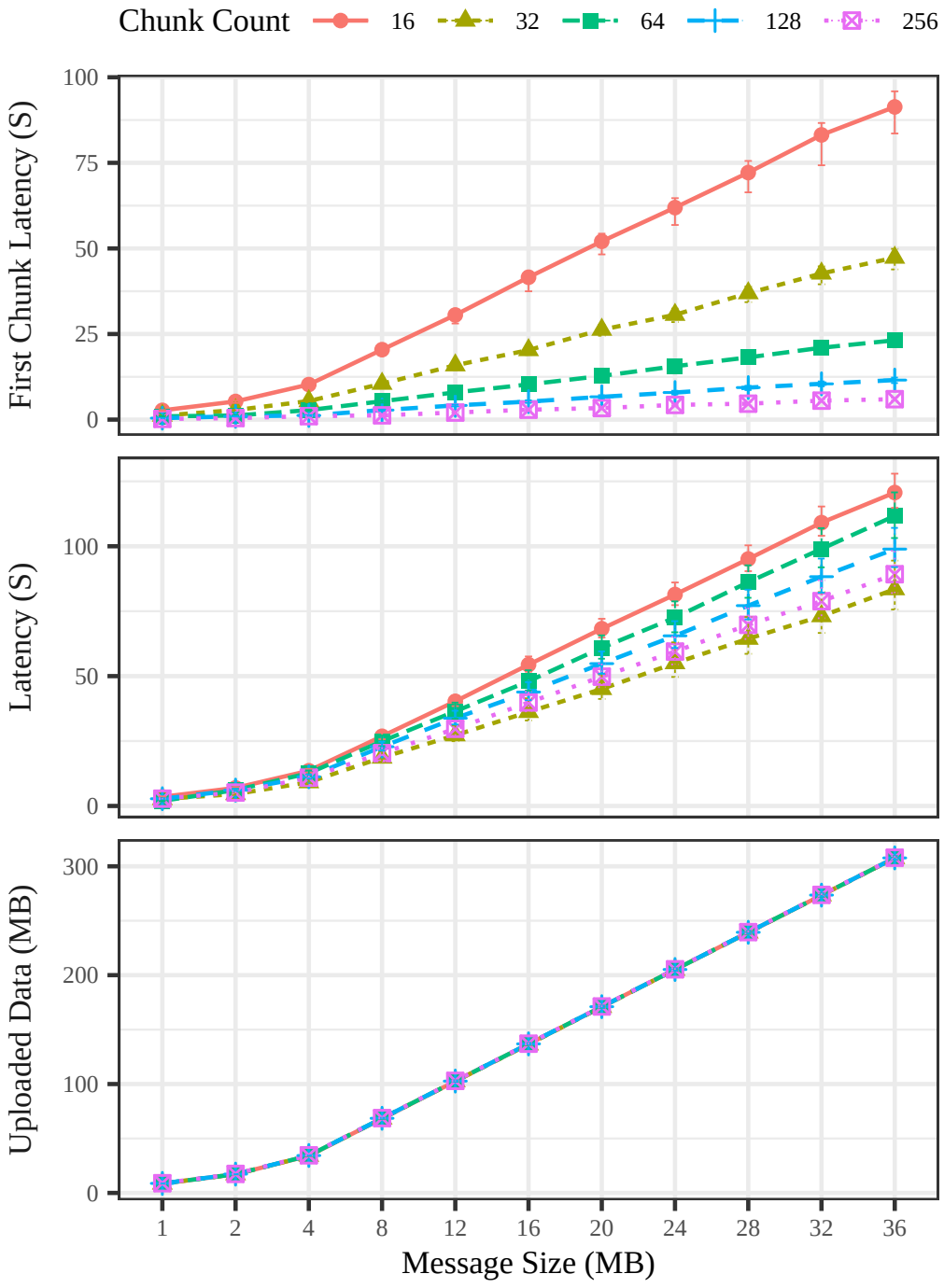

FIGURE 5.3: The behavior of *IDA-Gossip* with different chunk counts and message sizes.

5.3.4 *IDA-Gossip* with Faults

We now evaluate the performance of *IDA-Gossip* under faults by varying the proportion of byzantine nodes in the system, where byzantine nodes drop all the messages they receive. In our experiment, we exclude the byzantine behavior of altering the chunk content or the Merkle proofs as this does not impact the performance of *IDA-Gossip*. We set the message size to 2 MB, and vary the percentage of faulty nodes from 0% to 40%. We also explore the impact of varying the number of chunks (from $\kappa = 16$ to 256) in this faulty environment.

Fig. 5.4 shows the the evaluation results. In terms of first chunk latency, *IDA-Gossip* instances with high chunk counts (64, 128, and 256) provide stable results when increasing the percentage of byzantine nodes, whereas the other instances (with 16 and 32 chunks) provide slightly degraded first chunk latency.

The latency of all *IDA-Gossip* instances is increasing along with the ratio of faulty nodes. This increase is more pronounced in *IDA-Gossip* with $\kappa = 16$ and 32 chunks. In a fault-free system, *IDA-Gossip* with 32 chunks is providing the lowest latency but, under faults, its performance is affected more than other instances.

From these experiments, we conclude that *IDA-Gossip* with high chunk counts, *i.e.*, above 64 chunks, are more resilient to Byzantine behaviors, and provide smooth latency degradation when facing faults.

In terms of coverage, all instances of *IDA-Gossip* provide stable performance: with an increased fraction of faulty nodes we did not observe a significant decrease in coverage, and it is above 99% for all instances.

5.3.5 *IDA-Gossip* with different dissemination concurrency values

In case of big messages and limited bandwidth, a node should send messages to its neighbors one by one; otherwise, the sending might take longer time because concurrent send events will compete for the same bandwidth resource, and this will increase the latency of dissemination. *IDA-Gossip* chunks a large message into smaller pieces, and this makes the chunk-sending strategy vital for the performance of the system. In this subsection, we investigate the effect of dissemination concurrency on the performance of *IDA-Gossip* because choosing the wrong strategy might result in sub-optimal performance.

