Blockchain System Designs from the End-to-End Principle

Bowen Xue

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2023

Reading Committee:
Sreeram Kannan, Chair
Payman Arabshahi
James A Ritcey

Program Authorized to Offer Degree:
Electrical & Computer Engineering

University of Washington

Abstract

Blockchain System Designs from the End-to-End Principle

Bowen Xue

Chair of the Supervisory Committee:
Sreeram Kannan
Electrical & Computer Engineering

Blockchain is a system of computers and user endpoints owned by decentralized entities across the world and connected by an open Internet. It enables uncoordinated parties to interact with a shared state, but at every moment maintains a common view about the state among everyone without a centralized trusted entity. Those features have made Blockchain a unique digital platform for shared state applications, because it is open to all application developers and any users can permissionlessly interact with it. Under the hood, a blockchain is made of layers of distributed protocols to endure adversarial participants and network environment. As workload and interactions among protocols increase, it is necessary to decouple functionalities through abstraction, such that independent development on separate protocols is possible while maintaining interoperability to the rest of the system. But naive abstraction can introduce unnecessary complexity, especially when lower layers contain unwanted and redundant features. The end to end design principle provides a guideline to delineate boundary between layers for reducing those adverse effects. This thesis contains four innovations for blockchains at the layers of Network, Data Availability, Censorship Resistance and Fair Ordering. While taking consideration from the end-to-end principle, those protocols improve the performance of the existing protocols and expand additional functionalities. Combining them altogether creates a blockchain that offers both superior performance and maximal flexibility for applications.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I want to thank my advisor Sreeram Kannan for his mentorship in the past six years. He has given me numerous advice and provided abundant opportunities to explore new area of study. He always encourages new approaches and interpretation, and it feels like working with a fellow while developing the idea. Passion is a great component in conducting research and directing works, and he has shown it throughout the years. It was a great pleasure to work with him.

I also want to thank my parents, Dong Xue and Yuyin Wang, and my grandmother Shufang Zhao. They have played an important roles in my upbringing. I have been studied in the U.S for about 11 years. Although there is a long separation between us, my parents provide care and support to me throughout the time, it would be impossible to reach this far without their support. My grandmother has been an important figure since I was young. She is a doctor and I spent a lot of time with her in the hospital, though most of time I was just playing around. I have always admired her for how she treats patients.

At last, I want to thank my friends, those I know from the Internship: Soubhik Deb, Gautham Anant, Daniel Mancia, Sam Laferriere Cyr, Chris Moller, Siddhartha Jagannath, Robert Raynor and Jeffery Commons. Also I want to thanks those who I knew on the UW campus since my undergraduate and master here: Boling Yang, Jingxing Wang, Yiran Zhang, MingJian Fu. Also thanks others who helped me along the way: Hong Zhou.

DEDICATION

to my Family

Chapter 1

INTRODUCTION

Blockchain is an algorithmic tool that enables anonymous users to recognize a commonly shared state of applications without a trusted centralized entity. Any users can send transactions to mutate the state, and a blockchain provides a reliable expectation to everyone that if such mutation is valid, it will be executed correctly and observable to everyone.

This is typically achieved with two components: consensus and execution. A consensus component establishes a common set of transactions available to everyone. It assures users the platform is decentralized enough that their transactions would not get censored and the execution of transactions are executed correctly. The execution component processes the transactions and derives a final state. All the processing logic in the execution component is defined in smart contracts, which can be composed and sent as transactions by any users. The process of executing the transactions is deterministic, so given the consensus is correct, everyone can independently derive the same outcome. This thesis focuses on innovations in the consensus component.

Although any party can pemissionlessly participates both the execution and consensus, most blockchain participants only interact with executions by sending transactions for mutating the state. It is because running either consensus or execution software requires much heavier hardware. For consensus, the hardware needs to constantly communicate on the Internet and maintaining a database for the entire blockchain. For execution, the hardware needs to execute the transactions from the entire database to derive the final state in most cases. As the result, Blockchain in general consists of two types of participants that run different programs: client software(node) and user endpoints. A client node uses both consensus and execution software to establish agreement on all transactions and derive the final

state; whereas user endpoints only generate transactions and send it to some client nodes in the hope that those nodes would accept their transaction and incorporate them onto the blockchain. However, in blockchains all nodes are anonymous by nature, it is possible for nodes to launch attacks on some users or other nodes. The most severe attacks are the double spending attack and stalling attacks, which are direct violations of consensus component. Censorship attack is a more nuanced attack, it happens when nodes are colluding together to delay inclusion of transactions from certain users for a long enough time. At last, some domain specific applications in decentralized finance have an attack vector that relates to the transaction ordering; a sandwich attack can happen when nodes maliciously reorder the user's transactions for extracting profits.

Throughout the years, the scope of applications has increased beyond traditional cryptocurrencies to more sophisticated applications including decentralized exchange, auction platform and social network. Supporting all of them requires the consensus system to have high throughput and more robust protections against censorship and sandwich attacks. As we begin to optimize on the distributed protocol, it is important to separate the concerns and isolate the protocol complexity by separating consensus into modules.

A modular system consists of layers of protocols that define clear interfaces and functionalities. The lower layers provide properties necessary for the the upper layers. If designed well, new protocols can be readily built on top of the existing solution without starting from the basis, so they can immediately focus on the problems. The computer network is an accomplishment of layering architecture.

However a naive implementation of modular system in many cases hinders the performance and development of the system. If a protocol in the lower layer is on the critical path to many applications in the upper layer, then one unnecessary feature will incur large costs for those inheriting the additional features. On the other hand, even if protocols in lower layer contain properties desired for the upper layer, it might not meet the exact requirements that the upper protocols need, therefore the upper layer would have to implement features again. It is important to design modules carefully that provide just enough features without

incurring extra cost.

The end to end argument can be thought as a set of rational principles for organizing layered system. It is not an absolute rule, but a guideline that helps in application and protocol design analysis, as suggested by [136]. It encompasses design problem in many area. For example, In computer architecture design, RISC architecture can be seen as the result of applying end-to-end principle: the performance of a system is better if implementing exactly the instructions on a set of primitive tools. Essentially, any additional features imagined by the lower layer system designers are likely not meeting the demand of the upper applications, which would in the end provide its own implementation. However, applying end to end principle is a nuanced task, because sometimes a redundant feature in the lower layer are necessary to ensure the system running smoothly. For example, the link layer has a data re-transmission feature to handle packet drop. Although it is redundant to the re-transmission protocol at the higher TCP layer, removing it will incur overloading to the TCP layer and therefore increase the delay.

In this thesis, we discuss four protocols that divides a blockchain into an architecture of five layers. Figure 1.1 is a diagram that shows what each protocol is made of, some protocols might contain many subprotocols. Every layer in the diagram is responsible for specific functions.

- **The network layer** is responsible for transporting data among nodes, whose data can either be metadata from the consensus layer or actual transactions. It is the most basic requirement which all upper layer protocols rely on it.

- **The consensus layer** is responsible for agreeing at a common string of bits, provided from the upper layer. Although a blockchain can use the consensus layer to replicate the entire database of transactions among nodes, it would be inefficient, since some protocol in the upper layer does not require every node to download the entire data.

- **The data availability layer** provides minimal functionality to allow anyone to down-

load the data on demand. But by default, no one is required to download the entire database of transactions. It is one of the important ways to increase the transactions throughput. It can be used alone, but more importantly it can be used with consensus to drastically increase its throughput.

- **The censorship resistance layer** provides an additional properties which most leader based blockchain does not have, it guarantees to the users that the transactions will be included in the short term. It requires the lower layer to have the traditional properties offered by any consensus protocol.

- **The fair ordering layer** provides an even more refined properties than the censorship resistance. It requires that the transactions received by the blockchain must be sorted according to some fair ordering rule in the blockchain. It requires the lower layer to support censorship resistance.

For the rest of the Introduction, we provide a short description on how each protocol in the thesis innovates on different layers. Then each chapter dives into the details for those protocols.

1.0.1 ACeD - Decoupling Consensus and Data Availability

Traditional blockchain is inefficient, because it requires all nodes to download and execute the transactions. As more applications are using the share state of the blockchain, eventually the blockchain will be congested. There are many proposal to solve the issue, and one popular and simple solution is called sidechain. A sidechain has a smaller blockchain with a few nodes for running its own consensus; each node stores and executes the transactions; periodically every sidechain posts a hash of the latest blocks to the main blockchain. Because each smaller consensus does not send all its transactions to the main consensus layer, this method can drastically increase the throughput of the overall systems. However, sidechains are vulnerable to the Data Availability attack when the smaller consensus protocol withhold

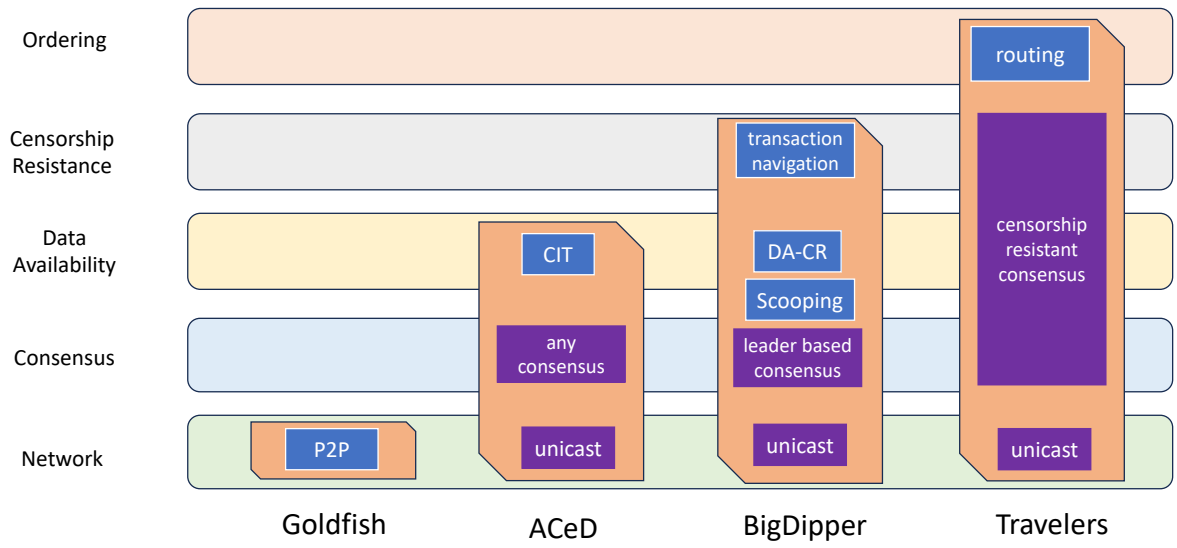

Figure 1.1: The blue boxes contain protocols based on innovations from this thesis. Each orange box represents a high level protocol.

data and can stall the blockchain forever. ACed is a protocol that investigates the relation between Data Availability and Consensus protocols. It prevents the Data availability attack while allowing each sidechain to just periodicaaly post hashes. ACeD [1] conceptualizes a module called Data Availablity oracle as a middleware, that separates the data availability out of the consensus from the main blockchain. No sidechain needs to store its own data for having its own data availability, but instead, a sidechain stores data inside the oracle and obtain a certificate from it. Then with the certificate, a sidechain can posts the hash in to the main chain to update its progress. Within the oracle, we developed an algorithm called coded interleaving tree(CIT) such that each node in the oracle does not have to download the entire data for all sidechains. We use erasure coding inside CIT to obtain a property that as more nodes join the oracle, each node would need to download less amount of data.

The concept of a oracle can increase the throughput without changing the existing consensus protocol. However, if we were to design the consensus protocol from the beginning, we can integrate the Data availability into the consensus protocol. BigDipper is a BFT (Byzantine Fault Tolerant) consensus system that integrates the Data availability among the consensus participants. We will discuss in fuller details in the next few sections.

1.0.2 Goldfish - Decoupling Consensus and Network

The peer to peer(P2P) decentralized network is the most common network protocol used by blockchains. The protocol is responsible for disseminating user's transaction, and propagating the meta-information important for the consensus block. In a typical P2P network, every node randomly selects a constant number of peers and establish persistent TCP network connections. Due to the randomization, with extreme high probability that the entire network is connected. To communicate between any two nodes, the number of intermediate nodes is approximately the logrithm of the total number of nodes. But there are multiple inefficient part in the P2P network that can increases the delay. First, all peer-to-peer connections in the network are logical; so it is possible that two nodes in Seattle are relayed through a node from Amsterdam. Second because the P2P network is inherently dynamic, whenever the network topology changes, it takes time for the P2P network to adapt to the new setting. As the result, there is a great efficiency loss in latency. Goldfish [12] is a distributed protocol that helps individual nodes to select best peers to connect with. The selection is based on the collected data from previous interactions with peers. Goldfish makes a decision on peer connections based on how well the peer can reduce its broadcast latency. The core of Goldfish is a matrix completion algorithm, that uses machine learning to infer how well a peer would perform if it was connected for broadcasting. With more information, every node can make a better choice on how to allocate its connections to existing peers.

Goldfish currently only uses a simple node selection algorithm, such that for every few blocks, it would switch its current connections to a fixed number of new peers to explore. It is possible to use more sophisticated bandit formulation to find solutions that allows variable

number of peers to explore in each time when connections are updated.

1.0.3 BigDipper - Censorship resistance layer

Conventional leader based consensus protocols focuses only on liveness which guarantees that all transactions are eventually included. But for many applications, it is important to have transactions included in a more precise manner, for example, an auction bid expiring in the next block height. BigDipper [149] is a consensus system that adds a layer of censorship resistant protocol, called transaction navigation, to guarantee the transactions are included in a timely manner. The navigation protocol expects the underlying layer provides a DA-CR property with abilities to quantify how well the censorship resistance is supported. DA-CR is an upgraded version of Data Availability, which stands for Data Availabilitu with Censorship Resistance. The BigDipper protocol provides three algorithm that implement DA-CR, each with varying degree of censorship resistance. Because DA-CR offers the additional property than the conventional Data availability, we need a special algorithm between consensus and DA-CR to preserve censorship resistance, and the algorithm that is called scooping algorithm. To implement DA-CR, we use a new approach to add erasure code (instead of CIT from ACeD) that allows more efficient encoding and assignment. Beside DA-CR, a blockchain can use other leaderless consensus protocol to replace DA-CR and Scooping.

In the next step, it would be interesting to test the BigDipper system in a decentralized setting that is reflective to the real world scenario, where the household Internet bandwidth is highly asymmetric. There are new consensus protocols designed each year, it would be interesting to see how BigDipper can integrate with others.

1.0.4 Travelers - Fair ordering layer

Fair ordering is a stronger property than censorship resistance. It requires that not only user's transactions are included, but they are included in a specific order based on certain notion of fairness. For example, the ordering of transactions could base on the reception time of each transaction when there is any node receiving it. **Travelers** is a performant system that uses

the layering architecture of blockchains and implements the most communication efficient fair ordering protocol. Unlike many other fair ordering prototype which redesigns the whole consensus protocol for the property, Travelers takes advantage of existing censorship resistant protocols, and then focuses primarily on the fair ordering property itself. The Travelers is based on a new fairness notion called probabilistic fair ordering, and a new protocol called routing protocol. In short, the new notion permits a small probability that in some blocks, the ordering of all transactions can be manipulated. But using the routing protocol can reduce the probability to exponentially small and hence deter the sandwich attack. The routing protocol has a major advantage that drastically reduces the message complexity of the system. After being processed by the routing protocol, every fairly ordered transactions will have a tag of timestamp. When consensus include those transaction, the ordering among transactions are simply decided based on the timestamp.

In Travelers, we use absolute timestamp to tag each transaction. However, it would be interesting to understand if we can use relative order to achieve a similar fairness notion while preserving the low communication complexity.

Chapter 2

ACED

2.1 Introduction

Public blockchains such as Bitcoin and Ethereum have demonstrated themselves to be secure in practice (more than a decade of safe and live operation in the case of Bitcoin), but at the expense of poor performance (throughput of a few transactions per second and hours of latency). Design of high performance (high throughput and low latency) blockchains without sacrificing security has been a major research area in recent years, resulting in new proof of work [33, 79, 150, 154], proof of stake [31, 62, 66, 84, 99], and hybrid [45, 122] consensus protocols. These solutions entail a wholesale change to the core blockchain stack and existing blockchains can only potentially upgrade with very significant practical hurdles (e.g.: hard fork of existing ledger). To address this concern, high throughput scaling solutions are explored via "layer 2" methods, including payment channels [67, 111] and state channels [92, 124, 143]. These solutions involve "locking" a part of the ledger on the blockchain and operating on this trusted, locked state on an application layer outside the blockchain; however the computations are required to be semantically closely tied to the blockchain (e.g.: using the same native currency for transactions) and the locked nature of the ledger state leads to limited applications (especially, in a smart contract platform such as Ethereum).

In practice, a popular form of scaling blockchain performance is via the following: a smaller blockchain (henceforth termed "side blockchain") derives trust from a larger blockchain (henceforth termed "trusted blockchain") by committing the hashes of its blocks periodically to the trusted blockchain (Fig.2.1a). The ordering of blocks in the side blockchain is now determined by the order of the hashes in the trusted blockchain; this way the security of the side blockchain is directly derived from that of the trusted blockchain. This mechanism is

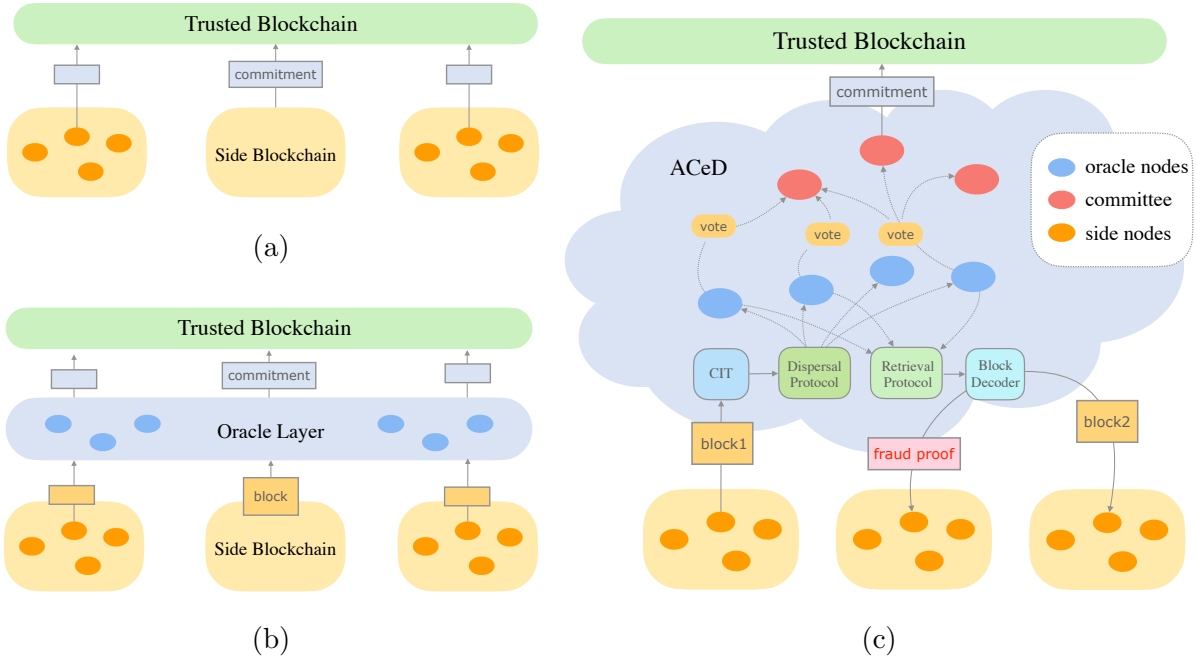

Figure 2.1: (a) Side blockchains commit the hashes of blocks to a larger trusted blockchain. (b) An oracle layer is introduced to ensure data availability. (c) ACeD is a data availability oracle.

simple, practical and efficient – a single trusted blockchain can cheaply support a large number of side blockchains, because it does not need to store, process or validate the semantics of the blocks of the side blockchains, only storing the hashes of them. It is also very popular in practice, with several side blockchains running on both Bitcoin and Ethereum; examples include donation trace (Binance charity [81]) and diplomas and credentials verification (BlockCert used by MIT among others [91]).

For decentralized operations of this simple scaling mechanism, any node in the side blockchain should be able to commit hashes to the trusted blockchain. This simple operation, however, opens up a serious vulnerability: an adversarial side blockchain node can commit the hash of a block without transmitting the block to any other side blockchain node.

Thus, while the hash is part of the ordering according to the trusted blockchain, the block corresponding to the hash is itself unavailable at any side blockchain node; this *data availability attack* is a serious threat to the liveness of the side blockchain. The straightforward solution to this data availability attack is to store all blocks on the trusted blockchain, but the communication and storage overhead for the trusted blockchain is directly proportional to the size of the side blockchains and the mechanism is no longer scalable.

We propose an intermediate "data availability oracle" layer that interfaces between the side blockchains and the trusted blockchain (Fig. 2.1b). The oracle layer accepts blocks from side blockchains, pushes verifiable commitments to the trusted blockchain and ensures data availability to the side blockchains. The N oracle layer nodes work together to reach a consensus about whether the proposed block is retrievable (i.e., data is available) and only then commit it to the trusted blockchain. The key challenge is how to securely and efficiently share the data amongst the oracle nodes to verify data availability; if all oracle nodes maintain a copy of the entire side blockchain data locally (i.e., repetition), then that obviously leads to simple majority vote-based retrievability but is not scalable. If the data is dispersed among the nodes to reduce redundancy (we call this "dispersal"), even one malicious oracle node can violate the retrievability. Thus it appears there is an inherent trade-off between security and efficiency.

The main point of this paper is to demonstrate that the trade-off between security and efficiency is not inherent; the starting point of our solution is the utilization of an erasure code such that different oracle nodes receive different coded chunks. A key issue is how to ensure the integrity and correctness of the coded chunks. Intuitively we can use a Merkle tree to provide the proof of inclusion for any chunk, but a malicious block producer can construct a Merkle tree of a bunch of nonsense symbols so that no one can successfully reconstruct the block. To detect such attacks, nodes can broadcast what they received and meanwhile download chunks forwarded by others to decode the data and check the correctness. Such a method is used in an asynchronous verifiable information dispersal (AVID) protocol proposed by [49]. AVID makes progress on storage savings with the help of

erasure code but sacrifices communication efficiency. Nodes still need to download all data chunks; hence, the communication complexity is $O(Nb)$. An alternative approach to detect incorrect coding attacks is via an *incorrect-coding proof* (also called fraud proof), which contains symbols that fail the parity check and can be provided by any node who tries to reconstruct the data; consider using an (n, k) Reed-Solomon code (which is referred as 1D-RS), with k coded symbols in the fraud proof, essentially not much better than downloading the original block (n symbols). For reducing the size of fraud proof, 2D-RS [23] places a block into a $(\sqrt{k}, \sqrt{k})$ matrix and apply $(\sqrt{n}, \sqrt{k})$ Reed-Solomon code on all columns and rows to generate n^2 coded symbols; 2D-RS reduces the fraud proof size to $O(\sqrt{b} \log b)$ if we assume symbol size is constant.

In summary (Table 2.1), to find a scalable solution for the data availability oracle problem, erasure code based methods must defend against the incorrect coding attack while minimizing the storage and communication cost. 1D-RS has low communication complexity but when the storing node is adversarial, the storage and download overhead are factor b worse than optimal. The storage and download overhead of AVID remains optimal even under adversarial storing node but the communication complexity is factor N worse than optimal. A full analysis on each performance entry in the table is provided in Appendix 2.14.

Our main technical contribution is a new protocol, called Authenticated Coded Dispersal (ACeD), that provides a scalable solution to the data availability oracle problem. ACeD achieves near-optimal performance on all parameters: optimal storage, download and communication overhead under the normal path, and near-optimal (worse by a logarithmic factor) storage and download overhead when the storing node is adversarial, cf. Table 2.1. We state and prove the security of the data availability guarantee and efficiency properties of ACeD in a formal security model (Section §2.3).

Technical summary of ACeD. There are four core components in ACeD, as is shown in Fig.2.1 with the following highlights.

- ACeD develops a novel coded commitment generator called *Coded Interleaving Tree*

Table 2.1: Performance metrics for different data availability oracles (N: number of oracle nodes, b: block size).

	maximal adversary fraction	normal case		worst case		communication complexity
		storage overhead	download overhead	storage overhead	download overhead	
uncoded (repetition)	1/2	$O(N)$	$O(1)$	$O(N)$	$O(1)$	$O(Nb)$
uncoded (dispersal)	1/N	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(b)$
AVID [49]	1/3	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(Nb)$
1D-RS	1/2	$O(1)$	$O(1)$	$O(b)$	$O(b)$	$O(b)$
2D-RS [23]	1/2	$O(1)$	$O(1)$	$O(\sqrt{b}\log b)$	$O(\sqrt{b}\log b)$	$O(b)$
ACeD	1/2	$O(1)$	$O(1)$	$O(\log b)$	$O(\log b)$	$O(b)$

Table 2.2: System Performance Metrics

Metric	**Formula**	**Explanation**
Maximal adversary fraction	β	The maximum number of adversaries is βN.
Storage overhead	$D_{\text{store}}/D_{\text{info}}$	The ratio of total storage used and total information stored.
Download overhead	$D_{\text{download}}/D_{\text{data}}$	The ratio of the size of downloaded data and the size of reconstructed data.
Communication complexity	D_{msg}	Total number of bits communicated.

(CIT), which is constructed layer by layer in an interleaved manner embedded with erasure codes. The interleaving property avoids downloading extra proof and thus minimizes the number of symbols needed to store.

- A dispersal protocol is designed to disperse tree chunks among the network with the least redundancy and we show how feasible dispersal algorithms ensure the reconstruction of all data.

- A hash-aware peeling decoder is used to achieve linear decoding complexity. The fraud proof is minimized to a *single parity equation*.

Performance guarantees of ACeD. Our main mathematical claim is that safety of ACeD holds as long as the trusted blockchain is secure, and ACeD is live as long as the trusted blockchain is live and a majority of oracle nodes are honest (i.e., follow protocol) (Section §2.5.1). ACeD is the first scalable data availability oracle that promises storage and communication efficiency while providing a guarantee for security with a provable bound and linear retrieval complexity; see Table 2.1 with details deferred to Section §2.5.2. The block hash commitment on the trusted blockchain and the size of fraud proof are both in constant size.

Incentives. From a rational action point of view, oracle nodes are naturally incentivized to mimic others' decisions without storing/operating on their own data. This "information cascade" phenomenon is recognized as a basic challenge of actors abandoning their own information in favor of inferences based on actions of earlier people when making sequential decisions [71]. In the context of ACeD, we carefully use the semantics of the data dispersal mechanisms to design a probabilistic auditing mechanism that ties the vote of data availability to an appropriate action by any oracle node. This allows us to create a formal rational actor model where the incentive mechanism can be mathematically modeled: we show that the honest strategy is a strong Nash equilibrium; the details are deferred to Appendix §2.12.

Algorithm to system design and implementation. We design an efficient system

architecture implementing the ACeD components. Multithreaded erasure code encoding and block pipelining designs significantly parallelize the operations leading to a high performing architecture. We implement this design in roughly 6000 lines of code in Rust and integrate ACeD with Ethereum (as the trusted blockchain). We discuss the design highlights and implementation optimizations in Section §2.6.

Evaluation. ACeD is very efficient theoretically, but also in practice. Our implementation of ACeD is run by lightweight oracle nodes (e.g.: up to 6 CPU cores) communicating over a wide area network (geographically situated in three continents) and is integrated with the Ethereum testnet Kovan with full functionality for the side blockchains to run Ethereum smart contracts. Our implementation scales a side blockchain throughput up to 10,000 tx/s while adding a latency of only a few seconds. Decoupling computation from Ethereum (the trusted blockchain) significantly reduces gas costs associated with side blockchain transactions: in our experiments on a popular Ethereum app Cryptokitties, we find that the gas (Ethereum transaction fee) is reduced by a factor of over 6000. This is the focus of Section §2.7.

We conclude the paper with an overview of our contributions in the context of the interoperability of blockchains in Section §2.8.

2.2 Related Work

Blockchain Scaling. Achieving the highest throughput and lowest latency blockchains has been a major focus area; this research has resulted in new proof of work [33, 79, 150, 154], proof of stake [31,62,66,84,99], and hybrid [45,122] consensus protocols. Scaling throughput linearly with the number of nodes in the network (known as *horizontal scaling*) is the focus of *sharding* designs: partition the blockchain and parallelize the computation and storage responsibilities [101,106,129]. Both horizontal and vertical scaling approaches directly impact the core technology (" layer 1") of the blockchain stack and their implementation in existing public blockchains is onerous. An alternate approach is to lock parts of the state of the blockchain and process transactions associated with this locked state in an application layer:

this approach includes payment channels [67, 111] and state channels [92, 124, 143]. In this paper we are concerned with a third (and direct) form of blockchain scaling: we interact very lightly with the trusted blockchain (only storing hashes of each block of the side blockchains) but design and implement an oracle that allows for scalable secure interactions between the side blockchains and the trusted blockchain.

Data Availability. Blockchain nodes that do not have access to all of the data (e.g.: light nodes in simple payment verification clients in Bitcoin) are susceptible to the data availability attack. One approach makes use of zero knowledge proof to represent the delta of the blockchain state, eliminating the need for the entire data [75]. Another approach is to have the light nodes rely on full nodes to notify the misbehavior of a malicious block proposer (which has withheld the publication of the full block). Coding the blockchain data to improve the efficiency of fraud proofs was first suggested in [22] (using 2D Reed-Solomon codes), and was strongly generalized by [155] via a cryptographic hash accumulator called Coded Merkle Tree (CMT) to generate the commitment of the block. Light nodes randomly sample symbols through anonymous channels and by collecting sampling results, an honest full node can either reconstruct the block or provide incorrect-coding proof. CMT reduces the proof size compared to [22]. However, the sampling mechanism is probabilistic (and not appropriate for implementation in the oracle setting of this paper); further CMT requires anonymous communication channels. In this paper, we propose an alternate coded Merkle tree construction (CIT) that is specific to the oracle operation: the method is equipped with a "push" scheme to efficiently and deterministically disseminate data among oracle nodes (no communication anonymity is needed).

Another perspective to understand the data availability problem is to find a secure, efficient way of dispersing data in the network and prove it is permanently retrievable. Such a problem has been discussed under the context of asynchronous verifiable information dispersal (AVID) [49], which applies erasure code to convert a file into a vector of symbols which later are hashed into a fingerprint. The symbols together with the fingerprint are disseminated to nodes. Then each node checks the hash and broadcasts its received copy

for guaranteeing retrievability. AVID improves storage saving but sacrifices communication efficiency; the performance is summarized in Table 2.1, with details in Appendix 2.14. The key insight of ACeD is that by using codes designed to have short fraud proofs, it is possible to avoid the additional echo phase in AVID where honest nodes flood the chunks to each other and thus provide near-optimal communication efficiency.

2.3 System and Security Model

The system is made up of three components: a trusted blockchain (that stores commitments and decides ordering), clients (nodes in side blockchains who propose data), and an intermediate oracle layer ensuring data availability (see Fig.2.1).

2.3.1 Network Model and Assumptions

There are two types of nodes in the network: oracle nodes and clients.

Oracle nodes are participants in the oracle layer. They receive block commitment requests from clients, including block headers, and a set of data chunks. After verifying the integrity and correctness of the data, they vote to decide whether the block is available or not and submit the results to the trusted blockchain.

Clients propose blocks and request the oracle layer to store and commit the blocks. They periodically update the local ledger according to the commitment states from the trusted blockchain and ask oracle nodes for the missing blocks on demand.

One of the key assumptions of our system is that the trusted blockchain has a persistent order of data and liveness for its service. Besides, we assume that in the oracle layer, there is a majority of honest nodes. For clients, we only assume that at least one client is honest (for liveness). Oracle nodes are connected to all clients. The network is synchronous, and the communication is authenticated and reliable.

2.3.2 Oracle Model

The oracle layer is introduced to offload the storage and ensure data availability. The network of oracle layer consists of N oracle nodes, which can interact with clients to provide data availability service. There exists an adversary that is able to corrupt up to βN oracle nodes. Any node if not corrupted is called honest.

The basic data unit for the oracle layer is a *block*. A data availability oracle comprises of the following primitives which are required for committing and retrieving a block B.

1. **Generate chunks**: When a client wants to commit a block B to the trusted blockchain, it runs ($\mathsf{generate\_commitment}(B, M)$) to generate a commitment c for the block B and a set of M chunks $c_1, ..c_M$ which the block can be reconstructed from.

2. **Disperse chunks**: There is a dispersal protocol $\mathsf{disperse}(B, (c_1, .., c_M), N)$ which can be run by the client and specifies which chunks need to be sent to which of the N oracle nodes.

3. **Oracle finalization**: The oracle nodes run a finalization protocol to finalize and accept certain blocks whose commitments are written into the trusted blockchain.

4. **Retrieve Data:** Clients can initiate a request $(\mathsf{retrieve}, c)$ for retrieving a set of chunks for any commitment c that has been accepted by the oracle.

5. **Decode Data**: There is a primitive $\mathsf{decode}(c, \{c_i\}_{i \in S})$ that any client can run to decode the block from the set of chunks $\{c_i\}_{i \in S}$ retrieved for the commitment. The decoder also returns a proof that the decoded block B is related to the commitment.

We characterize the security of the oracle model and formally define data availability oracle as follows,

Definition 1 *A data availability oracle for a trusted blockchain accepts blocks from clients and writes commitments into the trusted blockchain with the following properties:*

1. ***Termination:*** *If an honest client initiates a* disperse *request for a block B, then block B will be eventually accepted and the commitment c will be written into the trusted blockchain.*

2. ***Availability*** *If a dispersal is accepted, whenever an honest client requests for retrieval of a commitment c, the oracle is able to deliver either a block B or a null block $\emptyset$ and prove its relationship to the commitment c.*

3. ***Correctness:*** *If two honest clients on running* $(\mathsf{retrieve}, c)$ *receives B_1 and B_2, then $B_1 = B_2$. If the client that initiated the dispersal was honest, we require furthermore that $B_1 = B$, the original dispersed block.*

A naive oracle satisfying all above expectations is trivial to construct, e.g., sending all oracle nodes a full copy of data. However, what we want is a **scalable** data availability oracle. To better understand this motivation, in the next section, we will introduce some metrics to concretize the oracle properties.

2.4 Technical Description of ACeD

In this section, we describe the four components of ACeD: CIT, dispersal protocol, retrieval protocol and block peeling decoder.

2.4.1 Coded Interleaving Tree

The first building block of ACeD is a coded commitment generator which takes a block proposed by a client as an input and creates three outputs: a commitment, a sequence of *coded symbols*, and their proof of membership POM – see Figure 2.2. The commitment is the root of a coded interleaving tree (CIT), the coded symbols are the leaves of CIT in the base layer, and POM for a symbol includes the Merkle proof (all siblings' hashes from each layer) and a set of *parity symbols* from all intermediate layers. A brief overview to erasure coding can be found in Appendix 2.15.

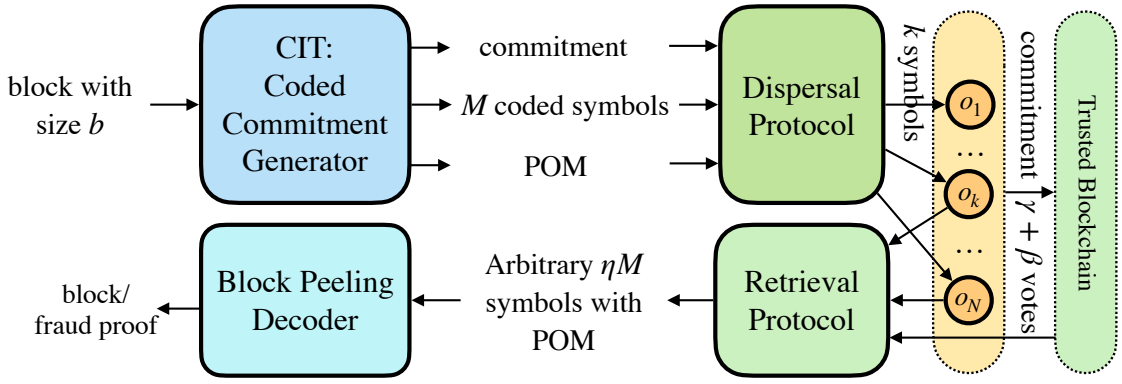

Figure 2.2: The pipeline for a block to be committed in trusted blockchain.

The construction process of an example CIT is illustrated in figure 2.3. Suppose a block has size b, and its CIT has ℓ layers. The first step to construct the CIT is to divide the block evenly into small chunks, each is called a *systematic symbol*. The size of a systematic symbol is denoted as c, so there are $s_\ell = b/c$ systematic symbols. And we apply erasure codes with coding ratio $r \leq 1$ to generate $m_\ell = s_\ell/r$ coded symbols in base layer. Then by aggregating the hashes of every q coded symbols we get m_ℓ/q systematic symbols for its parent layer (layer $\ell - 1$), which can be further encoded to $m_{\ell-1} = m_\ell/(qr)$ coded symbols. We aggregate and code the symbols iteratively until the number of symbols in a layer decays to t, which is the size of the root.

For all layers j except for root, $1 \leq j \leq \ell$, denote the set of all m_j coded symbols as M_j, which contains two disjoint subsets of symbols: systematic symbols S_j and parity symbols P_j. The number of systematic symbols is $s_j = rm_j$. Specifically, we set $S_j = [0, rm_j)$ and $P_j = [rm_j, m_j)$. Given a block of s_ℓ systematic symbols in the base layer, the aggregation rule for the k-th systematic symbol in layer $j-1$ is defined as follows:

$$Q_{j-1}[k] = \{\mathsf{H}(M_j[x]) \mid x \in [0, M_j), k = x \bmod rm_{j-1}\} \tag{2.1}$$

$$M_{j-1}[k] = \mathsf{H}(\mathsf{concat}(Q_{j-1}[k])) \tag{2.2}$$

where $1 \leq j \leq \ell$ and H is a hash function. $Q[k]$ is the tuple of hashes that will be used

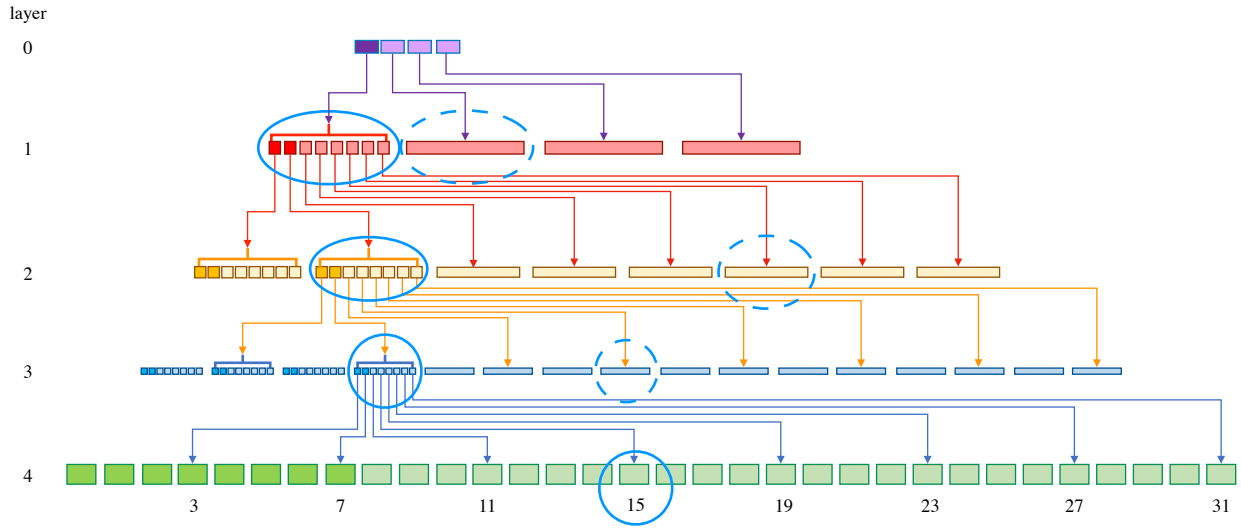

Figure 2.3: (1) CIT construction process of a block with $s_\ell = 8$ systematic symbols, applied with erasure codes of coding ratio $r = \frac{1}{4}$. The batch size $q = 8$ and the number of hashes in root is $t = 4$. (2) Circled symbols constitute the 15th base layer coded symbol and its POM . The solidly circled symbols are the base layer coded symbol and its Merkle proof (intermediate systematic symbols), the symbols circled in dash are parity symbols sampled deterministicly.

to generate k-th symbol in the parent layer and concat represents the string concatenation function which will concatenate all elements in an input tuple.

Generating a POM for a base layer symbol can be considered as a layer by layer sampling process as captured by the following functions:

$$f_\ell : [m_\ell] \to \binom{m_{\ell-1}}{2}, \cdots, f_2 : [m_\ell] \to \binom{m_1}{2};$$

Each function maps a base layer symbol to two symbols of the specified layer: one is a systematic symbol and the other is a parity symbol. We denote the two symbols with a tuple of functions $f_j(i) = (p_j(i), e_j(i))$, where $p_j(i)$ is the sampled systematic symbol and $e_j(i)$ is the sampled parity symbol, each is defined as follows:

$$p_j(i) = i \bmod rm_{j-1}; \;\; e_j(i) = rm_{j-1} + (i \bmod (1-r)m_{j-1}) \tag{2.3}$$

where $p_j : [m_\ell] \to [0, rm_{j-1})$ and $e_j : [m_\ell] \to [rm_{j-1}, m_{j-1})$. Note that the sampled non-base layer systematic symbols are automatically the Merkle proof for both systematic and parity symbols in the lower layer.