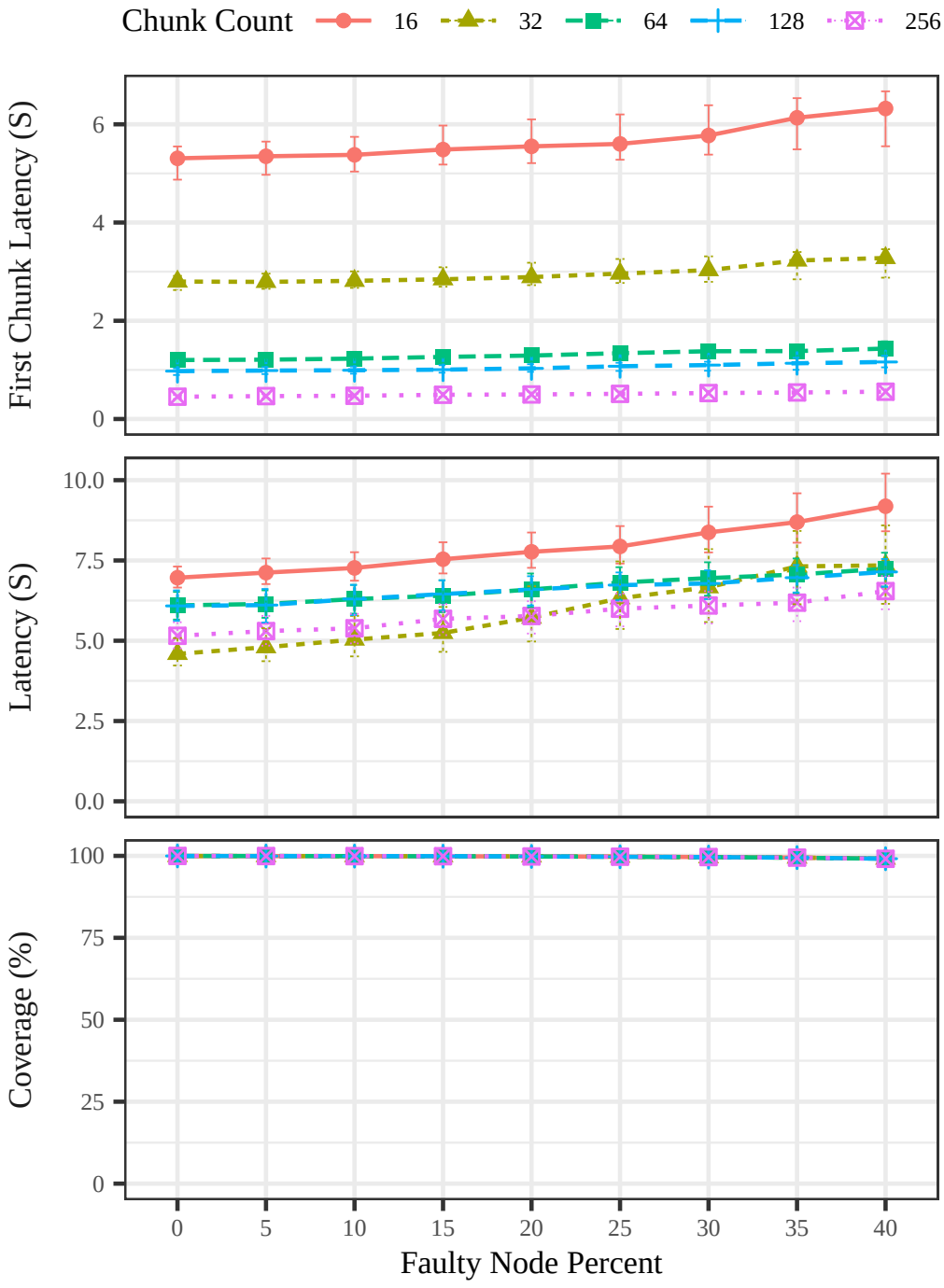

FIGURE 5.4: The behavior of *IDA-Gossip* with different percentages of Byzantine nodes.

We have conducted a set of experiments where we keep the message size constant—2MB—, and we vary the dissemination concurrency value between 1 and 128 so that a node serves a limited number of connections simultaneously. Without any restriction, dissemination concurrency is equal to the fanout value because a node opens a single connection to each sampled peer, and each peer connection is owned by a distinct thread that serves the connection. The value of dissemination concurrency 1 means that a node serves peers one by one, sequentially, and the value of 2 means that a node can serve up to two peers concurrently, and so on. To increase the dissemination concurrency above the fanout value, our implementation opens more than one connection to each peer; for example, with a fanout value of 8 to reach dissemination concurrency of 128, a node needs to open 16 (128/8) connections to each sampled peer.

In this set of experiments, as seen in Fig. 5.5, we have observed that smaller values of dissemination concurrency, like 1 and 2, provide the best first-chunk latency. When we increase the dissemination concurrency, the first chunk latency increases up to a point and later stays constant for all *IDA-Gossip* instances: This is because there is a limited number of chunks to disseminate. On the other hand, smaller values of dissemination concurrency values produce undesirable latency measurements for some *IDA-Gossip* instances: for example, *IDA-Gossip* with 256 and 128 chunks provides respectively 35 and 25 seconds latency with 1 simultaneously served connection. When we send a high number of chunks sequentially, the latency of the channel piles up, and it affects the final latency of dissemination.

Each *IDA-Gossip* instance provides the best latency with a different dissemination concurrency value: For example, *IDA-Gossip* with 16 chunks provides a latency of 5.153 seconds with dissemination concurrency of 4, and *IDA-Gossip* with 32 chunks provides a latency of 5.119 seconds with dissemination concurrency of 8. *IDA-Gossip* instance with 128 chunks provides the best latency with dissemination concurrency of 16.

In these experiments, we have considered only the message size of 2MB. The choice of chunk count,κ, and message size will change the characteristics because the size of a chunk depends on these parameters; therefore, to obtain optimal performance, one needs to examine the effect of dissemination concurrency according to considered chunk count, message size and considered network bandwidth capacity.

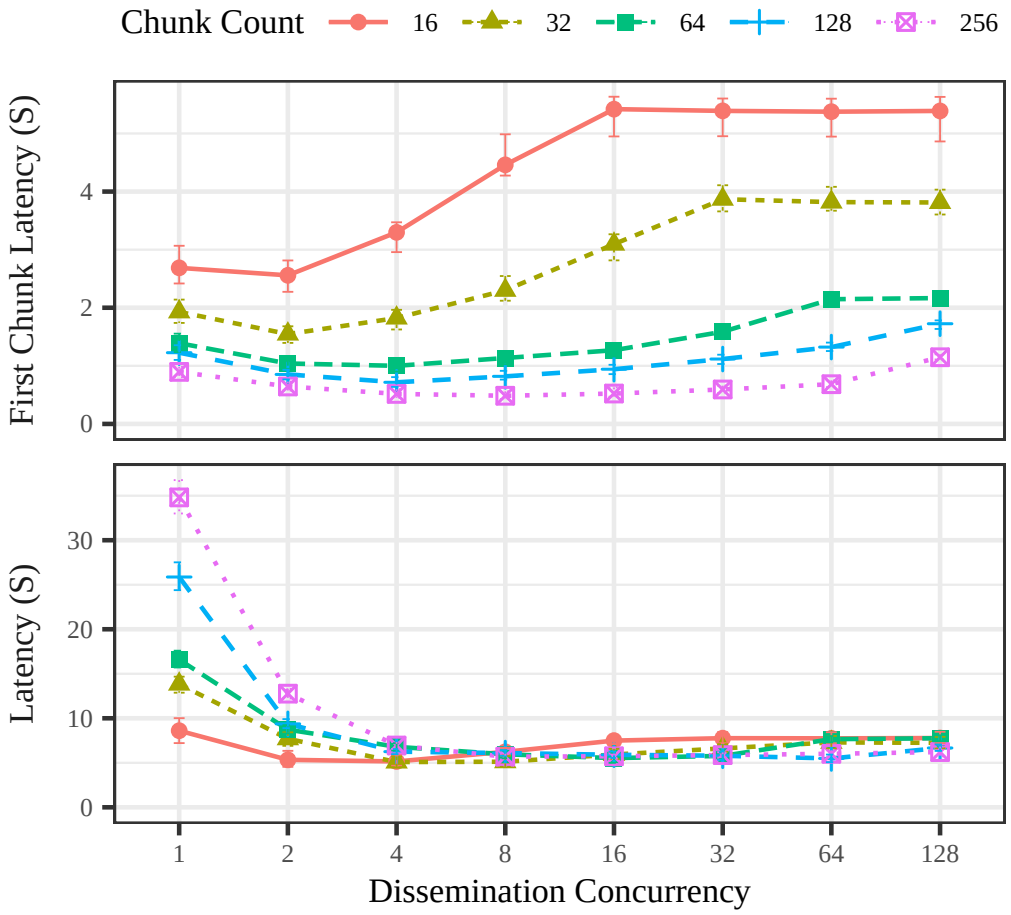

FIGURE 5.5: The behavior of *IDA-Gossip* with different dissemination concurrency values.

5.4 Evaluation with simulations

In this section, we compare the fault resilience of *IDA-Gossip* to the one of standard chunk-based gossip dissemination. We wish to quantify the resilience improvement brought by the use of erasure-coding in *IDA-Gossip*. To that end, we implement a simulation engine in Golang that simulates the two following gossiping strategies for disseminating a message M of size $|M|$: *IDA-Gossip* as described in this paper, and a chunk-based gossip dissemination where a message is chunked into n pieces.

5.4.1 Methodology

5.4.1.1 Simulations

Our simulation engine is discrete-time based where time is divided into consecutive rounds. In a round, a node can receive multiple chunks from different nodes. In the same round, a node forwards chunks that are received in the previous round. We use a fanout value of 8, therefore; for each message, a node samples 8 other nodes uniformly random and sends the message to these nodes.

The system consists of 4096 nodes, as in our experimental evaluation. A simulation run starts by having a source node selected uniformly at random sending chunks to its neighbors, and ends when there is no node with chunks to forward. We configure the simulations with a fixed proportion f of faulty nodes, that we vary from 0 to 99 with steps of 1. For each configuration, we run 1000 simulations. At the end of the 1000 runs, the simulator aggregates the results and computes the measured metrics.

5.4.1.2 Simulated strategies

In the *IDA-Gossip* strategy, the source node chunks the message M into $n = (1 - \phi)\kappa = 48$ data chunks and adds $\phi\kappa = 80$ parity chunks. Any set of $(1 - \phi)\kappa$ different chunks is enough to reconstruct the original message. Then, the source node samples $d = 16$ nodes from the system and sends them each $\kappa/d = 8$ chunks.

In the classic chunk-based gossip strategy, a source node splits M into $n = 48$ chunks and sends each chunk to 8 other nodes. To reconstruct the original message, a node requires all n chunks. In both strategies, each node stores and forwards each received chunk until being able to reconstruct the original message. Then, nodes may only receive other chunks but will ignore them.

5.4.1.3 Measured metrics

To evaluate the resilience of both protocols, we measured the following metrics: 1) coverage, 2) dissemination failure ratio, 3) received chunk count, and 4) delivered chunk count. *Coverage* is the percentage of nodes that delivers the full message at the end of a simulation run. The *dissemination failure ratio* is the percentage of simulation runs in which none of the nodes, except the source, managed to reconstruct the original message. The *received chunk count* is the number of chunks received by a node in a simulation run. Note that a node can receive multiple copies of the same chunk because our gossip engine implements a push-based gossip where no communication happens between nodes to identify chunks that should be sent/receive according to already received chunks. The *delivered chunk count* is the number of distinct chunks delivered by a node to reconstruct the original message. This must be equal to the chunk count of the original message which is 48 for both simulated strategies.

5.4.2 *IDA-Gossip* vs classic multi-chunk gossip dissemination

For all measured metrics, we display their mean values in Fig. 5.6. *IDA-Gossip* and chunk-based gossip protocols behave similarly when considering the numbers of received and delivered chunks. Indeed, we employ the same push-based gossip technique in both strategies and use the same data chunk count $n = 48$. When the system consists of correct nodes only, the mean number of received chunks per node is 384. With $n = 48$ data chunks, and each node forwarding each delivered chunk to its 8 neighbors, it implies that each node sends $48 \times 8 = 384$ chunks in a simulation run. Therefore, each node is expected to receive 384 chunks on average. Also, each node is expected to deliver 48 chunks to finish dissemination.

Regarding coverage, *IDA-Gossip* provides 100% coverage with up to $f = 40\%$ of Byzantine nodes in the system. On the other hand, the classic chunk-based gossip suffers from low coverage even with a low fraction of faulty nodes because the probability of not delivering a single chunk of a message with an increased number of data chunks is increasing. This is not the case for *IDA-Gossip* because different nodes forward different subsets of chunks, and any subset with cardinality 48 is enough to reconstruct the original message.

We observe that *IDA-Gossip* starts suffering from dissemination failures slightly earlier than classic chunk-based gossip. This is because the dissemination fails

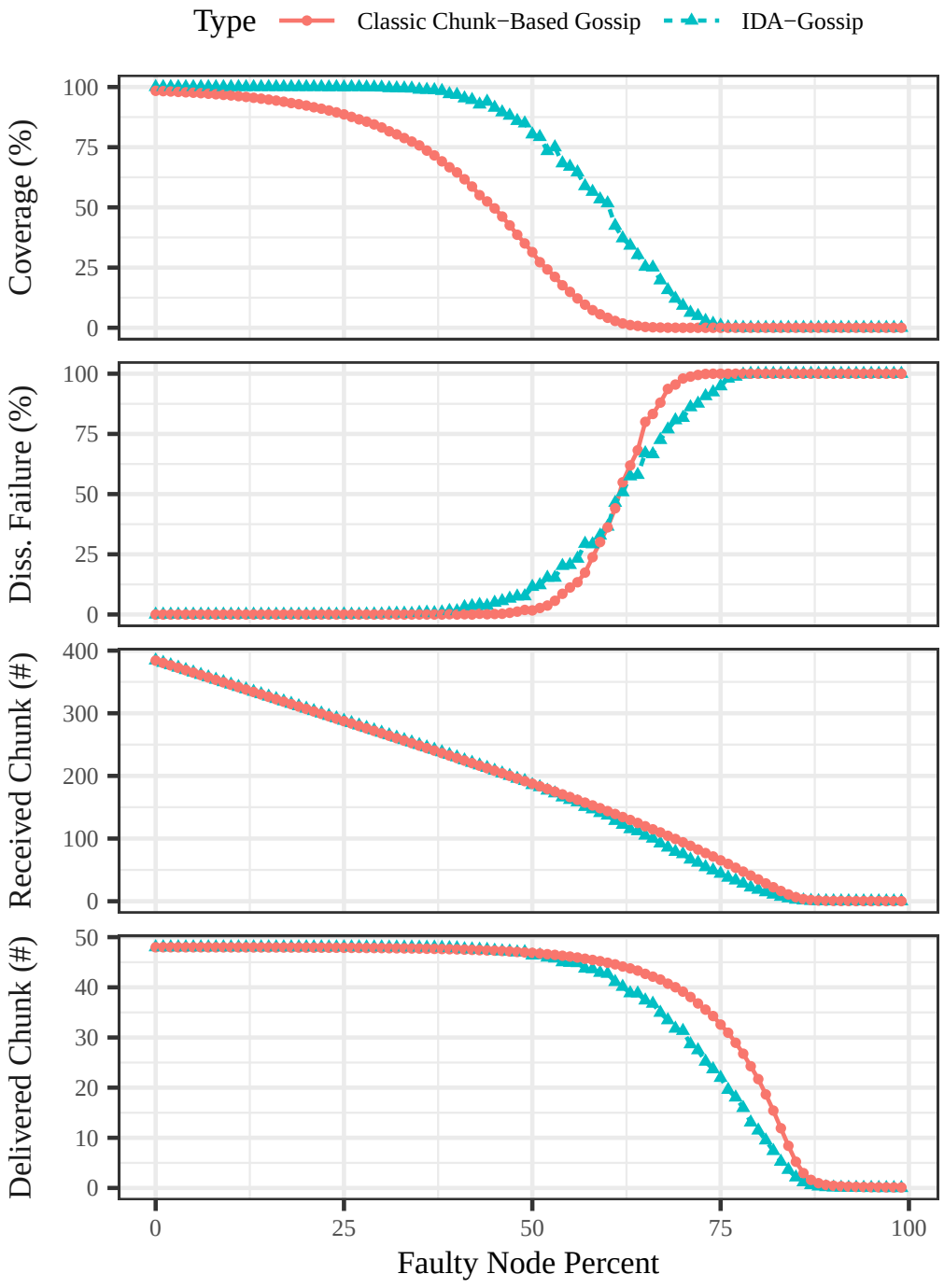

FIGURE 5.6: Comparison of *IDA-Gossip* with classic multi-chunk gossip under Byzantine behaviors.

if a source node samples more than 10 faulty nodes out of 16; in this case, an insufficient number of chunks is disseminated, and none of the nodes can reconstruct the original message except the source itself. The classic gossip approach suffers from this problem later than *IDA-Gossip* because each chunk is forwarded 8 times by the source node, diminishing the likeliness of losing a chunk because of faulty nodes.