There are two important properties of the sampling function. (1) It guarantees that if at least $\eta \leq 1$ ratio of distinct base layer symbols along with their POM are sampled, then in every intermediate layer, at least η ratio of distinct symbols are already picked out by the collected POM . It ensures the reconstruction of each layer of CIT(Lemma 2, *reconstruction* property). (2) All sampled symbols at each layer have a common parent (Lemma 3, *sibling* property), which ensures the space efficiency of sampling.

As described above, CIT can be represented by parameters $\mathcal{T} = (c, t, r, \alpha, q, d)$. All parameters have been defined except α, which is the *undecodable ratio* of an erasure code and d, which is the number of symbols in a failed *parity equation*. Both of them will be discussed in the decoder section (Section 2.4.3).

Comparison with CMT (1) CIT is designed for the Push model (the client sends different chunks to different nodes) whereas CMT is designed for the Pull model (the nodes decide which chunks to sample from the client). (2)The CIT provides an ideal property (Lemma 2) that ensures reconstruction from any set of nodes whose size is greater than a given threshold. Thus as long as enough signatures are collected and we can be assured that the honest subset represented therein is larger than this threshold, we are guaranteed reconstruction in CIT. This is not the case in CMT, whose guarantees are probabilistic since each node samples data in a probabilistic manner. (3) On the technical side, CIT requires a new interleaving layer as well as a dispersal mechanism which decides how to assign the different symbols from intermediate layers to different groups. (4) Furthermore, CMT requires an assumption of an anonymous network whereas CIT does not require an anonymous network.

2.4.2 Dispersal Protocol

Given the commitment, coded symbols and POM generated by CIT, the dispersal protocol is used to decide the chunks all oracle nodes need to store. Consider a simple model where there are $M = b/(cr)$ chunks to be distributed to N nodes (each node may receive k chunks) such that any γ fraction of nodes together contains η fraction of chunks. The efficiency of the dispersal protocol is given by $\lambda = M/(Nk)$. We define the dispersal algorithm formally as follows,

Definition 2 *The chunk dispersal algorithm is said to achieve parameters set* $(M, N, \gamma, \eta, \lambda)$ *if there is a collection of sets* $\mathcal{C} = \{A_1, ..., A_N\}$ *such that* $A_i \subseteq [M]$, $|A_i| = \frac{M}{N\lambda}$. *Also, for any* $S \subseteq [N]$ *with* $|S| = \gamma N$, *it holds that* $|\bigcup_{i \in S} A_i| \geq \eta M$.

To simplify the 5 dimensional region, we consider the tradeoff on the three quantities: a certain triple (γ, η, λ) is said to be achievable if for any N, there exists M such that the chunk dispersal algorithm can achieve parameters set $(M, N, \gamma, \eta, \lambda)$. In the ACeD model, γ is constrained by security threshold β since the clients can only expect to collect chunks from at most $(1 - \beta)N$ nodes. η is constrained by the undecodable ratio α of the erasure code since the total chunks a client collects should enable the reconstruction. So for a given erasure code, there is a trade-off between dispersal efficiency and security threshold.

Our main result is the demonstration of a dispersal protocol with near optimal parameters (Lemma 1).

Lemma 1 *If* $\frac{\gamma}{\lambda} < \eta$, *then* (γ, η, λ) *is not feasible. If* $\frac{\gamma}{\lambda} > \log(\frac{1}{1-\eta})$ *then* (γ, η, λ) *is feasible and there exists a chunk dispersal protocol with these parameters.*

We provide a sketch of proof. If $\frac{\gamma}{\lambda} < \eta$, the maximum number of distinct symbols is $\frac{M}{N\lambda} \cdot \gamma M = \frac{M\gamma}{\lambda} < \eta M$, so (γ, η, λ) is not feasible.

If $\frac{\gamma}{\lambda} > \log(\frac{1}{1-\eta})$, we prove that the probability of $\mathcal{C}$ is not a valid code can vanish to as small as we want. The problem is transformed to the upper bound of $P(Y < \eta M)$ use a standard

inequality from the method of types [59], where Y is the number of distinct chunks sampled. By sampling $\frac{\gamma}{\lambda}M$ chunks from a set of M chunks randomly, we have $P(Y < \eta M) \leq e^{-f(\cdot)\cdot M}$, where $f(\cdot)$ is a positive function. So the probability of $\mathcal{C}$ is a valid code can be positive, which proves the existence of a *deterministic* chunk distribution algorithm. And we can control the probability of $\mathcal{C}$ is not a valid code to be vanishingly small. The detailed proof of the result can be found in Appendix §2.9.1.

In the dispersal phase, all oracle nodes wait for a data commitment c, k assigned chunks and the corresponding POM . The dispersal is accepted if $\gamma + \beta$ fraction of nodes vote that they receive the valid data.

2.4.3 Retrieval Protocol and Block Decoding

When a client wants to retrieve the stored information, the retrieval protocol will ask the oracle layer for data chunks. Actually, given erasure codes with undecodable ratio α, an arbitrary subset of codes with the size of over ratio $1 - \alpha$ is sufficient to reconstruct the whole block. When enough responses are collected, a hash-aware peeling decoder introduced in [155] will be used to reconstruct the block. The decoding starts from the root of CIT to the leaf layer and for each layer, it keeps checking all degree-0 parity equations and then finding a degree-1 parity equation to solve in turn. Eventually, either all symbols are decoded or there exists an invalid parity equation. In the second case, a logarithmic size incorrect-coding proof is prepared, which contains the inconsistent hash, d coded symbols in the parity equation and their Merkle proofs. After an agreement is reached on the oracle layer, the logarithmic size proof is stored in the trusted blockchain to permanently record the invalid block. Oracle nodes then remove all invalid symbols to provide space for new symbols. In the special case when the erasure code used by the CIT requires more than $(1 - \alpha)$ ratio of symbols to decode the original block, oracle nodes need to invoke a bad code handling protocol to reset a better code. We leave details in the Appendix §2.16.

2.4.4 Protocol Summary

In summary, an ACeD system with N oracle nodes and block size b using CIT $\mathcal{T}$ and dispersal protocol $\mathcal{D}$ can be represented by parameters $(b, N, \mathcal{T}, \mathcal{D})$, where $\mathcal{T} = (c, t, r, \alpha, q, d)$ and $\mathcal{D} = (\gamma, \eta, \lambda)$. The pipeline to commit a block to the trusted blockchain is as follows (see Figure 2.2).

- A client proposes a block of size b, it first generates a CIT with base layer symbol size c, number of hashes in root t, coding ratio r and batch size q. There are $M = b/(cr)$ coded symbols in the base layer. And then it disperses M coded symbols, their POM and the root of CIT to N oracle nodes using the dispersal protocol $\mathcal{D} = (\gamma, \eta, \lambda)$.

- Oracle nodes receive the dispersal request, they accept chunks and commitment, verify the consistency of data, POM and root, vote their results. A block is successfully committed if there are at least $\beta + \gamma$ votes. Upon receiving retrieval requests, oracle nodes send the stored data to the requester. Upon receiving the fraud proof of a block, oracle nodes delete the stored data for that block.

- Other clients send retrieval requests to the oracle nodes on demand. Upon receiving at least $\eta \geq 1 - \alpha$ fraction of chunks from at least γ oracle nodes, they reconstruct the block, if a coding error happens, the fraud proof will be sent to the trusted blockchain.

2.5 Performance Guarantees of ACeD

Theorem 1 *Given an adversarial fraction $\beta < \frac{1}{2}$ for an oracle layer of N nodes, ACeD is a data availability oracle for a trusted blockchain with $O(b)$ communication complexity, $O(1)$ storage and download overhead in the normal case, and $O(\log b)$ storage and download overhead in the worst case.*

This result follows as a special case of a more general result below (Theorem 2).

Proof 1 *Suppose χ is an ACeD data availability oracle with parameters $(b, N, \mathcal{T}, \mathcal{D})$ where $\mathcal{T} = (c, t, r, \alpha, q, d)$ and $\mathcal{D} = (\gamma, \eta, \lambda)$. There are at most $\beta < \frac{1}{2}$ fraction of adversarial nodes in the oracle layer. Then by setting $r, q, d, t = O(1), c = O(\log b), b \gg N$, χ is secure as long as $\beta \leq \frac{1}{2}(1 - \lambda \log(\frac{1}{\alpha}))$; the communication complexity of χ is $O(b)$ because*

$$Nyt + \frac{b}{\lambda r} + \frac{(2q-1)by}{cr\lambda} \log_{qr} \frac{b}{ctr} = O(N) + O(b) + O(b) = O(b)$$

the storage and download overhead in the normal case is $O(1)$, because

$$\frac{Nyt}{b} + \frac{1}{\lambda r} + \frac{(2q-1)y}{cr\lambda} \log_{qr} \frac{b}{ctr} = O(1) + O(1) + O(\frac{1}{\log b} \log(\frac{b}{\log b})) = O(1)$$

the storage and download overhead in the worst case is $O(\log b)$, because

$$\frac{c(d-1)}{y} + d(q-1) \log_{qr} \frac{b}{ctr} = O(\log b) + O(\log_{qr}(\frac{b}{\log b}) = O(\log b)).$$

A complete description of the security and performance guarantees of ACeD is below.

Theorem 2 *ACeD is a data availability oracle for the trusted blockchain tolerating at most $\beta \leq 1/2$ fraction of adversarial oracle nodes in an oracle layer of N nodes. The ACeD is characterized by the system parameters $(b, N, \mathcal{T}, \mathcal{D})$, where $\mathcal{T} = (c, t, r, \alpha, q, d)$ and $\mathcal{D} = (\gamma, \eta, \lambda)$. y is a constant size of a hash digest, then*

1. *ACeD is secure under the conditions that*

$$\beta \leq \frac{1-\gamma}{2};\ \frac{\gamma}{\lambda} > \log(\frac{1}{1-\eta});\ \eta \geq 1 - \alpha$$

2. *Communication complexity is*

$$Nyt + \frac{b}{\lambda r} + \frac{(2q-1)by}{cr\lambda} \log_{qr} \frac{b}{ctr}$$

3. *In normal case, both the storage and download overhead are*

$$\frac{Nyt}{b} + \frac{1}{\lambda r} + \frac{(2q-1)y}{cr\lambda} \log_{qr} \frac{b}{ctr}$$

4. *In worst case, both storage and download overhead are*

$$\frac{c(d-1)}{y} + d(q-1) \log_{qr} \frac{b}{ctr}$$

Proof 2 *We prove the security and efficiency guarantees separately.*

2.5.1 Security

To prove that ACeD is secure as long as the trusted blockchain is persistent and

$$1 - 2\beta \geq \gamma;\ \frac{\gamma}{\lambda} > \log(\frac{1}{1-\eta});\ \eta \geq 1 - \alpha$$

, we prove the following properties as per Definition 1.

- ***Termination.*** *In ACeD, a dispersal is accepted only if there is a valid commitment submitted to the trusted blockchain. Suppose an honest client requests for dispersal but the commitment is not written into the trusted blockchain, then either the commitment is not submitted or the trusted blockchain is not accepting new transactions. Since $1 - 2\beta \geq \gamma$, thus $\beta + \gamma \leq 1 - \beta$, even if all corrupted nodes remain silent, there are still enough oracle nodes to vote that the data is available and the commitment will be submitted, hence the trusted blockchain is not live, which contradicts our assumption.*

- ***Availability.*** *If a dispersal is accepted, the commitment is on the trusted blockchain and $\beta + \gamma$ oracle nodes have voted for the block. Since the trusted blockchain is persistent, whenever a client wants to retrieve the block, it can get the commitment and at least γ nodes will respond with the stored chunks. On receiving chunks from γ fraction of nodes, for a CIT applying an erasure code with undecodable ratio α and a feasible dispersal algorithm (γ, η, λ)(Lemma 1), because $\eta \geq 1 - \alpha$, the base layer is reconstructable. Then we prove the following lemma ensures the reconstruction of all intermediate layers.*

  Lemma 2 *(Reconstruction) For any subset of base layer symbols W_ℓ, denote $W_j := \bigcup_{i \in W_\ell} f_j(i)$ as the set of symbols contained in* POM *of all symbols in W_ℓ. If $|W_\ell| \geq \eta m_\ell$, then $\forall j \in [1, \ell]$, $|W_j| \geq \eta m_j$.*

  The proof of Lemma 2 utilizes the property when generating POM *given base layer symbols. (See details in Appendix §2.9.2). Thus the entire tree is eventually reconstructed*

and the oracle can deliver a block B, and the proof for B's relationship to commitment c is the Merkle proof in CIT. If a client detects a failed parity equation and outputs a null block $\emptyset$, it will generate an incorrect-coding proof.

- ***Correctness.*** *Suppose for a given commitment c, two honest clients reconstruct two different blocks B_1 and B_2, the original dispersed block is B.*

  (1) If the client that initiated the dispersal was honest, according to the availability property, $B_1, B_2 \neq \emptyset$, both clients can reconstruct the entire CIT. If $B_1 \neq B_2$, the commitment $c_1 \neq c_2$, which contradicts our assumption that the trusted blockchain is persistent.

  (2) If the client that initiated the dispersal was adversary and one of the reconstructed blocks is empty, w.l.o.g suppose $B_1 = \emptyset$, the client can generate a fraud proof for the block. If $B_2 \neq \emptyset$, the entire CIT is reconstructed whose root is commitment c_2. Since there is no failed equation in the CIT of B_2, $c_1 \neq c_2$, which contradict our assumption that the trusted blockchain is persistent.

  (3) If the client that initiated the dispersal was adversary and $B_1, B_2 \neq \emptyset$, both clients can reconstruct the entire CIT. If $B_1 \neq B_2$, the commitment $c_1 \neq c_2$, which contradict our assumption that the trusted blockchain is persistent.

  Thus we have $B_1 = B_2$, and if the client that initiated the dispersal is honest, $B_1 = B$.

2.5.2 Efficiency

Prior to computing the storage and communication cost for a single node to participate dispersal, we first claim a crucial lemma:

Lemma 3 *For any functions $p_j(i)$ and $e_j(i)$ defined in equation 2.3, where $1 \leq j \leq \ell$, $0 \leq i < m_\ell$, $p_j(i)$ and $e_j(i)$ are siblings.*

Lemma 3 indicates that in each layer, there are exactly two symbols included in the POM *for a base layer symbol and no extra proof is needed since they are siblings (see proof details in §2.9.3). For any block B, oracle nodes need to store two parts of data, the hash commitment, which consists of t hashes in the CIT root, and k dispersal units where each unit contains one base layer symbol and two symbols per intermediate layer. Denote the total storage cost as X, we have*

$$X = ty + kc + k[y(q-1) + yq] \log_{qr} \frac{b}{ctr} \tag{2.4}$$

where y is the size of hash, b is the size of block, q is batch size, r is coding rate, and c is the size of a base layer symbol. Notice that $k = \frac{b}{Nrc\lambda}$, we have

$$X = ty + \frac{b}{Nr\lambda} + \frac{(2q-1)by}{Nrc\lambda} \log_{qr} \frac{b}{ctr}. \tag{2.5}$$

It follows that the communication complexity is NX. In the normal case, each node only stores X bits hence the storage overhead becomes $\frac{NX}{b}$, and similarly when a client downloads data from N nodes, its overhead is $\frac{NX}{b}$. In the worst case, we use incorrect-coding proof to notify all oracle nodes. The proof for a failed parity equation which contains d coded symbols consist of $d-1$ symbols and their Merkle proofs, denote the size as P, we have

$$P = (d-1)c + dy(q-1) \log_{qr} \frac{b}{ctr}. \tag{2.6}$$

The storage and download overhead in this case is $\frac{P}{y}$, the ratio of the proof size and the size of reconstructed data ,a single hash y.

2.6 Algorithm to System Design and Implementation

ACeD clients are nodes associated with a number of side blockchains; the honest clients rely on ACeD and the trusted blockchain to provide an ordering service of their ledger (regardless of any adversarial fraction among the peers in the side blockchain). A client proposes a new block to all peers in the side blockchain by running ACeD protocol. An honest client confirms to append the new block to the local side blockchain once the block hash is committed in

the trusted blockchain and the full block is downloaded. As long as there is a *single honest* client in the side blockchain, we have the following claim:

Claim 1 *Once a block is confirmed by an honest client, security is guaranteed as long as the trusted blockchain is safe, even if the oracle layer is dishonest majority. Liveness is guaranteed when the trusted blockchain is live and the majority of the oracle layer is honest.*

The claim indicates that in side blockchains, the safety of a confirmed block only relies on the trusted blockchain because the commitment on it is irrefutable once the trusted blockchain is safe, and the honest client has already downloaded the full block. So even if the oracle layer is occupied by the dishonest majority, those confirmed blocks are still safe. However, the liveness relies on the liveness of both ACeD and the trusted blockchain. As for the side blockchain network, because data persistence is guaranteed by ACeD, any client inside the network can safely conduct a transaction and reconstruct the ledger without worrying about a dishonest majority; similarly a valid transaction will eventually join the ledger as long as there is a single honest client who can include the transaction into a block.

Next we discuss the practical parameter settings for ACeD. We use these parameter choices to design and implement an ACeD oracle layer that interfaces with Ethereum as the trusted blockchain.

Parameter Specifications. We study an ACeD system with $N = 9000$ oracle nodes with adversarial fraction $\beta = 0.49$; the block size b is 12 MB and therefore $b \gg N$. In each block, the base layer symbol size c is $2000 \log b \approx 48$ kB, which corresponds to $\frac{b}{c} \approx 256$ uncoded symbols in the base layer. Within the setup, we construct a CIT of five layers with parameters: number of root symbols $t = 16$, hash size $y = 32$ bytes, coding ratio $r = 0.25$, aggregation batch size $q = 8$ and 4 erasure codes of size (256,128,64,32) for each non-root layer. For selecting erasure code properties, we use codes with undecodable ratio $\alpha = 0.125$, maximal parity equation size $d = 8$. In the dispersal protocol, we use $\eta = 0.875 = 1 - \alpha$, which translates to a dispersal efficiency $\lambda \leq \frac{1-2\beta}{\log \frac{1}{1-\eta}} = 1/150$, and therefore each oracle node needs to store roughly 17 symbols. With ACeD characterized by those parameters, the total

communication cost for a client to commit a 12 MB block is roughly 5.38 GB; this represents a $0.5N$ factor boost over storing just one block. In the normal path, after accepting the 12 MB block, each oracle node only has to store 448 kB of data, a 3.7% factor of the original data; if there is an incorrect-coding attack, oracle layer will together upload data of size 339 kB incorrect-coding proof to the trusted blockchain. To download a block in the normal path, a client can use a naive method of collecting all symbols from all oracle nodes. Despite the conservative approach at block reconstruction, the entire download takes 5.38 GB. A simple optimization involving selectively querying the missing symbols can save significant download bandwidth: a client only needs 896 coded symbols in the base layer to reconstruct the block; thus, in the best case only 42 MB is needed to directly receive those symbols. When there is an incorrect-coding attack, a new user only needs to download the fraud proof which has been generated by other nodes (either client or oracle nodes); the proof is only of the size of 339 kB. Table 2.3 tabulates the performance metrics of ACeD (and baseline schemes) with these parameter settings.

In the uncoded repetition protocol, each node stores the identical block, so the storage cost for both cases is 12MB. In the uncoded dispersal protocol, because all oracle nodes are honest, the block is equally divided into N nodes in both cases. In the AVID protocol, each message has the size $\frac{b}{N-2N\beta} + y(\log_2 N + 1) = 4.37$ kB, so the total amount of messages exchanged during the broadcasting stage equals to 354GB; AVID does not commit an invalid block, so only the block hash is required to show the block is invalid. When $\beta = 0.33$ 1D-RS uses the same erasure code as AVID to assign each node a symbol, so their normal case is identical; when there is an incorrect-coding attack, a node needs to download the whole block to reconstruct the block, therefore in the worst case both storage and download are 13.4MB. To tolerate $\beta = 0.49$, 1D-RS needs to decrease the coding ratio to 0.02 while increasing the symbol size, because 200 nodes are required to start the decoding. A lower coding ratio requires nodes to store more 67.1 kB data; the download cost are saved by reducing the number of Merkle proofs, because only 200 symbols are needed for decoding. Finally, we use the same method to evaluate ACeD with $\beta = 0.33$, both normal case storage and

Table 2.3: Performance metrics under a specific system parameterization.

	maximal adversary fraction	normal case		worst case		communication complexity
		storage cost*	download cost*†	storage cost*	download cost*	
uncoded (repetition)	0.49	12MB	12MB	12MB	12MB	108GB
uncoded (dispersal)	0	1.3kB	12MB	1.2kB	12MB	12MB
AVID [49]	0.33	4.37kB	13.4MB	32B	32B	354GB
1D-RS	0.33	4.37kB	13.4MB	13.4MB	13.4MB	39.4MB
1D-RS	0.49	67.1kB	12.1MB	12.1MB	12.1MB	604MB
2D-RS	0.33	5.4kB	16.6MB	232.1KB	232.1KB	48.9MB
2D-RS	0.49	72.1kB	13MB	925.6KB	925.6KB	648.6MB
ACeD	0.33	50.3kB	42MB	339kB	339kB	452 MB
ACeD	0.49	597kB	42MB	339kB	339kB	5.38GB

* cost is derived by $\frac{b}{N} \cdot$ overhead

† best case

communication complexity improves due to a better dispersal efficiency. In evaluating the metrics for 2D-RS, we first get a lower bound for the number of honest oracle nodes available in the decoding phase, which is used to compute the size of a matrix used in 2D-RS, and the goal is to decide a 2D-RS symbol size which guarantees each honest oracle node with at least one symbol. The 2D-RS looks over-perform ACeD when the adversarial ratio is 0.33. But if the block size increases from 12MB to 1.2GB while the ratio is 0.33, the worst case in ACeD needs 279kB, whereas 2D-RS becomes 21.8MB; when the adversary ratio is 0.49, ACeD needs 279 KB for a 1.2GB block, and 2D-RS requires a fault proof of 92.3MB. The downside of ACed is the communication complexity when the block size is large, a 1.2GB block requires each node communicating 64MB data, whereas 2D-RS only requires 6.6MB communication.

The ACeD oracle layer interacts with the side blockchains and the trusted blockchain. For a high performance (high throughput, low gas), the oracle layer and its interaction protocols have to be carefully designed for software implementation. In this section, we discuss our system design decisions that guided the implementation.

Architecture. We use Ethereum as the trusted blockchain. The oracle nodes interact with Ethereum via a special (ACeD) smart contract; the contract is owned by a group of permissioned oracle nodes, who individually cannot update the contract, except that a majority of the oracle nodes together agree and perform a single action on the smart contract. The contract only changes its state after accepting a multisignature of majority votes [41]. We implement both the oracle nodes and side blockchain nodes in RUST; the architecture is depicted in Figure 2.4. There are four important modules:

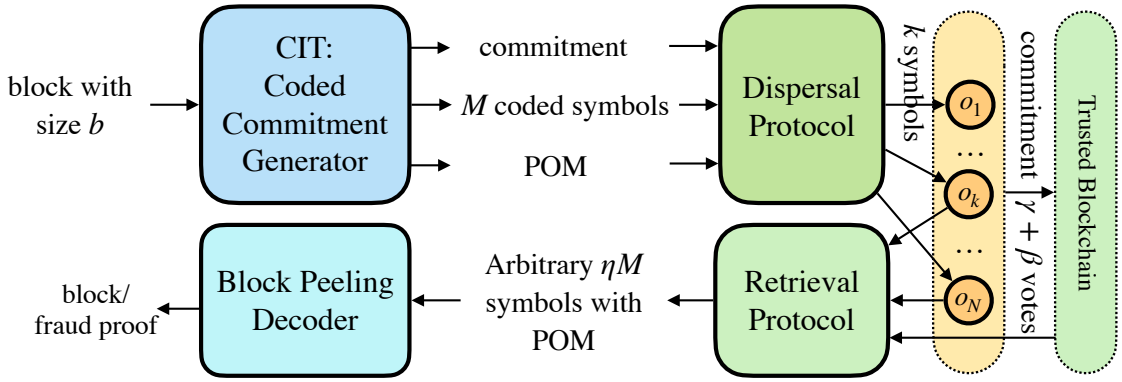

Figure 2.4: The left figure depicts a block diagram of a side node; the right figure depicts an oracle node. The sharp rectangle represents an active thread; round rectangles are system states.

(1) **Smart contract manager**, when used by a side blockchain node, is responsible for fetching states from the trusted blockchain; when used by an oracle node, it writes state to the trusted blockchain after the block manager has collected sufficient signatures. (2) **Block manager**, when used by a side blockchain node, creates a block (triggered by a scheduler), and distributes the coded symbols to oracle nodes; when used by an oracle

node and has received messages from side blockchain nodes, the module stores the received symbols and provides them on demand; When used by an oracle committee node, the module collects signatures and send an Ethereum transaction once there are sufficient signatures. (3) **Scheduler**, when used by a side blockchain node, decides when a node can transmit its block (and coded symbols) to oracle nodes; when used by an oracle node, the module decides who the correct block proposer is. (4) **Trusted Chain manager**, when used by a side node, periodically pulls state changes from the trusted blockchain; if a new block is updated, it collects block symbols from all oracle nodes; the module also maintains a blockchain data structure.

A detailed discussion of the design and implementation optimizations of these four blocks is provided in Appendix §2.13, which also discusses the data structures that maintain the state of ACeD.

Implementation Optimizations. The key challenge to a high performance implementation of ACeD is the large number of interactions among the various system blocks. Since the trusted blockchain is immutable, we optimize the oracle and side blockchain client implementations by utilizing parallelism and pipelining as much as possible. (1) **LDPC parallelization**2.15: we bring state of the art techniques from wireless communication design and encoding procedures (Appendix A of [131]) to our implementation of encoding a block by distributing the workload to many worker threads; we find that the encoding time is almost quartered using 4 threads. A detailed description is referred to Appendix §2.11. (2) **Block Pipelining**. We pipeline signature aggregation, block dispersal and retrieval so that multiple blocks can be handled at the same time. We also pipelined the trusted chain manager so that a side node can simultaneously request multiple blocks from oracle nodes. The local state variable, (block id, hash digest), is validated against the smart contract state whenever a new block is reconstructed locally.

2.7 Evaluation

We aim to answer the following questions through our evaluation. (a) What is the highest throughput (block confirmation rate) that ACeD can offer to the side blockchains in a practical integration with Ethereum? (b) What latency overhead does ACeD pose to block confirmation? (c) How much is the gas cost reduced, given the computation (smart contract execution) is not conducted on the trusted blockchain? (d) What are the bottleneck elements to performance?

Testbed. We deploy our implementation of ACeD on Vultr Cloud hardware of 6 core CPU, 16GB memory, 320GB SSD and a 40Gpbs network interface (for both oracle and side blockchain nodes). To simplify the oracle layer, we consider all of oracle nodes are committee members, which is reflected in the experiment topology: all the oracle nodes are connected as a complete graph, and each side blockchain node has TCP connections to each of the oracle nodes. We deploy a smart contract written in Solidity using the Truffle development tool on a popular Ethereum testnet: Kovan. Oracle nodes write their smart contracts using Ethereum accounts created with MetaMask, each loaded with sufficient Keth to pay the gas fee of each Ethereum transaction. We use an LDPC code of $\frac{b}{c} = 128$ input symbols with a coding rate of $r = 0.25$ to construct CIT with $q = 8, d = 8, \alpha = 0.125, \eta = 0.875, t = 16$. We fix the transaction length to be 316 bytes, which is sufficient for many Ethereum transactions.

Experiment Settings. We consider four separate experiments with varying settings of four key parameters: the number of oracle nodes, the number of side blockchain nodes, block generation rate, and block size. The experiment results are in Figure 2.5.

(1) **Throughput**. We measure the rate of state update in the trusted blockchain as viewed from each oracle blockchain nodes; the throughput is then the rate averaged across time and across oracle blockchain nodes. The throughput performance changes with four parameters: In experiments A and B, the throughput is not noticeably impacted as the number of oracles or side blockchain nodes increases. In experiment C, the block size has roughly a linear effect on throughput, which continues until coding the block is too onerous (either it costs

	# side blockchain nodes	# oracle nodes	Block size(MB)	Block generation rate(sec/blk)
A	5	5,10,15,25	4	5
B	5,10,20	10	4	5
C	5	10	4,8,16	5
D	3,5,8,10	10	4	8.33,5,3.125,2.5

Table 2.4: Four different experiments varying the parameters of ACeD.

too much memory or takes too long). In experiment D, we fix the product of the block generating rate and the number of side blockchain node to constant, while increasing the block generation rate, the throughput increases linearly; the performance will hit a bottleneck when the physical hardware cannot encode a block in a time smaller than the round time.
(2) The **latency** of ACeD is composed of three major categories: block encoding time, oracle vote collection time and time to update the trusted blockchain. We find latency stays relatively constant in all experiments (the exception is experiment C where block encoding is onerous).
(3) **Gas saving**. ACeD transactions cost significantly less gas than a regular Ethereum transaction, independent of the computational complexity of the transaction itself. The cost of an ACeD transaction voted by 10 oracles nodes on a 4MB block costs on average 570K gas. Based on a historical analysis of 977 Crytokitties transactions, we estimate a 220 byte transaction costing roughly 180,000 gas: ACeD gas savings are thus over a factor 6000. We emphasize that the saving in gas is independent of block generation rate or the block size, since the smart contract only needs the block header and a list of the oracle signatures.

2.8 Conclusion and Discussion

Interoperability across blockchains is a major research area. Our work can be viewed as a mechanism to transfer the trust of an established blockchain (eg: Ethereum) to many small (side) blockchains. Our proposal, ACeD, achieves this by building an oracle that guarantees

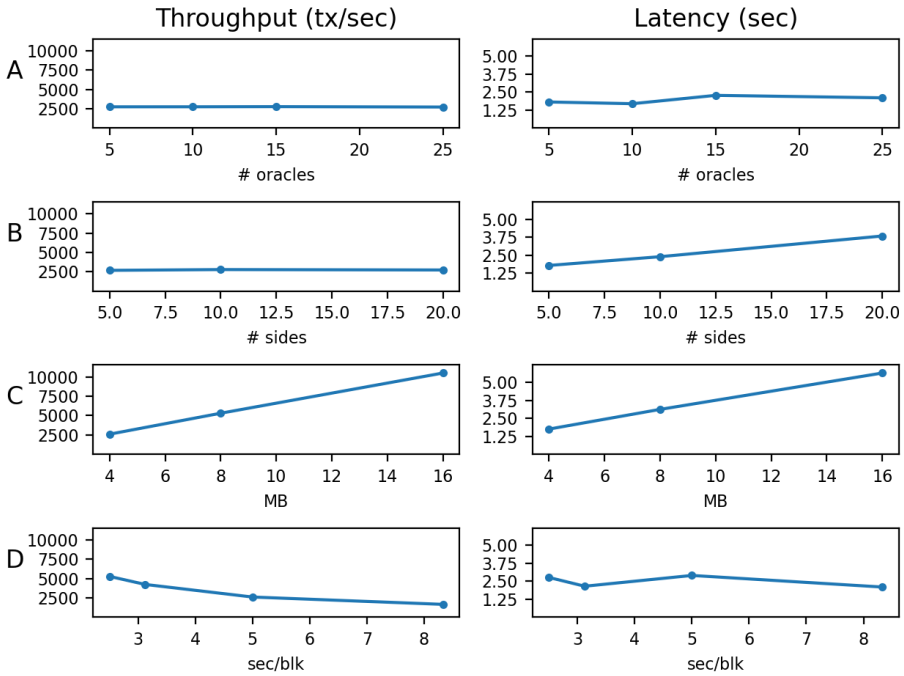

Figure 2.5: Throughput (left) and Latency (right) for the 4 experiments A, B, C, D.

data availability. The oracle layer enabled by ACeD is run by a group of permissioned nodes which are envisioned to provide an *interoperability service* across blockchains; we have provided a detailed specification of ACeD, designed incentives to support our requirement of an honest majority among the oracle nodes, and designed and evaluated a high performance implementation.

ACeD solves the data availability problem with near optimal performance on all metrics. However, when instantiating the system with specific parameters, we observe that some metrics have to be computed with constant factors which give rise to acceptable but non-negligible performance loss. The critical bottleneck for ACeD is the lower undecodable ratio (the fraction of symbols in the minimum stopping set) compared to 1D-RS coding; this undermines dispersal efficiency and increases the communication cost under the existence of strong adversary. Therefore, finding a *deterministic* LDPC code with a higher undecodable ratio will immediately benefit the practical performance of ACeD; the construction of such

LDPC codes is an exciting direction of future work and is of independent interest.

2.9 Proof of Lemmas and Theorems

2.9.1 Proof of Lemma 1

We show the existence of (several) chunk dispersal algorithms using the probabilistic method [24]. Consider a randomized chunk dispersal design, where each element of each A_i is chosen i.i.d. uniformly randomly from the set of M possible chunks. This gives raise to a randomized code $\mathcal{C}$. We prove the following statement:

$$\mathbb{P}\{\mathcal{C} \text{ is NOT a } (M, N, \gamma, \eta, \lambda) \text{ code}\} \leq e^{-n} \tag{2.7}$$

Let $k = M/(N\lambda)$ be the number of chunks at a given node.

$$\begin{aligned}
\mathbb{P}\{\mathcal{C} \text{ is not a valid code}\} &= P(\exists S \text{ with } |S| = \gamma N : |\bigcup_{i \in S} A_i| \leq \eta M) && (2.8)\\
&\leq \sum_{S \subseteq [M]:|S|=\gamma N} P(|\bigcup_{i \in S} A_i| \leq \eta M) && (2.9)\\
&\leq e^{NH_e(\gamma)} P(|\bigcup_{i \in S} A_i| \leq \eta M) && (2.10)
\end{aligned}$$

where we have used that $\binom{N}{\gamma N} \leq 2^{NH(\gamma)} = e^{NH_e(\gamma)}$, with $H(.)$ being the binary entropy function measured in bits (logarithm to the base 2) and H_e being the binary entropy measured in natural logarithm (nats), a standard inequality from the method of types [59].

The key question now is for a fixed S, we need to obtain an upper bound on $P(|\bigcup_{i \in S} A_i| \leq \eta M)$. We observe that under the random selection, we are choosing $\gamma/\lambda M$ chunks (sampled randomly with replacement). The mathematical question now becomes, if we choose ρM chunks randomly (with replacement) from M chunks, what is the probability that we get at least ηM distinct chunks.

Lemma 4 *We sample ρM chunks from a set of M chunks randomly with replacement. Let Y be the number of distinct chunks. Let $x = (1 - \eta)exp(\rho)$.*

$$\mathbb{P}(Y < \eta M) \leq e^{-(1-\eta)\frac{(x-1)^2}{x(x+1)}} \tag{2.11}$$

Proof 3 *Let $Z = M - Y$ be the number of chunks not sampled. Then $\mu := \mathbb{E}[Z] = (1 - \frac{1}{M})^{\rho M} \approx Me^{-\rho}$ (when M is large). We can write $Z = \sum_{i=1}^{M} Z_i$, where Z_i is the binary indicator random variable indicating that chunk i is not sampled. We note that a direct application of tail bounds for i.i.d. random variables does not work here. However, there are results analyzing this problem in the context of balls-in-bins and they show that the following tail bound does indeed hold (see Theorem 2 and Corollory 1 in [93]). We can write the tail bound on Z as follows:*

$$P(Z > \ell\mu) \leq e^{-\frac{(\ell-1)^2}{1+\ell}\mu} \quad \forall \ell > 1 \tag{2.12}$$

Let $\bar{\eta} = 1 - \eta$ and $\ell = e^{\rho}\bar{\eta}$, we get

$$P(Z > \bar{\eta}M) \leq e^{-\frac{(e^{\rho}\bar{\eta}-1)^2}{e^{\rho}(e^{\rho}\bar{\eta}+1)}M} \quad \text{if } e^{\rho}\bar{\eta} > 1 \tag{2.13}$$

Define the function, $f(\eta, \rho) = \frac{(e^{\rho}\bar{\eta}-1)^2}{e^{\rho}(e^{\rho}\bar{\eta}+1)}$ (we note that $f(\cdot)$ is a positive function), we have $P(Y < \eta M) \leq e^{-f(\eta,\rho)\cdot M}$ if $\rho > \log(\frac{1}{1-\eta})$.

Continuing from before, we get

$$\mathbb{P}\{\mathcal{C} \text{ is not a valid code}\} \;\; \leq \;\; e^{N \cdot H_e(\gamma) - M \cdot f(\eta,\rho)} \text{ if } \frac{\gamma}{\lambda} > \log\left(\frac{1}{1-\eta}\right) \tag{2.14}$$

If we fix a N and take M large enough, we can make the right hand side large enough to make the probability that $\mathcal{C}$ is not a valid code vanish to as small as we want. This result has two interpretations: (1) as long as the probability of being a valid code is strictly greater than zero, then it proves the existence of a *deterministic* chunk distribution algorithm, (2) if we can make the probability of being a invalid code arbitrarily close to zero, then we can use the randomized algorithm directly with a vanishing probability of error. Fixing a N and taking M large can make the RHS arbitrarily small and thus the stronger second interpretation can be used. This concludes the proof of the theorem.

2.9.2 Proof of Lemma 2

Proof 4 *By definition, $f_j(i)$ can be decomposed to two functions, $p_j(i)$ and $e_j(i)$. According to equation 2.3, $p_j(i)$ is computed by modulo, so there are at most $c = \frac{m_\ell}{rm_j} = \frac{m_j \cdot (qr)^{\ell-j}}{rm_j} = \frac{(qr)^{\ell-j}}{r}$ base symbols mapping to one symbol in layer j . In the worst case, γm_ℓ distinct base layer symbols map to $\frac{\gamma m_\ell}{c} = \gamma r m_j$ distinct symbols in layer j in the range $[0, rm_j]$. Similarly, there are $d = \frac{M_\ell}{(1-r)M_j} = \frac{M_j \cdot (qr)^{\ell-j}}{(1-r)m_j} = \frac{(qr)^{\ell-j}}{1-r}$ base layer symbols mapping to one symbol at layer j, and therefore in the worst case there are $\frac{\gamma m_\ell}{d} = \gamma(1-r)m_j$ distinct symbols in the range $[rm_j, m_j)$. Since two range do not overlap, in total there are γm_j distinct symbols for each layer. Hence it completes the proof.*

2.9.3 Proof of Lemma 3

Proof 5 *First we show for every $i \in [m_\ell]$ and $2 \leq j < \ell$, $p_j(i)$ and $e_j(i)$ are sibling. According to the definition,*

$$p_j(i) = i \bmod rm_{j-1} = i \bmod qr^{2m}_{j-2} \tag{2.15}$$

$$p_{j-1}(i) = i \bmod rm_{j-2} \quad \textit{(by definition)} \tag{2.16}$$

$$= p_j(i) \bmod rm_{j-2} \quad \textit{(discuss below)} \tag{2.17}$$

To prove equation 2.17, suppose $i = p_j(i) + xqr^2 m_{j-2}$ and $i = p_{j-1}(i) + yrm_{j-2}$ for some integer x, y. By rearranging terms, $p_j(i) - p_{j-1}(i) = rm_{j-2}(y - xqr)$. If $p_j(i) - p_{j-1}(i) \neq 0$, we mod rm_{j-2} on both sides, and therefore $p_{j-1}(i) = p_j(i) \bmod rm_{j-2}$. If $p_j(i) - p_{j-1}(i) = 0$, then obviously $p_{j-1}(i) = p_j(i) \bmod rm_{j-2}$. Therefore in any case, equation 5.3 holds, which means $p_{j-1}(i)$ is the parent of $p_j(i)$.

Then let's see another function $e_j(i)$, by definition we have

$$\begin{aligned} e_j(i) &= rm_{j-1} + (i \bmod (1-r)m_{j-1}) \\ &= rm_{j-2}qr + (i \bmod (1-r)qrm_{j-2}) \end{aligned}$$

To calculate its parent according to aggregation rules, we get similar result that $e_j(i) -$

$p_{j-1}(i) = rm_{j-2}(y - xq(1-r))$ *thus*

$$\begin{aligned} e_j(i) &= i \bmod (1-r)qrM_{j-2} \\ &= p_{j-1}(i) \bmod rm_{j-2} \end{aligned}$$

$p_{j-1}(i)$ *is also the parent of* $e_j(i)$. *Hence it completes the proof.*

2.10 Other Dispersal Protocol

To reconstruct blocks, a side node needs to collect enough coded symbols which are distributively stored among oracle nodes. There are several alternative dispersal protocols to decide which subset of data chunks a specific oracle node would receive. In Section §2.4 we introduce a scheme to automatically get coded symbols from intermediate layers given the set of symbols on the base layer and we claim the mechanism effectively save the message size while ensuring decodable property for each layer. For comparison, we discuss another two mechanisms to get data for decoding.

2.10.1 Layer-independent Dispersal

Recall that to decode a layer i of CIT with m_i coded symbols in total we need at least γm_i coded symbols to solve parity equations, i.e. an oracle node is supposed to store at least $t_i = \gamma m_i/(\theta N)$ symbols for layer i. The most straightforward way is to consider each layer independently, e.g. block proposer can designate the first t_i coded symbols on layer i to oracle node with id 1 and the second batch of size t_i to node 2 and so on. Given a reasonable partition mechanism for base layer, it can be applied to all the other intermediate layers independently. However, a POM is required to ensure membership for each symbol including $q-1$ sibling hashes from each upper layer. The size of POM for a oracle node to store is at most $\sum_{i=0}^{\log_{qr}\frac{k}{rt}-1} y(q-1)\log_{qr}(\frac{k}{rt}-i) = O(\log^2 b)$ where y is the size of hash (e.g., 32 bytes), k is the number of base layer symbols, r is coding ratio, $i = 0$ for base layer.

2.10.2 Sampling based Dispersal

Although Layer-independent sampling is succinct and easy to understand, it costs more bandwidth to downloading extra Merkle proofs for intermediate layer chunks. To optimally reduce the message size, the original CMT paper proposes a probabilistic sampling mechanism. The first main difference of this mechanism is that light nodes can determines what subset of data chunks they want to store by themselves. They send requests to block proposer once deciding and wait for response containing sampled symbols. When considering requested symbols, they divide the coded symbols from intermediate layers into two parts, the data symbols and parity symbols. Data symbols are automatically included in the POM of base layer symbols with no extra cost. But different from the deterministic method we use, CMT randomly samples about $(1-r)s_i$ parity symbols from layer i. Specifically, for every pair of parent and child intermediate layer, if a parent layer data symbol is sampled, then with probability $1-r$, one of its $q(1-r)$ child parity symbols from interleaved batch will be sampled uniformly at random. However, such self-determined sampling methods might bring security issues. For example, a selfish oracle node can choose to submit signature without sampling any coded symbols and borrow some symbols from other nodes once audited. We can not detect its behavior since the submitted symbols are valid and there's no evidence indicating that it didn't send sampling request.

2.11 Efficient Encoding for LDPC Codes

A block of transactions is coded to symbols before distributing them to the oracle nodes. We have implemented a CIT library in RUST; compared to CMT (coded Merkle tree) library, the major distinction is in the data interleaving algorithm. We find that the encoding time of CMT is very time consuming: on a Mac 2.6 GHz 6-Core Intel Core i7 computer, the library required about 34 seconds to encode a 4 MB block on a basis of 128 symbols with 0.25 coding ratio. The issue is due to a huge amount of XOR operations required for encoding the base layer. In our implementation, we have developed two strategies to overcome the problem

by adding parallelism to the encoding process and by devising a code transformation that reduces coding complexity. Parallelization involves two components: a light weight read-only hash map and a multiple-thread environment for partitioning the workload. An inverse map is created whose key is the index of input symbols, and value is a list of parity symbols to be XOR with according to the encoding matrix. We use a single-producer multiple-workers paradigm where the producer thread uses the inverse map to properly distribute the input symbol. Worker threads partition the parity set such that each thread is responsible for an equal number of parity symbols. The input symbol is communicated through a message queue, and once a worker receives the input symbol, it XORes the symbol locally. The producer distributes all input symbols and sends a final control message to all workers to collect the result. The optimization brings out a significant speedup, reducing the encoding time of a 4MB block from 31 second to about 10 sec using 4 threads. Adding as many as 10 threads further reduces the encoding time to 8.5 seconds, and eventually hitting physical (CPU) limits. In the second strategy, we attempted to change Coding matrix to reduce the number of XOR computation at the root. The conventional encoding complexity of LDPC codes is $O(n^2)$ where n is the length of coded symbols. We could transform the original encoding matrix to a upper triangular form that has a computation complexity of $O(n)$. But after a careful analysis, we discover an extra constant factor overkills the reduced coding complexity, so in the end we rely solely on parallelism to reduce coding latency.

2.12 Incentive Analysis

2.12.1 Motivation and Countermeasures

For oracle nodes, downloading and checking data requires nontrivial computation effort, selfish oracle nodes have an incentive to follow others' decisions and skip verification to compete for more reward by voting for more blocks. Such a phenomenon, known as *information cascade* in the literature, sets forth that people tend to make decisions according to the inferences based on the actions of earlier people. As a consequence, an oracle node may

submit a vote for a block but can not provide any data when a retrieval request is received. To solve this problem, we introduce an audit scheme. When a block is added to the side blockchain, with probability p_a, a voted node will be selected to submit the received data chunks. Our dispersal protocol guarantees that each oracle node ought to store a specific subset of data chunks. For those committed blocks which are not successfully appended in the side blockchain, all voted nodes should submit data or lose their stakes.

Even if there are several nodes in the aggregation committee, only the first one who submits the valid signature can claim the reward, thus rational committee members would choose to submit votes containing the first bath of signatures. And since only those oracle nodes whose signatures are used to aggregate the final signature would receive a reward, nodes with the largest network latency may never get any income. Furthermore, adversaries can violate an arbitrary subset of honest nodes' interests by excluding them intentionally. We can not influence the adversarial behavior, but to encourage selfish nodes to include more signatures, the committee reward will be proportional to the number of signatures in final aggregation.

2.12.2 Model

We consider a network to be a set of N configured oracle nodes (oracle nodes) $O = \{o_1, o_2, \cdots, o_N\}$. Suppose βN nodes in the network are *Byzantine*, who behave arbitrarily. And that every non-Byzantine nodes are *Rational*, they follow the protocol if it suits them, and deviate otherwise. In the registration phase, each oracle node needs to deposit an amount of stake stk_o for registering a pair of valid BLS key set.

There is a set of blocks proposed by side nodes. To propose a block, the proposer needs to deposit an amount of stake stk_b and will receive block reward B if successful. A fraction of block reward ηB will be used for rewarding oracle nodes. There is a small committee $C \subset O$ in the oracle layer. If a block is committed, the committee member who submits the commitment will get an extra reward, which comes from a constant submission fee r_m paid by other oracle nodes.

strategy	block proposer	oracle nodes	committee member
D	$-stk_b$	$-p_a stk_v + (1-p_a)\frac{\eta B}{k} - r_m$	$-stk_m$
O	0	0	0
C	$(1-\eta)B$	$\frac{\eta B}{k} - r_m - c_s$	$kr_m - c_m$

Table 2.5: Utility functions for different types of nodes. k is the number of signatures aggregated in committed signature.

Basically there are three general actions a node can take, cooperate, offline and defect, denoted as D, O, C respectively. We define *All-C*, *All-C* and *All-D* representing the strategy that all nodes choose to be cooperative, offline and defective. We design the utility functions for the block proposer, oracle nodes and committee members when a block is successfully committed in Table.2.5.

2.12.3 Incentive Compatibility

A protocol is incentive compatible if rational actors see no increase in their utility from unilaterally deviating from the protocol. To prove that, there are several things rational participants may care about.

For oracle nodes, there is a reward for having a block it successfully voted for added to the side blockchain. They cannot do much to influence the reward, except creating more voting accounts which means to store more data and deposit more stakes. For the block producer, there is a reward $(1-\eta)B$ when the proposed block is added to the side blockchain. Rational side nodes can't gain more rewards by deviating the protocol thus have no reason to violate the protocol. For the committee member, they claim rewards when a vote is accepted and more participants in aggregation bring more reward to the submission fee.

We prove that *All-C* strategy is a Nash Equilibrium below.

Theorem 3 *There exists a strategy $\{e_i^*\}_{i\in[1,n]}$ for all rational oracle nodes in system to reach Bayesian Nash Equilibrium: $\forall k \in [1,n]$, k is rational,*

$$\mathbb{E}[U_k(e_k^*)|\{e_i^*\}_{i\in[1,n]}] \geq \mathbb{E}[U_k(e_k)|\{e_i^*\}_{i\neq k}, e_k]$$

and the following statements hold:

1. *All-O strategy is a Bayesian Nash Equilibrium.*

2. *All-C strategy is a Bayesian Nash Equilibrium.*

Proof 6 *A protocol is a Byzantine Nash Equilibrium if rational actors see no increase in their utility from unilaterally deviating from the protocol. Firstly, consider $\{e_i^*\}_{i\in[1,n]} = All\text{-}O$, it can be easily derived that $\mathbb{E}[All\text{-}O] = 0$. Suppose any voter k wants to change the protocol, no matter it chooses to cooperate or defect, it can not change the consensus thus gets no reward and even pays some cost for computation when choosing to cooperate, thus All-O is a Bayesian Nash Equilibrium.*

Suppose a block is eventually committed. The expected utility for All-C is,

$$\mathbb{E}[All\text{-}C] = \frac{\eta B}{k} - r_m - c_s \tag{2.18}$$

The expected utility for k to choose to offline and defect is,

$$\mathbb{E}[e_k = O] = 0 \tag{2.19}$$

$$\mathbb{E}[e_k = D] = -p_a stk_v + (1-p_a)\frac{\eta B}{k} - r_m \tag{2.20}$$

Letting $\mathbb{E}[All\text{-}C] > \mathbb{E}[e_k = D]$ and $\mathbb{E}[All\text{-}C] > \mathbb{E}[e_k = O]$ derives that All-C strategy is a Bayesian Nash Equilibrium when $p_a(stk_v + 1) - c_s > 0$ and $\frac{\eta B}{k} > r_m + c_s$. Since both c_s and r_m are constant, we can always choose stake value to make it reach the equilibrium.

Table 2.6: List of symbols used in incentive analysis

Symbol	Definition
r_m	Rewards per signature for a committee member
B_j	Block rewards for block j
c_s	Costs for oracle nodes to verify data chunks
c_m	Costs for committee members to aggregate signatures
stk_o	Unit stake of oracle nodes
stk_m	Unit stake of committee members
stk_b	Stake of block proposer
β	Proportion of byzantine nodes
θ	Threshold of aggregated signature
η	Proportion of block rewards as block proposition cost
p_a	Probability to audit when a new block is added

2.13 Design and Implementation of ACeD Modules

The **Smart Contract manager** is an event loop which waits for the signal from other managers. Its main purpose is to communicate with the smart contract by either sending Ethereum transactions to write new states to the smart contract, or calling the smart contract to read the current state.

The **Trusted Chain manager** is an event loop used by any side nodes for periodically pulling the latest state of the main chain by asking the smart contract manager. Every side node uses it to monitor any state update from the oracle nodes. An update occurs when the smart contract block id is higher than the block id maintained in the blockchain. Each side blockchain node queries the trusted blockchain periodically to be aware of the state change. When a new state is received, the side node queries all oracle nodes using the block id, for assembling the new block. It then checks Merkle proof in the header to check the

block integrity. Then the module uses the new hash digest in the smart contract to verify the integrity of the header. The computation is done by hashing the concatenation of the previous hash digest and the hash digest of the new block header H; then compare it with the hash digest in the smart contract. The trusted blockchain manager continues until it completes to the latest state and has a consistent view of the blockchain compared to the states on the trusted blockchain.

The **scheduler** runs an endless loop which measures the current time to determine the correct side node proposer at its moment. It uses the current UNIX EPOCH to compute the time elapsed from a time reference registered at the contract creation, and divides Slot Duration, a system parameter, to get the current slot id. Therefore both oracle and side node can use slot id and the token ring to determine the valid block proposer at the current time. When a node proposes, its block id is set to the current slot id. After receiving block symbols using the dispersal protocol, oracle nodes verify the message sender using the identical scheduling rule. In the special situation while a block is pending in the network and a new block is proposed in the next slot, the pending block might be updated or rejected by the smart contract due to the block id rule constraint by the smart contract. Because a block id is accepted only if it is monotonically increasing, the releasing of the pending block does not hurt the system in either case: If the pending block arrives to smart contract after the new block, the pending block is excluded because it has a lower block id than the new block; if the pending block is included, there might be some conflicting transactions in the two blocks, but since the transactions in the pending block cannot be arbitrarily updated by adversaries after its release, the adversaries cannot adaptively damage the system. This property allows an optimization to improve the throughput (discussed later).

The **block manager** is an event loop that sends and receives message among peers. When an oracle node receives the header and coded symbols from a side node, it stores them into the symbol database, and submits its signature of the header to its Committee nodes after it receives a sufficient amount of symbols. If an oracle node is chosen as a committee node, the node receives the signatures and aggregates them until the number suggests that

there is a sufficient amount of symbols stored in honest nodes. The block manager asks the smart contract manager to send an Ethereum transaction to write the new state to the smart contract.

The states of ACeD are maintained in three data structures:

1. **block database**, residing on persistent storage, is used by Side nodes to store decoded blocks reconstructed from the oracle nodes

2. **symbols database**, residing on persistent storage, is used by oracle nodes to store coded symbols from the Side nodes.

3. **mempool**, transactions queued in memory to create a block.

The block database is a key-value persistent storage, where the key is block id and the value is a block $B = (H, C)$ which contains a header H and content C. The header is a tuple of $(blockid, D)$ where D is the digest of the content C. Instead of relying on a cryptographic puzzle for accepting blocks, each oracle node uses a deterministic token ring available from the smart contract to decide which block symbols to accept. The token ring is a list of registered side chain nodes eligible for proposing blocks. A block is considered valid only if it is received at the correct time from the correct side node.

2.14 Performance table

We compared five data dispersal protocols and analyzed their performances using maximal adversarial fraction and communication complexity; we further articulate the storage overhead for the oracle layer and download overhead for each client in two cases depending on whether there are invalid coding attacks in the system. We define those four key metrics of system performance; see Table 2.2. Let N be the number of nodes and b be the block size. "Maximal adversary fraction" measures the fault tolerant capability of the model. "Storage overhead" measures the ratio of total storage cost and actual information stored. "Download

overhead" measures the ratio of downloaded data size and reconstructed information size. "Communication complexity" measures the number of bits communicated

In the uncoded repetition data dispersal protocol, a block proposer client sends the original block to all oracle nodes to ensure the data availability. The system needs more than $1/2$ fraction of the votes for committing a block digest of constant y bits into the trusted blockchain. The total amount of bits communicated equals to Nb, where b is block size and N is the total number of oracle nodes. Hence the communication complexity is Nb. In the normal case, when a block is coded correctly, the storage overhead is Nb/b whereas the download overhead is b/b. When an adversary uploads an invalid block by distributing different blocks to each node, the worst case for storage overhead is identical to the normal case, because every oracle nodes have to store the data block to make sure the data is available when later adversaries oracle nodes decide to make their block available. The download efficiency is $O(1)$ because as long as a client contacts an honest oracle node, it is sufficient to check the data availability.

In the uncoded data dispersal protocol, the block proposer client divides the original block into N chunks, one for each oracle node. The system cannot tolerate any adversary because any withheld chunk can make the block unavailable. The total communication complexity is b since a block is equally distributed. In both normal and worst case, the storage overhead is b/b because every node only store b/N chunks. Similarly, in both normal and worst cases, the download overhead is b/b because a client downloads at most b/N chunks from N oracle nodes.

In the 1D-RS data dispersal protocol, at least $1/2$ fraction of oracle nodes need to be honest for them to agree to commit a block digest. The communication complexity is $O(b)$ using a chunk dispersal protocol discussed in section 2.4. In the normal case, each node only stores $O(b/N)$ number of chunks, and therefore the storage overhead is $O(1)$; similarly when a client node wants to download the block, it collects all chunks from the oracles node with download overhead $O(1)$. In the worst case, when an adversarial client disperses chunks with invalid coding, oracle nodes in the end would produce an invalid coding proof for proving

that the block is invalid to all querying clients. In 1D-RS, the fraud proof consists of all chunks as one needs to decode itself to detect inconsistency. Therefore, the overall storage is $O(N(b/N + \log b))$, when b is large ($b/N >> \log b$), the overall coding proof size is $O(b)$. For computing the storage overhead, we consider the total information stored in the denominator to be the size of the block digest, which is used by a client to start the block retrieval. When a client requests for an invalid coded block, the client needs to download the incorrect-coding proof to convince itself that the block is invalid, hence the download overhead is the same as the storage overhead.

In the 2D-RS data dispersal protocol, at least 1/2 fraction of nodes needs to be honest. The communication includes the chunks that make up the entire blocks and Merkle trees, and $O(\sqrt{b})$ of Merkle proof, so in total it takes $O(N\sqrt{b} + b + N \log b) = O(b)$ bytes. In the normal case, each node only stores $O(b/N)$ bytes, therefore the storage overhead is $O(1)$, similarly the download overhead is $O(1)$. In the worst case, when an adversary disperses invalid coded chunks, each node needs to download $\sqrt{b} \log b$ bytes for producing the invalid code proof, according to Table 2 of [23].

In the AVID protocol, oracle nodes run asynchronously with a security threshold 1/3 [49]. During the block dispersal, every oracle nodes needs to recollect sufficient symbols for reconstructing the original blocks, hence the total communication complexity is $O(Nb)$. In the normal case, after a block is checked valid at the dispersal phase, each node discard chunks except the ones sent from the block producer, hence the storage overhead is $O(1)$; similarly when another client downloads the data, it only needs to download $O(b)$ data, hence its download overhead is $O(1)$. When there is an invalid encoding attack , it can be detected by AVID, and then the nodes would discard the block. Therefore in the worst case, the storage overhead of AVID has the same performance as its normal case. Similarly, since AVID can detect invalid encoding before committing to the trusted blockchain, the worst case download overhead is $O(1)$, the same as its normal case.

In the ACeD dispersal protocol, at least 1/2 fractions of oracle nodes need to be honest to commit the block digest. The communication complexity is $O(b)$ by using the dispersal

protocol in Section 2.4. In the normal case, each node only stores $O(b/N)$ chunks hence the storage overhead is $O(1)$, and similarly the download overhead is $O(1)$. In the worst case, when an invalid block is dispersed, oracle layer generates an incorrect-coding proof of size $O(\log b)$ as shown in Theorem 1. The incorrect-coding proof is then stored in the trusted blockchain. When a client queries for such block, the proof is replied so that the client can be convinced that the block is invalid. For computing the storage and download overhead in the worst case, we consider both the stored information and the size of the reconstructed data to be a single hash which is the pointer to the data.

2.15 Erasure code

An erasure code encodes a string of input bytes into a string of output bytes which can tolerate missing bytes or check if bytes are modified by using some special decoding algorithms. Suppose we want to encode a block of b bytes, first we divide b bytes into k symbols(a fixed number chunks of bytes), each of $\frac{b}{k}$ bytes, some padding is used if they do not divide perfectly; then we choose an appropriate code which transforms k symbols to $n > k$ symbols for supporting erasure resistance. This ratio $\frac{k}{n}$ is called coding ratio, and usually it is a constant fraction like $\frac{1}{2}$, $\frac{1}{4}$. Many codes offer erasure resistance property including 1D-RS, 2D-RS [23], but we choose LDPC (linear density parity check) code for its special properties discussed below. LDPC decodes very efficiently by possibly sacrificing the encoding efficiency; in the encoding process, we apply n linear operations over various subsets of k symbols for generating n coded symbols, the operations can be represented as a encoding matrix; while at decoding side, only a few symbols out of n coded symbols are required to check the validity of those symbols; this can be checked by computing if they produce a zero vector after a linear operations, which is captured in a parity equation. We shown in the section 5 that the scheme we are using with LPDC only use number of symbol of roughly $O(\log b)$ bytes, whereas both 1D-RS and 2D-RS requires the decoder to download all b bytes and $O(\sqrt{b}\log b)$ bytes to run the decoding algorithm assuming symbol size is constant. Erasure Code provides erasure resistance, but there is a limit on how many modified or missing symbols can

be tolerated. To capture this property, every erasure code has a parameter called undecodable ratio. The set of symbols that meets the undecodable ratio is called a stopping set, and there might be multiple of such set. In the CMT [155] paper, the undecodable ratio is 0.125; both 1D-RS and 2D-RS can produce codes that tolerate any ratio of missing symbols. To compensate high encoding complexity, in Section 2.6 and Appendix 2.13 we discussed possible ways to parallelize LPDC encoding algorithm for a faster block generation rate. A formal introduction to LDPC can be found at [131] chapter 3.

2.16 Bad code handling

The LDPC codes are probabilistically generated, and some codes may have smaller stopping size (see Appendix 2.15) than designed due to randomness in the generation process. The CMT [155] paper section 5.4 table 3 provides the likelihood of such event that a bad code is generated: when the number of symbols, n, in a layer is 256, the probability of bad code is 0.886; when n=512, prob=5.3e-2; when n=1024, prob=2e-3. The calculation process can be found at [155] Appendix A. We layout the following protocol for explicitly handling such bad code.

A bad code handling protocol is triggered by an honest side blockchain node who is unable to reconstruct a block after receiving a sufficient number of valid chunks from the oracle layer. The honest side blockchain node sends a bad code message to all oracle nodes containing the block chunks for demonstrating the code is bad. Then all honest oracle nodes communicate with each other, and confirm if the code is bad by exchanging their local chunks to reconstruct the original block. If the code is bad, honest oracle nodes either reconstruct the block but with more chunks or fail to reconstruct the block. The honest oracle nodes then vote and write a proof to the main chain so that a new code will be accepted. The generating process of new codes can be done at each oracle node using a commonly agreed seed, which can be setup in the main chain after bad code proof is voted and accepted. The oracle node then communicates and votes for the new code, and eventually a valid new code is committed to the main chain.

Chapter 3

GOLDFISH

3.1 introduction

P2P overlay network is a fundamental layer required by any blockchain consensus protocol for block delivery and reaching consensus. In a proof of work blockchain like Bitcoin, a network with high delivery latency produces a higher chance of forking in the consensus layer, effectively reducing the security parameter for any adversary to overtake the honest chain [68]. In a proof of stake blockchain like Ethereum, a high latency network can delay the block propogation, and possibly reduce the participating reward because of missing protocol assigned tasks. This loss can discourage home validators(nodes) from running their own hardware, and increase the barrier for decentralization. Decentralization is a key defense mechanism against censorship attack, which is required for the liveness property of any consensus protocol. Recently there was an outcry within Ethereum community on large staking pools engaging in censoring of certain transactions [80, 144]. Such liveness concern can only be addressed by having a large set of decentralized nodes participating in the consensus protocol, which in turn would require underlying P2P protocol to be scalable. In the end, a node's safety also depends on security of the P2P layer. Inside the P2P network, if a node locally connects to the adversary only, its view on the state of the chain can be manipulated by feeding the wrong blocks. Such an attack is called eclipse attack [145]. Hence a P2P layer should secure nodes against connection manipulation. To summarize, a P2P protocol needs to address at least three aspects: latency, scalability and robustness to connection manipulation. We ask the question how we can design a broadcast P2P protocol that addresses those issues?

Suppose there are N nodes in a P2P network, a fully connected graph is a design that

has the most security. Its message delivery time is constant $O(1)$ and each node only receives the broadcast message only once. But such scheme is not scalable since the total number of connections in the entire network is $O(N^2)$. One can design a structured P2P network like Kademlia [110], where each node only connects to $O(log(N))$ number of peers based on consistent hashing. The network is scalable since each node only connects to a few nodes even within a large graph; it is also efficient since the network structure helps disseminate messages such that there is no redundant message received by each node [73]. Besides Kademlia, another design that reduces latency is to use synthetic coordinates [60] for creating a structured network. However, the structured design is not secure, since an adversary can always observe the network and pinpoint connections to compromise. Proposals have been made to modify structured P2P design to accommodate the security [29], but as far as authors' understanding such designs have not been adopted by scalable blockchains.

An unstructured p2p network is a balanced design between the above approaches, where connections are created by randomly choosing a constant number of nodes in the network. Such design is scalable because the node only connects to a constant number of peers. The security is also preserved since it is difficult for an adversary to predict and manipulate the node's local network given its connections are purely random out of all possible connections. The unstructured network has drawbacks of being less efficient because it relies on flooding its local network in order to broadcast the message; it is also more difficult to add latency optimization since there is no coordination among random connections from each node. However, in production, blockchains like Ethereum and Bitcoin both use the unstructured p2p design as its overlay network, for its good scalability and safety properties. The particular unstructured network design used by Ethereum 2.0 is descrribed in [145]. However, we still need to address the latency problem as it is a potential threat to the consensus protocol, and moreover, latency is an important factor in user experience for any protocol that wants to achieve fast time block finality.

Latency optimization in unstructured P2P networks is challenging, because an algorithm needs to be adaptable to variability introduced by network churn, and to the randomness

from the publishing sources. Work has been done to treat the peer selection problem as a bandit problem [109] in which a node makes exploration and exploitation to minimize the latency. However, such scheme relies on a relatively simple heuristic that only considers the peer performance at the moment, and discard all the information explored in the past. We noticed that there are still open design space inside peer selection problem, which improves latency while maintaining scalablility and security. Specifically, we can decouple the network modelling and peer selection into two parts, and we can optimize each of them with higher design freedom.

We propose Goldfish that treats the peer selection as a learning problem whose data points are continuously collected from the network to create a model, which predicts network latency to any nodes. The model is able to adjust itself across time as the network experiences churn or randomness from the publishing sources. Specifically, a Goldfish instance optimizes itself through four procedures: **data observation**, **matrix constructor**, **matrix completer** and **peer selector**. The **data observation** procedure takes care of collecting information from the network. The input data requires only the IP address from the TCP header, without extra information from the P2P overlay protocol. Moreover, in the context of blockchain, Goldfish can optimize connections to miners even if the block does not include timestamp of creation, which can provide additional privacy defense.

The core algorithm considers every input data point as a (peer, message, delivery time) tuple observed by a Goldfish instance. Those data points are used by the **matrix constructor** to fill in a matrix whose dimensions are message and peer. The missing entries represent the unknown delivery time for some messages with some unconnected peers. A naive implementation would be to create a large matrix for every possible message for every peer ever connected. But completing such matrix would be computationally intensive and slow, preventing the system to adapt with the network changes in real-time. In addition, using data from old history could also leads to inaccurate results, because data points collected long time ago are not reflective of the current network state. We solve both the issues by using a streaming algorithm to digest information in real time and deprioritize old information.

This design removes the constraint to interpolate the entire matrix, and is still able to attain optimal solution by incrementally finding the best peer within a local region.

The key step for interpolating the missing cells is to solve a non-convex optimization problem derived from the matrix, the procedure is carried out by a **matrix completer** procedure which uses an optimizer to find the best value such that the interpolated missing values are consistent to known value in the matrix. We use K nearest neighbor (see Section 3.4) to pick up the consistent peers. After interpolating the missing values, the **peer selector** procedure uses an altruistic exploitation strategy by choosing best peers from the existing set of peers that minimizes the broadcast latency. To dynamically adapt to the network as well as to explore new peers, a Goldfish establishes new connections to random peers every time the exploitation strategy is run. Each node can only have a constant number of outgoing connections, the exploration process requires to drop some existing peers. But since exploration in nature is random, sometimes a good performing peer is replaced by a random peer who turned out to be less performant. Goldfish inherently has the memory about which peers had good performance in the recent history, and therefore re-catches the good performing peers if they were dropped during exploration.

To validate the design of Goldfish, we create a simulation tool that measures latency among any pair of nodes in a network. The network can be generated with arbitrary size with ability to specify constraints on degree of any node. Every node assumes two roles. A node must be either a publishing or forwarding node for the first role; and a node must either be a static node that fixes its peers, or a adaptive node that actively selects its peers for the second role. The broadcast latency measures how well a node is choosing its peers, and is computed efficiently using the shortest path algorithm. We evaluate Goldfish under both analytical and complex dynamic settings. In the analytical setting, only 1 Goldfish node is adaptating and the number of publishing source is equal to the degree of the node. We can quantify the performance by clearly defining success and failure: if Goldfish can attain the global optimal solution which is direct connections to all publishing sources. We show that in a network size of 100 nodes with 3 publishers, a Goldfish node succeeds with 92.7%

probability from 200 randomly generated network by only exploring every peer only once. In more complex dynamic settings, many independent nodes are publishing and adapting simultaneously. Because the global optimal solution for all node simultaneously is computationally intractable, we compare Goldfish with a baseline algorithm Perigee [109] to solve the same problem. We show that Goldfish has strictly superior performance in all cases, and reduces broadcast latency by 14.5% than the baseline algorithm.

Section 3.2 summarizes the related works. Section 3.3 describes data measurement methods and network assumption. Section 3.4 dives into details of Goldfish. Section 3.5 presents the simulation tool and experiment results. The code is available at [12].

3.2 Related work

3.2.1 P2P network and Perigee

A P2P network is an overlay logical network that has a wide range of applications including distributed file storage [132], distributed lookup table [110]. Because connections on an overlay network are logical, the physical property like peer-to-peer distance is abstracted away, therefore the network topology has great impact on a node's shortest path to other nodes in the network. Perigee [109] is a system specifically built for blockchain, it treats peer selection as a tradeoff of exploration vs. exploitation. It uses local time measurement similar to Goldfish as the only source of information for making peer selections. However, unlike Goldfish, Perigee only uses current set of connections to decide which peers to keep. Its core (called subset) algorithm enumerates all possible combination of exploitation subset from the current peers, then associates each combination with a score generated using time observation in its current connections; it chooses the exploitation peers which achieves the best score. Perigee was shown to have similar, if not superior, performance than Kademlia in many network scenarios.

3.2.2 KNN and tensor graph

The Matrix completion problem in Goldfish can be categorized as a unsupervised learning problem, whose input data does not contain any label and is often represented in vector form. Goldfish employs K-nearest neighbor (KNN) to solve this learning problem, which is a method that looks for using K closest vector surrounding the interested vector for interpolation. Since the optimization problem in Goldfish is non-convex, we use Pytorch [16] to empirically evaluate variables for finding their local optimal solution. Pytorch is an optimized tensor library for deep neural network, which has shown been effective to train machine learning model with appropriate loss functions.

3.3 System Model

A node inside a P2P network can be an adaptation node, a publishing node or both. An adaptation node continuously optimizes its connections, as opposed to a static node which always keeps the same set of connections. A publisher node generates original messages depending on its publishing probability. In blockchain, publishers are the miners, and adapting nodes are full nodes that optimize their outgoing connections.

Every node has constraints of CPU and bandwidth, which translate to how many concurrent peers a node could maintain connections. In this work, we assume a model that every node can at most maintains a constant number of outgoing and incoming connections. A node has 2 actions on how to react to an incoming connection, by either rejecting it if the node has saturated all its connections or by accepting it. A node can freely choose any peer combinations for initiating its outgoing connections, though some outgoing connections might be rejected by the other node if the other node is saturated. In this work, we assume each node knows and has means to connect to all other nodes in the system, and therefore a Goldfish needs to make decisions from an exponential number of combinations of connections.

Every P2P node relays messages. Any connection is bidirectional, and every node is required to relay messages for all incoming and outgoing connections, except to the peer

from whom the node firstly received the message. All nodes are expected to receive multiple copies of the identical messages from all its peers. We do not assume a message to have timestamp or any other identification related to who generates the message. All nodes in the network need not synchronize their time. Instead, each node uses relative time measurement for each message to record its peer's performance. Suppose peer A sends a message, and the message is the first one that arrives at the local node at local time t, then peer A has a measurement of $t - t = 0$; peer B sends the same message at t', and has a measurement of $t' - t$ in local time. Such measurement is easy to implement and robust to time tempering since it only relies on local observation.

3.4 Goldfish

Goldfish is a distributed algorithm that each node runs locally to optimize its shortest topological distances to other nodes in the P2P network. Goldfish mimics an intelligent entity optimizing for its own improvements: it collects surrounding information, organizes it and recognizes pattern in order to make next move to fit in a ever-changing environment. They correspond to four components, as shown in Fig 3.1. They are Network-Storage, Matrix Constructor, K-NN Matrix completer and Peer Selector.

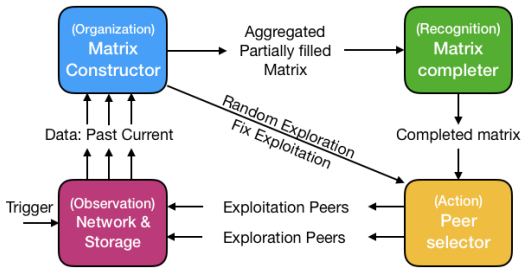

Figure 3.1: Goldfish components flowchart

We refer to the data forwarded by a node as a message, and refer to the content of a message as a block. A node can receive multiple messages of the same block. The Network-Storage component observes and keeps a memory about any delivered data in the local

storage and formats it as a (peer, block, delivery time) tuple. The delivery time is measured with relative timestamp, see Section 3.3.

At initial, Goldfish is triggered to collect data tuple, and uses a Matrix Constructor to organize observations from the local storage for creating data batches. A batch contains messages observed in a contiguous time span without connection changes, we use epoch $E_1, ..E_\ell$ to denote each of the ℓ batches corresponding to each contiguous time span.

For messages collected in a single epoch, we can create a 2D matrix, where each block has its own row vector, and each peer has its own column vector. The intersection of row and column stores the recorded latency. But to concatenate blocks from multiple epochs into a single matrix, every block vector has to insert empty cells for peers only present in other epochs. The **Matrix Constructor** creates a synthesized matrix by concatenating epochs along the block axis, and its peer axis is union of connections across epochs. A synthesized matrix from 2 epochs is shown in Table(3.1a) where $\star$ are missing observations, which is further discussed in the following sections.

Goldfish uses Matrix completer to estimate the missing cells. First, the completer classifies all cells in the matrix into 4 categories: **observed**, **symbolically observed**, **estimable** and **inestimable**. For all estimable missing cells, they are formulated as optimization variables in a non-convex optimization formulation. While designing the loss function, we have a key insight that block vectors exhibiting similar peer delivery pattern are more likely from the source-nodes close to each other. We can compare those block(row) vectors and use K-NN to interpolate the missing cell using nearest neighbor. To solve for solution, we create a pytorch tensor graph (see Section 3.2) to empirically optimize for the missing values.

After estimating the missing cells, Goldfish can use any peer exploration algorithm. One particular selection algorithm that helps to evaluate Goldfish performance is the depleting pool, where exploration peers are drawn from a depletion pool which includes all nodes in the network at the beginning; the peers are taken out of the pool without replacement, and resets itself until the pool is emptied. Exploitation peers are chosen based on an altruistic heuristics. We first identify a set of contributing peers who always deliver blocks fast to us,

and we credit them scores by counting the number of benefiting peers who receives block fast from us. The altruistic principle is based on an intuition: if a Goldfish can help so many other nodes, then itself must already be well connected, so a Goldfish should improve itself by the measure how helpful it becomes to the others. A Goldfish can occasionally skip recognition step if it wants to speedup exploration as represented by the diagonal in Fig. 3.1, and this is implemented inside Scheduler submodule.

Since the system is remarked by its short memory only to appreciate recent data (because historic data is not reflective to current network condition), the completer only needs to process a few epochs at a time. It greatly reduces the space and computation complexity. For the rest of the section, we characterize every component of Goldfish with greate details.

3.4.1 Matrix Constructor

Matrix constructor transforms the tuple data from multiple epochs to a single partially observed matrix. A concatenated matrix with 2 epochs in shown in Table (3.1a). In both epochs, the number of active connections is 3. In epoch E_1, the node is connecting to peers n_1, n_2, n_4; whereas in epoch E_2, the node is connecting to peer n_1, n_2, n_3. There are 4 blocks in each epoch being delivered to the node, with notation $m_1, ..., m_8$. Each superscript on the block denotes the original publisher for the block, which is used only to elaborate the algorithm, and Goldfish does not need this information for execution. As shown in Table(3.1a), after combining 2 epochs, each epoch contains a missing sub-column because those peers are not connected when the corresponding blocks were perceived. In the next step, Goldfish categorizes cells in order to formulate the optimization problem.

Preliminary Cell classification

A partially observed matrix, T, of size (p, q) contains 3 types of cells: observed, missing, and symbolically known cells. Every cell in the matrix is associated with 2 attributes: the delivering peer and the delivered block. A cell (i, j), where $0 < i \leq p, 0 < j \leq q$, is an observed cell if there is a peer n_i delivered a block m_j to the node, and $t_{i,j}$ is the associated

Table 3.1: Transformation from (a) Synthesized matrix to (b) Output of Matrix completer

	Block	Peers			
		n_1	n_2	n_3	n_4
E_1	$m_1^{n_5}$	0	$t_{1,2}$	$\star$	$t_{1,4}$
	$m_2^{n_2}$	†	0	$\star$	†
	$m_3^{n_5}$	0	$t_{3,2}$	$\star$	$t_{3,4}$
	$m_4^{n_6}$	$t_{4,1}$	$t_{4,2}$	$\star$	0
E_2	$m_5^{n_2}$	†	0	†	$\star$
	$m_6^{n_6}$	0	$t_{6,2}$	$t_{6,3}$	$\star$
	$m_7^{n_6}$	0	$t_{7,2}$	$t_{7,3}$	$\star$
	$m_8^{n_7}$	†	0	$t_{8,3}$	$\star$

(a) $\star$ is missing cell. † is symbolically known cell when a peer gets the first block from local node

$\Longrightarrow$

	Block	Peers			
		n_1	n_2	n_3	n_4
E_1	$m_1^{n_5}$	c_1	$c_1 + t_{1,2}$	a_1	$t_{1,4} + c_1$
	$m_2^{n_6}$	†	c_2	×	†
	$m_3^{n_6}$	c_3	$t_{3,2} + c_3$	a_2	$t_{3,4} + c_3$
	$m_4^{n_6}$	$t_{4,1} + c_4$	$t_{4,2} + c_4$	a_3	c_4
E_2	$m_5^{n_2}$	†	c_5	†	×
	$m_6^{n_6}$	c_6	$t_{6,2} + c_6$	$t_{6,3} + c_6$	a_4
	$m_7^{n_6}$	c_7	$t_{7,2} + c_7$	$t_{7,3} + c_7$	a_5
	$m_8^{n_7}$	†	c_8	$t_{8,3} + c_8$	×

(b) Completed matrix. × is infeasible cell. † is symbolically known cell. $a_1 - a_5$ are estimated missing values. $c_1 - c_8$ are offset.

measurement. The measurement for an observed cell (i, j) is computed as time difference between the peer n_i's delivery time and the earliest delivery time for block m_j among all q peers. If a peer n_i does not send the block m_j to the node, the cell (i, j) is categorized either as a missing cell($\star$), or as a symbolically known cell(†). A missing cell is identified if the cell is synthetically created as the result of merging multiple matrices along the block axis; a symbolically known cell is identified when the cell's associating peer does not forward the block to the Goldfish node; in which case, we can infer the local node (who is creating the table) is the first node who forwards the block to that peer (since a node should not send back the block to where it comes from). For example in Table(3.1a), cell $(2, 1)$ and $(2, 4)$ are

symbolically known, and we can infer the local node is the first one that is delivering the block m_2 to peer n_2, n_4. The table suggest a situation where n_1, n_3, n_4 are far away from the original publisher n_2 for m_2, and the local node is on the faster path to deliver the block to other peers. Although the symbolically known cells are missing measurement data, they still worth to consider because they convey the associated peer cannot be the fastest ones to deliver similar blocks to the local node. These 3 cell types partition $p \times q$ cells, and each cell type induces its own binary masking matrix on T.

Scheduler

A local node can combine arbitrary number of epochs into a matrix for cell interpolation. But as network continues to change, historical data might damage the completion accuracy because the network topology might have changed so much that those date are no longer useful to represent the current network. There are 2 design spaces to tune learning behavior: the size of the synthesized matrix determined by the number of epoch, and frequency to initiate the algorithm. For example, a node can set the matrix to contain 3 epochs, and choose to run matrix completion every 2 epoch. In which case, instead of running completion every time when peers change, the node can skip 1 learning epoch and use it to explore more peers. This speeds up node exploration, and saves resources on running computational tasks.

3.4.2 K-NN Matrix Completer and Missing Cell Classification

After data is formatted into a matrix, Goldfish uses K nearest neighbor to interpolate the missing cells. We begin by treating every row of the synthesized matrix, T, as a vector in the space $\mathbb{R}^q$, $(T_{i,j} = m_i[j])$, the goal is to cluster similar rows and use the available data to fill the missing ones.

Estimable Missing Cells

The missing cells are further classified into 3 categories: estimable, ambiguous (‡) and infeasible(×). A missing cell attributed by block r peer u, $m_r[u]$, is estimable if there exists another block m_i which has measured data from peer u, and both blocks m_i, m_r have measured data from least 2 common peers. The intuition is that, we can use the closeness of the observed value to derive the unobserved value. But since we use K nearest neighbor to interpolate the missing dimensions, we need to find K such blocks to complete the estimation. However, not all missing cells can find enough K blocks to satisfy the requirements. Those cells are deemed as ambiguous (‡). Infeasible cells are missing values that cannot be estimated, its definition is not central to peer selection, so we move it to the end.

Distance Metrics for Selecting K Nearest Neighbor

A metric function $g : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^+$ takes two block vectors and produce a non-negative value. A metrics is good if it produces a small value when two block sources are close and they have similar delivery viewed from a perspective of the local node. Let's first focus on a simple case where all publishers have perfect synchronous clocks and block creation time is printed on all messages. Then if a node receives multiple messages about the same block, it can deduce about its peers' topological positions in the network based on the messages arrival time. But in the real world, we cannot get accurate time measurement because time synchronization precision among computer can be low, and bad data can significantly damage the model. In Section 3.3, we discussed to use relative time to solve the problem, and it has an effect to offset the fastest route to 0. It appears we miss some information, because all values subtracts a common constant, but subtracting a offset does not change the differential value among vector's dimensions. Therefore, we can still use the differential characteristics of the block vector to infer if 2 block vectors are coming from close publishers. To construct the distance metrics, we subtract 2 vectors and take the unbiased variance on the peer dimensions where the local node has observations to both blocks, i.e. $g(m_r, m_i) = \mathrm{Var}(D_i)$

where $D_i = \{m_r[j] - m_i[j] : 0 < j \leq q, B_{r,j} = 1, B_{i,j} = 1\}$ (B is a binary matrix indicating if a cell has observation). At least 2 values are required to compute unbiased variance.