5.5 Conclusion

Our experimental evaluation revealed that *IDA-Gossip* provides significant latency improvement compared to classic store-and-forward gossip dissemination: for all considered message sizes, even with a message size of 1 MB, *IDA-Gossip* provides the lowest latency. Also, *IDA-Gossip* better utilizes system resources compared to classic gossip dissemination because nodes start contributing earlier to the dissemination. Parity chunks and chunk authentication mechanisms of *IDA-Gossip* do not incur a significant bandwidth usage overhead compared to classic gossip dissemination.

We have observed, all instances of *IDA-Gossip* with different chunk counts provide similar bandwidth usage when the message size is kept constant. In a fault-free setting, *IDA-Gossip* with 32 chunks provides the lowest latency. On the contrary, *IDA-Gossip* instances with a high number of chunk counts provide better system utilization because of lower first chunk latency. When we injected faults into the system, we observed that *IDA-Gossip* instances with a high chunks count provide more resilience and graceful performance degradation. Also, we have observed that the choice of chunk-sending strategy is vital to obtain optimum performance from *IDA-Gossip* instances.

Our simulations revealed that, in the presence of faulty nodes in the system, *IDA-Gossip* provides excellent coverage—above 99%—compared to classic multi-chunk gossip dissemination. *IDA-Gossip* starts to suffer from dissemination failures earlier than classic gossip dissemination where none of the nodes delivers the message except the source. Finally, our simulations showed that *IDA-Gossip* and classic multi-chunk gossip dissemination provide similar bandwidth usage characteristics in terms of received and send chunk counts. Therefore, the use of parity chunks does not incur an overhead on bandwidth usage.

6 Conclusions and future work

6.1 Conclusions

In this thesis, we propose solutions that aim to improve the performance of leader-based blockchains by removing bottlenecks in the consensus and the network layers. We made two contributions: we developed *ALDER*—a generic construction to enrich leader-based blockchains with multiple leaders, and we provided an in-depth analysis of $IDA-Gossip$—a chunk-based efficient gossip dissemination protocol. In more detail, we have the following contributions:

- The *ALDER* construction consists of three primitives to multiplex the consensus protocols of leader-based blockchains: transaction hash space partitioning, multiple leader election and bucket assignment, and multiplexed consensus and macroblocks. *ALDER* partitions the hash space of transactions into Cl disjoint buckets. *ALDER* updates the leader election mechanism of the base consensus protocol to elect multiple leaders and each elected leader is assigned a transaction bucket in a publicly verifiable manner. Leaders propose blocks, and nodes decide on a subset of them using the multiplex version of the base consensus protocol. Decided blocks are appended to the distributed ledger in the form of a macroblock that can contain up to Cl blocks. *ALDER* targets bottlenecks that stem from a single leader by spreading the cost of leadership among many leaders: multiple leaders process user requests concurrently. Also, the design of *ALDER* provides mechanisms to circumvent inefficiencies that originate from the network plane: multiple leaders of *ALDER* can submit smaller blocks that disseminate faster than one big block. We evaluated *ALDER* by applying it on top of three state-of-the-art blockchains: Algorand, Rapidchain, and Bitcoin. First, we revealed the extent of consensus and network layer bottlenecks on these blockchains using experiments. Later, we applied Alder on top of these blockchains by implementing enriched protocols. Next, we quantified the benefits of *ALDER* using large-scale experiments. Our experiments are designed to measure normal case and

scalability performance. In normal case experiments, we measured the throughput and latency characteristics of enriched versions of the protocols. In scalability experiments, we measured the throughput degradation and latency increase of implementations by increasing the number of nodes in the system. Our evaluations show that in the presence of consensus and network layer bottlenecks, $ALDER$ can provide up to 300% throughput and latency gain. Furthermore, when there are no bottlenecks, $ALDER$ preserves the performance of the base consensus protocol. Finally, our scalability experiment conducted on 100 machines deploying up to 10,000 processes revealed that $ALDER$ preserves the scalability properties of the base consensus protocol by providing the same throughput degradation and latency increase characteristics as the base consensus protocol.

- $IDA - Gossip$ is proposed in the context of blockchains: designed to improve the dissemination latency of the Rapidchain protocol. It disseminates messages using chunk-based gossip dissemination. It uses parity chunks to protect them against loss: the source node pieces a large message into chunks and calculates parity chunks using an erasure coding scheme. $IDA - Gossip$ uses Merkle trees to authenticate each chunk efficiently. The source node distributes chunks among its peers: the source node's peers disseminate chunks on its behalf. Although it has potential use in many blockchains, its properties have not been studied in depth, which is the main obstacle to its adoption. Our work fills a gap in the literature by investing its properties under different conditions using experiments and simulations: we compared the performance of $IDA - Gossip$ with classical gossip dissemination mechanisms, and we investigate the effect of protocol parameters on the performance of $IDA - Gossip$ in the presence of faulty nodes. Our experimental evaluation revealed that *IDA-Gossip* provides significant latency improvement compared to classic store-and-forward gossip dissemination where a message is disseminated in its entirety: With messages size 1, 2, 16, and 36 MB: *IDA-Gossip* provides latency values of 2.7, 6.0, 43.8, and 98.9 seconds while classic gossip instance provides 7.1, 13.3, 93.7, and 216.7 seconds respectively. Also, *IDA-Gossip* better utilizes system resources than classic gossip dissemination because nodes start contributing earlier to the dissemination: all *IDA-Gossip* instances start contributing dissemination within 11.5 seconds, while classic gossip dissemination requires around 230 seconds. Furthermore, parity chunks and the chunk authentication mechanism of *IDA-Gossip* do not incur a significant bandwidth usage overhead compared to classic gossip

dissemination, as nodes do not require disseminating extra chunks after delivering the original message. Also, we have observed that *IDA-Gossip* instances with a high chunk count provide more resilience and graceful performance degradation in the presence of faulty nodes. Also, choosing a chunk-sending strategy according to considered message size and chunk count is vital to obtain optimum performance from *IDA-Gossip* instances. Our simulations revealed that *IDA-Gossip* provides 100% coverage with up to $f = 40\%$ of Byzantine nodes in the system. On the other hand, the classic chunk-based gossip suffers from low coverage even with a low fraction of faulty nodes because the probability of not delivering a single chunk of a message with an increased number of data chunks is increasing. Furthermore, *IDA-Gossip* suffers from dissemination failures earlier than classic multi-chunk gossip dissemination, where none of the nodes delivers the message except the source. Finally, our simulations showed that *IDA-Gossip* and classic multi-chunk gossip dissemination provide similar bandwidth usage characteristics regarding received and sent chunk counts. Therefore, parity chunks do not incur an overhead on bandwidth usage compared to classic multi-chunk gossip dissemination.

6.2 Future work

$ALDER$ defines a set of reusable primitives that need to be implemented on top of a blockchain. It could be interesting to transform the $ALDER$ construction into a reusable blockchain framework for developing blockchains. Our evaluation of $ALDER$ considered three different blockchains. Applying $ALDER$ on top of other blockchains can help reveal its performance benefits. Also, our evaluations considered non-faulty nodes, and it remains a future work to understand the behavior of $ALDER$ with faulty nodes. Finally, we have provided arguments for the correctness of $ALDER$ construction, and it could be interesting to develop a formal proof of $ALDER$ using a proof assistant system.

Our evaluation of $IDA-Gossip$ was conducted in a blockchain-agnostic way. It could be interesting to test $IDA-Gossip$ with different blockchains to quantify its practical use and illuminate possible problems. Orthogonal to our work, the $IDA-Gossip$ protocol can be further improved using hybrid gossip dissemination techniques to increase dissemination efficiency.

In this thesis, we have mainly focused on the performance problems of leader-based blockchains because they are the most commonly deployed family of

blockchains. Although non-leader-based blockchains are not widely adopted, they are promising alternatives to leader-based blockchains because they do not suffer from consensus level and network level bottlenecks as leader-based blockchains. Investigation of the practical use of non-leader-based blockchains remains as a future work.

A Publications and Awards

A.1 Published papers in international peer reviewed conferences

- K. Korkmaz, J. Bruneau-Queyreix, S. Ben Mokhtar and L. Réveillère, "ALDER: Unlocking blockchain performance by multiplexing consensus protocols," 2022 IEEE 21st International Symposium on Network Computing and Applications (NCA), Boston, MA, USA, 2022, pp. 9-18, doi: 10.1109/NCA57778.2022.10013556.
- K. Korkmaz, J. Bruneau-Queyreix, S. Delbruel, S. B. Mokhtar and L. Réveillère, "In-depth analysis of the IDA-Gossip protocol," 2022 IEEE 21st International Symposium on Network Computing and Applications (NCA), Boston, MA, USA, 2022, pp. 139-147, doi: 10.1109/NCA57778.2022.10013564.

A.2 Awards

- The best student paper award – *21st IEEE International Symposium on Network Computing and Applications (NCA 2022) for the paper "In-depth analysis of the IDA-Gossip protocol"*

Bibliography

Alon, Noga et al. (2003). “Addendum to "Scalable Secure Storage when Half the System is Faulty"”. In: *Information and Computation* 2004.

Amir, Yair et al. (July 2011). “Prime: Byzantine Replication under Attack”. In: *IEEE Transactions on Dependable and Secure Computing* 8.4. Conference Name: IEEE Transactions on Dependable and Secure Computing, pp. 564–577. ISSN: 1941-0018. DOI: 10.1109/TDSC.2010.70.

Androulaki, Elli et al. (Apr. 2018). “Hyperledger fabric: a distributed operating system for permissioned blockchains”. In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys ’18. New York, NY, USA: Association for Computing Machinery, pp. 1–15. ISBN: 978-1-4503-5584-1. DOI: 10.1145/3190508.3190538. URL: https://doi.org/10.1145/3190508.3190538 (visited on 11/02/2022).

Aublin, Pierre-Louis et al. (Jan. 2015). “The Next 700 BFT Protocols”. In: *ACM Transactions on Computer Systems* 32.4, 12:1–12:45. ISSN: 0734-2071. DOI: 10.1145/2658994. URL: https://doi.org/10.1145/2658994 (visited on 01/03/2023).

Aumasson, Jean-Philippe and Daniel J. Bernstein (2012). “SipHash: a fast short-input PRF”. In: *International Conference on Cryptology in India*. Springer, pp. 489–508.

Bach, L. M., B. Mihaljevic, and M. Zagar (May 2018). “Comparative analysis of blockchain consensus algorithms”. In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 1545–1550. DOI: 10.23919/MIPRO.2018.8400278.

Badertscher, Christian et al. (2018). “Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 913–930.

Balouek, Daniel et al. (2013). "Adding Virtualization Capabilities to the Grid'5000 Testbed". en. In: *Cloud Computing and Services Science*. Ed. by Ivan I. Ivanov et al. Communications in Computer and Information Science. Cham: Springer International Publishing, pp. 3–20. ISBN: 978-3-319-04519-1. DOI: 10.1007/978-3-319-04519-1_1.

Ben-Or, Michael (Aug. 1983). "Another advantage of free choice (Extended Abstract): Completely asynchronous agreement protocols". In: *Proceedings of the second annual ACM symposium on Principles of distributed computing*. PODC '83. New York, NY, USA: Association for Computing Machinery, pp. 27–30. ISBN: 978-0-89791-110-8. DOI: 10.1145/800221.806707. URL: https://doi.org/10.1145/800221.806707 (visited on 01/03/2023).

Bentov, Iddo, R. Pass, and E. Shi (2016). "Snow White: Provably Secure Proofs of Stake". In: *IACR Cryptol. ePrint Arch.* URL: https://www.semanticscholar.org/paper/Snow-White%3A-Provably-Secure-Proofs-of-Stake-Bentov-Pass/ac482f29106a16778805db32a4e71f77737f8f3e (visited on 12/16/2022).

Bessani, Alysson, João Sousa, and Eduardo E.P. Alchieri (June 2014). "State Machine Replication for the Masses with BFT-SMART". In: *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. ISSN: 2158-3927, pp. 355–362. DOI: 10.1109/DSN.2014.43.

Bitcoin Transactions Per Day (Jan. 2023). URL: https://ycharts.com/indicators/bitcoin_transactions_per_day (visited on 01/19/2023).

Bitnodes (Aug. 2023). en. URL: https://bitnodes.io/ (visited on 01/08/2023).

Bloom, Burton H. (July 1970). "Space/time trade-offs in hash coding with allowable errors". In: *Communications of the ACM* 13.7, pp. 422–426. ISSN: 0001-0782. DOI: 10.1145/362686.362692. URL: https://doi.org/10.1145/362686.362692 (visited on 01/01/2023).

Bolze, Raphaël et al. (2006). "Grid'5000: A large scale and highly reconfigurable experimental grid testbed". In: *The International Journal of High Performance Computing Applications* 20.4, pp. 481–494.

Budhiraja, Navin et al. (1993). "The primary-backup approach". In: *Distributed systems* 2, pp. 199–216.

Castro, Miguel, Barbara Liskov, et al. (1999). "Practical byzantine fault tolerance". In: *3th USENIX Symposium on Operating Systems Design and Implementation (OSDI 99)*.

Castro, Miguel et al. (2003). "Splitstream: High-bandwidth multicast in a cooperative environment". In: *In SOSP'03*.

Cohen, Bram (2003). "Incentives build robustness in BitTorrent". In: *Workshop on Economics of Peer-to-Peer systems*. Vol. 6. Berkeley, CA, USA, pp. 68–72.

Corallo, Matt (2015). *Bitcoin Relay Network*. URL: https://bitcoinrelaynetwork.org/ (visited on 01/01/2023).

– (Apr. 2016a). *Compact Block Relay*. original-date: 2013-11-19T17:18:41Z. URL: https://github.com/bitcoin/bips/blob/15c8203eb36304efa1e4588b950f62a5bb32f965/bip-0152.mediawiki (visited on 12/31/2022).

– (2016b). *FIBRE Fast Internet Bitcoin Relay Engine*. URL: https://bitcoinfibre.org/ (visited on 01/01/2023).

Countries Where Bitcoin Is Legal and Illegal (Jan. 2023). en. URL: https://www.investopedia.com/articles/forex/041515/countries-where-bitcoin-legal-illegal.asp (visited on 01/19/2023).