The differential variance metrics has a good property that groups blocks whose miners are close to each other. The intuition behind is that publishers with close topological locations generate blocks that exhibit similar differential variance. We illustrate this property by an example. From the Table(3.1a), since $m_4[4] = 0$ we know n_4 is close to n_6 compared to other nodes, such that n_4 is the first one to deliver m_4 (mined by n_6) to the local node, but n_4 is not fast enough to make the local node be the first forwarding node to n_1, n_2, i.e. n_1, n_2 have their separate but slightly worse path than n_4 relative to the miner n_6. In epoch E_2, peer n_4 is dropped by the local node to connect with n_3. Suppose this is the only connection change in the entire network, then relative time difference for n_1, n_2 to deliver the block is identical as before, i.e. $t_{4,2} - t_{4,1} = t_{6,2} = t_{7,2}$. The differential variance metrics $g(m_4, m_6)$ is computed as $\text{Var}(\{t_{4,1}, t_{4,2} - 4_{6,2}\}) = \text{Var}(\{t_{4,1}, t_{4,1}\}) = 0$, and similarly $g(m_4, m_7) = 0$. Therefore m_4, m_6, m_7 are grouped together in the next section for optimizing the missing cells.

Optimization Formulation.

After defining the optimization space $\mathbb{R}^q$ and the metric to find K nearest neighbors, we start to formulate the unsupervised optimization problem. Suppose there are in total s estimable missing values, whose set is denoted as S. In the partially observed matrix T of dimension $p \times q$, we associate each missing value with an optimization variable, and denote them as $A = a_1, ..., a_s$. Each missing value in the matrix has a row and column indices, which can be encoded by 2 functions: $\textbf{row} : [s] \to [p] \quad \textbf{col} : [s] \to [q]$. Notation $[n]$ denotes a integer set $\{1, .., n\}$. For each missing cell $a_v \in S$, its k nearest vectors can be pre-processed using the metrics defined earlier, and the processed result can be summarized by 2 functions. Function $\mathbf{N} : [s] \to [p]^k$ encodes for each missing cell a_v, which k block vectors out of p block vectors are closest to the block $m_{row(a_v)}$ where a_v is a part of. Furthermore, the loss calculated by metrics $Var(D)$ between $m_{row(a_v)}$ and any of its k nearest vector can be encoded as a function

$\mathbf{w} : [s, p] \to \mathbb{R}$. Since we only assign weight to k such vector, other vectors has a weight of 0. We further normalize the k weight so that their sum equals to 1 using softmax [147]. We also define a helper function $\mathbf{L} : [p, p] \to \{0, 1\}^q$ which takes 2 integers i, j and returns a q dimensional binary vector indicating peers who deliver block m_i, m_j to the local node.

To illustrate how to construct the loss function, we begin with a simpler task by optimizing for a single missing cell a_v. We can get its k closest vectors and the corresponding weight using the function defined earlier. $N(v)$ gives k peers and $w(row(v), o)$ gives their weights. Since all latency are measured with relative time, we have a problem that any 2 row vectors are not comparable to each other due to a lack of common reference time. To illustrate it, we performed a step-by-step cells completion for both blocks m_4 and m_6 in the Table(3.1a). Suppose m_4, m_6 are grouped together and they are generated from one publisher, we can calculate the vector difference based on measured time and use that to complete the missing ones. In this case, the difference are $t_{4,1} - 0 = t_{4,2} - t_{6,2}$. As the result, the missing cell $m_4[3]$ should equal to $t_{4,1} + t_{7,3}$ and the missing cell $m_7[4]$ should equal to $-t_{4,1}$. As we can see, although we can compute those missing values, we are not able to compare latency across rows. Specifically block m_7 is slower than m_4 by $t_{4,1}$, but the negative value in $m_7[4]$ is the lowest number. Consequently, it is much more helpful if there is a common reference such that we can easily compare latency among multiple row vectors based on absolute number. For that purpose, we introduce extra optimization variable, $C = c_1, ..., c_p$ for each row. We show an example of a compensated matrix with all optimization variables in Table(3.1b). With all variable defined, the optimization formulation is shown below

$$\arg\min_{A,C} \sum_{v=1}^{s} \sum_{o \in N(v)} w(v, o) \|I(\Delta_{v,o}, \Theta_{v,o})\|_2 + \|A\|_2 + \|C\|_2 \tag{3.1}$$

$$\text{s.t.} \quad \Delta_{v,o} = L(row(a_v), o) + \vec{\boldsymbol{e}}_{col(a_v)} \tag{3.2}$$

$$\Theta_{v,o} = T_o + c_o \mathbf{1}^q - T_{row(a_v)} - c_{row(a_v)} \mathbf{1}^q - (a_v - c_{row(a_v)}) \vec{\boldsymbol{e}}_{col(v)} \tag{3.3}$$

We use this formulation to continue to optimize for a single cell a_v. After we identify k nearest vectors and their weight, we optimize for the ℓ_2 norm on the difference between the 2 vectors. The vectors subtraction is shown in equation 3.3, where T is a matrix containing all observed latency data, and all unobserved data are filled with 0; notation T_o is the o-th row of the matrix T. As shown in the equation 3.3, $\Theta_{v,o}$ is the difference between the row vector $m_{row(v)}$ where a_v comes from, and the neighboring vector o. But since each of 2 vectors contains many other missing cells, and those cells do not contribute to the estimation of a_v. They are not considered in the metrics weight calculation. It is done by creating an indicator vector $\Delta_{v,o}$ in equation 3.2 with the helper function $L(row(a_v), o)$ and an elementary vector $\vec{\boldsymbol{e}}_{col(v)}$ to specify which dimensions should be contributing to the loss. We enable it with a mask selector function $I(a, b) = \{b_j : 0 < j \leq q, a_j = 1\}$ function to combine equation 3.2 and 3.3 in equation 3.1. To optimize for all optimization variable, we optimize for sum of loss and add two regularization terms to keep optimization solution bounded.

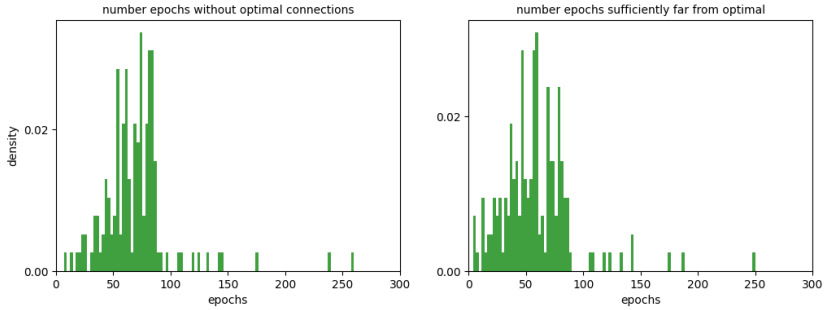

Figure 3.2: Histogram of number of non-optimal epochs

Optimization Solver.

The stated optimization problem is an unconstraint non-convex optimization problem, which is difficult to solve by analytically deriving gradients for each variables. Instead, we use a popular pytorch autograd package proven to be successful in many machine learning tasks.

See Section 3.2 for pytorch tensor graph. Solving the formulated problem requires 3 components: node definitions, a loss function that defines graph edges, and an optimizer. We define A, C as the edge nodes that requires gradients and each observation row vector T_i as constant nodes, so that the back-propagation only optimizes variables A, C. The loss function defined in the last section gives straightforward instructions to construct DAG tensor graph: the binary vector $\Delta_{v,o}$ decides edges for the graph, and the computation logic is contained in $\Theta_{v,o}$; in the end, we regularize A, C by adding them to the final cost. Since none of variables A, C is constrained, we can use optimizer provided from pytorch like adam, which has been highly optimized, and gives us a better performance in short time. It is important to balance the complexity of edges among tensor nodes.

Infeasible Missing Cells

A missing cell can be infeasible ($\times$) if its value cannot be estimated, which can occur in 2 situations. First, when there is only 1 numeric observation in vector m_r. It happens when the local node is the fastest relaying node. In the Table(3.1a), cell $m_2[3], m_5[4]$ are infeasible. Second, a missing cell, $m_r[u]$, is infeasible if there is no vector m_i satisfying 2 properties: the local node has numeric observation for block i from peer u; the local node has numeric observations for block i and r from at least 2 peers (number of 2 is discussed in Distance metrics).

3.5 Experiment and Evaluation

We aim to investigate the following questions through evaluations: in a random network (1) Can a single Goldfish instance use a short memory to find and retain the global optimal connections quickly when the unique global optimal solution is direct connections to all publishers? (2) What is the performance of multiple simultaneous Goldfish running under varying number of publishers when global optimal for all adapting nodes is computationally intractable? (3) What is the performance of multiple Goldfish in networks with real world propagation latency under varying number of publishers? We address (1-3) with a simulation

tool.

3.5.1 Simulation Framework

For answering (1) and (2), we create multiple artificial networks inside a 2D plane, where locations of nodes are uniform randomly generated along both axes. For answering (3), we use measured latency [148] among randomly selected cities in the real world to create a 3D network graph. Every node is constraint by 8 incoming connections and 4 outgoing connections; any pair of nodes need to respect both constraint before setting up a connection. Every node needs to relay messages as specified in Section 3.3. Network operates in round fashion. Only one publisher generates one message in each round, and each node has a fixed publishing probability, together sum to 1. Every message is broadcast until every node has a copy. The delivery latency between 2 directly connected peers has two components: fixed 20ms node (including processing and transmission) delay and propagation delay. The latency of delivering a message between any 2 nodes is the shortest path on the weighted graph, whose edge weight is propagation latency. We use Dijkstra algorithm to find the shortest path using only the exploitation edges. Each experiment is initialized with a random network topology (respecting the edge constraints) using a random seed. Every Goldfish node uses 3 outgoing connections for exploitation and 1 for exploration. Every epoch contains 40 messages(rounds), and the matrix constructor combines 3 epochs, where the middle epoch is always the pure exploration epoch, see Section 3.4 Scheduler. Matrix completer sets $K = 2$ for nearest neighbor problem, and runs at most 2000 optimization steps. The typical amount of time to run a matrix of size 120×7 requires around 20 seconds on a 6 cores 2.6G Hz 16 GB Mem laptop.

3.5.2 Global Optimal

We claimed that Goldfish can use short memory to find and retain the best connections with high probability only by exploring every peer once. To evaluate the idea, we setup experiments where the number of publishers is less than or equal to exploitation limit; in such

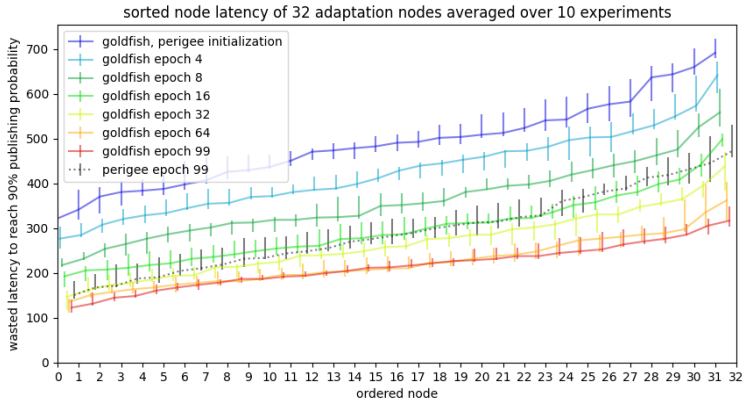

Figure 3.3: Performance comparison between Goldfish and Perigee in a random graph of 100 nodes with exponential publishing probability. Solid line connects median values, upper bar shows the 75th wasted latency, and bottom bar for 25th of 10 experiments. The dashed line shows captures the network that uses Perigee.

situation, the global optimal solution is direct connections to all publishers. The experiments contain 200 random graphs on a 500×500 plane, and each experiment runs on a distinct graph configuration of random node locations and initial network topology for 300 epochs. In each graph, 3 random nodes out of 100 nodes are assigned with publishing probability $\frac{1}{3}$, and 1 other node is randomly selected to run Goldfish for changing outgoing peers (3 exploitation and 1 exploration) per epoch; all other nodes remain static. Note that a Goldfish has to spend at most 96 epochs to discover every peer once.

Fig. 3.2(a) is a histogram generated from the 200 random experiments whose x axis is the number of epochs that a Goldfish node does not attain optimal solution. This condition captures both the convergence rate to find the optimal solution and tendency to diverge from the optimal solution. It shows that in most (around 92.7%) cases Goldfish can find and retain the optimal solution by exploring every peer only once.

Fig. 3.2(b) is a similar histogram that depicts how many epochs the Goldfish node is sufficiently far away from its unique optimal solution. Let $\lambda_i(e)$ be the latency difference at

Table 3.2: Goldfish Perigee Latency on 10 experiments under both random and real model. measured in millisecond

topo	#	pub	#	Goldfish				Perigee				ratio
type	pub	prob	adapt	25th	50th	75th	*mean*	25th	50th	75th	*mean*	mean
random	100	exp	10	249	288	342	299	262	319	370	323	0.926
			32	181	218	259	222	237	296	356	306	0.725
			64	155	185	217	191	188	227	270	235	0.813
			100	134	162	193	169	151	173	201	178	0.949
	32	unif	10	261	294	354	308	288	315	381	332	0.928
			32	197	228	264	234	255	293	330	298	0.785
			64	164	188	218	194	189	218	260	229	0.847
	64	unif	10	287	320	358	328	316	348	390	358	0.916
			32	250	282	312	284	260	301	342	306	0.928
			64	195	218	247	226	195	221	257	229	0.987
real	100	exp	10	130	154	188	162	144	181	218	184	0.880
			32	114	131	158	140	128	152	193	165	0.848
			64	100	114	131	120	118	136	163	145	0.828
			100	86	99	114	104	98	111	130	120	0.867

epoch e between the optimal solution and its current Dijkstra distance; we compute and plot the sum of their difference for every epoch every experiment $\lambda(e) = \sum_{i=1,2,3} \lambda_i(e)$; the criteria for an epoch to be sufficiently far is determined as $\lambda(e)/\lambda(0) > 0.05$, where $\lambda(0)$ is latency which a Goldfish node gets from random connections. From the figure, the distribution is shifted downward, around 61.5% cases Goldfish finds sufficiently close solution within 48 epochs.

3.5.3 Multiple Goldfish *on Many Publishers Network*

When a Goldfish is optimizing for more publishers greater than its exploitation limit, the node needs to compromise connections to balance latency to all publishers. In such complex system, finding the global optimal solution is intractable, hence we evaluate performance by comparing Goldfish against a baseline algorithm Perigee [109], see Section 3.2. Figure 3.3 is a comparison for a 100 nodes random graph which contains 32 adapting nodes, 68 static nodes, and all nodes have publishing probabilities based on an exponential distribution (80% probability is concentrated on 20 nodes). For evaluating broadcast latency of each node, we use only exploitation edges to compute the weighted shortest distance to all publishing sources. The broadcast latency is identified as the distance to reach 90% of the publishing probability. To visualize how much extra time wasted on the topological relay as opposed to direct connection, we subtract each node's topological latency with their direct connection latency, and plot the sorted broadcast wasted latency. To get a robust comparison, we run 10 experiments with random graph to get the 25th, median, 75th percentile of i-th node performance. The last epoch (99) of Perigee is shown as the dashed line in the figure. Fig. 3.3 shows Goldfish keeps improving performance until it converges at a region around epoch 64 and 99; Goldfish has better performance than Perigee.

To robustly compare Goldfish and Perigee, we run both systems under various network scenario, and each scenario is repeated with 10 different network initialization. Table 3.2 records the performance at their final epochs after summarizing over 10 runs. We use the identical evaluation method to show broadcast wasted latency. As the number of adapting nodes increase, both systems produce better performance, but Goldfish is consistently better in all categories. We observed that when there are many adaptation nodes, Goldfish is only slightly better than the Perigee; the difference is most significant when around 30% nodes in the network are adapting.

3.5.4 Evaluation based on real world latency

We use real world geolocation and latency to setup the network. The result is shown in the lower part of Table 3.2. Compared to Perigee, Goldfish on average uses 14.5% less time.

3.6 Conclusion and Discussion

P2P is a fundamental component in many decentralized applications. Our work addresses improvement on broadcast latency and exploration convergence by bringing idea from matrix completion and streaming algorithm to synthesize a general view about the network. When the number of publisher is equal or less than the number of exploitation connections, the Goldfish can find and retain the global optimal connections. When the network is dynamic and complex, Goldfish saves 14.5% time than its baseline.

Chapter 4

BIGDIPPER

4.1 Introduction

Short term censorship resistance provides an unusual property that differentiates from the conventional liveness property in BFT protocols. It guarantees transaction inclusion in the next block even if the current and future leaders have the intent of censoring. This property is especially useful for transactions whose value is sensitive to block heights when they are included; for example, an auction bid trying to enter at the last permissible block height. The benefits of th property can be generalized further, such as periodic inclusion at precise block height and real time interactions like high frequency trading, collateral liquidation and web3 social network. In a partially synchronous network, conventional leader based BFT protocols [42, 51, 107, 153] cannot provide this property, because the leader unilaterally decides which transaction appear at what place. The middleman nature of the leader creates a principle agent problem between the BFT protocol and the leader, where the leader can extract benefits from the protocol through arbitrage, sandwich attack and bribery for censorship. There are prior works [97, 98] that use a notion of fair ordering to constrain the leader. However, those protocols require every transaction to be received by majority of replicas, so that a fair order consensus can be reached based on the relative ordering of receiving time for producing a first come first serve (FCFS) ordering. Both fairness and censorship resistance are addressed in one shot, but it comes with downsides of limited scalability and high protocol complexity. Moreover, the rationale and implication of FCFS is still at debate. BigDipper treats the problem differently by focusing primarily on the property of short term censorship resistance while leaving an interface to support any ordering mechanism. The benefits are twofold. First the property of short-term censorship resistance itself is useful for many

applications, by removing fair ordering constraint the throughput of the protocol can scale linearly as the number of replicas joining the system increases. Second, by decomposing the protocol's stack with modular interfaces, new protocols can work directly at the fair ordering problem by stacking on top of existing protocols. A concurrent submission called Travelers makes use of this modular approach to develop a scalable fair ordering protocol, BigDipper can be directly plugged in to its protocol stack as its censorship resistance BFT consensus.

Transactions differ by their urgency for censorship resistance, like an expiring auction bid is more important than some dinner fees transfer among friends. But this information is known by the clients only, who are also dynamically changing over time. So a flexible protocol is unlikely to be one that offers a rigid inclusion guarantee for everything. The Shard protocol in Appendix 4.8 fails on this part. Essentially between the protocol and clients, there are two information asymmetries, one is about time sensitivity on the value, the other is about which replicas are malicious. Although clients can uses transaction value or tips to differentiate its urgency from others, but those information alone are not enough for censorship resistance, because determined malicious replicas can censor regardless of its tips.

In BigDipper, we offer a robust and flexible censorship resistance. It is achieved by an architectural change to the conventional BFT protocol: all transactions need to submit to a new component called **Data Availability and Censorship Resistant (DA-CR)** layer which is maintained among the BFT replicas, then a valid block can be created by a leader by including all the transactions available within DA-CR. If the leader fails to include some transactions from it, the leader will fail to reach consensus and will be replaced for violating the liveness. But the leader enforcement alone is insufficient to achieve censorship resistance, we need all replicas be able to permissionlessly add transactions to DA-CR. Because the BFT replicas are decentralized, it is used as the source for providing censorship resistance. From the perspective of a consensus protocol, DA-CR is an intermediate layer that restores the decentralization from the leader. In order to do that, replicas collect and batch transactions into a new data structure called mini-block, which is the basic unit to send to DA-CR. The

final BFT block is made of those mini-blocks. A client uses a transaction broadcast protocol to have its transactions batched into mini-blocks by replicas.

We made two contributions for the DA-CR component. First we delineate the properties by three parameters, (ρ, t, η), and we produce three designs that demonstrate different trade-offs with regard to the message complexity, trust assumption and degree of censorship resistance. The parameterization of each property allows us to evaluate the effectiveness of the protocol: a protocol is more useful if it can achieve a desired property with low cost. Specifically, ρ measures the data tampering resistance to adversarial attacks, t measures the amount of censorship resistance, i.e. how many honest mini-blocks have to be included in the final block, and η measures how much influence a leader can impose on the inclusion of mini-blocks. We elaborate the exact definitions in Section 4.3. Each of three protocol designs has distinct (ρ, t, η) properties. The three DA-CR designs can be categorized into two general design patterns depending on if they support accountability. In an accountable protocol, every replica knows whose mini-blocks are included in the final block. In Section 4.3 and 4.4, we delineate their differences and implications on the derived protocols.

The second technical contribution is a transaction broadcast protocol that enables clients to adjust the amount of resources spent on censorship resistance. We use probability of inclusion as a quantifiable measure for the amount of censorship resistance from which a client can choose the desired quantity. Once a probability is selected, this quantity can translate directly to the number of transaction copies that should be sent to distinct replicas. The more copies of a transaction a client sends, the higher chance the transaction would get included, up to probability 1. Allowing clients to express inclusion urgency through probability simplifies protocol complexity. We discuss spamming and transaction de-duplication in Appendix 4.20, the central idea is to correlate the spent resources with transaction fees.

A final technical contribution is an enforcement rule inserted into the BFT consensus path, so that no leader can append a BFT block to the ledger unless all the data from the DA-CR are included. We provide an integration with a two-phase leader based BFT protocol, Hotstuff-2 [107], in Section 4.6.2, and prove both safety and livenesss. The integration fits

all DA-CR parts inside the two-phase BFT protocol, and requires no change to pacemaker during leader rotation.

Why we prefer leader based protocol

It is an empirical observation that resource distribution are highly asymmetrical. For example, the top three miningpool on Bitcoin occupy 65.9% of total hash power [34]. In Ethereum post-merge, Flashbot relay propagated 80% of blocks [152]. We expect similarly for parties that runs hardware (replicas). Because BFT protocols rely on at most 1/3 replicas are malicious, it is less likely that decentralized parties would collude together when the total number of replicas are large. To achieve that, we need the communication complexity on each node to be small. But since resourceful parties still exists and participates the system, they can play the role to perform heavy computation, as long as they are rewarded accordingly. Leader based protocol fits well to this model. More discussions in Appendix 4.9.

4.1.1 Comparison with Other BFT protocols

Many asynchronous BFT protocols by nature provide censorship resistance, including HoneyBadger [112] and DispersedLedger [151]. In HoneyBadger, every replica creates its own mini-block and runs a binary agreement protocol in parallel to commit at least $n - f$ mini-blocks. There is no centralized leader in the system, and because of that, the asynchronous system enjoys censorship resistance inherently. Many DAG-basd BFT protocols like Bullshark [141], Tusk [64], Dumbo [82], VABA [20] are also leaderless and therefore censorship resistant. But without coordination, the leaderless protocols have a all-to-all communication pattern which hurts communication complexity.

To understand the protocol's trade-offs, we use three metrics to systematically compare their performance: **worst case inclusion distance**, **system throughput**, and **block interval**. The worst case inclusion distance measures the largest number of blocks malicious replicas can prevent a transaction from entering the blockchain in the worst scenario.

Second, a performant blockchain should support many clients and satisfy throughput requirements for a wide range of applications. Assuming each replica has a constant bandwidth C_{band}, ideally we want the maximum system throughput to increase linearly as more replicas join the network. As more replica joins, it reinforces both the decentralization of blockchain and the system performance. To capture the scalability of the system, we measure it as a ratio between the total system throughput and individual bandwidth.

The third metric focuses on the latency of the block interval, because the shorter the interval is, the faster a client can receive transactions confirmation. The latency can be decomposed to two parts: number of round trips and communication latency which is a ratio between the communication bits and bandwidth.

Table 4.1: Comparison of BFT protocol in a system of n replicas

BFT protocols	Coordinating Leader	**Per Replica Communication**			System / Replica Throughput	Number Comm Steps	Sum Comm. Complexity	Worst Inclusion (blocks)	Max Adv. Ratio
		DA		**Consensus**					
		Complexity	Download	Complexity					
Hotstuff [153]	✓	$O(nb)$	✓	$O(1)$	$O(1)$	4	$O(nb)$	$f+1$	0.33
HoneyBadger [112]	×	$O(nb+\lambda n^2 \log n)$	✓	$O(n^2 \log n)$	$O(1)$	$3+\log n$	$O(nb+n^2 \log n)$	1	0.33
DispersedLedger [151]	×	$O(b+n^2)$	×	$O(n^2 \log n)$	$O(n)$	$3+\log n$	$O(b+n^2 \log n)$	1	0.33
Prime [25]	✓	$O(nb+n^2)$	✓	$O(n^2)$	$O(1)$	6	$O(nb+n^2)$	1	0.33
Dumbo-2 [85]	×	$O(nb+n^2 \log n)$	✓	$O(n^2)$	$O(1)$	20	$O(nb+n^2 \log n)$	1	0.33
Dumbo-NG [82]	×	$O(nb)$	✓	$O(n^2 \log n)$	$O(1)$	9	$O(nb+n^2 \log n)$	1	0.33
VABA [20]	×	$O(nb)$	✓	$O(n)$	$O(1)$	13	$O(nb)$	1	0.33
Dag-Rider+AVID [95]	×	$O(nb+n^2 \log n)$	✓	$O(n^2 \log n)$	$O(1)$	12	$O(nb+n^2 \log n)$	1	0.33
Narwhal-Hotstuff [64]	✓	$O(nb+n^2)$	✓	$O(1)$	$O(1)$	6	$O(nb+n^2)$	1	0.33
Tusk [64]	×	$O(nb+n^2)$	✓	$O(n^2 \log n)$	$O(1)$	9	$O(nb+n^2 \log n)$	1	0.33
Bullshark [141]	×	$O(nb)$	✓	$O(n^2 \log n)$	$O(1)$	4	$O(nb+n^2 \log n)$	1	0.33
BigDipper-¼	✓	$O(b)$	×	$O(1)$	$O(n)$	4	$O(b)$	1	0.25
BigDipper-7	✓	$O(b+n)$	×	$O(1)$	$O(n)$	4	$O(b+n)$	1	0.33
BigDipper-Lite	✓	$O(b+\log n)$	×	$O(1)$	$O(n)$	4	$O(b+\log n)$	1	0.33

Table 4.1 summarizes the metrics for most recent and important BFT protocols in the past. The analysis of the table is provided in Appendix 4.11. At the high level, we can decompose any consensus protocol into data availability and consensus. Data availability

(DA) ensures that everyone has access to a common data in the unit of blocks, whereas consensus restricts the ordering among blocks. Recent DAG based consensus protocols decouples into consensus and a transport layer which satisfies DA. DA can be either achieved by requiring every replica to have the full data while reaching agreement, or achieved by applying erasure code on the data such that replicas do not have the full data after finishing the agreement. We add a sub-column to table, **Download**, to signify for each protocol. We separately display per replica complexity for DA and consensus.

If a protocol requires every replica to have the entire BFT block while making agreement, then maximal system throughput is upper bounded by each replica's bandwidth. So the system to replica throughput ratio is always $O(1)$. In contrast, with erasure code it is possible to reach agreement on the data without downloading it. Every replica's bandwidth is spent on distinct set of transactions, so the overall system throughput scales linearly as n increases. We note that there are types of applications that do not require full data for execution. For example, a sequencer [74] in the rollup architecture does not need to execute the transaction. But more generally, as roles are diversified in the infrastructure software, like consensus and execution separation in Ethereum. It is possible only a handy number heavy-duty nodes are providing the execution API. As long as, the result is provided with vaild proof, regular replicas does not need to have all transaction data to re-execute everything.

In Hotstuff, the worst inclusion distance is $f + 1$, because there can a sequence of f malicious leader. Narwhal achieves probabilitic censorship resistance with DAG. Prime [25] is an early leader based protocol with censorship resistance, but its is designed for defense against performance degradation. All leaderless BFT protocols has a distance of 1 given sufficient replicas have the transaction.

When computing the confirmation latency, the communication step is a fixed number. The variable part comes from the communication bit complexity, which is the sum of the DA and consensus complexity. Most leader based protocol has a small complexity on consensus, because there is a central role to coordinate communication among replicas to reach consensus. Most leaderless asynchronous BFT protocols have high complexity on both DA

and consensus, those also include DAG based protocols like Dag-Rider [95], Bullshark [141], Dumbo [82, 85].

BigDipper-¼ achieves constant overhead while achieving linear system capacity but it requires a stronger security assumption of $n \geq 4f+1$. BigDipper-7 restores the 1/3 assumption, but each replica incurs a linear overhead at the download path. At last, BigDipper-Lite is a modification based on BigDipper-7, that achieves all desired properties with latency of $O(\log n)$.

4.2 Background and Related Works

4.2.1 DA, VID and Erasure code

Data availability is implicit by all BFT protocols such that all replicas have access to a common ordered data at any time. Most leader based BFT protocol uses a simple DA by having the leader broadcast the data to everyone. There are more sophisticated designs of DA, including AVID [49], AVID-FP [88], AVID-M [151], SEMI-AVID-PR [116]. We compare our censorship resistant DA with them in Appendix 4.13.2. Verifiable Information dispersal(VID) is a technique that uses Reed-Solomon code to split data into $O(n)$ chunks, and each replica only stores a constant number of chunks to ensure the data is reconstructable. The verifiability ensures any two clients always retrieve identical data.

A Reed-Solomon(RS) erasure code is specified by a pair (k, h), where k is the number of systematic chunks, created from the input data, and the second parameter h is the coding redundancy. The newly generated data are called parity chunks. The ratio between k and h is called the coding ratio. The encoding process of RS code is based on the idea of polynomial interpolation [130], which is a linear operation. Given an input array, the encoding can be implemented so that the output is a concatenation of input data and parity chunks. We use $rsEncode, rsDecode$ to refer to the encoding and decoding operations. Appendix 4.12.3 contains more detailed introduction.

4.2.2 Polynomial Commitment and KZG

A polynomial commitment scheme [94] treats data as a polynomial and provides primitives including *commit*, *createWitness*, and *verifyEval*. The *commit* primitive provides a concise and unique representation of a polynomial. It has a Polynomial binding property such that it is computationally infeasible to find two polynomials that commit to the same commitment. With primitives *createWitness*, *verifyEval*, the scheme allows any party to reveal(open) evaluations at specific indices along with witnesses(proofs) without disclosing the entire polynomial. When presented with a correct witness, any verifier can use *verifyEval* and check that the evaluations are indeed correct against the commitment. The scheme has a Evaluation binding property such that it is computationally infeasible to find two witnesses for two different polynomial evaluations at the same evaluation index that passes the *verifyEval*. Kate-Zaverucha-Goldberg (KZG) [94] is a polynomial commitment scheme that is linear in commitments and witnesses.

KZG can be applied on a two dimensional matrix. Suppose we have n polynomials, each of degree $d-1$, and we want to encode it with a coding ratio of $\frac{1}{3}$. A matrix can be created where every column contains one polynomial. The RS encoding is applied to each of d row by extending the evaluations to $2n$ more points. The commitment of every newly generated $2n$ polynomials along the columns are the polynomial extension of the first k column commitments. It is due to the linearity property of KZG and RS [44]. A good reference can is available at [116]. Figure 4.2 provides a visualization for such 2D encoding. More discussion can be found in Appendix 4.12.3 and 4.12.5.

4.2.3 Combined Signature Scheme and CRHF

We use combined signature to refer to a signature generated from either a multi-signature or threshold signature scheme. We will use the exact name when a specific signature scheme is used. On paring friendly curves, a practical way to aggregate multiple signatures on a common message into one signature is to use BLS multi-signature [37]. The signature

scheme provides three secure primitive $\sigma_i \leftarrow$ *ms-sign*(sk_i, m); $\sigma, I_\sigma \leftarrow$ *ms-agg* $(pk_1, \sigma_1 \cdots pk, \sigma_t)$; *bool* $\leftarrow$ *ms-verify*(I_σ, m, σ), where I_σ is an indicator vector for the signers, which has $O(n)$ size. In contrast to multi-signature, a threshold signature show out of n signers at least some ratio number of signers have signed a common message, and it requires only constant size as opposed to $O(n)$ size. To convert a list of data into a single message, we assume a collision resistant hash function($CRHF$) with negligible probability for two different data to hash into the same digest. A combined signature is valid if it contains $n - f$ signatures.

4.2.4 Security Assumptions for BigDipper

The network is assumed to be partial asynchronous. In this networks, there is a notion of an unknown Global Stabilization Time (GST), after which all transactions can arrive within a known Δ duration. The security assumption is $n \geq 3f + 1$. all malicious replicas can have arbitrary network latency. The malicious replicas can be separate or coordinated by a single adversary. Transaction censoring occurs when malicious replicas (including the leader) exclude some transactions on purpose (it is different from the situation when an honest replica is impatient of waiting and miss transactions,it is not on purpose).

4.3 Censorship Resistant DA

4.3.1 Primitives

BigDipper is built on three protocol components for providing short term censorship resistance in leader based BFT protocols. The first component delivers clients' transactions to BFT replicas; DA-CR ensures the censorship resistance; the last component integrates the DA-CR with a standard BFT protocol, like Hotstuff. The DA-CR is a critical for BigDipper BFT system as its properties determine the degree of censorship resistance. In this section, we outline the interfaces and their properties.

The DA-CR provides a Disperse invocation that distributes transactions data among replicas, and a Retrieve invocation to safely retrieve the data from honest replicas. For

censorship resistance, we require the Disperse invocation be initiated in a distributed manner from at least $n - f$ replicas, and the final dispersal data B assembled by the leader must also contain at least $n - f$ mini-blocks. For the Retrieve invocation, either replicas or clients can invoke the procedure to retrieve partial or full dispersed data.

Every invocation is initiated against a DA-CR instance with a unique ID, which is used by replicas and clients to identify the correct context for intended invocations. The Disperse invocation is associated with two events, a Start event and a Finish event. The Finish event evaluates to the following types: Complete or Incomplete. Since a malicious leader can stall the protocol by inactivity, the Incomplete type serves as a trigger for the leaders replacement. If a Disperse invocation finishes with the type Complete, it will return a commitment C and its associated combined signature from $n-f$ out of n replicas. It's possible that a Completed Disperse invocation may not contain mini-blocks from all n replicas. This is because either some honest replicas are late for the leader's collection time, or some malicious replicas refusing to participate. For the missing parts, a protocol designer can meditate on the trade-offs and decide if to allow the leader to fill in its own data. We elaborate on this trade-off in Appendix 4.18.

4.3.2 Properties of DA-CR protocols

A leader based DA-CR protocol that offers censorship resistance must provide the following properties for all DA-CR instances. The DA-CR protocol relies on partial synchronous assumption, which is different from asynchronous protocols like AVID [49], AVID-M [151].

- **Termination**: If some replicas invoke Disperse(B), then those replicas eventually Finish the dispersal. If at least $n-f$ replicas invoke Disperse(B) and the leader is honest with a network after GST, then the dispersal Finishes with type Complete.

- **Availability**: If an honest replica Finished a dispersal with type Complete, then any honest retriever can invoke Retrieve and eventually reconstruct a block B'.

- **Correctness**: If an honest replica Finished a dispersal with a type Complete, then any honest retriever always retrieves the same block B' by invoking Retrieve. If the leader is honest, then $B = B'$, where B contains all the mini-blocks available to the honest leader at dispersal.

- **Commitment Binding**: If an honest replica Finished a Disperse(B) invocation with a type Complete and a valid combined signature on its commitment, no adversary can find another different B' that verifies correctly against the combined signature.

- **Inclusion-t**: If an honest replica Finished a Disperse(B) invocation with type Complete, the dispersed data must include at least t mini-blocks from honest replicas.

- **Space-Capturing**-η: If an honest replica Finished a Disperse(B) invocation with type Complete, the dispersed data contains at most η captured mini-blocks by the leader.

- **Data-tampering Resistance**-ρ: In an accountable inclusion mechanism, if an honest replica Finished a Disperse(B) invocation with type Complete, the dispersed data B must ensure at least ρ mini-blocks are not tampered in the DA-CR instance. If $\rho = n$, protocol guarantees no data can be tampered.

The last three properties are new for delineating the censorship resistance. As mentioned in Section 4.1, (t, η, ρ) characterize the properties of censorship resistance. We will use an important concept called **Attestation** later. In short, it is a signature on a mini-block by a replica to declare the replica intends to include the mini-block. We refer the readers to Appendix 4.13.1 for formal definitions of **Space-capturing**, **Accountable mechanism** shown in the properties.

Next, we describe three DA-CR protocols, Table 4.2 summarizes their properties. For ease of reference, we use the suffix of each protocol combining with CARD[1] as the name for

[1]CARD stands for **c**ensorship **r**esistance **d**ata **a**vailability with flexible ordering.

DA-CR component. VANILLA is a protocol that takes minimal changes from a conventional leader based BFT protocol to a censorship resistant one,it is presented in Appendix 4.16.1.

Table 4.2: Comparison of DA-CR component of BFT protocols

BIGDIPPER protocol	DA-CR protocol	BFT ($n \geq$) assumption	ρ	t	η	message complexity	hyperscale throughout
BIGDIPPER-7	CARD-7	$3f+1$	n	$n-2f$	0†	$O(b+\lambda n)$	✓
BIGDIPPER-¼	CARD-¼	$4f+1$	-	$n-3f$	$n-f-1$	$O(b+\lambda)$	✓
BIGDIPPER-LITE	CARD-LITE	$3f+1$	n	$n-2f-o(f/\log(f))$	$o(f/\log f)$†	$O(b+\lambda\kappa\log n)$	✓

λ is a security parameter with fixed size, which uniquely represents the data block. κ is a small constant. † means the parameter η can increase up to $\eta = f$, see Appendix 4.18.

4.4 DA-CR Protocol Designs

We begin by presenting a DA-CR design called CARD-7 which has properties of ($\rho = n, t = n-2f, \eta = 0$). We present its variant CARD-¼ in Appendix 4.15. We provide an outline for CARD-LITE that achieves $O(\log n)$ complexity at the end.

4.4.1 CARD-7 *Protocol*

At a high level, the CARD-7 protocol relies on the Reed-Solomon code, a 2D KZG and an aggregate multi-signature scheme to support the invocations with mentioned properties.

It is designed to be an accountable inclusion mechanism, so that every replica know whose mini-blocks get included. The formal protocol is given in Algorithm 2 from Appendix 4.14.1. In the following sections, we present an outline of both Disperse and Retrieve invocations in sequence, and proof is given in Appendix 4.14.5.

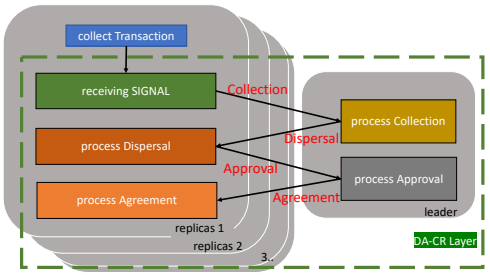

Figure 4.1: Flow diagram for a DA-CR protocol

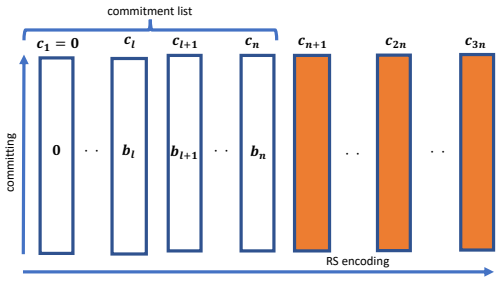

Figure 4.2: A 2D matrix data structure used by the leader for producing $3n$ commitment. Replica 1 is excluded, as its data is 0. b denotes mini-block(systematic chunk), orange are for parity chunks.

Disperse Invocation

The Disperse invocation for a DA-CR instance involves two round trips between replicas and the leader, and its workflow is shown in Figure 4.1. For ease of reference, we name the four communication messages as collection, dispersal, approval and agreement in sequence. In the beginning of every BFT round, there is special message called Signal which is received from a pacemaker inside a BFT protocol. More details is given in Section 4.14.1.

When replicas initiate the Disperse invocations distributedly for some DA-CR instance, each one of them generates its own mini-block, and a corresponding **attestation** by signing the kzg commitment of the mini-block. A **collection message** is sent by each replica to transfer (attestation, mini-block) to a leader. Because at most f malicious replicas could simply stop responding, the leader waits for receiving at least $n - f$ mini-blocks and the corresponding attestations. We refer the collected attestations as the *attestation set* whose cardinality is m equal to the number of received mini-blocks.

When the leader is processing the collection messages shown as the yellow box of Fig-

ure 4.1, data is encoded with Reed-Solomon code. Figure 4.2 provides a visualization of the data structure. The received mini-blocks are laid out as columns in a matrix, and parity chunks are generated by interpolating additional values in each row on the matrix. The first n columns of the matrix are block space designated for mini-blocks from n replicas, we refer their corresponding commitments as the *commitment list*. If a replica i is excluded (when $m < n$), the corresponding i-th column contains identically zero data and its commitment is 0; otherwise the column is filled with data from the received mini-block.

For every row, the coding ratio used by the erasure code is $\frac{1}{3}$, which is equivalent to say two parity chunks (columns) are generated per mini-block. Appendix 4.14.2 discuss the calculation to arrive at the $\frac{1}{3}$ ratio. After encoding, the leader prepares a **dispersal message** for each of n replicas. Every message contains identical *commitment list* of size n and an *attestation set* of size m, but also contains two parity chunks unique for every replica. This ensures every chunk is stored by only one replica, which is essential for safe retrieval.

When receiving a dispersal message, a replica has several verification procedures, shown in Algorithm 2 (line 10-12). They verify the leader has included at least $n - 2f$ honest mini-blocks, and parity chunks are generated correctly and the leader has not tampered any data. If all verifications pass, a replica computes a final commitment C with a collision resistant hash function ($CRHF$) on the *commitment list*. The final commitment C is used as the signing message for approving the dispersal by the multi-signature scheme (4.2.3). After that, a replica sends a **approval message** containing the signature to the leader. An honest replica only signs once for every unique DA-CR instance, and it is strictly increasing as the BFT view number.

In Figure 4.1, after receiving approval messages, a leader waits to accumulate $n-f$ multi-signature before proceeding. Although f out of them might be malicious, the remaining $f+1$ approval from honest replicas is sufficient to ensure the data availability since $3(f+1) > n$. With sufficient approval, the leader generates an aggregate signature and signer bitmap (Section4.2.3), and sends the identical **agreement message** to all n replicas. When a replica receives a valid aggregate signature with sufficient signers, the corresponding DA-CR

instance is considered Finished with a Complete type. Otherwise after timeout, a replica can conclude that the DA-CR instance Finishes with a Incomplete type.

Retrieve Invocation

The retrieval algorithm is formally defined in Algorithm 4 from Appendix 4.14.3. The **Retrieve** invocation guarantees any parts from dispersed data in any DA-CR instance are retrievable by anyone. If the interested data is one mini-block, a Retrieve invocation is first sent to the leader and the replica who possess the data; upon responded with data, the retriever checks correctness by computing polynomial commitment on the data and comparing with the *commitment list*. If both deny the request, a replica queries and finds $f + 1$ other honest replicas which responds correct chunks consistent to the commitments. Because all replicas have distinct chunks, the retriever can perform *rsDecode* to get the original dispersed data, which contains all mini-blocks. For partial retrieval from a mini-block b_p, the invocation is similar to full mini-block. More importantly, if both deny, the retriever can query partial data and does partial reconstruction without needs to construct the entire dispersed data. Because each replica possesses three chunks, a retriever needs only contact roughly $\frac{4f}{5}$ random replicas to reconstruct the data with high probability. Appendix 4.14.4 presents analysis and figures.

4.4.2 Card-Lite *Protocol*

Card-Lite is an improved protocol based on Card-7 that only requires $O(\log n)$ communication overhead. But it has slightly weaker censorship resistance, as shown in Table 4.2. We outlines the intuition behind the protocol and provide the complete protocol and proofs in Appendix 4.17.

How to differentiate two types of leaders is the key problem for any honest replica when asked to approve a dispersal. In the context of censorship resistance, an honest leader is required to include at least $2f + 1$ mini-blocks, whereas the censoring leader includes less than $2f + 1$ mini-blocks. It is easy to detect the malicious leader in Card-7, because

everyone receives the entire *commitment list*. But we want each replica receives only $O(\log n)$ commitments and attestations.

The core idea is to distinguish the two types of leader through random sampling. If a leader does not include sufficient mini-blocks in its commitment, there will be some random samples which the leader is unable to answer. If the leader is unable to answer too many of them, the leader would not be able to collect $2f + 1$ signatures to make progress.

The idea assumes that the leader cannot lie about the relation between responses and its commitment. To achieve that while keeping a low complexity, we require the leader to use a doubly homomorphic commitment scheme [43] on the *commitment list* (see Appendix 4.17.1 for background). So in the protocol, a sample for chunk i would contain: attestation from replica i, commitment c_i, final commitment C, a kzg binding proof for c_i with respect to C). Because the proof is aggregatable and has a size of $O(\log n)$, the overall complexity per sample is just $O(\log n)$. A random query is success if the leader can provide the valid response, and failure otherwise.

Suppose a leader has constructed a 2D KZG matrix and receives a random query, clearly the honest leader type has higher chance of being able to response as opposed to the malicious type. Then if a batch of k random queries are requested, the probability of having all k success query is exponentially decreasing for both types. But if we independently retry (boost) L number of batches, eventually the probability of having at least one successful batch would be sufficiently high. A key observation is that the honest type requires much less boosting, the exact difference depends on the number of mini-block s which the malicious type censors. As shown in Appendix 4.17, if the leader is censoring $s = \Omega(f/\log f)$ mini-blocks, and all honest replicas sample $L = o(n)$ each, the extremely high probability the malicious type is exposed, but the honest type can always pass the query in all cases.

If done naively, every replica need to send kL number of random queries, which is much larger than $O(\log n)$. But with Fiat-Shamir transform on the final commitment, we can make the process non-interactive. The leader deterministically generates random queries and sends back only one successful batch to the replica to prove non-censorship. Upon receiving a batch,

a replica can independently verify the tuple and correctness of the randomness generation. The full analysis are presented in Appendix 4.17.

4.5 Transaction Broadcast

In the new architecture, all replicas can add transaction to the final block, the client now has more options to choose which replicas to send to. We provide a transaction broadcast mechanism which clients can use to achieve a desired amount of censorship resistance.

4.5.1 Navigator *Protocol*

The interface to the protocol is a submit function call, shown in Algorithm 5 from Appendix 4.19.2. To send a transaction, a client provides two inputs: the transaction and number of copies x. The algorithm will send the transaction to x random replicas available for the view number. Replicas continuously wait for transactions to arrive, and include them at best effort.

The difficulty with the interface is to decide how many copies to send for achieving some desired probability for inclusion in the next block. To do that, we create Table 4.3 based on the worst case assumption that all f malicious replicas are censoring transactions all the time. Depending on the type of leader and number of included honest mini-block q, we can compute the probability of inclusion as a function of x. However, since both the leader type and the number q is unknown to the client, we have to model the leader with a bernoulli variable with $p = 2/3$ for each possible leader, and a uniform variable for all possible q. Later, as the clients gets more data, it is possible to update those prior based on Bayesian method. The lower bound for q is determined by the Inclusion-t property offered by the DA-CR. When a leader is honest, q has a range from t to $n-f$. On the other extreme, when the leader is malicious, there can be at most t collected honest mini-blocks. At runtime, the actual number of q would fall anywhere between t and $n-f$. In the Table, if a client sends $x \ll f$ copies, determined malicious replicas can censor it with certainty. However, in Appendix 4.19.6, we show that although a malicious leader can censor some clients who

sends only a few copies, the probability to censor u different transactions altogether decreases exponentially as u increases. The rest of the section delineates how to compute the Table.

Table 4.3: Probability of inclusion as a function of number of copies and number of honest inclusion

Leader type	Malicious	Honest	Honest
number honest mini-block	$q = t$	$q = t$	$q = n - f$
$n - t + 1 \leq x \leq n$	$\Pr(IN) = 1$	$\Pr(IN) = 1$	$\Pr(IN) = 1$
$f + 1 \leq x < n - t + 1$	see 4.5.2	see 4.5.2	$\Pr(IN) = 1$
$0 < x < f + 1$	$Pr(IN) = 0$	see 4.5.2	see 4.5.2

The cases with probability 1 can be proved by the pigeonhole principle. For instance, suppose $x = n - t$ and $q = t + 1$, then there must be at least one mini-block from an honest replica which includes the transaction, and similarly for any pairs with $x + q \geq n + 1$. On the other extreme, when a client sends less than $x \leq f$ transactions to replicas and if a malicious leader has intent of censoring the transaction, the adversary can selectively including those $f + 1$ honest mini-blocks without the transaction.

4.5.2 Probabilistic Transaction Inclusion

Probability of Inclusion with an Honest Leader

Given the leader is honest and the client chooses x replicas randomly without replacement, let X be a random variable indicating the number of honest replicas that have the client's transaction. For $0 < i \leq \min(x, n - f)$, the probability of the transaction being included by $X = i$ honest replicas is $\Pr(X = i; x) = \binom{n-f}{i}\binom{f}{x-i} / \binom{n}{x}$. Suppose f malicious mini-blocks are collected by the honest leader, the probability is

$$\Pr(IN;\, x,q) = 1 - \sum_{i=1}^{x} \Pr(None \mid X = i; q) \Pr(X = i; x) \tag{1}$$

where $\Pr(None\ |X;q)$ is the probability none of X honest mini-block containing the client's transaction. Suppose all honest replicas have independent and identically distributed network latency, $\Pr(None\ |X=i;q) = \prod_{j=0}^{q-1} \frac{n-f-i-j}{n-f-j}$, and the probability of exclusion decreases exponentially as q increases. We provide an empirical evaluation in Appendix 4.19.4 and an table of probability in Figure 4.7.

Probability of Inclusion with a Malicious Leader

When a leader is malicious, but a client sends $x \geq f+1$ copies of a transaction, the probability of inclusion equals to $\Pr(IN) = \sum_{i=f+1}^{x}(\binom{n-f}{i}\binom{f}{x-i}/\binom{n}{x})$, where $f+1 \leq x < n-t+1$. Further analysis is provided in Appendix 4.19.5.

4.6 BigDipper *with BFT Integration*

4.6.1 Overview

BigDipper is a system that provides censorship resistance to leader based BFT protocols [42, 51, 153]. After transactions are included in the DA-CR, a leader is required to take all transactions into its next block. By Commitment binding and Correctness property of DA-CR, every replica can arrive at an identical order. In the following, we specify the Scooping algorithm which would be inserted into the consensus path in order to check correct inclusion and DA. Then we demonstrate an integration with the two phase Hotstuff-2 [107] protocol which achieves all the properties in Appendix 4.10.1.

4.6.2 Scooping *Algorithm*

The Scooping algorithm is simply a procedure called verifyCard. It performs the operation to verify if the combined signature returned by the DA-CR in **Agreement** is correct. This procedure is inserted into the BFT consensus path such that it would not proceed unless the replica can looking up its local storage to find a valid combined signature. The algorithm is defined in Appendix 4.22.1.

Algorithm 1 Hotstuff-2 adaptation

1: **Modification to Enter(1)**
2: **As a leader**, no modification. if entering the view with double certificate, $(C_{v-1}(C_{v-1}(B_{k-1})))$, then proceed directly to Propose(2); otherwise the leader sets a timer of 3Δ then proceeds to the Propose(2)
3: **As a replica**, if entering a view with a double certificate, then proceed directly to Vote-and-commit(3); otherwise the replica sends its locked certificate and the corresponding valid combined signature of DA-CR to the leader along with **Collection** message prepared from Alg.CARD-7 line 3- 8
4: **Modification to Propose(2)**
5: The leader instead of include the BFT block, it performs Alg.CARD-7 line 22- 26 to encode the collected mini-blocks, then broadcast the Propose and **Dispersal** message together. Any certified block must come with a valid combined signature, otherwise it must not be used for extending the next block.
6: **Modification to Vote and commit(3)**
7: Before proceeding any execution, the replica performs Alg.CARD-7 line 10- 17 to checks the dispersal correctness and send **Approval** message, then proceeds to Hotstuff-2
8: **Modification to Prepare(4)**
9: The leader runs Alg.CARD-7 line 27- 30 and send **Agreement** message, then proceeds to Hotstuff-2
10: **Modification to Vote2(5)**
11: Replicas performs procedures from Alg.Scooping 2- 3 before locking the BFT block. If succeed, send along with the vote2 message a **Collection** message prepared from Alg.CARD-7 line 3- 8.

4.6.3 BIGDIPPER-7 *with Hotstuff-2*

DA-CR protocol can integrate with any leader based BFT that uses lock-commit paradigm [107]. When some replicas commit a BFT block, the lock-commit paradigm ensures safety by having at least $n-f$ replicas to lock the committed value, such that no other value can be committed at the same block height. HotStuff [107,153], Tendermint [42], PBFT [51] all use the paradigm. They differ from how liveness property is achieved when the leader is honest. If a leader is malicious, it cannot violate the safety due to the lock, and it would be replaced if violating liveness. We provide an integration of CARD-7 with Hotstuff-2, which has great properties including its simplicity, optimal responsiveness and the two phase latency.

The integrated protocol is defined in Algorithm 1, which highlights the additional procedures on top of the Hotstuff-2 protocol. We provide a brief terminology of Hotstuff-2 protocol in the following; a certified block B at view v, whose certificate is denoted as $C_v(B)$, is a block with $n-f$ combined signature, certified blocks can be ranked by height; a certificate is locked if the replicas is guarding its block; a BFT block has a double certificate if it has been voted by $n-f$ replicas in two phases, and is safe to commit. To help understand the protocol integration, we create a flow diagram in Figure 4.10 in Appendix 4.23.

In the new algorithm, different from the original Hotstuff-2, a leader does not include the entire BFT block in the Propose(2) message. Instead it uses information dispersal and runs the encoding parts of the CARD-7 protocol line 22- 26. The data availability is confirmed in exactly one round trip later before locking any certificate. CARD-7 does not affect Hotstuff-2 safety because the data availability is checked before locking. The abstract signal mentioned in Section 4.14.1 can be implemented by the Hotstuff-2 pacemaker [107]. The complete proof for **Safety** and **Liveness** are presented in Appendix 4.24.

4.7 Conclusion

BIGDIPPER is a hyperscale BFT system with short term censorship resistance for leader based protocols. It is based on the idea a DA layer can the decentralize transaction inclusion and clients can choose amount of censorship resistance.

4.8 Shard *algorithm*

A simple protocol, called SHARD, with the new architecture can achieve the censorship resistance property with sharding and cryptographic signatures. At a high level, a client sends a transaction to a shard for inclusion. If a sufficient number of replicas in a shard receive the transaction, SHARD guarantees the transaction is included in the final BFT block. Since there are malicious replicas in any shard, a client should send multiple transactions to a shard's replicas to ensure that some honest replicas has received it. All replicas then send batched transactions (mini-block) to leader for inclusion. However, the leader might be malicious and exclude mini-blocks that include a certain transaction. To prevent that, the leader has to include a threshold of mini-blocks from replicas in a shard, which can be enforced by requiring a threshold of digital signatures from members of the shard.

The shard size should be sufficiently large and rotated frequently to ensure that each shard has a sufficient amount of honest replicas to prevent possible collusion (in the most secure case, all replicas are in a single shard). Then the collected mini-blocks can be aggregated in the DA running among all replicas. In the end, the leader has to include all the data from

the DA in the final BFT block. If the leader censors a transaction from the DA, anyone can prove to the BFT protocol of such censorship by providing a proof which indicates the missing inclusion of the transactions in the final block. The challenger can access the unordered transactions by querying the DA. The SHARD protocol assumes that there is a random beacon such that any client can calculate the membership of each shard, and use it to send transactions to the replicas.

However, such SHARD protocol relies on a random beacon assumption and has to calculate a delicate parameter, the shard size, to balance the trade-off between communication efficiency and security against dynamical adversaries and leader-shard collusion. A large shard size causes a client to send more copies of the transaction; whereas a small shard lowers the probability that there are sufficient honest replicas within a shard. In the event that malicious replicas occupy a significant portion of the shard such that the honest replicas cannot have sufficient signature to proceed, the protocol has to decide whether to allow the leader to proceed when the malicious shard refuses to participate in the protocol. If we don't allow the leader to proceed, the BFT protocol has a liveness attack when a shard get dynamically corrupted; if we allow the leader to proceed, then a malicious leader can censor any arbitrary shard to exclude all transactions. It is because we cannot distinguish between the case when a leader is malicious or a majority of a shard is malicious. Both of them can unilaterally censor a mini-block. As the result, we cannot detect censorship when a leader is allowed to proceed with all transactions missing from a shard, and there is no forensics when censorship happens. But since the replica allocation to shards is publicly accessible in order for clients to submit their transactions, the dynamic adversary can acquire this information and slowly corrupt the shard. To lower the probability the above liveness and censorship attack from happening, the protocol has to rotate the shard very frequently to prevent the worst case liveness attack. But shard rotation requires intricate handoff mechanism that increases the protocol's complexity and decreases the system's throughput capacity.

4.9 Proof of Stake BFT Leader Assumption

In the production leader based proof of stake blockchain, like Tendermint [42], a replicas' chance to become a leader is weighted proportional to replicas' voting power. There is a natural implication from this model to our assumption that leader is much capable such that it has high bandwidth and computing facility. Essentially, the leader with large percentage of stake in the partial synchronous setup already put down large capital investment in the form of stake, but the hardware requirement is relatively cheap. For example, a powerful instance m7g.16xlarge [30] on AWS EC2 with 64 vCPU, 256 GB memory and 30 *Gbps* optimized bandwidth requires only $\$2.6112/hour$ in a long term plan.

On the other hand, the censorship resistant property in BigDipper does not give the leader much higher potential to manipulate the transactions inclusion. Therefore from the protocol's view, it is fine to have a few replicas with large amount of stake, and many replicas with less stake and regular hardware, provided that there is one powerful leader to drive liveness.

A more uniform stake distribution is desirable, in which scenario, replicas with insufficient hardware can choose third party services to temporarily serve the leader's role. Or the protocol has to downgrade the performance for a brief time proportionally to the replicas' stake.

To avoid the problem of downtime or third-party trust relation, the leader selection part of BFT protocol can have an opt-in mechanism, such that only interested parties can volunteer to becomes the leader. If some leaders are persistently censoring, eventually replicas from community can opt-in to improve the honest ratio in the leader set.

4.10 BigDipper *Properties*

4.10.1 BigDipper *Properties*

BigDipper is a BFT system that provides consensus on a string of bits. It requires a transaction broadcast protocol to accept client transactions, a DA-CR for guaranteeing transaction

availability and inclusion, and finally an integration with a BFT protocol. BigDipper relies on an execution engine like the Ethereum Virtual Machine to handle transaction validity. BigDipper satisfies the following additional properties besides safety and liveness of the BFT protocols. Suppose there are n total replicas, and f of them are malicious. In the property description, a replica is available if it can include any transaction before sending its mini-block:

- **Strict Transaction Inclusion**: If a DA-CR guarantees at least t mini-block inclusions from honest replicas, an honest client can send $n-t+1$ copies of a transaction to distinct available replicas for guaranteed inclusion regardless of whether the leader is malicious.

- **Probabilistic Transaction Inclusion Under Honest Leader**: Under an honest leader, if a DA-CR guarantees at least t mini-block inclusions from honest replicas, an honest client can send x copies of a transaction to random distinct available replicas for probabilistic inclusion, the probability monotonically increases with x and t. If network latency of honest replicas are i.i.d, the probability increases exponentially with x and t.

- **Probabilistic Transaction Inclusion Under Malicious Leader**: Under a malicious leader, with a DA-CR instance with t guaranteed honest mini-block inclusions, an honest client can send x copies of a transaction to random distinct available replicas for probabilistic inclusion, the probability monotonically increases with x, up to 1 when $x = n - t + 1$. When $x \leq f$ the malicious leader can censor one transaction with probability 1, but to censor u different transactions each of x copies altogether, the probability of censoring decreases exponentially as u increases.

- **2 Δ Probabilistic Confirmation for Clients**: If a DA-CR guarantees at least t mini-block inclusions from honest replicas, an honest client can send x copies of a transaction and compute the probability of inclusion right after 2 Δ network latency by using number of responses from x replicas and belief on the leader's type.

- **Hyperscale Throughput**: The throughput of reaching consensus on a sequence of bits is a constant ratio to system capacity, which scales linearly as the number of replicas joining in the system.

4.11 BFT protocol complexity analysis

4.11.1 Prime

Prime [25] is a classic leader based protocol whose goal is to defend against adversarial attack on performance degradation while ensuring basic safety and liveness of a BFT protocol. The malicious replicas can perform arbitrary operations but correctly enough for not being identified as malicious. The paper defines a new correctness criteria that takes performance into account for defending against performance degradation attack. For example, a malicious leader can introduce latency in the consensus critical path to delay the rate of confirmation. The paper relies two insights, most of procedures in the protocol does not require malicious replicas to take action; second, if the workload for a leader has a predictable bound, all peers can observe the network and access the leader's performance to decide if the leader should be replaced. To achieve them, the protocol devises multiple sub-protocols, each responsible for a specific tasks. When a replica receives a order (transaction), it disseminate it to all other replicas. The pre-ordering sub-protocol is responsible for confirming the data availability. A data structure called **PO-SUMMARY** is invented to record from each replica's view, how much data the other replicas currently have. Essentially, **PO-SUMMARY** is a vector recording cumulative sum about what other replicas have. Then each replica broadcast **PO-SUMMARY** to everyone including the leader. The exchange about **PO-SUMMARY** requires a $O(n^2)$ communication, where n is the number of replicas. The leader then uses all **PO-SUMMARY** to create a global ordering, if a transaction has been recorded by at least $2f+1$ replicas. BigDipper is different from Prime although both are leader based protocols, because in BigDipper there is never a all-to-all communication pattern, and in addition, the inclusion of transaction does not require consensus of $2f + 1$ to receive the transaction.

Although Prime uses erasure code to reconciliate data availability, the erasure coding is done to per transaction basis. The regular Ethereum transaction size if roughly 200Byte, then if the network has size of equal size, the overhead per transaction is outweighting the benefits of erasure coding. Whereas in BigDipper, the erasure code is done all transactions batch received by a BFT replica. Applying erasure code to the batch does not introduce too much overhead for indicating which indices its points are evaluating on. At latency perspective, Prime extends 2-Phase PBFT by 1.5 phase (round-trip) trips message, which are **PO-REQUEST**, **PO-ACK**, **PO-SUMMARY**, whereas BigDipper perfectly fits into the 2-Phase hotstuff protocol.

4.11.2 HoneyBadger and DispersedLedger

HoneyBadger is a leaderless asynchronous BFT protocol, made of two protocols: a reliable broadcast protocol (RBC) and an asynchronous binary agreement protocol. To reach consensus, HoneyBadger runs n reliable broadcast protocol asynchronously. The reliable broadcast is modified based on AVID [49], where every replica produces its own merkle tree, and broadcast the merkle leaf and proof to all other replicas. The total communication complexity for one replica to broadcast its data of size b bytes is $O(nb+\lambda n^2 \log n)$; whereas using the classic Bracha [40] reliable broadcast would take $O(n^2 b)$. Whenever a replica completes a reliable broadcast j, it starts to contribute to the binary agreement protocol for j. An asynchronous binary agreement protocol accepts inputs from all replicas. Since the binary agreement(BA) protocols based on construction from [113] requires $O(n^2)$ communication complexity per round, and it takes $O(k)$ rounds to complete with probability $1-2^{-k}$. The overall complexity for BA is $O(n^2 \log k)$. To ensure all n instances of BA to terminate with high probability, the expected number of round is $\log n$. The overall communication complexity for all n replicas to complete a round is $O(n^2 b+\lambda n^3 \log n + n^3 \log n) = O(n^2 b+\lambda n^3 \log n)$, where b is inputs from each replicas. The number of communication steps to complete a honeybadger RBC (which is modified based on AVID) is 3. The first step sends merkle leaf to everyone, the second step sends echo among all replicas, and the final step for sending ready messages.

DispersedLedger has a very similar structure as HoneyBadger. DispersedLedger divides the protocols into two parts: a data availability(DA) protocol and a binary agreement protocol. The data availability is different from reliable broadcast protocol in HoneyBadger, because data availability can be achieved by the Disperse invocation alone, and its retrieval part can be deferred in the later stage. The construction of DA in DispersedLedger is called AVID-M, which has very similar structure as the merkle construction in reliable broadcast in HoneyBadger and AVID-H in AVID [49]. In DispersedLedger, the communication complexity for each replica is $O(b+\lambda n^2)$, the binary agreement part is identical to HoneyBadger which takes $O(n^2 \log k)$ per replicas, that terminates with probability $1-2^{-k}$. So together the total communication complexity is $O(bn+\lambda n^2+n^3 \log n)$. The number of steps for DA is 3, the sender replica first sends the merkle leaf and proof, the each replica sends **GotChunk** that contains only the merkle root, this is the step that differs significantly from AVID-modified RBC from HoneyBadger; the last step is to send **ready** message. Although both DispersedLedger and HoneyBadger has censorship resistance property, it is implemeneted by a all-to-all communication pattern. Whereas in BigDipper, there is an active coordinating leader preventing the quadratic communication pattern.

4.11.3 VABA

Validated Asynchronous byzantine Agreement is the first protocol that achieves multivalue byzantine agreement with $O(n^2)$ complexity. Previous method by [47] of MVBA requires $O(n^3)$ communication complexity. The key construction element is called provable broadcast, which is also a key building block for conventional leader-based BFT protocol like [42,51,153], where the leader has an active role to coordinate replicas. For VABA, each Atomic broadcast instance is associated with an ID; within each ID, four consecutive PB are invoked to ensure data is committed without worries of hidden lock. All n replicas performs 4-stages PB, and a leader is chosen retrospectively after all replicas have abandoned their PB procedures. If the selected leader prematurely terminates the 4 stage PB, a view change repeats the 4 stage PB until an externally valid value is confirmed by all honest replicas. BigDipper is a leader

based BFT protocol, that avoids all-to-all communication pattern, unlike VABA.

4.11.4 Dumbo, sDumbo-DL and Dumbo-NG

Dumbo is BFT protocol built upon the framework presented from HoneyBadger. There are two versions of the protocol: Dumbo1, and Dumbo2. The Dumbo1 identified n parallel asynchronous Binary Agreement protocol as the bottleneck to the system throughput. Then it provides an optimization to replace n Asynchronous Byzatine Agreement(ABA) instance. The technique is to replace n ABA with a $\log n$ reliable broadcast(RBC) to ensure one honest replica can provide a batch of transactions from $n - f$ replicas. Then all n nodes runs $\log n$ ABA to agree on which one of the $\log n$ Index from RBC are correct, and the data is received by them. In total the step which Dumbo1 takes to finish the protocol is $2 + 2 + \log\log n$ for Value-RBC, Index-RBC and ABA.

Dumbo2 is a further optimized version of Dumbo1. Instead of using n Value-RBC, $\log n$ Index-RBC followed by $\log n$ ABA, Dumbo2 replaced the reliable broadcast part with a new protocol called provable RBC, that uses threshold signature to ensure the data broadcast by Index-RBC are indeed correct. The agreement part is replaced with only one Multivalue Byzantine Agreement (MVBA).

4.11.5 DAG-Rider, Narwhal-Hotstuff and Tusk

DAG-Rider, Narwhal-Hotstuff and Tusk are consensus protocols that adopts a design that separate mempool and consensus, where the mempool is implemented as Direct acyclic graph. DAG-rider is an asynchronous protocol that arrives at consensus every four rounds, where each round consists of a reliable data broadcast (RBC) and is not a point-to-point communication. When a RBC is implemented by AVID, the communication complexity is $O(nb + \lambda n^2 \log n)$. Because each RBC takes 3 communication steps (point to point communication), the overall steps it takes to confirm a block is 12. Because every vertex must contain n meta-data of size $\log n$, the broadcast would consume $O(n^2 \log n)$ bits for each replica.

Narwhal is a mempool module that is dedicated to produce data protocol that has properties of Integrity, block-availability, containment, 2/3-causality and 1/2-chain quality. The data broadcast is achieved with provable broadcast by using certificate from $n - f$ replicas' signature. Narwhal-Hotstuff-2 is a partial synchronous protocol that uses Hotstuff-2 for reaching consensus. In the table, we use the latest Hotstuff-2 as the consensus module to reduce latency. In the Narwhal-Tusk paper, the communication steps is 8, because it used the old 3 phases Hotstuff protocol.

Tusk is an asynchronous protocol that is an implementable version of DAG-Rider, and it includes optimization to reduce the expected communication steps to 9 as indicated in the paper [64].

4.12 Background Appendix

4.12.1 Validity requirement in BFT problem

Often external validity [47] is added to the BFT requirement, which requires a polynomial-time computable predict for deciding if a transaction is valid. BigDipper uses standard Virtual machine to handle transaction validity, the protocol by itself serves to reach consensus on a sequence of bits.

4.12.2 DA using outsourced components

There are many ways to incorporate [21,138] a DA into a BFT protocol, either by borrowing external DA oracle or internalizing it with the BFT replicas. The advantage of the second option is a single unified trust assumption. The advantage of former options usually come with high throughput and simpler application design logic due to modularity.

4.12.3 Reed Solomon Erasure coding

The encoding process of Reed-Solomon code is based on the idea of polynomial interpolation [130]. Essentially, given a list of n points, there is a unique polynomial of degree

$n-1$ that interpolates those points. The redundancy generation is basically evaluating the polynomial at more points. Where those new points locate depends on specific encoding scheme. However, one encoding scheme that uses Fast Fourier Transform is particularly popular due to its fast $O(n \log n)$ computation complexity. It only requires the field which evaluations indices belong to have root of unity, ω, such that $\omega^z = 1$, where z is power of 2. Given some data D, the exact operation can be summarized as $IFFT(concate(FFT(D), \widehat{0}))$, where $IFFT$ is inverse FFT, and the length of $\widehat{0}$ depends on size of redundency. Since Fast Fourier Transform is a linear operation, i.e. $\alpha FFT(a) + \beta FFT(b) = FFT(\alpha a + \beta b)$. We will use the FFT encoding scheme on Elliptic Curve [158] to work with KZG polynomial commitment.Given an input array $[d_1 \cdots d_n]$ with coding ratio 1/3, RS encoding can be implemented such that the output is $[d_1 \cdots d_n, p_{n+1} \cdots p_{3n}]$, where the input data take identical locations, and parity data p_* are generated at new indices.

RS encoding can be used to introduce redundancy in numerous ways. Figure 4.4 presents a way by laying all data in one dimensional line, and apply RS encoding. Another way is to format the data into a 2D matrix like Figure 4.2. Inside the matrix, data are partitioned into columns, and redundancy is applied by taking FFT extension on each rows, such that redundant columns are generated. The matrix format is pioneered by [21] and used by Ethereum Danksharding [77] in the future roadmap. Unlike those works, our construction the matrix does not add redundancy along columns. Our RS construction is also different from Semi-AVID-FR [116], that every replica would possess one system chunks and two parity chunks.

4.12.4 BLS Elliptic Curve and Signature

This section serves as a very brief coverage of basic elements of BLS12-381 elliptic curve and their properties. We also briefly cover BLS signature which can be instantiated on the BLS12-381 curve.

An elliptic curve consists of three group G_1, G_2, G_T or order p, a pairing is an efficiently computable function $e : G_1 \times G_2 \to G_T$. Group G_1, G_2 have generator g_1, g_2 respectively,

and all members in the group can be obtained by enumerating powers on the generator. The generator of G_T equals to $e(g_1, g_2)$. A pairing function provides a following property that $e(X^a, Y^b) = e(X, Y)^{ab}$, where X, Y are elements from elliptic curve group G_1, G_2, and a, b are integer from their corresponding finite field. BLS12-381 is one elliptic curve allows for pairing (bilinear) operations.

On top of BLS12-381 curve, we can instantiate BLS public key signature [39] scheme, which allows clients to sign and verify signature. Beyond simple signature, both threshold signature and multi-signature can be developed.

4.12.5 Polynomial Commitment and KZG

We provide a more technical definition for primitives that uses KZG scheme that uses Discrete Logarithm(DL) and t-SDH assumptions [94]. First, KZG requires a trusted setup of $\{g^z \cdots g^{z^t}\}$, where z has to be a secret to anyone and g is a generator for some elliptic curve group that has bilinear pairing property [37]. We use Setup to presents the procedures to get those powers of generators, which requires a security parameter λ. The security of a scheme is measured against a probabilistic polynomial time (PPT) adversary to break the properties. A secure scheme requires all polynomial time adversary to have negligible probability to break the scheme. The primitive $Commit$ of a function $\phi = \sum_j \phi_j x^j$ can be implemented as $C = \prod_{j=0}^{deg(\phi)} (g^{z^j})^{\phi_j}$, where g^{z^j} is provided from the public setup, and ϕ_j are the coefficient for j-th monomial. The primitive $creatWitness$ for some index i with evaluation $\phi(i)$ is defined as $\psi(x) = \frac{\phi(x)-\phi(i)}{x-i}$, and we let $w_i = g^{\psi(z)}$, which is the outcomes of $Commit(\psi)$. The primitive $VerifyEval$ on some evaluation a tuple $(i, \phi(i), w_i)$ is defined as $e(C, g) = e(w_i, g^z/g^i)e(g, g)^{\phi(i)}$, where e represents the bilinear pairing operation. A proof for its correctness can be found in [94].

Definition 3 *(KZG Commitment scheme)*

1. ***Polynomial-binding***. *For all PPT adversaries* $\mathcal{A}$

$$\Pr\left(\begin{array}{c} pk \leftarrow Setup(1^\lambda) \\ (C, \phi(x), \phi'(x)) \leftarrow \mathcal{A}(pk) \end{array} : \begin{array}{c} Commit(\phi) = C \wedge \\ Commit(\phi') = C \wedge \\ \phi \neq \phi' \end{array}\right) \leq \epsilon(\lambda).$$

2. ***Evaluation-binding***. *For all PPT adversaries* $\mathcal{A}$

$$\Pr\left(\begin{array}{c} pk \leftarrow Setup(1^\lambda), \\ (C, \langle i, \phi'(i)), w_i\rangle, \\ \langle i, \phi'(i)), w_i'\rangle \\ \leftarrow \mathcal{A}(pk) \end{array} : \begin{array}{c} VerifyEval(pk, C, i, \\ \phi(i), w_i) = 1 \wedge \\ VerifyEval(pk, C, i, \\ \phi'(i), w_i') = 1 \wedge \\ \phi \neq \phi' \end{array}\right) \leq \epsilon(\lambda).$$

ϵ represents a negligible function such that $\epsilon(n) \leq \frac{1}{n^c}$ for all n greater some function of c. $\mathcal{A}$ denotes an adversary.

It can be shown that KZG provides linearity in commitments and witnesses, such that $commit(\alpha u + \beta v) = \alpha commit(u) + \beta commit(v)$, $createWitness(\alpha u + \beta v) = \alpha createWitness(u) + \beta createWitness(v)$. Here we slightly abuse the notation a bit to use + to present the group operation $\cdot$, it is useful to illustrate the following property.

The linearity of KZG commitment and RS encoding provide a powerful way to encode systematic chunks such that commitments of parity chunks interpolates to a polynomial, which is generated purely from commitments of systematic chunks.

There are existing works [77, 88, 116] that study and apply this property. Generating witnesses(proofs) is a computational intensive process. Techniques including KZG multi-reveal [78] and Universal Verification equations [83] from Ethereum provide both fast way to compute batch and evaluate proofs.

4.12.6 Pedersen commitment

Pedersen commitment is another polynomial commitment scheme [123]. Assuming discrete logarithm is hard, suppose we have a cryptographic group of $(G, \cdot)$ where $\cdot$ is the group

operation, that has order q and generator g. Suppose we choose n random point, $\mathbf{g} = g_1, \cdots, g_n$, from G, we can commit two field element $\mathbf{a} = a_1, \cdots, a_n$ by defining the commit function as

$$commit(\mathbf{g}, \mathbf{a}) = \prod_i {g_i}^{a_i}$$

Since Discrete Logarithm is assumed to be hard, no adversary can find another a', b' that commits to the same value. where product is carried over group operation $\cdot$.

4.12.7 Ordering Functions

BigDipper prioritizes transaction inclusion, and provide a modular interface for any ordering modules. From the perspective of BigDipper, the other end of the interface is treated as a **ordering** function, that returns a deterministic permutation of transactions for data corresponds to some BFT view number. For each replica, the function could be a simple deterministic implementation. For example, a shuffling function can be implemented by computing the hash digest of all sorted transactions; another example would be using round robin to take transaction one-by-one from each replica. Then if all replica only access to the same data, they all arrive to the same permutation. However, simple ordering function suffers a limitation that a malicious leader can grind its mini-blocks to alter transaction locations. More sophisticated oracle usually requires inputs from other replicas or external source for providing the ordering. For example, logical timestamp among transactions is required for creating a FCFS fair ordering [97]; auction mechanism needs to collect bids from possible external parties; a verifiable random function(VRF) requires randomness inputs from other replicas. Most implementations can be optimized with coordination from the leader. Appendix 4.21 displays a sketch about using permutation argument from zero knowledge to facilitate correct ordering.

4.13 DA-CR **Primitives and Definition**

4.13.1 DA-CR Definitions

For a DA-CR protocol, its censorship resistance depends on the characteristics of the Disperse invocation. Definitions (4, 5) quantify the amount of censorship resistance with a single parameter. Given a desired parameter, the protocol has a large design space for choosing an inclusion mechanism for meeting the requirement. Definition (7) classifies the design space into two categories, based on the idea of whether a leader informs every replica about whose data are included. It is achieved with the use of attestation in definition (6). If a design uses attestation in conforming the censorship resistance parameter, definitions (8, 9) capture possible outcomes when a leader interferes with the inclusion mechanism. In Section 4.4, we provide protocols designed base on either of the two categories.

Definition 4 (Replica-Exclusion) *Suppose a Disperse invocation finishes with a Complete type with a valid combined signature, a replica p is excluded if the leader does not include the mini-block from p for the DA-CR instance. Replica-exclusion happens even when the leader is honest.*

Definition 5 (Censorship Resistance t) *A protocol is censorship resistant if it guarantees that at least t honest replicas are not excluded in any DA-CR instance. If $t = 0$, the protocol is not censorship resistant.*

Definition 6 (Data-Attestation) *If a replica proposes some non-zero data as its mini-block for a DA-CR instance, the replica declares it to a leader by generating a signature, called attestation, based on data in the mini-block. An attestation from replica p is denoted a_p.*

Definition 7 (Accountable-Inclusion-Mechanism) *To conform with the censorship resistance t, an accountable inclusion mechanism requires a leader to explicitly demonstrate*

the required number of inclusion by showing attestations from other replicas, whereas a non-accountable inclusion mechanism allows the leader to demonstrate conformity without attestations.

Definition 8 (Data-tampering) *In an accountable mechanism, suppose a replica p participates in a Disperse invocation with a non-zero mini-block and the leader used the attestation a_p for demonstrating sufficient inclusion of mini-blocks to other replicas. Let b_p be the mini-block in the actual dispersed data corresponding to replica p, the data of replica p is tampered if a_p does not bind to b_p.*

Definition 9 (Space-capturing) *Suppose in a Disperse invocation a replica p is excluded from the DA-CR instance, the data space designated for the replica p is captured by a leader if in the actual dispersed data b_p, it is not set to a system default value reserved for the excluded replicas.*

The default value for the excluded replicas can be arbitrary, we let the value be identically 0. To handle a corner case when a replica has no data to upload, the protocol defines a special NULL transaction for differentiating scenario. With respect to definition(7), a great difference between the two types of inclusion mechanism is shown in Lemma 5:

Lemma 5 *Replica-Exclusion and Space-capturing in an accountable mechanism is publicly attributable; whereas in a non-accountable mechanism, both of them are only known privately by the leader and the excluded replica.*

The proof is a direct consequence of using attestations. In a non-accountable mechanism, there is a lack of association between dispersed mini-blocks and their senders, so the leader can overwrite some space with its own data as long as it satisfies some requirement of censorship resistance t. In contrast, an accountable inclusion mechanism keeps track of what mini-block is sent by which replica. See Appendix 4.18 for more discussion.

When an accountable inclusion mechanism is used, it exposes an attack vector based on definition(8), which does not exist for protocols that use non-accountable inclusion mechanisms. The goal of the attack is to introduce inconsistency between what is being attested

and what is actually dispersed. An honest leader never tampers data; whereas a malicious leader can tamper any mini-blocks including those produced by other malicious replicas. With the definitions above, we describe properties required for a DA-CR protocol.

4.13.2 DA-CR relationship with AVID protocol family

AVID is not communication efficient since every replica downloads the entire data. AVID-FP fixes it by using homomorphic fingerprinting, but requires a $O(n^2\lambda)$ number of overhead per node. AVID-M overcomes the quadratic overhead by using merkle commitment, but it allows the room for inconsistent coding, which either complicates the trust assumption or prolongs the confirmation latency. Semi-AVID-PR is designed for rollup, so it does not provide **Agreement** and **Termination** properties unlike others, but it shares many similarity with DA-CR on data structure and properties. Compared to those protocols, there are two great differences in DA-CR. First, DA-CR has a network assumption of partial synchrony, which allows for a leader to intermediate among replicas; it is the key that allows DA-CR to avoid the second $O(n)$ broadcast step in AVID-M [151] needed to ensure the Termination property. Second, AVID family protocols has a client-server model and the client has the entire data; whereas DA-CR is closer to a P2P network, where every replica has partial data and serves as both a client and a server. The DA-CR is similar to AVID-FP, but avoids $O(n^3)$ messages complexity, because the leader can coordinate communication in the partial synchronous network. At the downside of the DA-CR, a malicious leader can refuse to proceed the protocol at any stage. A timeout and a leader replacement mechanism are required to properly terminate a round.

4.13.3 DA-CR properties discussion

We note that the Inclusion-t, Data-tampering Resistance-ρ and Space-Capturing-η are extensions to the Correctness property which is offered by many DA-CR protocols. Those additional properties refine the relation between the original data from an honest replica and its final dispersed mini-block even when the leader is malicious. Data-tampering Resis-

tance is different from Commitment Binding, because the later guarantees there is only one data block that matches to the final commitment and its corresponding combined signature; whereas for Data-tampering Resistance in an accountable mechanism, ρ number of dispersed mini-blocks must indeed be consistent to the published data attestations binding to the final commitment.

To emphasize the importance of ρ, suppose $\rho = 0$, then the leader is free to mutate any bits in any replicas' mini-blocks. The malicious leader can then attack the system by posting the original attestations, but in fact dispersing something else. Without careful protocol design, one can come up with scheme that violates the data tampering. The inclusion-t property directly corresponds to the censorship resistant in Definition 5, and similarly, Space-Capturing-η for Definition 9.

4.14 DA-CR Protocol Designs

4.14.1 BigDipper-7

Collection Frequency and Waiting Time

In each DA-CR instance, every replica waits for its local *Signal* to send mini-blocks to the leader. The frequency of the signal must equal to the frequency of BFT blocks. If there are more than one DA-CR instance per consensus instance, a malicious leader can censor all but keep one DA-CR instance to avoid stalling the BFT protocol. If the frequency is lower, the chain cannot progress fast enough. Since an instance has the same pace as the BFT protocol, we let the signal to carry the replica's local view number from the BFT protocol. In the actual implementation, the signal can come from a pacemaker from a leader based BFT protocol [107]. Section 4.6.3 discuss an integration with Hotstuff-2.