Crain, Tyler, Christopher Natoli, and Vincent Gramoli (2021). "Red Belly: a secure, fair and scalable open blockchain". In: *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P'21)*.

Crain, Tyler et al. (Nov. 2018). "DBFT: Efficient Leaderless Byzantine Consensus and its Application to Blockchains". In: *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pp. 1–8. DOI: 10.1109/NCA.2018.8548057.

Croman, Kyle et al. (2016). "On Scaling Decentralized Blockchains". en. In: *Financial Cryptography and Data Security*. Ed. by Jeremy Clark et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 106–125. ISBN: 978-3-662-53357-4. DOI: 10.1007/978-3-662-53357-4_8.

Cryptocurrency Prices, Charts And Market Capitalizations (2023). en. URL: https://coinmarketcap.com/ (visited on 01/19/2023).

Demers, Alan et al. (Dec. 1987). "Epidemic algorithms for replicated database maintenance". In: *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*. PODC '87. New York, NY, USA: Association

for Computing Machinery, pp. 1–12. ISBN: 978-0-89791-239-6. DOI: 10.1145/41840.41841. URL: https://doi.org/10.1145/41840.41841 (visited on 10/25/2022).

Dhillon, Vikram, David Metcalf, and Max Hooper (2017). "The hyperledger project". In: *Blockchain enabled applications*. Springer, pp. 139–149.

Dingledine, Roger, Nick Mathewson, and Paul Syverson (2004). *Tor: The second-generation onion router*. Tech. rep. Naval Research Lab Washington DC.

Dorri, Ali, Salil S. Kanhere, and Raja Jurdak (Apr. 2017). "Towards an Optimized BlockChain for IoT". In: *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 173–178.

Douceur, John R. (2002). "The sybil attack". In: *Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers 1*. Springer, pp. 251–260. ISBN: 3-540-44179-4.

Dziembowski, Stefan et al. (2015). "Proofs of Space". en. In: *Advances in Cryptology – CRYPTO 2015*. Ed. by Rosario Gennaro and Matthew Robshaw. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 585–605. ISBN: 978-3-662-48000-7. DOI: 10.1007/978-3-662-48000-7_29.

Eischer, Michael and Tobias Distler (2019). "Scalable Byzantine fault-tolerant state-machine replication on heterogeneous servers". In: *Computing* 101.2, pp. 97–118.

Ekblaw, Ariel and Asaf Azaria (Apr. 2016). "MedRec: Medical Data Management on the Blockchain". en. In: *Viral Communications*. URL: https://viral.media.mit.edu/pub/medrec/release/1 (visited on 01/08/2023).

Eyal, Ittay et al. (Oct. 2015). "Bitcoin-NG: A Scalable Blockchain Protocol". In: URL: https://www.semanticscholar.org/paper/Bitcoin-NG%3A-A-Scalable-Blockchain-Protocol-Eyal-Gencer/d28dd2fe450d50a414e83461bfa3449750a573d0 (visited on 12/12/2022).

Eyal, Ittay et al. (2016). "Bitcoin-NG: A Scalable Blockchain Protocol". In: *13th USENIX symposium on networked systems design and implementation (NSDI 16)*, pp. 45–59.

Felber, Pascal et al. (2011). "PULP: an Adaptive Gossip-Based Dissemination Protocol for Multi-Source Message Streams." en. In: *Peer-to-Peer Networking*

and Applications 5.1, p. 74. DOI: 10.1007/s12083-011-0110-x. URL: https://hal.inria.fr/hal-00646616 (visited on 08/27/2022).

Feldman, Michal and John Chuang (July 2005). "Overcoming free-riding behavior in peer-to-peer systems". In: *ACM SIGecom Exchanges* 5.4, pp. 41–50. DOI: 10.1145/1120717.1120723. URL: https://doi.org/10.1145/1120717.1120723 (visited on 01/31/2023).

Fischlin, Marc et al. (2010). "Random oracles with (out) programmability". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, pp. 303–320.

Frey, D. et al. (Aug. 2010). "Boosting Gossip for Live Streaming". In: *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*. ISSN: 2161-3567, pp. 1–10. DOI: 10.1109/P2P.2010.5569962.

Gao, Yue et al. (June 2019). "Topology Measurement and Analysis on Ethereum P2P Network". In: *2019 IEEE Symposium on Computers and Communications (ISCC)*. ISSN: 2642-7389, pp. 1–7. DOI: 10.1109/ISCC47284.2019.8969695.

Gilad, Yossi et al. (2017). "Algorand: Scaling byzantine agreements for cryptocurrencies". In: *Proceedings of the 26th symposium on operating systems principles*, pp. 51–68.

Godfrey, P. Brighten, Scott Shenker, and Ion Stoica (Aug. 2006). "Minimizing churn in distributed systems". In: *ACM SIGCOMM Computer Communication Review* 36.4, pp. 147–158. ISSN: 0146-4833. DOI: 10.1145/1151659.1159931. URL: https://doi.org/10.1145/1151659.1159931 (visited on 11/03/2022).

Golan Gueta, Guy et al. (June 2019). "SBFT: A Scalable and Decentralized Trust Infrastructure". In: *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. ISSN: 1530-0889, pp. 568–580. DOI: 10.1109/DSN.2019.00063.

Goodrich, Michael T. and Michael Mitzenmacher (Sept. 2011). "Invertible bloom lookup tables". In: *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 792–799. DOI: 10.1109/Allerton.2011.6120248.

Guerraoui, Rachid et al. (2010). *LiFTinG: Lightweight Freerider-Tracking Protocol in Gossip*. Research Report RR-6913. INRIA, p. 21. URL: https://hal.inria.fr/inria-00379408 (visited on 09/29/2022).

Gupta, Suyash, Jelle Hellings, and Mohammad Sadoghi (2021). "Rcc: Resilient concurrent consensus for high-throughput secure transaction processing". In: *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, pp. 1392–1403.

Han, Meng et al. (Sept. 2018). "A Novel Blockchain-based Education Records Verification Solution". In: *Proceedings of the 19th Annual SIG Conference on Information Technology Education*. SIGITE '18. New York, NY, USA: Association for Computing Machinery, pp. 178–183. ISBN: 978-1-4503-5954-2. DOI: 10.1145/3241815.3241870. URL: https://doi.org/10.1145/3241815.3241870 (visited on 01/08/2023).

Huh, Seyoung, Sangrae Cho, and Soohyung Kim (Feb. 2017). "Managing IoT devices using blockchain platform". In: *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pp. 464–467. DOI: 10.23919/ICACT.2017.7890132.

Juels, Ari and Burton S. Kaliski (Oct. 2007). "Pors: proofs of retrievability for large files". In: *Proceedings of the 14th ACM conference on Computer and communications security*. CCS '07. New York, NY, USA: Association for Computing Machinery, pp. 584–597. ISBN: 978-1-59593-703-2. DOI: 10.1145/1315245.1315317. URL: https://doi.org/10.1145/1315245.1315317 (visited on 12/16/2022).

Kogias, Eleftherios Kokoris et al. (2016). "Enhancing bitcoin security and performance with strong consistency via collective signing". In: *USENIX Security Symposium*, pp. 279–296.

Kokoris-Kogias, Eleftherios et al. (May 2018). "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding". In: *2018 IEEE Symposium on Security and Privacy (SP)*. ISSN: 2375-1207, pp. 583–598. DOI: 10.1109/SP.2018.000-5.