When a leader waits for **collection** messages in the beginning of the workflow, the leader can add a new condition that allows it to wait extra time after receiving $n - f$ messages. In Algorithm 2, we denote the condition as *reach-proc-time*, standing for *reaching processing time*, which can be implemented as a short timer that evaluates to `true` on timeout. The

Algorithm 2 CARD-7

At the replica p :
$myView \leftarrow 0,\ aggSigs[\cdot] \leftarrow \emptyset,\ db \leftarrow \emptyset$
upon *receiving* $SIGNAL(v)$ **do**
  $myView \leftarrow v$
  $b_p^v \leftarrow mempool$
  $c_p^v \leftarrow kzgCommit(b_p^v)$
  $a_p^v \leftarrow sign(c_p^v||v)$
  send $\langle \mathsf{Collection}, a_p^v, b_p^v \rangle$ to leader
upon *receiving* Dispersal **do**
  $\{c_i\}_{1\le i\le 3n} \leftarrow extendEval(\{c_i\}_{1\le i\le n})$
  $a \leftarrow (checkCom(c_p, b_p^v) \vee c_p = 0) \wedge checkInc(S, \{c_i\}_{1\le i\le n})$
    $\wedge\ checkCom(c_{p+n}, chunk_{p+n}) \wedge checkCom(c_{p+2n}, chunk_{p+2n})$
  if $a =$ `true` **then**
    $db[c_p, c_{p+n}, c_{p+2n}] \leftarrow (b_p^v, chunk_{p+n}, chunk_{p+2n})$
    $C \leftarrow CRHF(\{c_i\}_{1\le i\le 3n})$
    $\sigma_p \leftarrow ms.sign(sk_p, C)$
    send $\langle \mathsf{Approval}, \sigma_p \rangle$ to leader
upon *receiving* Agreement **do**
  if $ms.verify(m.\sigma_{agg}, m.I_{\sigma_{agg}}, m.C)$ **then**
    $aggSigs[m.g, m.v] \leftarrow (m.\sigma_{agg}, m.I_{\sigma_{agg}})$
At the leader:
upon *receiving* Collection $\wedge$ $validCollect(m)$ $\wedge$ *reach-proc-time* **do**
  $B \leftarrow \{m.b_p^v | m.a_p^r \in S\}$
  $\{c_i, chunk_i\}_{1\le i\le 3n} \leftarrow encode2d(B)$
  for $i \leftarrow 1 \cdots n$ **do**
    send i $\langle \mathsf{Dispersal}, \{c_i\}_{1\le i\le n}, chunk_{i+n}, chunk_{i+2n}, S \rangle$
upon *receving* Approval $\wedge$ $validApproval(m)$ **do**
  $\sigma_{agg}, I_{\sigma_{agg}} = \text{ms.agg}(\{m.\sigma_{ip}, pk_{ip}\}_{1\le p\le n-f})$
  for `i = 1..n` **do**
    send replica i $\langle \mathsf{Agreement}, I_{\sigma_{agg}}, \sigma_{agg}, C \rangle$

Algorithm 3 utility functions

procedure checkCom($commitment$, $chunk$)
    return $kzgCommit(chunk) = commitment$
procedure extendEval($\{c_i\}_{1 \le i \le n}$)
    return $rsEncode(\{c_i\}_{1 \le i \le n}, 1/3)$ *# 1d RS to 3n with systematic encoding*
procedure checkInc(S, $\{c_i\}_{1 \le i \le n}$) *S is the attestation set*
    $O \leftarrow \emptyset$, $A \leftarrow \{1 \cdots n\}$
    for a_p^v *in* S **do**
        if $verifySig(a_p^v || myView, c_p) = \texttt{true} \wedge c_p \neq 0$ **then**
            $O \leftarrow O \cup p$
        else
            return `false`
    for $p \leftarrow A - O$ **do**
        if $c_p \neq 0$ **then**
            return `false`
    return $|O| \geq n - f$
procedure validCollect(m)
    if $verifySig(msg.a_p^v || myView, \text{kzgCommit}(m.b_p^v))$ **then**
        $S \leftarrow S \cup m.a_p^v$
    if $len(S) \geq n - f$ **then**
        return `true`, S
procedure validApproval(m)
    if $ms.verify(\sigma_p, pk_p)$ **then**
        $M \leftarrow M \cup (m, pk_p)$
    if $len(M) \geq n - f$ **then**
        return `true`, M
procedure encode2d(B)
    $\{chunk_i\}_{1 \le i \le 3n} \leftarrow \text{rsEncode2d}(B, 1/3)$
    for $i \leftarrow 1 \cdots 3n$ **do**
        $c_i \leftarrow \text{kzgCommit}(chunk_i)$
    return $\{c_i, chunk_i\}_{1 \le i \le 3n}$

number of included mini-blocks has a pivotal role for good censorship resistance in the transaction broadcast protocol discussed in Section 4.5. In short, the more mini-blocks gets included, the less resource a client needs to spend to achieve some certain probability of inclusion. If *reach-proc-time* is always true, the leader proceeds immediately after having $n-f$ mini-blocks. If the timer is set long enough, all mini-blocks from all honest replicas can be collected. The timer can be tuned to create a balance with the optimistic responsiveness property of Hotstuff.

4.14.2 Calculating Coding Ratio for DA-CR

The coding ratio is set to be 1/3, i.e. 2 parity chunks per one systematic chunk. The number three comes from the assumption that only f out of $3f+1$ BFT nodes are malicious. To prevent liveness attack from f malicious replicas, the leader should be allowed to proceed to complete a DA-CR instance as long as there is a combined signatures from $2f+1$ replicas. But since f out of the $2f+1$ signatures could come from the malicious replicas, we require the remaining $f+1$ replicas to be able to reconstruct the whole data. Although the exact ratio should have been $\frac{f+1}{3f+1}$, the erasure coding is more efficient for integer and easy for reference. Since $\frac{1}{3} < \frac{f+1}{3f+1}$, the data is safe.

4.14.3 Retrieve Algorithm

A retriever can invoke either **retrieveChunk** or **retrievePoint** in Algorithm 4 for receiving the data.

4.14.4 Retrieval success as a function of number requested replicas

Because every replica has three data chunks, a retriever at most needs to query $\lceil \frac{f+1}{3} \rceil + f$ replicas to reconstruct the interested rows in the data matrix. Let $a = \lceil \frac{f+1}{3} \rceil$. Figure 4.3 plots the probability of successful reconstruction as a function of random connections with varying number of nodes $n = 3f+1$.

Algorithm 4 Retrieve Invocations

```
1: At the replica p :
2: upon receiving RetrieveChunk do
3:     if m.c_p in db then
4:         send back ⟨db[m.c_p]⟩
5: upon receiving RetreivePoint do
6:     if m.c_p in db then
7:         φ_j, w_j ← createWitness(db[m.c_p], j)
8:         send back ⟨φ_j, w_j⟩
9:
10: procedure RETRIEVECHUNK( p, {c_i}_{1≤i≤n} ) # if p=-1, then retrieve all chunks
11:     if p ≠ −1 then
12:         for k ← {p, l} do
13:             chunk ← send⟨RetrieveChunk, c_p⟩ to replica k
14:             if checkCom(c_p, chunk) then
15:                 return chunk
16:     chunks = ∅    # when not comply
17:     {c_i}_{0≤i≤3n} ← extendEval({c_i}_{1≤i≤n})
18:     for j ← shuffle(1 · · · n) do
19:         chunk ← send⟨RetrieveChunk, c_j⟩ to replica j
20:         if checkCom(c_j, chunk) = true then
21:             chunks[j] ← chunk
22:     upon len(chunks) = f + 1 do
23:         B ← rsReconstruct(chunks)
24:         if checkCom(c_p, B[·, p]) then # if p=-1, loop for all chunks
25:             return B[·, p]
26: procedure RETRIEVEPOINT( p, j, {c_i}_{1≤i≤n}) # for a point at B[j,p]
27:     for k ← {p, l} do
28:         φ, w ← send⟨RetrievePoint, c_p, j⟩ to replica k
29:         if kzgEvalVerify(c_p, j, φ, w) then
30:             return φ
31:     {c_i}_{0≤i≤3n} ← extendEval({c_i}_{1≤i≤n})
32:     for i ← 1 · · · n do
33:         chunk ← send⟨RetrieveChunk, c_j⟩ to replica j
34:     B ← retrieveChunk(−1, {c_i}_{0≤i<3n})    # when not comply
35:     return B[j, p]
```

Since each honest replica randomly select others to connect with, the probability of connecting to a honest replicas, $\Pr(success)$, can be calculated by the following Equation:

$$\Pr(success) = \sum_{b=a}^{y} \frac{\binom{2f+1}{b}\binom{f}{y-b}}{\binom{3f+1}{y}} \tag{1}$$

As we increase the number of replica, the ratio of achieving successful reconstruction is high even at the ratio of 0.6(f+a). Each node only need to connect to approximately $4f/5$ replicas to reconstruct the data if it uses query replicas randomly.

The data reconstruction can be run asynchronously, and takes one round trip time to complete.

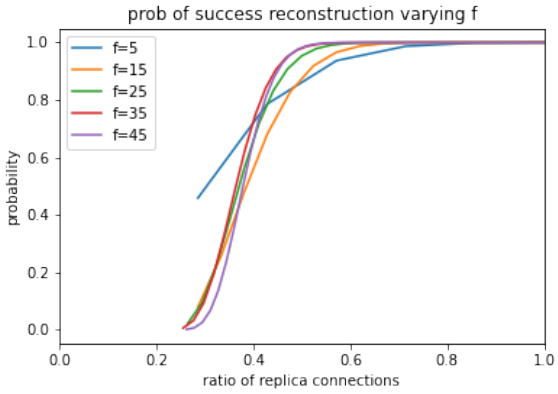

Figure 4.3: The probability to connect to a replicas as we change the total number of replicas $n = 3f + 1$. The x axis is presented as the ratio of $\frac{y}{f+a}$. The max value of x-axis is $f + a$.

4.14.5 CARD-7 *Properties Proofs*

Data-tampering Resistance-n: We prove by contradiction. Suppose a mini-block of some replica p has been tampered by the leader in some dispersal with a valid aggregate signature. By definition, the leader must have accepted p's attestation, and included it as a part of the *attestation set*. By intersection argument, $f + 1$ honest replicas must have signed a common

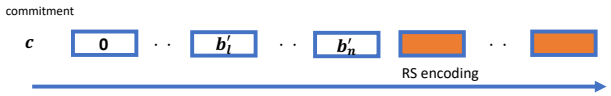

Figure 4.4: The data structure for 1 Dimensional KZG RS encoding

commitment list as its message for the aggregate signature. By (Alg.Card-7 line 11), we know the attestation signature of the tampered mini-block must verify correctly to p-th commitment in the *commitment list*, otherwise no honest replica would send the Approval. Because each of $f+1$ honest has three chunks and has verified their consistency with the received *commitment list* (Alg.Card-7.line 11-12), they in total possess $3f+3 > n$ consistent chunks. The $3f+3$ chunks are sufficient to interpolate every row to get the evaluations for the p-th column in the dispersed data, denote it as b'_p. By linearity of Reed-Solomon code and KZG commitments, b'_p must have committed correctly to p-th location in the *commitment list*. But by definition of data-tampering, b'_p does not bind to a_p, then the adversary must have broken the polynomial binding property of a polynomial commitment scheme.

Inclusion-(f+1): By (Alg.Card-7 line 11), the leader has to claim to at least include $2f+1$ valid attestations in the leader's mini-block. Out of which, at least $m \geq f+1$ mini-blocks come from honest replicas. By the Data-tampering Resistance property, all $f+1$ mini-block must be original from honest replicas without tampering.

Space-Capturing-0: Since all mini-blocks corresponding to the excluded replicas has to be 0 according to (Alg.Card-7 line 12), and because a malicious leader cannot forge signature, no adversary can insert its own data into mini-block designated for other honest replicas.

Termination: Suppose some replicas invoke Disperse(B). Because all honest replicas

can timeout, they eventually finish with type Incomplete even if the protocol has stalled due to malicious leader or network asynchrony. When the leader is honest and the network is after GST, there would be at least $2f + 1$ Collection messages with correct commitment signatures. All verification in the Dispersal messages would pass, and the leader is able to gather at least $2f + 1$ aggregate signatures. As the result, all honest replicas can finish with type Complete.

Availability: An honest replica has finished with a DA-CR instance with a type Complete, it must have received an aggregate signature with at least $2f + 1$ signer on a common message. $f+1$ of them must come from honest replicas. Because the coding ratio is $\frac{1}{3} < \frac{f+1}{3f+1}$, there are sufficient honest replicas to complete reconstruction. As the result any honest retriever can retrieve and reconstruct some data B'.

Commitment Binding: We prove by contradiction. Suppose a dispersal of B completed with an aggregate signature, and adversary finds another B' that also verify correctly with the signature. Since the aggregate signature requires at least $n - f$ signatures, at least $f + 1$ honest replicas have signed a common message. By intersection argument, there is at most one valid aggregate signature per instance. Because the hash function is collision resistant and there is only one aggregate signature, the underlying *commitment list* for producing both B and B' must also be also be identical. Suppose B and B' are different at the p-th mini-block. By polynomial binding, an adversary cannot find two b_p and b'_p that commits to identical commitment. Hence all $f + 1$ honest replicas must have cheated.

Correctness: An honest replica has Finished a dispersal with a type Complete, then by **Availability**, the retriever must be able to retrieve some data B'. Because the dispersed data B' is Commitment Binding, any retriever can only retrieve identical mini-blocks that binds to the *commitment list*. For those do not bind to the *commitment list*, the kzg commitment cannot match. Let B denote the mini-blocks that a leader collects right before starting the RS encoding. If the leader is honest, then $B = B'$.

4.15 Card-¼ *Protocol*

In every dispersal instance of Card-7, a replica needs to download both the *commitment list* and the *attestation set* whose size is linear to the number of replicas. In this section, we develop a non-accountbale protocol called Card-¼ that has constant message complexity as shown in Table 4.2. It also provides us a chance to analyze how accountability affects DA-CR properties by comparing two protocols. For the rest of the section, a threshold signature scheme is used for achieving a constant signature overhead.

At a high level, Card-¼ is very similar to Card-7, both of them use identical message flow depicted in Figure 4.1. However, there are three changes that differentiate the two protocols. First, when a replica sends its mini-block to the leader, no attestation is ever generated. Second, after a leader collected at least $q \geq n - f$ mini-blocks, the leader concatenates all mini-blocks and use the single polynomial to performs the RS encoding. The result is a single commitment which can be used in the approval message. Figure 4.4 in Appendix 4.16 displays the data structure. The third difference happens at the dispersal step, the leader only sends coded data to those q replicas whose mini-blocks are collected. As the result, there is no linear overhead in the number of replicas. However removing the accountability component, i.e. *attestation set*, the protocol also suffers some drawback. We state it in the Lemma 6:

Lemma 6 *If a protocol uses an accountable inclusion mechanism, the leader can send dispersal to all replicas, and all honest replicas (including excluded ones) can sign the approving signature without violating censorship-resistance. For honest replicas in a non-accountable inclusion mechanism, only those whose mini-blocks are included by the leader can approve the dispersal.*

In Card-7, a leader sends an identical tuple (*attestation set*, *commitment list*) to all replicas. Once the tuple is received, any replica including the excluded one can participate the approval step, because anyone can verify Alg.Card-7.line 11-12 to determine if the leader has included at least $n - f$ mini-blocks. However in the non-accountable mechanism

like CARD-¼, because attestation is not available, no replica can associate a mini-block to its original sender except a mini-block is created by itself. An **excluded** honest replica cannot distinguish between the case when an honest leader including $n - f$ replicas and the case when a malicious leader includes $n - f$ arbitrary mini-blocks. Consequently, only replicas whose mini-blocks are included can send approval. In non-accountable mechanisms, a leader has to be allowed to proceed with even $n - 2f$ approving signature because there is a case when f honest replicas are excluded, and at the same time f malicious replicas refuse to send approval message. The security assumption for CARD-¼ becomes $n \geq 4f + 1$. It is required for efficient information dispersal. Because f replicas out of $n-2f$ approving replicas can be malicious, we require $n-3f$ replicas to be able to reconstruct the whole data. With $4f + 1$, we can perform the RS coding with a chunk size $O(1/f)$ of the final BFT block size. Other properties and proofs are provided in Appendix 4.16.3.

4.15.1 Necessary condition for $\rho = n$

When an accountable inclusion mechanism is used by a DA-CR protocol, we want every dispersal to be tamperment proof, as it removes the requirement to detect the inconsistency. We provide Lemma 7 that states the necessary condition for $\rho = n$ and shows 1D KZG alone in CARD-¼ is insufficient:

Lemma 7 *If a dispersal scheme provides Data-tampering resistance-n, all honest signing replicas that approve the dispersal must have together ensured the consistency of every data in the dispersed block against the attestations.*

The proof to Lemma 7 can be derived from the statement. Suppose there is one data point not checked by any honest signing replicas, then the malicious leader can mutate that data point to introduce inconsistency. If CARD-7 uses 1D KZG for encoding, a leader can mutate mini-blocks from some $z \leq f$ honest replicas. But the rest $n - z$ replicas would still verify correctly that their received chunks are consistent to a global commitment, and the adversary can progress the protocol with tampered data. The 2D KZG construction

from CARD-7 satisfies the Lemma condition on every data point, because the interpolating polynomial of the *commitment list* can implicitly perform cross-columns checking. The proof of Data-tampering resistance-n in Section 4.14.5 demonstrates this key idea.

4.16 Non-accountable inclusion mechanisms

4.16.1 VANILLA *Protocol*

VANILLA uses unaccountable inclusion mechanism. The biggest advantage is the constant message complexity as shown in Table 4.2. The construction and message flow are very similar to CARD-7 depicted in Figure 4.1 and Algorithm 2. Here we overload the term VANILLA for both entire protocol and DA-CR.

VANILLA is a protocol with minimal changes from regular BFT protocols [42,51,107,153]. The BFT block is still assembled from mini-blocks, but it the dispersal step, the whole BFT block (containing all mini-blocks) is broadcast to everyone, instead of using verifiable information dispersal. Although inefficient, VANILLA has censorship resistance of $t = n-3f$. A leader first waits for $n - f$ mini-blocks to assemble the BFT block; for each replica who received the BFT block, it verifies if the block contains its own mini-block, and only its own data is included it would send the approving partial signature. For the same reason stated in Lemma 6, the excluded replicas cannot participate the approving procedure because it cannot verify if the leader has indeed include other's mini-blocks. Consequently, the leader is required to collect $n - 2f$ aggregate signature before proceeding, because there could be f malicious replicas within the $n - f$ replicas whose mini-blocks are collected. At least, since an honest leader only needs to wait for $n - 2f$ approving message, there could be f malicious replicas. So the Inclusion-t of DA-CR property for VANILLA is only $n - 3f$. So when $n = 3f + 1$, the minimal mini-blocks coming from honest replicas in the final BFT block is only one.

4.16.2 Proofs for Vanilla

In this section, we provide DA-CR property proofs for Vanilla. First briefly recapping the algorithm. It belongs to non-accountable inclusion mechanism. The sequence of message exchange follows the same workflow shown in Figure 4.1 as to Card-7 protocol discussed in the main text.

The biggest difference in Vanilla compared to Card-7 is that dispersal message contains the entire BFT block, as opposed to a chunk. In both scheme, the approving message is signed by a threshold signature scheme. We now presents proofs for Vanilla protocol. Because every replica in Vanilla receives the entire data, **Availability**, **Correctness**, **Commitment Binding** are inherited by default. Because the protocol uses the same message sequence and timeout, **Termination** proof is identical to Card-7. Because it uses non-accountable mechanism, **Data-tampering Resistance**-ρ is not applicable. **Inclusion-(n-3f)** is discussed in the last section. Essentially, because the leader can wait for at least $n - f$ mini-blocks. After dispersing the BFT block, in the worst case only $n - 2f$ can send approval message, and we must allow the leader to proceed with that amount of approval. In the remaining replicas, f can be malicious, and therefore the included honest replicas is $n - 3f$. The proof for **Space-Capturing**-$2f$ is due to non-accountability. A malicious leader can falsely submit mini-block for any $2f$ mini-block space, as long as there is one honest replica who sends the approval message, the malicious leader can move to the next round, because there are f malicious replicas to approve the message. So in the end, the leader captures $n - f - 1$ mini-blocks.

4.16.3 Proofs for Card-¼

Now we show proofs for Card-¼. One Big difference from Card-7 is the usage of 1D KZG scheme shown in Figure 4.4 and threshold signature. All mini-blocks are first concatenated into one dimensional line, and then the leader performs RS encoding to extends n chunks to $3n$ chunks. Then the leader produces only one polynomial commitment, which is used

as approval message. Another big difference comes from an increase of required security assumption of $4f + 1$ in order to enable information dispersal.

Availability, **Correctness** follows the same proof as CARD-7, since there are at least $f+1$ honest replicas each holding three correct chunks. **Commitment Binding** is provided by KZG polynomial commitment scheme. Because we do not change the message sequence and timeout, **Termination** is identical. Because it is not an accountable mechanism, **Data-tampering Resistance**-ρ is not applicable. **Inclusion-*n-3f*** is stated in the Section 4.15, and the argument is identical to proofs of VANILLA. At last, **Space-Capturing**-$2f$ are identical proofs for VANILLA.

4.17 Card-Lite *Protocol and Analysis*

4.17.1 Doubly Homomorphic Commitment Scheme

In this section, we provide a summary on how to use Doubly Homomorphic Commitment Scheme [43] in CARD-LITE to achieve $O(\log n)$ overhead complexity while still ensuring correct RS encoding. For explicit construction of polynomial commitment scheme that allows opening with witnesses, [43] contains more details on the construction.

Since CARD-LITE is a 2D KZG RS construction, the leader has a *commitment list* of size n. The censorship resistance protocol described in Section 4.4.2 requires each replica to have 2 parity chunks and 2 commitments as usual, but it additionally requires samples other columns. The two parity chunks are used for ensuring data availability. To ensure all sampled commitments corresponds to a single commitment polynomial of degree $n - 1$, every commitment must come with a logarithmic sized proof binding to the commitment C. Because otherwise, the leader can attack the system by incorrect coding, such that no retrieve can reliably retrieve data binding to the *commitment list*.

Regular merkle tree vector commitment provides the first requirement (logarithmic proof size), but fails the second requirement, because there is no way to verify all $3n$ commitments are interpolating a polynomial of degree $n - 1$. The doubly homomorphic commitment

scheme allow us to perform commitment of commitments, which treats the *commitment list* of 2D KZG as n evaluation points, which is points from G_1 group in some elliptic curve. Then with a polynomial with n evaluations, we can find out their corresponding coefficients, and get the final commitment with Pedersen Commitment scheme (see Appendix 4.12.6) by taking the inner product between the n coefficients from group G_1 and n basis from group G_2. We require the underlying elliptic curves has pairing capability such that we can derive a final commitment in G_T, see Appendix 4.12.4.

With construction of generalized inner product argument (GIPA) [43], we can use the polynomial commitment scheme with ability to open indices with witness size $O(\log n)$. To save verification cost, one can use a structured setup on G_2 similar structure in the setup of KZG 4.12.5 (g^{z^i}. However computation is not our concerns. But we can use SRS to limit the size of polynomial to degree $n-1$, which eliminates the needs of low degree testing to ensure the degree polynomial is indeed low degree of $n-1$.

Otherwise, we can use a low degree testing, by repeatedly reducing the degree of polynomial by splitting even and odd monomial, then committing the combined polynomial of half of the degree for $O(\log n)$ times, until there is a constant polynomial.

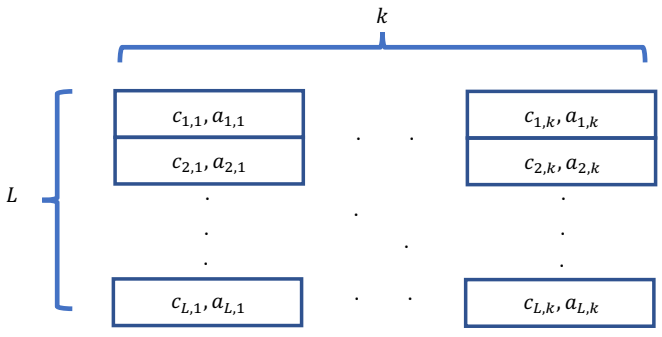

Figure 4.5: The sampling matrix for one replica.

4.17.2 Configuring Parameter for boosting

As described in Section 4.4.2, we recap the full CARD-LITE protocol and present analysis to derive proper parameters to achieve properties stated in Table 4.2. Since the leader cannot send *commitment list* of length n to replica, it has to use a doubly homomorphic commitment scheme to reveal commitments with $O(\log n)$ size proof. All replicas sign the final commitment C (which is commitment of the commitments) as its approving message. After collecting mini-blocks and creating an *attestation set*, the leader creates n sampling matrix, and each matrix has the format shown in Figure 4.5. The matrix entry at coordinate (i, j) consists of a tuple of $(a_p, c_p, \textit{proof of } c_p)$ corresponding to the some random query that requests an attestation from replica p, where $1 \leq p < n$. There are in total kL random query. In Figure 4.5, we use subscript to denote each sample, and omits their proofs. Clearly, if some replica q wants to create such a matrix, it cannot send kL random queries to create the matrix. We require the leader to use Fiat-Shamir Transform to generate randomness for replica q based on the commitment C by using hash function h to generate $r_{i,j} = h(C, i, j, q)$, where we use $(r_{i,j} \bmod n)$ to decide which replicas the random query wants to receive the tuple for. The leader has to present n matrix for each replica. Let's now focus on one such matrix.

If a leader excludes attestations from some replicas, some entries in the matrix must be empty, because no leader can provide proof for something that is not on the polynomial or attestations is not available. It is orthogonal to Data tampering attack, which is handled by the 2D KZG construction.

Suppose there are two types of leaders, both types collect *attestation set* from replicas, and use 2D KZG scheme shown in Figure 4.2 for encoding, and generate a *commitment list*. For simplicity of presentation, we don't allow any space capturing, hence all excluded mini-blocks must be 0.

We want to understand the worst case scenario on how much a malicious leader can censor. The honest leader type does not censor mini-blocks, whereas a malicious leader

censors at least $s \geq 1$ mini-blocks on top of f allowable exclusion. Now in the random query game, the probability that a random number mod n picks on the included mini-block is $\Pr_m(\text{valid}) \leq p_d = (2f+1-s)/(3f+1)$ for a malicious leader (inequality in case the malicious leader censors more); whereas for an honest leader is $\Pr_h(\text{valid}) = p_h = (2f+1)/(3f+1)$ (inequality in case malicious replicas complies with the protocol).

Then if k of such random queries are requested, the probability of providing all k valid responses together is at $g_d \leq p_d^k$ for malicious and $g_h \geq p_h^k$ for honest leader respectively. Clearly, $g_d < g_h$. As in the main text, we call the k query together as a batch.

Now we can compute a value $\rho < 1$ such that ${p_d}^\rho = p_h$, and equivalently $\rho = \frac{\log(1/p_h)}{\log(1/p_d)} = \frac{\log(1/g_h)}{\log(1/g_d)}$. Similarly compute a value k, such that ${p_d}^k = \frac{1}{f^\tau}$, where τ is a small positive value $\tau > 1$, hence equivalently $k = \frac{\tau \log f}{\log 1/p_d}$. We can bound g_d and g_h by

$$\begin{aligned} g_h \geq {p_h}^k &= {p_h}^{\frac{\tau \log f}{\log(1/p_d)}} \\ &= f^{-\frac{\log(1/p_h)}{\log(1/p_d)}} \\ &= f^{-\rho} \\ g_d \leq p_d^k &= f^{-1} \end{aligned}$$

The second equality comes from logarithmic rule $a^{\log b} = b^{\log a}$. Now suppose the node sends L batches to the leader. Now define two probability P_1, P_2, which equal to the probability that none of L batch is successful for honest and malicious leaders respectively.

Suppose we choose $L = \tau f^\rho \log f$, where τ is some constant.

$$\begin{aligned} P_1 \geq 1 - (1-g_h)^L &= 1 - (1 - f^{-\rho})^{\tau f^\rho} \\ &= 1 - e^{-\tau \log f} \\ &= 1 - f^{-\tau} \\ P_2 \leq 1 - (1-g_d)^L &= 1 - (1 - f^{-1})^{-\tau f^\rho \log f} \\ &= 1 - e^{-\tau f^{\rho-1} \log f} \end{aligned}$$

Clearly, P_1 approaches to 1 for any $\tau > 1$. Whereas for P_2, we need additional procedure to derive bounds. Since exponential is continuous function, we can bring the limit to the exponent,

$$\lim_{f\to\infty} e^{-\tau f^{\rho-1}\log f} = e^{\lim_{f\to\infty} -\tau f^{\rho-1}\log f}$$

we then need to calculate ρ to derive bounds on s, such that for the malicious type the limit on the exponent goes to 0, and consequently $P_2 \to 0$.

Since $\lim_{f\to\infty}(f^{-1/\log f})^{\tau} = e^{-\tau}$. We want to get a ρ such that $\rho - 1$ equal to $\frac{-\tau}{\log f}$. By definition, $\rho = \frac{\log(1/p_h)}{\log(1/p_d)}$.

We can approximate ρ with Taylor series with some additive second order error $\frac{6}{(3f+1)^3}$, which is close to 0 as f increases.

$$p_h = \frac{2f+1}{3f+1} = 1 - \frac{f}{3f+1} \approx e^{-\frac{f}{3f+1}}$$
$$p_d = \frac{2f+1-s}{3f+1} = 1 - \frac{f-1+s}{3f+1} \approx e^{-\frac{f-1+s}{3f+1}}$$

hence $\rho = \frac{f}{f-1+s}$. With the equality $\rho - 1 = \frac{-\tau}{\log f}$. Now compute s

$$\log f = \frac{\tau}{1-\rho} = \frac{f-1+s}{s-1}\tau$$
$$s = \frac{f\tau}{\log f} = \Theta(\frac{f}{\log f})$$

As the result,

$$P_2 \leq 1 - e^{-\tau f^{\rho-1}\log f} = 1 - e^{-\tau e^{-\tau}\log f} = 1 - f^{-\tau e^{\tau}}$$

Clearly, $e^{-\tau}$ decreases much faster than τ for $\tau > 1$, P_2 approaches 0 exponentially fast

as we increase τ. Therefore

$$P_1 \geq 1 - f^{-\tau}$$
$$P_2 \leq 1 - f^{-\tau e^{-\tau}}$$

Even with a small τ, the honest replicas can differentiate two types with extremely high probability.

The leader prepare one $k \times L$ matrix for every one replica. Each matrix corresponds to a sampling replica p. If the leader finds a valid batch (row), it can demonstrate that sufficient many mini-blocks are included, by sending the particular valid row(batch) in the matrix to the replica. For an honest leader, it needs to find the working batch for at least $2f + 1$ replicas, hence the probability of is

$$\begin{aligned}
\Pr(\text{success} \text{ — honest}) = {P_1}^{2f+1} &= (1 - f^{-\tau})^{2f+1} \\
&\approx e^{-\frac{2f+1}{f^{\tau}}} \\
&\approx e^{-2f^{1-\tau}} \ \textit{where } \tau > 1 \\
&\approx 1
\end{aligned}$$

However, if the leader is malicious, it needs to convince at least $f + 1$ honest replica.

$$\begin{aligned}
\Pr(\text{success} \text{ — malicious}) = {P_2}^{f+1} &= (1 - f^{-\tau e^{-\tau}})^{f+1} \\
&\approx e^{-\frac{f+1}{f^{\tau e^{-\tau}}}} \\
&\approx e^{-f^{(1-\tau e^{-\tau})}} \\
&\approx e^{-f} \\
&\approx 0
\end{aligned}$$

Hence for any number of censoring with $s \geq \Theta(\frac{f}{\log f})$, there is extremely low chance that all $f + 1$ would find valid batches. So a malicious leader can at most censor $s = o(\frac{f}{\log f})$. In

summary, the leader needs to create n sample matrix for each replicas, each of which has $(\tau f \frac{f}{f-1+s} \log f)$ number of rows and $(\frac{\tau}{\log 1/p_d} \log f)$ number of column. Since $p_d = \frac{2f+1-s}{3f+1}$ and $1 \leq s \leq o(\frac{f}{\log f})$, we can derive a more concise inequality for the number of columns, i.e batch size as

$$\begin{aligned} k &= -\tau(\log f) \log p_d \\ &\leq -\tau(\log f) \log(\frac{2f - \frac{f}{\log f}}{3f}) \\ &\leq \tau(\log f) \log 2/3 \\ &\leq \kappa \log f \end{aligned}$$

where κ is a small positive constant.

For every dispersal, a leader has in total $O(f^2 \log^2 n)$ rows for all replicas. A malicious leader cannot grind the final commitment C to increase the probability of convincing $f+1$ replicas if it censors $s = \Theta(\frac{f}{\log f})$, because it needs to grind at least Z times, such that $Z * e^{-f}$ equals to some constant, where n can take asymptotic value.

The overall overhead message size is therefore just the $\kappa \log f$ tuples consisting of $(a_p, c_p, proof)$. The proof part takes $O(\log n)$ size because the polynomial commitment scheme uses IPA. However, since Pedersen commitments are aggregatable, the $\log f$ proofs can collapse together in a single proof. Hence the overall overhead complexity is still $O(\log f)$.

4.17.3 Card-Lite *Proof*

We now prove the DA-CR properties for Card-Lite property.

Termination: Since we don't touch the protocol mechanism, and there is still timeout to detect livelessness, Termination proof is identical to the Card-7.

Availability: If an honest replicas received an threshold signature, there must be at least $2f+1$ signer, and $f+1$ are honest replicas. Since there are at least $f+1$ honest replicas that signed the message, eventually an honest retrieve can find a honest replicas to get the correct *commitment list* that commits to the final commitment C. After getting the

commitment list, since coding ratio is 1/3, any honest retriever can reconstruct the whole BFT block without concerning about the degree of the polynomial and incorrect coding. This is because we use IPA for commitment to the polynomial and can use a fixed size SRS to ensure the commitment is consistent to a polynomial of degree $n-1$. (we can optionally use a low degree testing for the same purpose).

Commitment Binding: The idea is similar in proof for CARD-7 except we replace the collision resistant hash function(CRHF) with a polynomial commitment scheme (generalized inner product argument), which is also collision resistant.

Correctness: The proof is identical to CARD-7.

Data-tampering Resistance-n: Since IPA is commitment binding, no adversary can find another *commitment list* that maps to the same final commitment. As the result, the proof is identical to CARD-7, since all signing replicas is using a common *commitment list*.

Inclusion-$(f - o(f/\log n))$: The proof is a direct consequence from Section 4.17.2.

Space-Capturing-$(o(f/\log f))$: Since the Data-tampering Resistance is $n - 2f - z$ where $z = o(f/\log f)$, the malicious leader can capture z mini-blocks without honest replicas noticing. The malicious leader cannot capture more than z because the correct batch of size k which the leader has to provide to all honest replicas must come with k original attestations.

One thing worth noticing is that the protocol designer also has a coarser control on how much to allow the leader to do space capturing, instead of a batch containing all k original attestation, it could allow $k-1$ attestations but allow the leader to have 2 attestations.

4.18 Stealing or LSC*?*

So far, we have yet discussed Leader Space Capturing (LSC) in depth. This section serves for developing difference and clarifying some internal tradeoff. In developing censorship resistance property, we use decentralization as the underlying force to achieve it. It is therefore natural to adhere to a principle that block building is a collective work that everyone should be given a fair share of space resource. However, a space capturing is a phenomenal when a leader has taken data space from other replicas which can potentially have positive effects

on the system utilization. On the other hand, if without constraint a malicious leader can capture too much space leaving only one guaranteed inclusion of honest replica; in the protocol Vanilla, as n becomes larger, the protocol is as close to as a regular leader based BFT protocol.

LSC is intractable with non-accountable inclusion mechanism because there is no mapping about whose mini-blocks get included. On the other hand, accountable inclusion mechanism allows protocol designers to tune the amount of LSC. In both accountable mechanisms we discussed, the parameter η is configured as 0, however with little modification, all accountable protocols can increase the number, by allowing a leader to replace some 0 in the *commitment list* with leader's own attestations and mini-block.

Given possible choices of LSC, we now elaborate in what ways LSC can be beneficial and harmful, which would be helpful for protocol designers to decides if to allow LSC and at what amount.

As mentioned, space capturing is unobservable in non-accountable mechanisms. However, if it is allowed in accountable mechanism, η is the only mechanism which can control the amount of LSC. Recall in in section 4.14.1, we discuss a local parameter *read-proc-time* which determines how much longer an honest leader waits after receiving $n - f$ mini-blocks. But since the *read-proc-time* is an unobservable parameter to the outside parties, all leaders can set it to maximize LSC, and the protocol has no means of enforcement. On the positive side, when the network is bad and many replicas cannot submit mini-blocks fast enough before the dispersal period ends, LSC allows the leader to make use of available space for higher system throughput. On the negative side, if we attempt a rational model for a moment, and assume replica's revenue increases with the number of included mini-block, then LSC encourages any leader to take data space from other replicas. However, at the same time if everyone gets the chance to be a leader, all can do LSC while at the same time increasing system utilization.

If LSC is allowed, there is subtle difference of LSC depending on the accountability of the inclusion mechanism. If non-accountable mechanism is used, a malicious leader can apply

censorship on the basis of replicas, and the replica which repeatedly do not get mini-blocks included cannot even prove itself being censored. And vice versa for a malicious replicas falsely accuses a leader when its data is included. In accountable mechanism, although we cannot attribute which party is at fault, we can at least know without false alarm there is something wrong in the protocol such that we can perform analysis on the leader or even increase network waiting time.

If we view LSC from a broader perspective that takes clients into considerations, the next section shows that the LSC parameters η has a great impact on the amount of censorship resistance to client's experience and system efficiency. In general, the less is the η value, the better is the client's experience and system efficiency.

4.19 Navigator protocol

4.19.1 Properties

A replica is available for a DA-CR instance if it can insert some transactions into its mini-block before sending it to a leader. Given f malicious replicas in the system, the NAVIGATOR protocol satisfies the following properties:

- **Strict Transaction Inclusion in DA layer**: Given a DA-CR instance with t guaranteed honest mini-block inclusion, if an honest client sends $n-t+1$ copies of a transaction to distinct available replicas, then the instance is either Incomplete, or Complete with the client's transaction.

- **Probabilistic Transaction Inclusion in DA layer Under Honest Leader**: Given a DA-CR instance with t guaranteed honest mini-blocks inclusion under an honest leader, if an honest client sends x copies of a transaction to random distinct available replicas and the instance Completes, the probability of inclusion monotonically increases with x and t. If network latency of honest replicas are i.i.d, the probability increases exponentially with x and t.

- **Probabilistic Transaction Inclusion in DA layer Under Malicious Leader**: Given a DA-CR instance with t guaranteed honest mini-blocks inclusion under a malicious leader, if an honest client sends x copies of one transaction to random distinct available replicas and the instance Completes, the probability of inclusion monotonically increases with x, up to 1 when $x = n - t + 1$. When $x \leq f$ the malicious leader can censor the transaction with probability 1, but to censor u distinct transactions each of x copies altogether, the probability of censoring decreases exponentially as u increases.

- **2 Δ Probabilistic Confirmation in DA layer**: Given a DA-CR guarantees at least t mini-blocks inclusion from honest replicas, if an honest client sends x copies of transaction for inclusion, the client can compute the probability of inclusion right after 2 Δ network latency based on number of responses from x replicas and belief on the leader's type.

Those properties are proved in Appendix 4.19.8.

4.19.2 Algorithm

The Algorithm is presented in Algorithm 5. Appendix 4.19.3 provides a concrete implementation for the procedure call at line 4, which allows client to get a list of available replicas.

4.19.3 Client and Replicas time synchronization

We present a time synchronization protocol in Algorithm 6. Essentially, the algorithm requires the client to periodically pings each replicas about their status about when they would submit the next mini-block. Assuming the worst cast network delay Δ after GST, the client can assure they can make the deadline. If the network has not reached GST, the BFT protocol might not reach agreement. The accuracy of the status synchronization depends on the ping frequency, which is a tunable parameter for each client.

Algorithm 5 NAVIGATOR

```
1: At the client:
2: procedure submit( x, tx)
3:     v ← currentBlockNum
4:     S ← getAvailableReplicas(x, v), cn ← 0
5:     if |S| < x then
6:         return FAIL, |S|
7:     for j in S do
8:         r ← send ⟨tx, v⟩ to replica j
9:         cn ← cn + r
10:    return cn
11: At the replica p :
12: upon receiving ⟨tx, v⟩ do
13:    if (v > myView) ∨ (v = myView ∧ waiting for Signal) then
14:        insert ⟨tx, v⟩ to mempool  and respond 1
15:    else
16:        respond 0
```

Algorithm 6 NAVIGATOR Synchronization

```
1: S_v ← ∅, v ← 0 ··· INF, timer
2: At the client:
3: procedure cronJob
4:     for i ← 1 ··· n do
5:         s ← send⟨PING, v, timer.now⟩      # timeout 2Δ
6:         S_v ← S_v ∪ s
7:     # Δ is latency in network model
8:     S_v ← {s : s ∈ S_v, timer.now < s.deadline}
9: At the replica p :
10: upon receiving ⟨PING, v, tn⟩ do
11:    if (v > myView) ∨ (v = myView ∧ currAt = SIGNAL) then
12:        deadline ← tn + 2Δ + timeToNextSignal
13:        send⟨PONG, deadline⟩
14:    else
15:        send⟨PONG, 0⟩
16: procedure GETAVAILABLEREPLICAS( x, tv)
17:    return shuffle(S_tv)[x :] #  return only x replicas
```

4.19.4 Empirical Evaluation on Inclusion Probability when Leader is honest

Figure 4.7 plots the probability of censorship for a transaction using a DA-CR with $t = f+1$ and $n = 3f + 1$ when the leader is honest, such that $t \leq q \leq n - f$, where $f = 10$ and $n = 3f + 1 = 31$.