Kotla, Ramakrishna et al. (Oct. 2007). "Zyzzyva: speculative byzantine fault tolerance". In: *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. SOSP '07. New York, NY, USA: Association for Computing Machinery, pp. 45–58. ISBN: 978-1-59593-591-5. DOI: 10.1145/

1294261.1294267. URL: https://doi.org/10.1145/1294261.1294267 (visited on 01/03/2023).

Krawczyk, Hugo, Mihir Bellare, and Ran Canetti (1997). *HMAC: Keyed-hashing for message authentication*. Tech. rep. ISBN: 2070-1721.

Küfeoglu, S. and M. Özkuran (May 2019). *Energy Consumption of Bitcoin Mining*. en. Working Paper. Accepted: 2019-06-28T08:38:17Z. Faculty of Economics, University of Cambridge. DOI: 10.17863/CAM.41230. URL: https://www.repository.cam.ac.uk/handle/1810/294129 (visited on 01/07/2023).

Lamport, Leslie (2001). "Paxos made simple". In: *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pp. 51–58.

– (2006). "Fast paxos". In: *Distributed Computing* 19.2. ISBN: 1432-0452 Publisher: Springer, pp. 79–103.

Lau, F. et al. (Oct. 2000). "Distributed denial of service attacks". In: *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0*. Vol. 3. ISSN: 1062-922X, 2275–2280 vol.3. DOI: 10.1109/ICSMC.2000.886455.

Li, Harry C. et al. (2006). "{BAR} Gossip". en. In: URL: https://www.usenix.org/conference/osdi-06/bar-gossip (visited on 09/29/2022).

Li, Jiaxing et al. (Nov. 2020). "Blockchain-based public auditing for big data in cloud storage". en. In: *Information Processing & Management* 57.6, p. 102382. ISSN: 0306-4573. DOI: 10.1016/j.ipm.2020.102382. URL: https://www.sciencedirect.com/science/article/pii/S0306457320308773 (visited on 01/08/2023).

Lo Cigno, Renato, Alessandro Russo, and Damiano Carra (June 2008). "On Some Fundamental Properties of P2P Push/Pull Protocols". In: *2008 Second International Conference on Communications and Electronics*, pp. 67–73. DOI: 10.1109/CCE.2008.4578935.

Luu, Loi et al. (Oct. 2016a). "A Secure Sharding Protocol For Open Blockchains". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. New York, NY, USA: Association for Computing Machinery, pp. 17–30. ISBN: 978-1-4503-4139-4. DOI: 10.1145/

2976749.2978389. URL: https://doi.org/10.1145/2976749.2978389 (visited on 11/02/2022).

Luu, Loi et al. (Oct. 2016b). "Making Smart Contracts Smarter". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. New York, NY, USA: Association for Computing Machinery, pp. 254–269. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978309. URL: https://doi.org/10.1145/2976749.2978309 (visited on 01/08/2023).

Mao, Yanhua, Flavio P. Junqueira, and Keith Marzullo (2008). "Mencius: Building Efficient Replicated State Machines for WANs". In: *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*. OSDI'08. San Diego, California: USENIX Association, 369–384.

Mattern, Friedemann (1988). *Virtual time and global states of distributed systems*. Univ., Department of Computer Science.

Maurer, Ueli (1996). "Modelling a public-key infrastructure". In: *European Symposium on Research in Computer Security*. Springer, pp. 325–350.

Maymounkov, Petar and David Mazieres (2002). "Kademlia: A peer-to-peer information system based on the xor metric". In: *International Workshop on Peer-to-Peer Systems*. Springer, pp. 53–65.

Merkle, Ralph Charles (1979). *Secrecy, authentication, and public key systems*. Stanford university. ISBN: 9798660696336.

Micali, Silvio, Michael Rabin, and Salil Vadhan (1999). "Verifiable random functions". In: *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE, pp. 120–130.

Miller, Andrew et al. (May 2014). "Permacoin: Repurposing Bitcoin Work for Data Preservation". In: *2014 IEEE Symposium on Security and Privacy*. ISSN: 2375-1207, pp. 475–490. DOI: 10.1109/SP.2014.37.

Milosevic, Zarko, Martin Biely, and André Schiper (2013). "Bounded delay in Byzantine-tolerant state machine replication". In: *2013 IEEE 32nd International Symposium on Reliable Distributed Systems*. IEEE, pp. 61–70.

Milosevic, Zarko, Martin Hutle, and Andre Schiper (Oct. 2011). "On the Reduction of Atomic Broadcast to Consensus with Byzantine Faults". In: *2011 IEEE 30th International Symposium on Reliable Distributed Systems*. ISSN: 1060-9857, pp. 235–244. DOI: 10.1109/SRDS.2011.36.

Minoli, Daniel and Benedict Occhiogrosso (Sept. 2018). "Blockchain mechanisms for IoT security". en. In: *Internet of Things* 1-2, pp. 1–13. ISSN: 2542-6605. DOI: 10.1016/j.iot.2018.05.002. URL: https://www.sciencedirect.com/science/article/pii/S2542660518300167 (visited on 01/08/2023).

Mockapetris, P. and K. J. Dunlap (Aug. 1988). "Development of the domain name system". In: *Symposium proceedings on Communications architectures and protocols*. SIGCOMM '88. New York, NY, USA: Association for Computing Machinery, pp. 123–133. ISBN: 978-0-89791-279-2. DOI: 10.1145/52324.52338. URL: https://doi.org/10.1145/52324.52338 (visited on 01/07/2023).

Mokhtar, Sonia Ben, Jérémie Decouchant, and Vivien Quéma (Oct. 2014). "AcTinG: Accurate Freerider Tracking in Gossip". In: *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*. ISSN: 1060-9857, pp. 291–300. DOI: 10.1109/SRDS.2014.12.

Nakamoto, Satoshi (2008). "Bitcoin: A peer-to-peer electronic cash system". In: *Decentralized Business Review*, p. 21260.

NIST (Aug. 2015). *Secure Hash Standard (SHS)*. en. Tech. rep. Federal Information Processing Standard (FIPS) 180-4. U.S. Department of Commerce. DOI: 10.6028/NIST.FIPS.180-4. URL: https://csrc.nist.gov/publications/detail/fips/180/4/final (visited on 01/01/2023).

Novo, Oscar (Apr. 2018). "Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT". In: *IEEE Internet of Things Journal* 5.2. Conference Name: IEEE Internet of Things Journal, pp. 1184–1195. ISSN: 2327-4662. DOI: 10.1109/JIOT.2018.2812239.

Ongaro, Diego and John Ousterhout (2014). "In search of an understandable consensus algorithm". In: *2014 USENIX Annual Technical Conference (Usenix ATC 14)*, pp. 305–319. ISBN: 1-931971-10-2.

Ozisik, A. Pinar et al. (Aug. 2019). "Graphene: efficient interactive set reconciliation applied to blockchain propagation". In: *Proceedings of the ACM Special Interest Group on Data Communication*. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, pp. 303–317. ISBN: 978-1-4503-5956-6. DOI: 10.1145/3341302.3342082. URL: https://doi.org/10.1145/3341302.3342082 (visited on 01/01/2023).

Paar, Christof and Jan Pelzl (2009). *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media. ISBN: 3-642-04101-9.

Park, Kihong and Heejo Lee (Aug. 2001). "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets". In: *ACM SIGCOMM Computer Communication Review* 31.4, pp. 15–26. ISSN: 0146-4833. DOI: 10.1145/964723.383061. URL: https://doi.org/10.1145/964723.383061 (visited on 01/31/2023).

Park, Sehyun et al. (2019). "Nodes in the Bitcoin Network: Comparative Measurement Study and Survey". In: *IEEE Access* 7. Conference Name: IEEE Access, pp. 57009–57022. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2914098.

Park, Sunoo et al. (2018). "SpaceMint: A Cryptocurrency Based on Proofs of Space". en. In: *Financial Cryptography and Data Security*. Ed. by Sarah Meiklejohn and Kazue Sako. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 480–499. ISBN: 978-3-662-58387-6. DOI: 10.1007/978-3-662-58387-6_26.

Pass, Rafael, Lior Seeman, and Abhi Shelat (2017). "Analysis of the blockchain protocol in asynchronous networks". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EuroCrypt)*. Springer, pp. 643–673.

Pass, Rafael and Elaine Shi (2017). "Fruitchains: A fair blockchain". In: *Proceedings of the ACM symposium on principles of distributed computing (PODC)*, pp. 315–324.

Reed, I. S. and G. Solomon (June 1960). "Polynomial Codes Over Certain Finite Fields". In: *Journal of the Society for Industrial and Applied Mathematics* 8.2. Publisher: Society for Industrial and Applied Mathematics, pp. 300–304. ISSN: 0368-4245. DOI: 10.1137/0108018. URL: https://epubs.siam.org/doi/abs/10.1137/0108018 (visited on 09/21/2022).

Ren, Ling et al. (2017). "Practical synchronous byzantine consensus". In: *arXiv preprint arXiv:1704.02397*.

Reyna, Ana et al. (Nov. 2018). "On blockchain and its integration with IoT. Challenges and opportunities". en. In: *Future Generation Computer Systems* 88, pp. 173–190. ISSN: 0167-739X. DOI: 10.1016/j.future.2018.05.046.

URL: https://www.sciencedirect.com/science/article/pii/S0167739X17329205 (visited on 01/08/2023).

Rocket, Team et al. (2019). "Scalable and probabilistic leaderless BFT consensus through metastability". In: *arXiv preprint arXiv:1906.08936*.

Sanghavi, Sujay, Bruce Hajek, and Laurent Massoulie (Dec. 2007). "Gossiping With Multiple Messages". In: *IEEE Transactions on Information Theory* 53.12. Conference Name: IEEE Transactions on Information Theory, pp. 4640–4654. ISSN: 1557-9654. DOI: 10.1109/TIT.2007.909171.

Simion, Robert-George and Mihai-Lica Pura (May 2019). "A BitTorrent DHT Crawler". In: *2019 IEEE 13th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pp. 155–160. DOI: 10.1109/SACI46893.2019.9111636.

Sompolinsky, Yonatan and Aviv Zohar (2013). "Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains". In: *IACR Cryptol. ePrint Arch.* URL: https://www.semanticscholar.org/paper/Accelerating-Bitcoin's-Transaction-Processing.-Fast-Sompolinsky-Zohar/401680ef12c04c247c50737b9114c169c660aab9 (visited on 12/12/2022).

Stathakopoulou, Chrysoula, Tudor David, and Marko Vukolic (2019). "Mir-BFT: High-throughput BFT for blockchains". In: *arXiv preprint arXiv:1906.05552*.

Stathakopoulou, Chrysoula, Matej Pavlovic, and Marko Vukolić (2022). "State machine replication scalability made simple". In: *Proceedings of the Seventeenth European Conference on Computer Systems*, pp. 17–33.

Tanenbaum, Andrew S. and Maarten van Steen (2006). *Distributed Systems: Principles and Paradigms (2nd Edition)*. USA: Prentice-Hall, Inc. ISBN: 978-0-13-239227-3.

Truong, Nguyen Binh et al. (2020). "GDPR-Compliant Personal Data Management: A Blockchain-Based Solution". In: *IEEE Transactions on Information Forensics and Security* 15. Conference Name: IEEE Transactions on Information Forensics and Security, pp. 1746–1761. ISSN: 1556-6021. DOI: 10.1109/TIFS.2019.2948287.

Tschipper, Peter (Jan. 2016). *Xtreme Thinblocks*. original-date: 2016-09-06T01:50:00Z. URL: https://github.com/BitcoinUnlimited/BUIP/blob/d818f89647bc3f499b2896cd3655d6f349e3beb3/010.md (visited on 01/01/2023).

Turek, J. and D. Shasha (June 1992). "The many faces of consensus in distributed systems". In: *Computer* 25.6. Conference Name: Computer, pp. 8–17. ISSN: 1558-0814. DOI: 10.1109/2.153253.

Wang, Jiaping and Hao Wang (2019). "Monoxide: Scale out Blockchains with Asynchronous Consensus Zones". In: URL: https://www.semanticscholar.org/paper/Monoxide%3A-Scale-out-Blockchains-with-Asynchronous-Wang-Wang/ea639e0bb4e4ef7ec5cbc7c0915033de4c89fd65 (visited on 12/12/2022).

Wei, PengCheng et al. (Jan. 2020). "Blockchain data-based cloud data integrity protection mechanism". en. In: *Future Generation Computer Systems* 102, pp. 902–911. ISSN: 0167-739X. DOI: 10.1016/j.future.2019.09.028. URL: https://www.sciencedirect.com/science/article/pii/S0167739X19313494 (visited on 01/08/2023).

Wohrer, Maximilian and Uwe Zdun (2018). "Smart contracts: security patterns in the ethereum ecosystem and solidity". In: *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, pp. 2–8. ISBN: 1-5386-5986-7.

Wood, Daniel Davis (2014). "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER". In: URL: https://www.semanticscholar.org/paper/ETHEREUM%3A-A-SECURE-DECENTRALISED-GENERALISED-LEDGER-Wood/da082d8dcb56ade3c632428bfccb88ded0493214 (visited on 12/12/2022).

Yin, Maofan et al. (July 2019). "HotStuff: BFT Consensus with Linearity and Responsiveness". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. PODC '19. New York, NY, USA: Association for Computing Machinery, pp. 347–356. ISBN: 978-1-4503-6217-7. DOI: 10.1145/3293611.3331591. URL: https://doi.org/10.1145/3293611.3331591 (visited on 01/03/2023).

Yu, Haifeng et al. (May 2020a). "OHIE: Blockchain Scaling Made Simple". In: *2020 IEEE Symposium on Security and Privacy (SP)*. ISSN: 2375-1207, pp. 90–105. DOI: 10.1109/SP40000.2020.00008.

Yu, Haifeng et al. (2020b). "Ohie: Blockchain scaling made simple". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 90–105.

Zamani, Mahdi, Mahnush Movahedi, and Mariana Raykova (Oct. 2018). "RapidChain: Scaling Blockchain via Full Sharding". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. New York, NY, USA: Association for Computing Machinery, pp. 931–948. ISBN: 978-1-4503-5693-0. DOI: 10.1145/3243734.3243853. URL: https://doi.org/10.1145/3243734.3243853 (visited on 08/29/2022).

Zamani, Mahdi et al. (2018). "Rapidchain: Scaling blockchain via full sharding". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 931–948.

Zhaofeng, Ma et al. (Mar. 2020). "A Blockchain-Based Trusted Data Management Scheme in Edge Computing". In: *IEEE Transactions on Industrial Informatics* 16.3. Conference Name: IEEE Transactions on Industrial Informatics, pp. 2013–2021. ISSN: 1941-0050. DOI: 10.1109/TII.2019.2933482.

Zhu, Liehuang et al. (Feb. 2019). "Controllable and trustworthy blockchain-based cloud data management". en. In: *Future Generation Computer Systems* 91, pp. 527–535. ISSN: 0167-739X. DOI: 10.1016/j.future.2018.09.019. URL: https://www.sciencedirect.com/science/article/pii/S0167739X18311993 (visited on 01/08/2023).