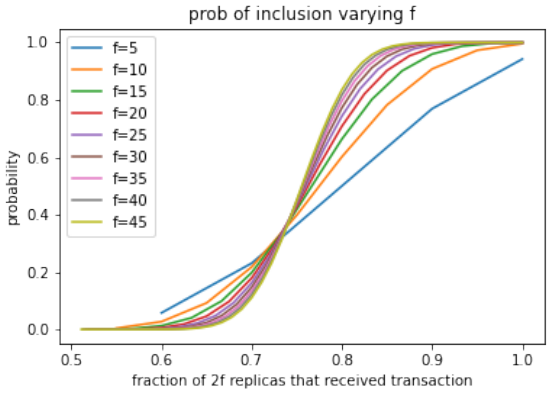

Figure 4.6: The probability of inclusion given the leader is malicious with varying f. The x-axis is a ratio of the number of copies of transactions and $2f$

4.19.5 Analysis on probability of inclusion given malicious leader with $f + 1 \geq x$

Figure 4.6 plots the probability of inclusion with varying number of replicas $n = 3f + 1$. The inclusion can be satisfied with high probability even the client sends less than $2f + 1$ copies, approximately $1.8f$. The probability of inclusion decreases as the clients send less number of copies of transactions.

4.19.6 Upper Bound on Total Censored transactions

In Appendix 4.19.7, we prove with Hoeffding's inequality and union bound that the probability of removing α fraction out of total T transactions is upper bounded by $2\binom{n}{n-f} \exp\{-2(p + \alpha - 1)^2 T\}$, where $p = 1 - \prod_{i=0}^{x-1} \frac{n-f}{n-i}$. As T increases, the probability decreases exponentially.

4.19.7 Analysis on the upper bound on the total number of censored transaction

Suppose there is a total T unique transactions, denoted as $t_1, ..t_T$ by all clients, where each transaction has x copies which are randomly sent to replicas without replacement. For a small constant ratio α, it is exponentially unlikely that the leader would be able to censor more than αT transaction.

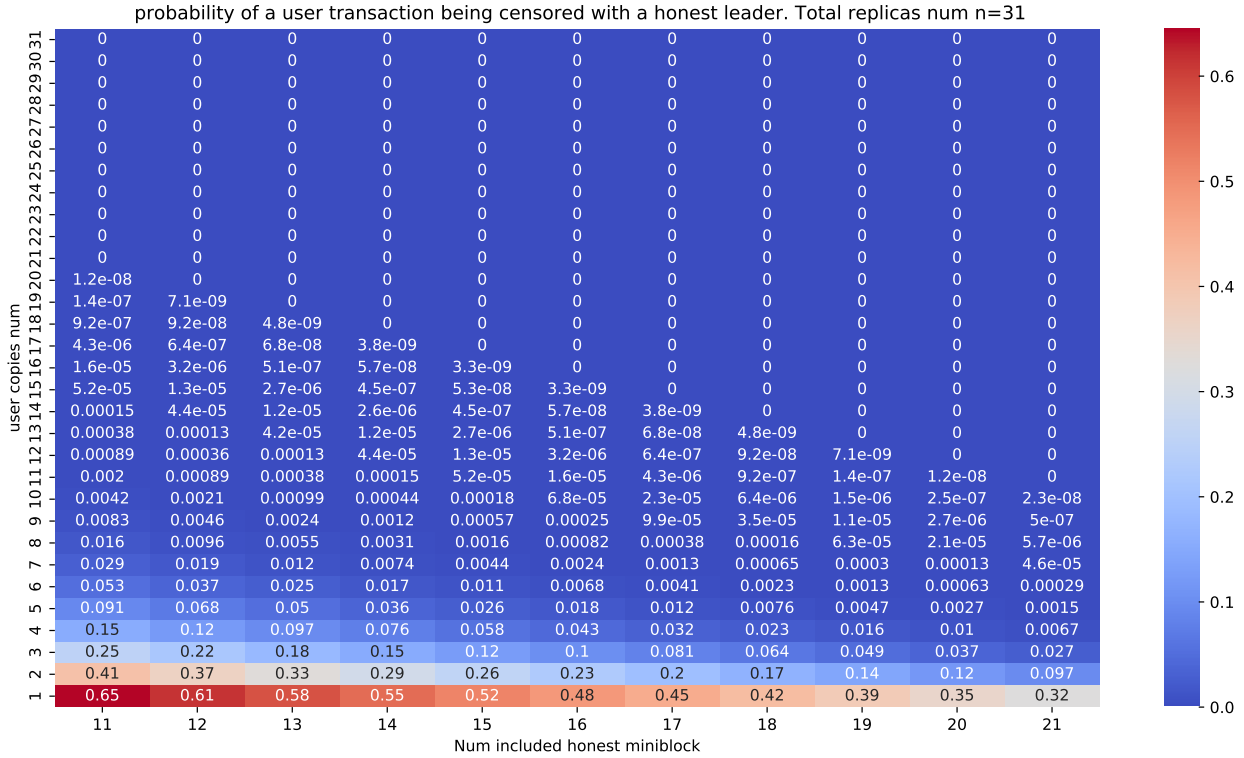

Figure 4.7: Under honest leader, the probability of a transaction being censored as a function of (x, q), where $1 \leq x \leq n,\ f < q \leq 2f + 1$

Define a set of mini-blocks S of cardinality $n-f$, we want to compute the probability that a malicious leader can find a particular S such that the union of all transactions from mini-blocks in set S has a size less than $(1-\alpha)T$. For the same reason we discussed in Section 4.5.2, the probability of inclusion is entirely depend on how likely a client sends its transaction to honest replicas; since adversary can always remove those transaction by selectively accepting other honest mini-block. The probability of such set S exists can be written as Equation 7, where every transaction corresponds to an independent random indicator variable, $\mathbb{1}_{t_i \in S}$. The variable is independent because every transaction runs its own instance of Algorithm 5. It is a Bernoulli random variables that evaluates to 1 if condition $t_i \in S$ holds true; otherwise 0. The probability of the condition being true is $p = 1 - \prod_{i=0}^{x-1} \frac{n-f}{n-i}$.

$$\Pr(\textit{remove } \alpha T) = \Pr_{|S|=n-f}\Big\{\exists S : \sum_{i=1}^{T} \mathbb{1}_{t_i \in S} < (1-\alpha)T\Big\} \tag{7}$$

$$\leq \binom{n}{n-f} \Pr\Big\{S : \sum_{i=1}^{T} \mathbb{1}_{t_i \in S} < (1-\alpha)T\Big\} \tag{8}$$

The inequality from Equation 7 to 8 comes from Boole's inequality. To get the probability on the right high side, we use a random variable $\frac{\sum_{i=1}^{T} \mathbb{1}_{t_i \in S}}{T} = X$, whose expected value is X is p, by rearranging terms of Equation 8, we get

$$\Pr\Big\{S : \sum_{i=1}^{T} \mathbb{1}_{t_i \in S} < (1-\alpha)T\Big\} = \Pr\Big\{X < 1-\alpha\Big\} \tag{9}$$

$$= \Pr\Big\{E[X] - X > E[X] + \alpha - 1\Big\} \tag{10}$$

$$\leq \Pr\Big\{|E[X] - X| > E[X] + \alpha - 1\Big\} \tag{11}$$

$$\leq 2\exp\left\{-2(p+\alpha-1)^2 T\right\} \tag{12}$$

In Equation 11, we let $\alpha > 1 - E[X] = \prod_{i=0}^{x-1} \frac{n-f}{n-i}$, inequality is due to $\Pr\Big\{|E[X] - X| > E[X] + \alpha - 1\Big\} = \Pr\Big\{E[X] - X > E[X] + \alpha - 1\Big\} + \Pr\Big\{X - E[X] < -E[X] - \alpha + 1\Big\}$. Inequality 12 comes from Hoeffding's Inequality.

The probability of removing αT transactions is therefore upper bounded by $2\binom{n}{n-f} \exp\left\{-2(p+\alpha-1)^2 T\right.$

As it can be shown that given 300 replicas and 200 transactions per mini-block, a malicious leader can censor more than 5% of the transactions.

4.19.8 Navigator Properties Proof

In this section we prove the four properties stated in Section 4.5.

Strict Transaction Inclusion in DA layer: Suppose there is a DA-CR instance with t guaranteed honest mini-block inclusion, and the client sends $n - t + 1$ copies of the same transaction to distinct replicas which are available for the instance round, then

suppose the leader is malicious and only includes the minimal t mini-blocks from honest replicas, which is required by the DA-CR. By pigeonhole principle, there must be at least one transaction received by one of the t honest replicas whose mini-block would get included. Similar argument can be made for honest leader.

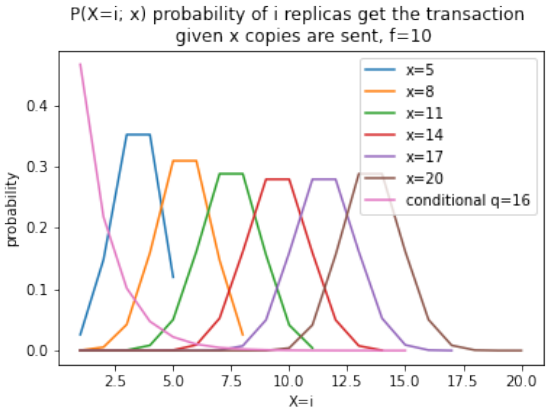

Figure 4.8: The probability of that , $X = i$, i number of honest replicas receiving the transaction given x copies was sent. We show one conditional probability when $q = 16$ assuming network latency is i.i.d to illustrate the point that both q, x needs to work together to increase the probability of $\Pr(IN)$.

Probabilistic Transaction Inclusion in DA layer Under Honest leader: Suppose a DA-CR instance with t guaranteed honest mini-blocks inclusion, if an honest client sends x copies of a transaction to random distinct available replicas. We first show that the probability of inclusion increases with x and t in the feasible region where a leader cannot exclude all x copies without making the instance Incomplete. From Section 4.5 and Table 4.3, we can define the feasible region as $(0 < x < n, t \leq q \leq n - f)$ when the leader is honest; By our first properties, we already knows when $x \geq n - t + 1$, the probability of inclusion is 1. We now focus on the rest of the case.

When the leader is honest and $(0 < x < n - t + 1, t \leq q < n - f)$, Section 4.5.2 shows

that

$$\begin{aligned}\Pr(IN;\, x,q) &= 1 - \sum_{i=1}^{x} \Pr(None \mid X = i; q) \Pr\left(X = i; x\right) \\ &= 1 - \sum_{i=1}^{x} \Pr(None \mid X = i; q) \frac{\binom{n-f}{i}\binom{f}{x-i}}{\binom{n}{x}} \end{aligned} \tag{13}$$

where X is a random variable indicating the number of honest replicas that has client's transaction. $P(q,i) = 1 - \Pr(None \mid X = i; q)$ is the probability none of X honest mini-block containing the client's transaction. We note that $P(q,i)$ is monotonically increasing in q, it is because as there are more guaranteed honest mini-block inclusion, the chance that one of the i mini-blocks containing the transaction gets included is non-decreasing. Since $q \geq t$, $P(q,i)$ increases as t increases. Because $\Pr\left(X = i; x\right)$ does not depend on t, the overall probability $\Pr\left(IN\right)$ as a function of q increases as $P(q,i)$ increases. Therefore $\Pr\left(IN\right)$ is monotonically increasing as q increases.

We now argue $\Pr\left(IN\right)$ is monotonically increasing as x increases. We first show that $P(q,i)$ is monotonically increasing in i while fixing q. Since the honest leader is collecting q honest mini-blocks out of $n - f$ honest replicas. If the number of honest replicas whose mini-block containing the transaction increases, then the chance of $P(q,i)$ is at least non-decreasing. Now in the Equation 13, there is a conditional sum over all possible value which X can take up to x; in the worst case when $\Pr(None \mid X = i; q)$ is a constant ratio, then the final probability $\Pr\left(IN\right)$ is also be a constant. But as the conditional probability starts to decrease, the overall $\Pr\left(IN\right)$ increases. In addition to that, as x increases, the distribution of $P(X = i; x)$ gets shifted rightward, such that it is more common (higher probability) that more honest replicas receiving the transaction. The intuition behind the phenomena is as x increases, the numerator terms of $\frac{\binom{n-f}{i}\binom{f}{x-i}}{\binom{n}{x}}$ finds its peak also at higher i. Figure 4.8 plots distribution for different x when $f = 10$. Hence if the conditional probability drop fast enough, a client only needs to send a few copies x, such that the permutation term (binomial like) raises up just after the conditional term drops, the probability can be kept very low.

In Figure 4.8, the conditional probability is using i.i.d assumption with number of included mini-block $q = 16$, out of 21 possible honest mini-blocks with $n = 31$.

Now, we show the second statement that if the network latency is i.i.d, the inclusion probability increases exponentially with x and t. Since network latency is i.i.d, the conditional part can be computed as

$$\begin{aligned}
\Pr(\mathit{None} \,|X = i; q) &= \prod_{j=0}^{q-1} \frac{n-f-i-j}{n-f-j} \\
&= \prod_{j=0}^{q-1} \frac{a-i}{a} \\
&< (1 - \frac{i}{n-f})^q \\
&\approx e^{-q\frac{i}{n-f}}
\end{aligned}$$

Hence we know the term $\Pr(\mathit{None} \,|X = i; q)$ is exponentially small when q increases. However, we need also show that the summation term in Equation 13 is small.

First note that, the permutation term, $\Pr(X = i; x)$, is always less than 1, so we can just focus on the case when i is small, because when i is large, the conditional probability part is already very small.

Now we use the binomial upper bound, $\binom{n}{k} < (\frac{ne}{k})^k$ [65], for numerator, and binomial lower bound, $(\frac{n}{k})^k < \binom{n}{k}$, for denominator to show that in worse case, when i is small the probability is exponential decreasing when x increases

$$\begin{aligned}
\frac{\binom{n-f}{i}\binom{f}{x-i}}{\binom{n}{x}} &< \frac{(\frac{(n-f)e}{i})^i(\frac{fe}{x-i})^{x-i}}{(\frac{n}{x})^x} \\
&= \frac{(n-f)^i f^{x-i} x^x e^x}{i^i (x-i)^{x-i} n^x} \\
&= \frac{(n-f)^i f^x x^x e^x}{i^i x^x n^x} \\
&\approx \frac{(n-f)^i}{i^i}(\frac{e}{3})^x
\end{aligned}$$

Clearly for small i, as x increases the value is upper bound by a exponentially decreasing value. Hence it completes the proof.

Probabilistic Transaction Inclusion in DA layer Under Malicious Leader: The first part of the statement that the probability increases as x increases comes directly from the fact from Section 4.5.2 where $f+1 \leq x < n-t+1$ that

$$\Pr(IN) = \sum_{i=f+1}^{x} \frac{\binom{n-f}{i}\binom{f}{x-i}}{\binom{n}{x}}$$

Clearly, $\Pr(IN)$ increases as x increases. If $x \leq f$, the probability is 0, because the malicious leader can choose the rest $f+1$ honest replicas which does not have the transaction.

To show that a malicious leader cannot censor all u unique transactions altogether, we use the result from Section 4.19.6, where $p = 1 - \prod_{i=0}^{x-1} \frac{n-f}{n-i}$

$$\Pr(\textit{Censor all } u \textit{ txs}) = 2\binom{n}{n-f} \exp\left\{-2(p+\alpha-1)^2 T\right\}$$

As we can see, given a fixed pair of n, x, the probability that the censoring attack succeeds is decreasing exponentially in either T, α or x. By letting $u = \alpha T$, we get our property.

2 Δ Probabilistic Confirmation in DA layer: By Algorithm 5, the clients is able to get the response if some replicas are able to submit transactions. The client can calculate its probability of inclusion based on the response. Although some responses might be sent from malicious replicas, the calculation stated in the previous properties can apply directly to derive the probability result, since they already assume the malicious replicas are not including the transaction. Since each categories have the explicit formula to compute the probability, the client can use the received the responses, the number of duplication and current belief on the leader to compute the probability of inclusion.

4.20 Spamming Attack and Transaction De-Duplication

BigDipper expects clients to duplicate its transactions among replicas to increase censorship resistance. However, without proper mechanism clients are inclined to spam the BFT replicas for increasing the chance of inclusion. A direct way of controlling the spamming problem is to

increase the cost of transaction. We could use a price mechanism, like EIP1559 [135] deployed in Ethereum, which uses posted price when the network is not congested, and otherwise use first price auction. But the price mechanism alone does not the problem, since the price is raised for everyone. The final cost for a client should also correspond to the number of copies a transaction has been sent. This can be enforced by requiring the execution layer (virtual machine) to also charge fees by transactions counts as a meta information when calculating the spent resources.

Another method that does not rely on in-protocol mechanism like 1559 is to use market mechanism entirely, such that every BFT replicas is free to post its own prices. The n replicas creates a competitive market. If we consider there are 2 types of transaction hold by various clients, and let the BFT replicas being the first mover in the information asymmetry game, a natural solution is to use screening mechanism. We can find a separating subgame perfect Nash equilibrium if the number of high value transaction type is relatively small compared to the low value type transaction. The screening mechanism has been been well studied in the field of contract theory [134]. However in the screening model, the BFT replicas would lose some profits from high value transaction, because the high value type would mimic the type of low value transactions. Then the replicas can use the value to sort the transaction and only include the ones that fit into the mini-blocks.

For spamming concern at network level, we can rely on standard approaches including content distribution network, proxy networks which have been already used in the real world scenario.

4.20.1 De-duplication

As client sends multiple copies of the same transaction, it might be worrisome that use of bandwidth is inefficient. However, as how to handle the spamming problem, the usage of block space is essentially economical. If a client desires more properties that take more block space, the client should pay for more. It is beneficial if the system can remove the inefficiency entirely, but given that BigDipper is hyperscalable, having some copies of transaction is

considered acceptable. For most time-insensitive transaction, censorship resistance is not a strict requirement.

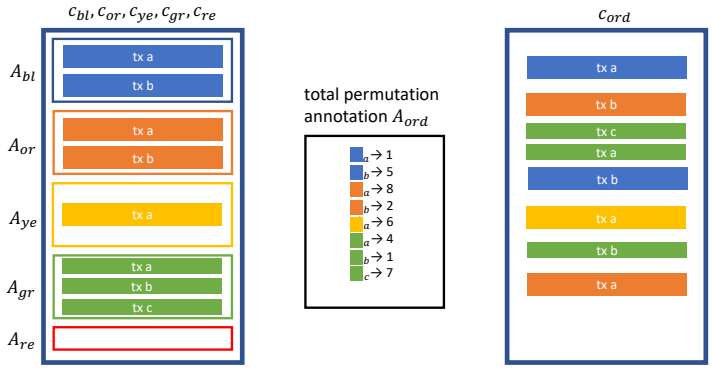

Figure 4.9: Transaction reordering from 5 mini-blocks, each contains varying number of transactions. The commitment is denoted by first two letter of their color ($c_{re} = 0$). c_{ord} is the commitment of ordered transactions.

4.21 Zero knowledge permutation argument

4.21.1 Permutation Argument

We define the property that an ordering function has to satisfy in order to ensure censorship resistance of the protocol. We focus on an ordering function that can be not deterministically derived from the data, and requires exogenous input and leader's coordination. Those include partial block auction, whole block auction, VRF.

- **Correct Permutation**: If an honest replicas has decided on some ordered transactions for some view in the BFT protocol, then the decided order must be a correct permutation of the dispersed data without corrupting integrity of any transactions.

Suppose B_{ord} is the resultant ordering after the exogenous ordering function. In the simple case, if the entire ordered block B_{ord} is broadcast by the leader and each replica

Algorithm 7 *Scooping*

1: **At the replicas** p:
2: **procedure** $verifyCard(\ commit, v)$
3: **return** ms.verify($aggSigs[\ commit, v]$)

has retrieved all data by using the Retrieve invocation, each replica can enforce the correct permutation property by checking if the ordered dispersed block is a subset of the ordered block. We now discuss the case when no replica retrieves the entire data. It is achieved with permutation argument. A visualization of a permutation argument is shown in Figure 4.9. The leader needs to sends three items for replicas to construct the permutation circuit used for verifying validity, which include a permutation annotation(function), annotations of the transactions size and hash inside each mini-block and a polynomial commitment after the ordering. The annotations of transactions A_i for mini-block i can be provided and encoded by replicas when creating the mini-block. To prevent the leader from corrupting the annotation of transactions, the leader has to provide witnesses against the *commitment list* for all A_i to every replicas.

Then the leader publishes the commitment of the ordered transactions by sending the actual ordered block B_{ord}, or its commitment c_{ord}. A permutation circuit with Halo2 [158] Plonkish arithmetizaton can be built to ensure a correct permutation with inputs from A_i and A_{ord}, such that all transaction boundary are preserved.

4.22 Scooping Protocol

4.22.1 Scooping Algorithm

The scooping algorithm is shown in Algorithm 7.

4.23 Integration with Hotstuff-2

Figure 4.10 depicts a flow diagram that integrates Card-7 with Hotstuff-2. We encourage readers to read about Hotstuff-2 [107] protocol. $C_v(\cdot)$ represents a certificate at view v,

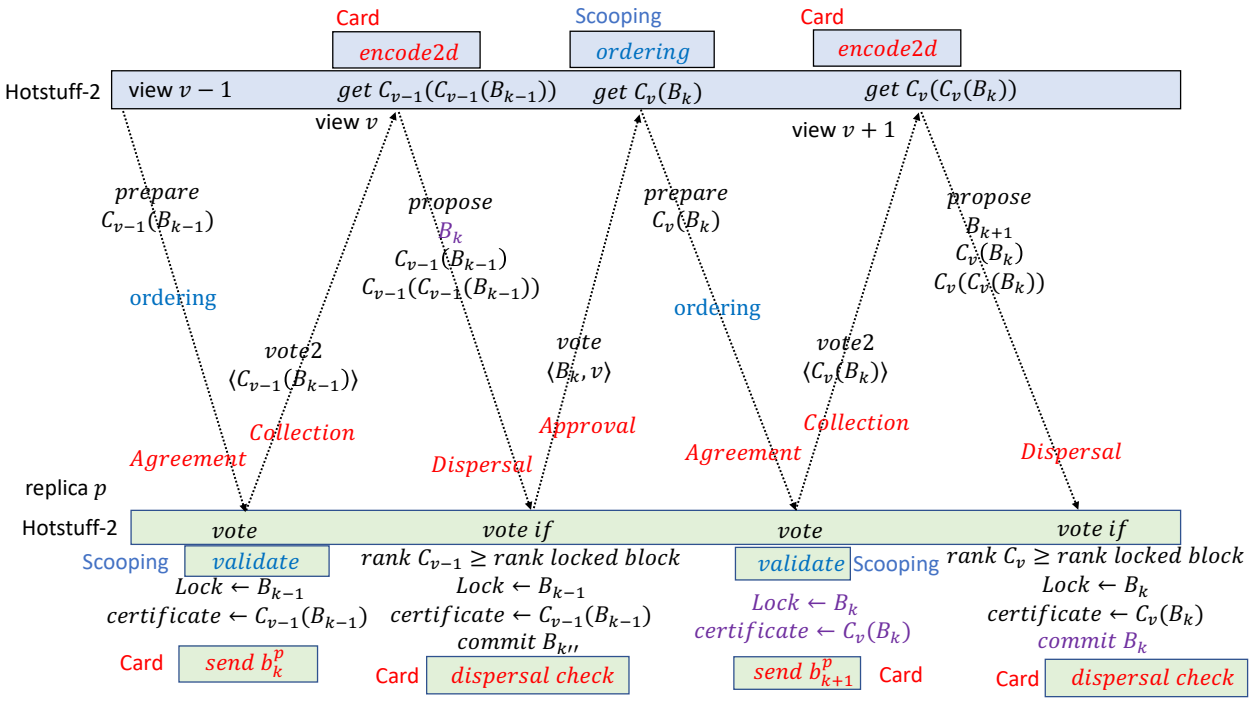

Figure 4.10: A flow diagram depicting an adaptation of BigDipper with Hotstuff. The procedures from Hotstuff-2 protocol is marked in black; Card-7 protocol is marked with red, and Scooping protocol is marked in blue. The signature validation procedure has to be verified after receiving the settlement message.

$C_v(C_v\cdot)$ represents a double certificates, which indicates the block at view v can be safely committed by anyone. k denotes block height, since no all view can successfully commit a block, hence k is different from v.

On the diagram, the blue bar represents the Hotstuff-2 leader replica, the green bar at the bottom represents a non-leader replicas. Procedures needs to be carried out after receiving the messages are written close to the message arrow, and the sequence of execution is from top to down.

Because we are using verifiable information dispersal, all block B from the Hotstuff-2 protocol actually means the metadata overhead described in the Card-7 protocol, which

are *commitment list* and *attestation set*.

All replicas need to three algorithms: Hotstuff-2 (in black and purple), CARD-7 (in red) and Scooping (in blue). The time flows from left to right, transitioning from view $v-1$ to v. Starting from *Collection* message from the left, it belongs to the CARD-7 protocol when a replica tries to get its mini-block included by the leader; the *vote2* message belongs to Hotstuff-2 protocol, the leader needs to collect at least $2f+1$ vote2 messages before proceeding; *vote2* message is sent at a stage that the replica has locked some block, and is planning to vote the second time to commit the block. The subscript indicates that the block B_{k-1} at blockchain height $k-1$ happens at view number $v-1$.

When the leader collects at least $2f+1$ Collections messages and $2f+1$ vote2 messages. The leader performs the 2D RS encoding and sends the *Dispersal* message of the CARD-7 protocol, at the same time the leader also sends the *Propose* message along with the double certificates to the replica. This is the official start of the new block, the block is tagged with view number v; in the diagram we colored all procedures relates to this block in purple. At the same time, the previous block B_{k-1} has just been able to commit after creating a double certificate by collecting signatures from $2f+1$ *vote2* messages, the double certificate are piggybacked to the replicas.

After receiving those messages at the replica side, the replica would directly commit the block that has double certificates, in addition it would indirectly commit all blocks that are prefix of the block just being committed. In the diagram, we represents them as $B_{k''}$. The non-leader replica then needs to perform the dispersal check which corresponds to the kzg commitment checking and signature checking from CARD-7 Protocol. After everything passes, the replica sends the *Approve* message of CARD-7 and *vote* message of Hotstuff-2 to the leader.

The leader waits to receive messages from $2f+1$ replicas and creates the combined signature for *Agreement* message of CARD-7 and similarly a certificate $C_v(B_k)$ for *prepare* message of Hotstuff-2. Optionally the leader can assign order of transaction in this step; however, for us we use deterministic ordering so the *ordering* message should be empty.

At the replica side, after receiving the first certificate for B_k, the first step is to check if the validate procedure from the SCOOPING protocol is successful. If it is not, then the data is not available, so the replica should not lock the certificate. Otherwise, the replica proceeds with regular Hotstuff-2 procedures. At the same time, the leader is preparing for a new view at $v+1$, so the replica should prepare its new mini-block b_{k+1} and sends to the leader. And we just complete a cycle of optimistic path when nothing wrong happens.

4.24 Properties proofs to BigDipper-7 *Hotstuff-2 integration*

4.24.1 Properties proofs to Safety and Liveness

We prove that the integrated protocol satisfies safety and liveness properties of BFT protocol, which has great similarity to the proof in [107]. All properties listed in Section 4.10.1 can be proved by applying properties proven in CARD-7 and NAVIGATOR protocols.

Safety: We prove by contradiction, suppose two blocks B_l, B'_l commit at the same height l. Since DA-CR has Correctness, Commitment binding and Data-tampering Resistance properties, both block must be made available and properly certified in different BFT instances, otherwise it cannot pass Alg 1 line 7. Suppose both are committed as result of direct commits of B_k, B'_k at view v, v' respectively. Suppose $v \leq v'$ (if $v = v'$, then suppose $k \leq k'$), by Lemma 8 from Appendix 4.24.3, certificate $C_{v'}(B'_{k'})$ must extend B_k, hence $B_l = B'_l$.

Liveness: The integrated protocol uses identical pacemaker as Hotstuff-2. Since DA-CR has Termination property and Hotstuff-2 is live by itself, the protocol continues to next view if a malicious leader stalls the protocol or the network is asynchronous. When the leader is honest and the network is synchronous, in the optimistic path where the double certificate is received, the leader can collect at least $n-f$ mini-blocks (line 11) from replicas. If the honest leader is not in the optimistic path, the leader must wait for 3Δ before proposing, Line 3 requires the replica to send the collection message along with the locked certificate. Consequently the leader always have data to propose in both cases. The Termination property of DA-CR ensures dispersal can finish with type Complete, so BFT can make progress.

4.24.2 Properties proofs to Addditional Censorship resistance properties

Strict Transaction Inclusion: Because CARD-7 provides Inclusion-(t) with $t = f + 1$, the strict inclusion directly come from Strict transaction in DA layer property provided by the NAVIGATOR algorithm. Since the integration with Hotstuff-2 is safe and live, under an honest client, a client can have guarantee of inclusion with probability 1 by sending $n-t+1$ copies of transaction. If a malicious leader excludes the transaction that has been sent to $n - t + 1$ replicas, also due to Strict transaction in DA layer property provided by the NAVIGATOR algorithm, the DA-CR results in an Incomplete state, which no valid combined signature can be generated, hence the integrated BIGDIPPER-7 Hotstuff-2 integration would not proceed because no honest replica can pass the SCOOPING Algorithm, which is inserted into the consensus path in Alg.SCOOPING line 11.

Probabilistic Transaction Inclusion Under Honest Leader: The proof is almost identical as above. Since the integration with Hotstuff-2 is safe and live, a client can send x copies of transaction to probabilistically reach sufficient replicas, the rest follow through the Probabilistic Transaction Inclusion in DA layer Under Honest Leader from NAVIGATOR. Since the integrated protocol is live and safe, the probability of client's transaction is included increases exponentially with x and t.

Probabilistic Transaction Inclusion Under Malicious Leader: The proof is almost identical as above. If the leader is malicious and the instance Completes, we use can use Probabilistic Transaction Inclusion in DA layer Under Malicious Leader from the NAVIGATOR protocol, to derive that the inclusion probability is monotonically increasing with x, and when $x \leq f$, the probability of censoring u transactions altogether decreasing exponentially as u increases.

2Δ Probabilistic Confirmation for Clients: It is a direct consequence of CARD property and NAVIGATOR property from Section 4.5.

Hyperscale Throughput: Since all BIGDIPPER protocols use verifiable information dispersal, the throughput of reaching consensus to a sequence of bits in not bottleneck from

downloading the whole data of size $O(bn)$, . The system capacity is summation of bandwidth from all replicas, which is $O(C_{band}n)$, where $C_{band}$is bandwidth for each replica. Then the limit on rate of information dispersal for each replicas needs to be $O(b) \leq C_{band}$, so at maximum the rate of reaching consensus on a sequence of bits is only a constant of system capacity.

4.24.3 Lemma in support for safety proofs of BigDipper *Hotstuff-2 integration*

We show that lemma4.1 from Hotstuff-2 [107] still holds in the integrated protocol.

Lemma 8 *If a party directly commits block B_k in view v, then a certified block that ranks no lower than $C_v(B_k)$ must equal or extend B_k.*

Proof 7 *We use the same approach as the Hotstuff-2 proof, we are mostly concerned that the data is unavailable, we present the whole proof and shows the data has to be available for any committed blocks. Suppose we consider a block $B'_{k'}$ that is certified in view v' to produce certificate $C_{v'}(B'_{k'})$ has a rank higher than $C_v(B_k)$ if $v' > v$. We will use induction on view v'.*

At the base case $v = v'$, $C_{v'}(B'_{k'})$ is certified in view v. But it is impossible, since no honest replica would vote twice per a single view.

At the induction step, since $C_v(C_v(B_k))$ exists, there must be a set of replicas S whose size is no less than $2f + 1$ that possess $C_v(B_k)$. Since a double certificate can only be created with at least $f + 1$ honest replicas, we know the data B_k is available. All replicas in S must have locked on B_k or a block that extends B_k at the end of v. By induction assumption, any certified block that ranks equally or higher than v must equal or extend B_k. By the end of v', the replicas in S are still locked at B_k or blocks that extends B_k. The leader has two ways to enter a new view. In the first case when the leader has a double certificate, then at least $f + 1$ honest replicas have ensured the data is available, so it is safe to extend on top of it. In the second case, the leader finds out the highest certified block by waiting 3Δ to receive all honest replicas. Since the leader verifies that all locked certificate must have a valid aggregate

signature corresponding to it, this step is ensured by Alg 1 line 5. So we know it is safe for the leader to build on top of it. Resume the Hotstuff-2 proof. Consider a proposal at view $v'+1$, if the leader makes a proposal $B'_{k'}$ that does not extend B_k, its certificate must be ranked lower than $C_v(B_k)$. As the result, no honest replica would vote for it, so the certificate $C_{v'+1}(B'_{k'})$ cannot be formed. Hence if $B'_{k'}$ was committed, $C_{v'+1}(C_{v'+1}(B'_{k'}))$ is formed, $B'_{k'}$ must extend B_k.

To clarify special cases, when a malicious leader correctly runs the Card protocol but refuses to broadcast the aggregate signature to anyone, the malicious leader eventually gets replaced. However, since no one has the aggregate signature, a new block proposed by an honest leader would not build on top of that block after waiting 3Δ. If a malicious leader broadcasts the aggregate signature, but refuses to create the double certificates, any honest replica can send the locked certificate along with the aggregate signature to convince the leader to build on top of it.

Chapter 5

TRAVELERS

5.1 Introduction

Cryptocurrency transaction has been a major application to blockchain for more than ten years since the start of Bitcoin. With the programmable features, Ethereum has enabled an array of decentralized financial (Defi) applications including exchanges, lending platform and stable coins. A great amount values are transacted every day, the daily trading volume on Ethereum is more than five Billions US dollar [56]. With so much values attracted into the blockchain, people starts to exploit some unspecified area in the protocol to extract Maximal Extractable Value (MEV) from users. The extraction is possible because most consensus protocols used today does not concern about the ordering of transactions.

For example, automated market maker (AMM) is an type of programmable exchange on blockchain. If a user sends a transaction to purchases asset, adversary can make profits out of user by placing two transactions before and after the user' transaction. Such attack is called sandwich attack, and has been extensively documented and analyzed in the past, see [63,87]. Besides the sandwich attack, people have discovered various other attacks specific to the Defi protocols, see [152] for a more comprehensive list.

To counter the problem, people have proposed at least three categories of solutions. First, notions of fair ordering have been proposed by many consensus protocols, including [48,57,97,157,160]. Most of them align on a common FIFO principle: a transaction perceived by the protocol first, should be executed before others, though the way they measure the ordering are different. Second, many large projects including Ethereum and application-chains from Cosmos adopt an approach called, MEV auction platform. With this approach, MEV is not alleviated, but optimized to extract the greatest value and then redistribute the

revenue back to the block proposers who are parts of ecosystem. The third approach can be categorized as content-agnostic ordering [146, 152], essentially the transaction content is committed and threshold encrypted to hide it from the adversary, so that by the time it is revealed, the transaction is already confirmed on the blockchain. Compared to fair ordering protocol, the content-agnostic approach offers a weaker guarantee, because it provide no ordering properties about its transactions. For example, a transaction submitted much later in a block can still front-run others transaction.

In the current blockchain ecosystem, the MEV auction approach has demonstrated to be the most popular. There is a flourishing ecosystem built on top of foundation of economic incentives, that includes Flashbot [80], Bloxroute [36], Blocknative [35], Eden [72], Skip Protocol [140]. The auction solution has a great advantage of $O(1)$ communication complexities. This is achievable because only a few centralized entities need to collect transactions in order to build a best block to win the auction. The second popular approach is the content-agnostic ordering. Although it offers less properties than fair ordering protocols, many projects like Shutter network [117], Sikka [139] have implemented and started to contribute the blockchain ecosystem, because of its simplicity and wide spectrum of applications. With regard to the category of fair ordering protocols, till now as far as author is aware, there is no well known project that operates a fair ordering protocol (except Chainlink has promised it since September 2020 [52]).

We notice all existing fair ordering protocols have rather stiff ordering policy enforcements regardless of the underlying fairness notions. Out of all practical solution, Themis [97] is the most communication efficient protocol, but still requires $O(nTL + n^2T)$ communication complexity, where L is the transaction size, T is number of transactions, and n is total number of nodes[1]. Importantly, Themis consensus protocol is entangled with complexity to deal with Condorcet Cycle, which is a phenomenon that the ordering among transactions are

[1]Themis SNARK [97] has $O(nTL+nT)$ communication complexity. But SNARK requires cryptographic computation order of magnitude higher than commitments or hashes. It serves only the theoretical purposes, as self-described by [97]

chained together to form nested loops, as the result those transactions have to be batched together. Although the protocol is carefully designed, the resulting protocol entails a great complexity.

This work explores a new notion of fair ordering, referred as probabilistic fair ordering. The purpose is to relax the criteria of existing fair order, by removing the requirement to have all nodes receiving identical transactions. We then introduce a class of protocols, called Travelers, based on a new notion called probabilistic ordering linearizability. Ordering linearizability is proposed by Pompē [160], that requires a real clock on each node to lock a transaction with a timestamp. Because timestamps are linear, all locked transactions with timestamps can be totally ordered by a simle sorting. It is a key insight mentioned in Pompē [160] that avoids the complex puzzle of Condorcet cycle. But even with a clock, Pompē still requires a communication complexity of $O(n^2TL + n)$. With the new relaxed notion of probabilistic ordering linearizability, Travelers can achieve $O(c\log(n)TL+D)$ communication complexity with error probability $\epsilon = \frac{1}{n^c}$ for some system parameter c, where D is the total communication complexity for some consensus protocol. In Travelers, the precision of fairness notion depends on how large the mismatch δ among nodes' clock and network latency Δ. We elaborate on the probabilistic notion in Section 5.3 and provide a concrete fairness definition for one Travelers protocol in Section 5.5.2.

5.1.1 Travelers system

At the high level, we can use a framework to split the system into two components: transaction submission and consensus. Transaction submission is the part responsible for getting sufficient nodes to receive the transaction, and it is where the transaction size L shows up in the complexity analysis. The consensus part is responsible for making agreement on some data critical to the consensus protocol. We will further elaborate this framework in the next Section. Unlike other fair ordering protocol, the submission component in Travelers neither requires a P2P dissemination network, nor it is required to send transactions to all correct nodes. Reducing the dissemination phase is important to improve the first term of

the communication complexity from $O(nL)$ to sublinear in L, see Table 5.1.

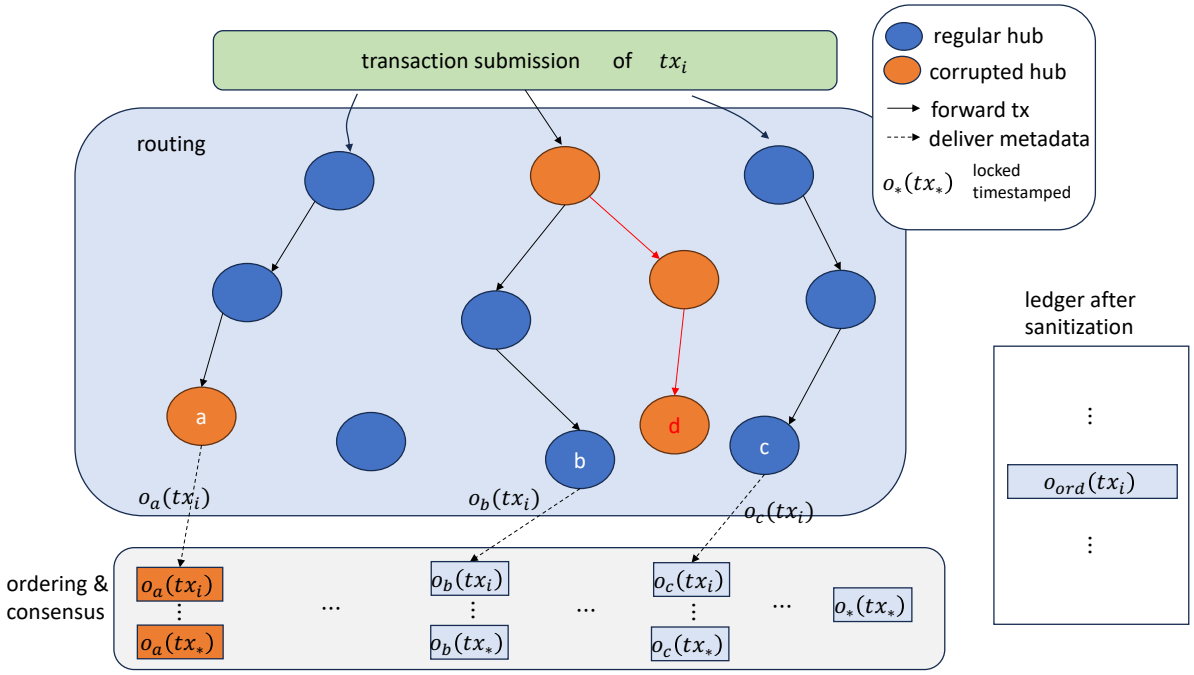

Figure 5.1: Travelers architecture. Suppose every hub consists of a single node. Each of three copies of transaction has distinct path. Nodes a, b, c are the nodes from the final hubs. The path containing node c consists of regular hub only. We use o_{ord} to denote the locked timestamp.

In our submission component, we design a routing protocol that moves a transaction through a series of hubs on a path. The transaction receives timestamp at each hub, and all timestamps are used together by the routing protocol to produces the locked timestamp, which is used for ordering. Figure 5.1 displays several paths each contains three hubs. A hub is a collections of nodes, whose membership and size are determined by some assignment function decided by the protocol. We will discuss more about the assignment function from Appendix 5.8.3. However, in Figure 5.1 each hub is assigned with only one unique node.

When a transaction arrives, a hub timestamps its arrival time and approves it with a

cryptographic signature on both metadata and the transaction payload. A corrupted hub indicates the set has significant amount of malicious nodes, such that they alone can approve the transaction with arbitrary timestamp. On the other hand, if honest nodes alone can approve the transaction, it is called the regular hub. If all of hubs in a path approved the transactions, the nodes from the last hub deliver all timestamps to the consensus along with signatures and transaction payloads. We define maximal timestamps in the path as the **locked** timestamp for the transaction on that path.

In Figure 5.1, we show a red path consisted entirely of malicious hubs. It is referred as the corrupted path, and similarly a path of all regular hub as the regular path. We want to decrease its probability of corrupted path such that it is less than ϵ. Because if no regular hub can bound the timestamp, an adversary can lock arbitrary timestamp for any transactions that travel the path. Not only we need to lower ϵ, the routing protocol also needs to cap the total possible number of paths by using a deterministic random function. Because there are combinatorial number ways to arrange hubs into distinct paths, no matter how small ϵ be, an adversary can find a corrupted path if given sufficient chances.

When a client sends multiple copies of the same transaction to traverse distinct paths, each would generate a different locked timestamp. It is the ordering rule's responsibility to determine the canonical timestamp for the transaction. Once one canonical timestamp is selected by the ordering rule, because timestamps are linear by nature, the total order of transactions can be derived by a simple sorting algorithm. The right side of Figure 5.1 provides global view about the ledger after the sanitization.

The notion of ordering linearizability decouples the ordering and consensus, as pointed by many works [87,157,160]. Because timestamps lock transactions, the only part we need from the standard consensus is just an agreement on a stream of bit, which is data availability, so that anyone can be retrieved them later. Standard asynchronous and partial asynchronous BFT protocol inherently satisfy data availability, but make it a requirement for every node to download the entire data during the agreement protocol. For fair ordering protocol, **censorship resistance** is an important consensus property but missed by many standard

leader based protocol . It guarantees some transaction is included in the next block even if the current and future leaders have the intent of censoring. Travelers requires the underlying consensus to have this property in order to commit the locked timestmap. Most standard leaderless BFT protocol has censorship resistance in nature. For example, DispersedLedger [151] is an asynchronous protocol that uses Erasure code to improve throughput. VABA [20], Dumbo-MVBA [105] are leaderless protocols that have communication complexity of $O(n^2)$. Because timestamps lock transaction before consensus, the confirmation latency required by the consensus protocol does not affect final ordering. An concurrent submission, called **BigDipper**, contains more rigorous formulation and presentation, and shows that the complexity can be further reduced to $O(\log(n)\lambda)$, resulting to a protocol with only $O(c\log(n)TL + \log(n)n\lambda)$ overall communication complexity.

Similar to many protocols based on the notion of ordering linearizability, Travelers can further protect transaction with encryption. The idea is to let the client encrypt the transaction, so that the ciphertext can only be decrypted toward the end of the path, such that no adversary has sufficient time to frontrun the transaction. We discuss it in Section 5.4.5.

5.1.2 Comparison with other approaches

There are two general approaches inside the fair-ordering category. Almost all of them can be split into the two components we used in the last section. The first approach uses the notion of ordering linearizability, represented by protocols including Pompē [160], Decentralized Clock Network(DCN) [87] and Lyra [157]. At the high level, every transaction first goes through a transaction submission component in order to lock a timestamp. For every transaction, Pompē devises its broadcast protocol to ensure everyone agrees on the median timestamp with $O(n^2)$ complexity. DCN created a timestamp agreement protocol that has $O(n^3)$ complexity. Lyra designs a Validated Value broadcast protocol based on Binary Agreement protocol [113] which take $O(n^2)$ complexity. All of them requires all nodes to receive the transaction. For the consensus component, they use some modified version of standard consensus protocol for replicating transaction along with the timestamp. Both Lyra [157] and

DCN [87] claim they do not assume a synchronous clock. Those protocol can easily incorporate threshold encryption like content-agnostic approach to provide further front-running protection.

The second common notion for fair ordering is batch order and differential order fairness [48, 96, 97]. They rely on relative transaction ordering to derive the final canonical ordering. In the transaction submission component, all transaction is assumed to be propagated in a dissemination network which ensures all honest nodes to observe everything eventually. This is the necessary requirement for those protocols, so that every honest node can create its local logical ordering to decide what can be ordered safely. In the consensus component, because there are possible Condorcet cycle among the transaction, the consensus protocol needs to handle more works to ensure only subset of observed transactions can be confirmed.

Table 5.1 summarizes a comparison among each algorithm, and its analysis is presented in Appendix 5.7. Like Quick Fair Ordering [48], we include the transaction size L in the table, since transactions might take significant size as they become more complex. The number of transactions T is a part of analysis, because clients' traffic is out of protocol control, this is beneficial to take it into consideration to understand how system behave in response to workload. λ denote the size of signature and hash value, and is typically 32 bytes. Compared to others, Travelers reduces communication complexity in both submission and consensus component. Travelers is the first fair ordering protocol that achieves sublinear communication complexity in the transaction submission phase, it is because we use probabilistic fair ordering that allows some ϵ error. Since Travelers is family of protocols, in Table 5.1 we include two versions of it. Travelers-Speed is optimized for faster latency to traverse a path. Travelers-Light is optimized for the lowest communication complexity. Not only Travelers is efficient, it is also flexible because both the routing protocol and the BFT consensus can be replaced easily by protocols with similar properties. For example, all DAG based protocols can use Travelers to get fair ordering, like BullShark [141], Tusk [64]. In the routing protocol, both hub size and path length can be tuned; the assignment functions can also be designed to work with a topology consisntent to distributed validators technology like Obol [120] to achieve

decentralization and threshold encryption.

Table 5.1: Difference of ordering notion and complexity

Fairness Notion	Algorithm Complexity	Security Assumption	Tx Submission Per payload Complexity	Total T Payload Complexity
Batch-Order-Fairness [97]	Themis	1/4	$O(nL)$	$O(nTL + n^2T\lambda)$
Differential-Order-Fairness [48]	Quick o.-f.	1/3	$O(n^2L + n^3\lambda)$	$O(n^2TL + n^3T\lambda)$
Ordering Linearizability [160]	Pompē	1/3	$O(n^2L)$	$O(n^2TL + n\lambda)$
Prob. Ordering Linearizability	Travelers-Speed	1/3	$O(c\log^2(n)L)$	$O(c\log^2(n)TL + n^2\lambda)$
Prob. Ordering Linearizability	Travelers-Light	1/3	$O(c\log(n)\lambda + L)$	$O((c\log(n)\lambda + L)T + \log(n)n\lambda))^a$

[a]By using iterative routing from Appendix 5.8.2, and BigDipper for consensus

In Section 5.4.4, we delineates the adversary action spaces, and in Section 5.5.1 we reason about how Travelers defends against MEV attacks.

5.2 Background

5.2.1 BFT problems

A Byzantine Fault Tolerant(BFT) protocol is a system of n nodes that reaches consensus on a common state. Clients submit transactions to some honest nodes in order to modify the state. To reach agreement, nodes communicate among in a network which can be modeled as asynchronous, partial asynchronous and synchronous. Most real system choose to use either asynchronous or partial asynchronous network model. In a partial asynchronous network, there is a notion of Glocal Stabilization Time, after which all transaction arrive at known bounded duration. Whereas in the asynchronous network model, messages among nodes are eventually delivered, but its time is unknown. A BFT protocol has to satisfy the safety and liveness properties:

- Safety: At any time, the ledger of every pair of honest nodes is a prefix of another

- Liveness: Transactions submitted by honest clients are eventually added into the ledger of honest nodes.

There is a lower bound for the number of malicious nodes. For most standard BFT protocol, it is $n \geq 3f + 1$. For Themis, it is effectively $n \geq 4f + 1$.

5.2.2 Condorcet Cycle and Relaxation

Drawn from Social choice theory, a Condorcet cycle is a phenomenon that groups of transitive relation results into a final non-transitive relation. For example, suppose three voters a, b, c rank three commands, c_1, c_2, c_3. A Condorcet cycle arise if the following ranks are given: $c_1^a \prec c_2^a \prec c_3^a, c_2^b \prec c_3^b \prec c_1^b, c_3^c \prec c_1^c \prec c_2^c$, where the superscript denotes the voter and $\prec$ is some transitive relation. At the end, there is no conclusion which command is ranked the highest. Due to this fact, previous work [98] shows that it is impossible to define fairness based on reception order and instead define batch-order-fairness, such that all transactions in the cycle are delivered altogehter. However, sometimes when there is Condorcet cycle spanning across multiple blocks intervals, a transaction might have been already recognized by the consensus, but needs to wait for multiple blocks before confirming the transaction. It is due this fact, the consensus protocol based on batch order fairness needs not only just the data availability, it needs to decide what transactions are allowed to confirm. Ordering linearizability avoids this problem and has been pointed by Pompē, Lyra, DCN [87,157,160]. In Themis, nodes and the leader exchange not only the transactions list, but also the leader is responsible for aggregating the updates messages from the nodes. In order to prevent a violation to a weak notion of liveness, special techniques like unspooling or deferred ordering are introduced to Themis.

5.3 Probabilistic Order Fairness

We introduce a high level notion called Probabilistic Order Fairness. We start by introducing Ordering Linearizability, and based on that we state where the probabilistic component arise

in the new notion. In simple term, Ordering Linearizability requires that

- if the highest timestamp of a transaction tx_a provided by any honest nodes is lower than the lowest timestamp of a transaction tx_b provided by any honest nodes, and if both transactions are committed, then tx_a will occur before tx_b in the final totally ordered ledger.

In Pompē, it is implemented by locking timestamps for all commands, so that the ordering linearizability can be enforced. To lock a transaction, all nodes need to first receive the transaction, then broadcasts and agree on the median timestamp, which is both upper and lower bounded by timestamp generated by honest nodes.

However, when Pompē defines the fairness, it is implicit that every node has a vote on the final results, which is achieved by having everyone receiving the transaction. In our case, only nodes along the path will receive the transaction, and there is a case that some transaction whose delivery is entirely determined by the malicious hubs. So in the probabilistic version, we made the following modification based on the previous definition:

- For every new block in the BFT protocol, there is ϵ probability that some transaction tx_c provided by a corrupted path can be added into the block at any locations, for the remaining $1 - \epsilon$ probability, if timestamp of a transaction tx_a provided by a regular path is x duration before the timestamp of a transaction tx_b provided a regular hub, and both transactions are committed, then tx_a will occur before tx_b in the final totally ordered ledger.

where x is the duration parameter affected by the length of the path, clock mismatch among all nodes and exact routing algorithm. We elaborate in Section 5.4.

Travelers is a set of possible realization for such probabilistic notions, and it is likely other systems might achieve the similar properties. Compared to Pompē, Travelers relaxes the assumption that everything is received by everyone, hence it has to add new definition to cover the case.

5.3.1 Error Interpretation and Probability in Consensus

The probabilistic notion relax the condition which a protocol has to satisfy, and therefore allows us to simplify protocol. The error probability ϵ can be set in two ways depending on the interpretation. Algorithmically, we can set the target ϵ to be negligible by letting $\epsilon = O(1/n^c)$ where n is the number of nodes. We can also interpret ϵ economically. Suppose the cost of attack per block is C, and the revenue from block reordering is D, then as long as $\frac{C}{\epsilon} >> D$, there is no benefits for adversary to initiate an attack.

We note that there has a history to introduce probability into the consensus properties. In the classic partial synchronous BFT protocols, like Hotstuff, PBFT, the liveness property guarantees an eventual inclusion of a transaction depending on if the next leader is honest, which is a probabilistic statement. For asynchronous BFT protocols, both safety and liveness are probabilistic requiring a random coin for ensuring the correctness. In the next section, we elaborate on ordering linearizability, which is used for developing a specific version of fairness.

5.4 Routing Protocols

At the high level, the goal of a routing protocol is to deterministically assign nodes to hubs, and to create a fixed number of random paths for transactions to traverse. After a client submits copies of a transaction to expected number of paths, it is the goal of the routing protocol that there is a constant probability that at least one of the paths is made of entirely regular hubs. Similarly for adversary, the protocol ensures there is negligible probability that there is a path made of entirely corrupted hubs, even if the adversary sends the transaction to all possible paths. Those paths are randomly generated for preventing adversary from manipulating the creation process. In every block, there are only a fixed number of path available regardless of transactions, so that no adversary can grind its transactions to discover the corrupted path. To achieve the above property, the protocol design makes use a key assumption from the standard BFT protocol that there are at most 1/3 malicious nodes. In

the following, we show how to use a common technique in computer science called boosting to design the routing protocols. Due to the size limitation, we state the properties of the routing protocol in Appendix 5.8.1. The path traversal mechanism is stated in Appendix 5.8.2. The Assignment functions in Appendix 5.8.3.

5.4.1 Technique of Boosting

Suppose there are two types of hubs: a regular hub which occurs at probability of p_h, and a corrupted hub with the probability p_d, such that $p_h > p_d$. Since $p_d > p_k$, we can represent $p_d^\rho = p_h$, for some $\rho < 1$. By arranging the term, we got $\rho = \frac{\log 1/p_h}{\log 1/p_d}$. Suppose all path has a length of k, the probability of a path of entire honest nodes is therefore $g_h = p_h^k$, and for a path of corrupted hubs is $g_d = p_d^k$. As length of path k increases, the probability of occurrence for both type becomes negligible. Although both are very small, there is a large gap between the two small number. With some calculation, we can show $g_d^\rho = g_h$ or equivalently $\frac{\log 1/g_h}{\log 1/g_d} = \rho$. Suppose all paths are created independently, as the client can try distinct paths linear number of time, the probability for both type increases linearly. Up to some point after trying L different paths, there is a constant probability the client would hit a honest path. But since there is a large gap between the two probabilities g_h and g_d. It is possible to find a L, such that the product $g_h L = O(1)$, but $g_d L$ is still negligible. By keeping the total number of possible path low enough, such that there is no much room beyond L tries, we can keep the chance for any adversary to find any corrupted paths negligible. In the following, we demonstrate two different designs of the routing protocol using this common technique. However, as the hub size and path length can be adjusted, there are much more possible design there.

5.4.2 Path of Singleton

Suppose every node becomes its own hub, then by BFT adversary assumption, $p_d = 1/3$ and $p_h = 2/3$. As the result, $\rho = 0.369$. Because we want $g_d = p_d^k = \frac{1}{n^\tau}$ be negligible, where $(\tau \geq 1)$. We can rearrange the term to get the path length $k = \frac{\tau \log n}{\log 1/p_d}$, then probability of

both paths can be computed as

$$g_h = {p_h}^{\frac{\tau \log n}{\log (1/p_d)}} = n^{-\frac{\log (1/p_h)}{\log (1/p_d)}\tau} = n^{-\rho\tau} = n^{-0.369\tau} \tag{5.1}$$

$$g_d = n^{-\tau} \tag{5.2}$$

The second equality in Eq(5.1) comes from the logrithmic rule $a^{\log b} = b^{\log a}$. If a client retry $n^{0.369\tau}$ different paths, there is a constant probability which one of the transaction will go through a regular path made of entirely honest nodes. By limiting the total number possible paths to $o(n^{\tau})$, the probability is always negligible for adversary to get a path made of entirely malicious nodes.

With those numbers, we can easily build a simple protocol with communication complexity of $O(n^{0.369(1+c)})$ with $\epsilon = 1/n^c$. The system allows n total possible path, such that every node is a starting point of some path. By letting $\tau = c + 1$, although adversary can try n different times, by taking the union bound, the overall probability is n^{-c}. On the other hand, if a client sends transaction to $n^{0.369(c+1)}$ nodes. The probability with a regular path is 1. However, in order to ensure the communication complexity be sublinear, we require $c < 1.71$. To further reduce the complexity, we need to increase the hub size.

5.4.3 Non-Singleton hubs

In the last section, p_h and p_d only differs by a constant factor, so the approach we use to differentiate them is by increasing the path length in order to increase the gap between the two probability. However, if p_h and p_d already differs by more than constant factor, we can reduce the length of path and therefore decrease the number of tries for transactions to find a regular path.

Suppose now each hub is made of q distinct nodes and every node is draw independently with replacement. We know that it is binomial distribution with mean equal to 2/3, and the binomial distribution is a good approximation to a distribution when nodes are draw without replacement if n is much larger than q. Similarly for the distribution for malicious nodes, it can also be modeled as a binomial distribution with mean equal to 1/3. The left diagram of

Figure 5.2 displays both distributions, when $q = 6$. Since the hub is not singleton, we need to define a threshold, t, which is the minimal number of signatures to get approved by the hub. A small t makes it easy to pass a hub, but it also increases the chance that a hub gets corrupted. We start with $t = \frac{2n}{3}$. In the left Figure 5.2, we denotes the mass of probability for passing event in the shaded area. However, as we increase the size of hub, as shown in Figure 5.2(b), while keeping $t = \frac{2n}{3}$, the difference between p_h and p_d becomes much more significant. Since t is equal to the mean of binomial distribution for honest nodes, $p_h = 1/2$. For computing p_d, we can derive it from Chernoff bound ([26] Theorem 1), where D is the Kullback-Liebler distance, and $p = 1/3$

$$p_d \leq \exp\{-qD(2/3||1/3)\} \tag{5.3}$$

$$= \exp\{-q(\frac{2}{3}\log\frac{2}{3p} + \frac{1}{3}\log\frac{1}{3(1-p)})\} = \exp\{-\frac{q}{3}\} \tag{5.4}$$

By letting $q = 3\log n$, we have $p_d = n^{-1}$. To arrive at $\epsilon = 1/n^c$, there are two approaches. We can let the path length $k = c$. Or alternatively, we could let $q = 3(c+1)\log n$, let k be a small constant. If we choose choose $k = 2$ and let $t = 2q/3$, then $p_h = 1/2$. The client only needs to submit its transaction to four different path to ensure one regular path.

In summary, by letting $t = 2q/3$ and k be a small constant, we can achieve $O(c\log n)$ communication complexity to ensure an error probability of $\epsilon = 1/n^c$. However, the protocol does not have to set $t = 2q/3$, the threshold can be adjusted by the system to balance between p_d and p_h.

5.4.4 Timestamp rendered by types of paths

The compositions of a path affect the properties of final timestamps. In this section, we delineate all types of hubs, and analyzes how they affect the properties of a path and the resulting timestamps. Understanding them is important to reason about how Travelers defend against the MEV attacks in the Section 5.5.1.

We start by examining the definition of hubs. Suppose the size of a hub be q and the passing threshold be t. Depending on how many nodes of each type are present in the hub

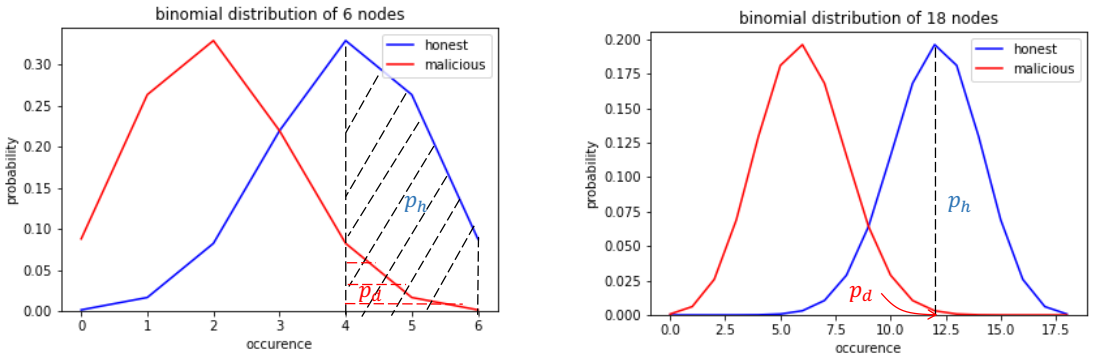

Figure 5.2: Hub Distributions

compared to t. Let's define an binary event **PASS** by comparing if certain types of node alone can approve the transaction, i.e. if the number of certain type nodes is at least t. We can define four types of hubs, as shown in in Figure 5.3.

A hub of Type 3 occurs when both honest and malicious nodes alone can approve the transaction. But since we want to decrease the chance of a corrupted hub, we can simply rmove this case by setting $t > q/2$. A hub of Type 1 is equivalent to our previous definition

	Events	Honest Nodes Alone: PASS (regular hub)	Honest Nodes Alone: NOT PASS
Malicious Nodes Alone	NOT PASS	Type 1	Type 2
Malicious Nodes Alone	PASS (corrupted hub)	~~Type 3~~	Type 4

Figure 5.3: Four types of Hubs

Outcomes	Types of path
True timestamp	Path made of entirely Type 1 hub
Manipulated timestamp	Path made of entirely Type 4 hub
Advancing timestamp	Path made of any consecutive Type 4 hub OR Path whose last hub is Type 4
Delaying timestamp	Path made of any hub with Type 2 or Type 4

Figure 5.4: Timestamp Types to paths

of regular hub, and a hub of Type 4 is equivalent to our definition of corrupted hub. The last type of hub we have not explored is Type 2, when a hub is in the situation of impasse if neither honest and malicious nodes alone can approve the transactions, and we want to avoid it for liveness reason.

After defining all types of hub, we can start analyzing types of path made of those hubs. In a path made of corrupted hubs, a transaction does not need to be sent on the network to get approval, because the adversary who holds the private keys can approve everything in a central location, any possible timestamp can be manipulated as long as the timestamp does not violate consensus. But this is rare by lowering ϵ. Similarly, a regular path is made of regular hubs. It produces a true timestamp reflective of current network and clock condition. But the most common case are paths made of a mixture of corrupted and regular hubs.

When adversary wants to delay an transaction, as long as there exist one hub along the path that honest nodes cannot approve the transaction alone, the adversary can control to delay the transactions. A delayed timestamp must have a timestamp later than the earliest true timestamp, otherwise it is advancing timestamps of the transaction.

Sometimes an adversary wants to speedup the delivery and therefore advance the timestamp. Suppose the last x hubs on some path are corrupted type 4 hubs, adversary can use a simple method to make it deliver faster, by reusing the last timestamp from honest hubs. There is another way to advance the timestamp if there is a consecutive corrupted hubs on the path (not necessarily from the end). Since all malicious nodes can be controlled by a single adversary, the first corrupted hub can create the approval for all the following corrupted hubs, and therefore help the transaction moves faster than it should have. Figure 5.4 summarizes four types of outcomes depending on how a path is constituent of.

Depending on the purpose of adversary, it can selectively choose either tools to advance or delay some transactions. For sandwich attack, adversary can setup a sandwich trap, and then either advance or delay some victim transaction, in order for the victim fall into the trap. However, by devising the ordering rule properly, we can prevent the adversary from using delaying timestamp entirely. The idea is to always use the earliest timestamp as

the canonical timestamp, we will elaborate more in Section 5.5.3 and 5.5.1. Then the only tactics which adversary can use is to advance the timestamp of the victim's transaction. But it makes the sandwich attack much harder because adversary has to setup the trap earlier than victim's transaction. By the property of automated market maker, the adversary incurs a loss if attack is not successful.

5.4.5 Encrypting the transaction

However, if we assume adversary has an advanced infrastructure that is magnitude faster than regular Internet backbone, there is a chance that adversary can peek the content of the victim transaction and still have time to setup a sandwich trap. In this section, we look at how to incorporate encryption to delay the time when adversary can know about a transaction, therefore making it impossible to setup a trap early enough.

There are multiple ways to implement such encryption scheme. At the high level, there must be at least one hub that can decrypt the entire transaction. If the hub contains only one node, the decryption is simple. However, if the hub size has q nodes, there are exponential signature combinations. Unless q is small enough, otherwise we need a threshold encryption scheme to make it efficient. Let's define a decryption set S that contains hubs capable of performing decryption. The hubs can either be group of nodes capable of threshold decryption, or hub whose size is small enough. For example 6 choosing 5 contains only 6 possibilities. For any path that has the ability to perform decryption, it must contain at least one hub from the decryption set S. Ideally the decryption hub is at the end of the path, so that the routing protocol can hide the transaction as long as possible. If we use iterative path traversal method discussed in Appendix 5.8.2, the initiator needs to provide all approval signatures from the beginning of the path to convince the decryption hub that enough time is spent on traversal. A transaction can be encrypted layer by layer if multiple hubs in the decryption set is available. It offers a better protection in case one decryption hub is corrupted. Our scheme is slightly different from the threshold encryption used by Shutter or anything that uses commit-and-reveal scheme. In our scheme, the plaintext will

be available after the last hub, so data can be read immediately without further processing.

5.5 Travelers

Travelers is a consensus protocol that provides probabilistic fair ordering notion we introduced in Section 5.3. In this section, we first provide an overview on the strategy which Travelers use to prevent ordering attacks. We then show how to design the ordering rule to support such strategy by removing delayed timestamps. Because ordering linearizability decouples the ordering from the underlying consensus protocol, the BFT protocol is very modular. However, it is important to ensure the BFT protocol has censorship resistance. We provide an overview on the category of BFT protocols that satisfy the requirements.

5.5.1 Strategy to prevent reordering attacks

In Section 5.4.4, we show that four types of locked timestamps are generated depending on the paths they are from, as shown in Figure 5.4. For an adversary, since the true timestamp is undesirable, and manipulated timestamp is bounded with $O(n^{-c})$ probability, the adversary only has two tools for arranging its attack: by advancing timestamps and by delaying timestamps. In the Section 5.5.3, we show by designing proper ordering rules, we can remove the delayed timestamps from the attack vectors, as long as there is at least one true timestamps being committed by the consensus protocol. So the only tool which an adversary can affect the ordering is using advanced timestamps by speeding up its or victim's transactions. However, by using encryption in Section 5.4.5, we can ensure that the clients' transactions have completed most of its traversal on the path. At last, we note it is up to clients to decide how much front-running protection being offered, a worried client would sends transaction to many paths to ensure the true timestamps from the regular hub is not censored, and the client would use the most secure encryption discussed in Section 5.4.5 to make sure the transaction is not front-runnable.

5.5.2 $1-\kappa$ Probabilistic ordering linearizability Property

We now formally define the probabilistic properties which overall Travelers has, based on the probabilistic fair ordering notion. The proof is provided in Appendix 5.9.1.

- For every new block in the BFT protocol, there is $1/n^c$ probability that some transaction tx_d provided by a corrupted path can be added into the block at any locations. For the remaining $1-1/n^c$ probability, if there is $1-\kappa$ probability that the timestamps of both transaction tx_a, tx_b provided by some regular paths are committed, then with $1-\kappa$ probability the following fairness notion holds: if the timestamp of tx_a is $4k\Delta+2\delta$ duration before the timestamp of tx_b, then tx_a will occur before tx_b in the final totally ordered ledger, where k is the number of hubs in a path.

We added a new probability $1-\kappa$ to ensure that the protocol covers the case that both transactions are committed. Im compute the resolution $4k\Delta + 2\delta$, the protocol uses the iterative routing, which doubles the latency to traverse a path. If we choose to use a leaderless asynchronous BFT protocol which is inherently censorship resistance, then κ can be decreased to negligible. DispeseredLedger [151] invents a technique called inter-node linking that guarantees every collected transaction is delivered. Similarly, all DAG-based protocols can fetch transactions that were submitted but not confirmed in the past, and confirm them in the new blocks. However,it requires longer confirmation latency, which we discuss in Section 5.5.3.

5.5.3 Ordering Rule

When a client sends a transaction on multiple paths, the same transaction would have been locked many timestamps. The ordering rule decides which one becomes the canonical timestamp used by the totally ordered ledger.

The core observation is that we always have a set of true timestamps committed in the final ledger. By letting the earliest time among them as the canonical timestamps, we filter

out all any delayed timestamps. By definition any timestamps that is earlier than the earliest true timestamp belongs to the category of advancing timestamps.

Note that the protocol does not need to process anything to specify which timestamp is canonical for each transaction. It is because if a party retrieves all confirmed transactions from the BFT protocol, the party can collects all the timestamps corresponding to a transaction and determine the canonical timestamp by a sorting algorithm. For this reason, consensus is simply a matter of arriving agreement on a stream of bit, which is much simpler than deducing Condorcet cycle and mitigating the nested loops.

Sometimes there might be a new transaction that locks to a timestamp earlier than what is already confirmed in the past. There are two methods to deal with it. The simplest approach is to add the transaction to the latest unconfirmed block with a new timestamp as early as possible without violating the block boundary. Another way would be altering confirmation rule at the application layer, such that the most recent x block are subject to changes, depending the timestamps of transactions in the new block. However, we have to modify the property a bit to cover the effect when there is a corrupted path.

5.5.4 Consensus and Censorship Resistance

Travelers is intended to be modular systems. However, there is another desired property called censorship resistance, which is not covered by most BFT protocol. It is important for Travelers, because if the leader is malicious, it can selectively remove the earlier timestamps, so that only the delay timestamped manipulated by the adversary is regarded the canonical timestamp for the transactions. Then the adversary can setup a sandwich and delay the victim transactions to fall into the trap.

It can be fixed by requiring the underlying consensus protocol be censorship resistance. For example, most leaderless protocols are censorship resistant. But they all have at least communication complexity of $O(n^2)$ because of all-to-all communication pattern. Both DispersedLedger and Dumbo-MVBA [105] use erasure code to make agreement on a stream of bits without downloading the whole data. Dumbo-MVBA has a communication complexity

of $O(n^2)$. A concurrent submission called BigDipper provides a more systematic presentation, and offers $O(n^2)$ or $O(n \log n)$ communication complexity.

5.6 Conclusion

We present Travelers composed of a flexible routing protocol and a censorship resistant BFT protocol. It achieves a notion called probabilistic ordering linearizability with $O(c \log(n)L + n^2)$ communication communication complexity. However, the probabilistic notion can also be applied to batch-order-fairness. It will be interesting to discover new protocols based on relative transaction order.

5.7 Complexity Analysis

5.7.1 Themis

Themis requires a desemination network, such that everyone can receive the transaction of size L, therefore the total communication complexity is $O(nL)$. In the consensus part, suppose there are T total transaction, each node has to provide the relative ordering among the T transactions to the leader. So the leader has a complexity of $O(nT\lambda)$, where λ is the hash digest to used to denote a transaction among the relative ordering. Because the leader in the practical non-SNARK version needs to send the relative ordering of all nodes to every node, the total complexity is $O(n^2T)$ for consensus, and $O(nTL)$ for transaction submission.

5.7.2 Quick Ordering Fairness

Quick Ordering Fairness protocol consists of a BCCH (Byzantine FIFO consistent broadcast channel) and a VBC (validated byzantine consensus). To send one transaction, BCCH requires $nL + n^2\lambda$ communication complexity, as provided by the Quick Ordering Fairness paper [48] itself. The VBC part is modular, the most efficient asynchronous consensus requires $O(n^2)$. Although it can also be used with partial synchronous protocol, its complexity is dominated by the BCCH.

5.7.3 Pompē

The Pompē protocol also requires all nodes to receive a transaction. There is $O(n^2)$ term, because there is a **Sequence** message to all nodes, and the message contains collected timestamps from $O(n)$ nodes. It is required to let every node to make agreement on the median timestamp. For Pompē, it uses a standard leader based Hotstuff protocol, which does not offer censorship resistance, and its the complexity is simply $O(n\lambda)$. Overall the complexity for T transactions is $(n^2TL + n\lambda)$.

5.7.4 Travelers

In travelers, a transaction passes the the submission protocol if it can traverse a path. In Section 5.4.3, we show if the hub size is $O(\log n)$, and the length of path is constant. A client can send the transaction to constant number of path. The overall complexity per transaction is $O(\log n(\lambda)$.

In Table 5.1, we list two versions of Travelers. In Travelers-Speed, the protocol uses recursive routing, so that the same transaction is sent every node in the hub, therefore it is $O(\log(n)(\lambda + L))$. Recursive routing is useful for reducing the time for the traversal time. See Appendix 5.8.2 for more information. In evaluation the complexity for consensus, we can either use VABA [20] or Dumbo-MVBA [105], they require $O(n^2)$ communication for reaching consensus on a stream of bit. In BigDipper from the concurrent submission, it contains a protocol with consensus complexity of $O(n^2)$.

In the other version, Travelers-Light optimizes for low communication complexity. It uses iterative routing, so only the hash of transaction needs to be sent to nodes, which is used for signing. The initiator in the recursive routing can just collect the signatures, and deliver the transaction along with the signatures to the consensus protocol. Therefore the communication complexity per transaction is $O(c \log n\lambda + L)$. In BigDipper from the concurrent submission, we show it is possible to design a censorship resistance protocol with total communication complexity of $O(\log(n)n)$. Each node only needs to spend $O(x/n+\log n)$

complexity in consensus, where x is total amount of data required to be reached agreement on after traversing the routing protocol from all nodes, which equals to $\log(n)TL$.

The overall complexity for T transaction is therefore $O((c \log n\lambda + L)T + \log(n)n)$.

5.8 Routing Protocol

5.8.1 Routing Protocol Properties

The routing protocol satisfy the following properties:

- $O(1)$ Probabilistic Regular path: For any transaction, there is $O(1)$ probability that the protocol can generate a set of timestamps from a path made of entire regular hubs.

- ϵ Probabilistic Corrupted path: In each block, there is only ϵ probability that an adversary can arbitrarily create certificate with any metadata for all transactions in the block.

5.8.2 Path Traversal Mechanism

In this section, we delineate the procedure how a transaction starts to be processed by the system and how a hub delivers its approval to the next hub. When a client sends a few copies of its transaction to the routing protocol, with high probability there is at least one honest node following the protocol to forward the transaction to the next hubs. Let's call the head of a path the **initiator**. The initiator is always a single node. If the initiator is trusted by the client, it can simplify protocols like encryption in Section 5.4.5.

Since a threshold of signatures are required for an approval from a hub, we need an efficient way to aggregate those signatures, because all signatures need to be carried to the end, which can be verified to show the transaction has traversed the entire path. We choose to use BLS signature for its property to combine signature. Once the first nodes in the first hub receives the transaction, honest nodes in the hub need to combine the signature and deliver them to nodes in the next hub.

Iterative vs. Recursive routing

Suppose hub A is a regular hub that has sufficient honest nodes to approve the transaction. There are multiple solutions to solve the problem. We can categorize them as either iterative or recursive solution. When using iterative queries, like DNS, the initiator sends hashes of transactions to every nodes in the hub, and waits to collect any aggregate signature. The initiator repeats for all hubs. This approach increases the latency to finish a query and increases the workload for the initiator. But it has benefits of being simple. Because the initiator is the single entity, we can easily add more features like encryption we mentioned later. It also offers a benefits of communication efficient, because to create a signature, the node does not need the entire transaction, but just its hash digest.

On the other hand, in a recursive routing, the initiator only participates in the beginning. Once sufficient honest nodes in the first hub receives the transactions, the hub works with the next hub directly to deliver the signature. The solution requires all honest nodes in the hub to broadcast its signatures to all nodes in the next hubs, once sufficient honest nodes in the next hub collect sufficient signatures, they combine the signature from previous hub locally and repeat the same procedures until the end of a path. This approach is faster in latency but requires q^2 communication complexity per hub. All signatures along the hub needs to be carried to the last hub, so that the consensus is able to verify that the transaction has traversed sufficient hubs.

5.8.3 Assignment functions

So far, we simply assume there is a hub assignment function from nodes to hub, and a path assignment function from hubs to path. In this section, we provide a general description on those functions. Let $\mathcal{D}$ denotes the set of all nodes, $\mathcal{H}$ denotes be the power set (set of all possible subset) of $\mathcal{D}$. Let's denote the hub assignment functions as $f : \mathcal{D} \to \mathcal{H}$. Let k be the path length for all paths, then the path assignment function is $g : \mathcal{H} \to \mathcal{H}^k$.

Multiple ways are possible to create those assignment function. One method requires

the use of a random source that produces a random number in every block, let's denote the randomness as r. For block number b, node i can computes q different members in the j-th hub by taking $Hash(i, j, k, b, r)\ mod\ n$, where n is number of nodes, and $0 < k < q$ (if one node is selected twice, then replace k with $2k, 3k..$ until a distinct node is found). Then repeat the procedure for $0 < j < k$, to obtain all nodes on its path. Because the randomness is globally visible by every nodes, all nodes would be able to obtain all paths by computing locally. Since eac Randao in Ethereum currently offer such randomness.

Another method does not require a global visible randomness, it can be achieved with verifiable random function (VRF)). A VRF is cryptographic tool that takes inputs of a seed and secret key of a key pair, and outputs a random number and a proof that the number is generated correctly. Every node then uses its private key as the source of randomness and generate all the nodes on its path.

Because the node assignment function can be very generic, we can add some structure to the function for enabling some features. For example, if we know some group of nodes already have the threshold signature setup, and the group of nodes has sufficiently high security assumption. We can group those nodes as a super node, so that when creating a new hub, if the node assignment function pick the super node, it would not select other nodes that is outside the group to be part of the new hub. In Obol [120], it is possible to have a threshold setup with 7 honest out of 10 nodes.

We can also change the path assignment functions to improve encryption scheme from Section 5.4.5. For example, the assignment function always include a hub from the decryption hub set in the end of every path. The assignment function can also change the size of hubs on a path, as long as the final probabilities for both corrupted path and regular path satisfy the protocol requirement. Some smaller hubs are therefore eligible to be a part of decryption hub set.

5.9 Travelers Consensus

5.9.1 Property Proof

An adversary can arbitrarily add new transaction into any location in the ledger if it can find a corrupted paths. Otherwise it will only be able to advance or delay the timestamps. By the properties of the routing protocol, there is the error probability of $\epsilon = 1/n^c$ for some $c \geq 1$ for that to happen. In the rest of case, no such event can happen.

Suppose there is $1 - \kappa$ probability for a transaction tx_a to get included in the ledger, then by the routing protocol there is $O(1)$ probability that this transaction has a regular path. Similarly for the other transaction tx_b. Suppose every path has a length of k hubs and network latency is bounded by Δ. In reference to the true timestamp produced by the regular path, due to the network uncertainty, those two transactions are separable as long as the true timestamps between the two differ by $2k\Delta$.

Suppose we choose the iterative routing mechanism to move from hubs to hubs which is the longer compared to the recursive routing, the total time spent on the traversal is doubled as $4k\Delta$, because of the round trip.

Suppose the mismatch among nodes' clock is bounded by δ, then the maximal mismatch between two true timestamps are 2δ. Hence if transaction tx_a is committed and separated by duration $4k\Delta + 2\delta$ from the transaction tx_b, then tx_a is ordered before tx_b.

BIBLIOGRAPHY

[1] Aced library.

[2] Beaker browser. https://beakerbrowser.com/.

[3] Compact block relay. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki.

[4] Cryptocurrency market capitalization. https://coinmarketcap.com/.

[5] Cvxpy. https://www.cvxpy.org/.

[6] Distribution of bitcoin mining hashrate from september 2019 to april 2021, by country. https://www.statista.com/statistics/1200477/bitcoin-mining-by-country/.

[7] Dsn bitcoin monitoring. https://www.dsn.kastel.kit.edu/bitcoin/videos.html.

[8] Dtube. https://d.tube/.

[9] Ethereum. https://ethereum.org/en/.

[10] Ethereum name service. https://ens.domains/.

[11] Fibre. https://bitcoinfibre.org/.

[12] Goldfish library.

[13] Interplanetary file system. https://ipfs.io/.

[14] Namecoin. https://www.namecoin.org/.

[15] Non-fungible tokens. https://ethereum.org/en/nft/.

[16] Pytorch.

[17] Raiden network. https://raiden.network/.

[18] Steemit. https://steemit.com/.

[19] Yacy. `https://yacy.net/`.

[20] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Validated asynchronous byzantine agreement with optimal resilience and asymptotically optimal time and word communication. *arXiv preprint arXiv:1811.01332*, 2018.

[21] Mustafa Al-Bassam. Lazyledger: A distributed data availability ledger with client-side smart contracts. *arXiv preprint arXiv:1905.09274*, 2019.

[22] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities. *arXiv preprint arXiv:1809.09044*, 2018.

[23] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities, 2019.

[24] Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2004.

[25] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. Prime: Byzantine replication under attack. *IEEE transactions on dependable and secure computing*, 8(4):564–577, 2010.

[26] Richard Arratia and Louis Gordon. Tutorial on large deviations for the binomial distribution. *Bulletin of mathematical biology*, 51(1):125–131, 1989.

[27] Sepehr Assadi and Chen Wang. Exploration with limited memory: streaming algorithms for coin tossing, noisy comparisons, and multi-armed bandits. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1237–1250, 2020.

[28] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[29] John Augustine, Soumyottam Chatterjee, and Gopal Pandurangan. A fully-distributed scalable peer-to-peer protocol for byzantine-resilient distributed hash tables. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 87–98, 2022.

[30] AWS. Aws instance type.

[31] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 913–930, 2018.

[32] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Deconstructing the blockchain to approach physical limits, 2019.

[33] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 585–602, 2019.

[34] blockchain.com. Bitcoin hashrate.

[35] blocknative. blocknative.

[36] bloxroute. bloxroute.

[37] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part II*, pages 435–464. Springer, 2018.

[38] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 435–464. Springer, 2018.

[39] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings 7*, pages 514–532. Springer, 2001.

[40] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[41] Lorenz Breidenbach, Phil Daian, Ari Juels, and Emin Gün Sirer. An in-depth look at the parity multisig bug. 2017. *URL: http://hackingdistributed.com/2017/07/22/deepdive-parity-bug*, 2017.

[42] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018.

[43] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27*, pages 65–97. Springer, 2021.

[44] Vitalik Buterin. 2d data availability with kate commitments.

[45] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[46] Olivier Bégassat, Blazej Kolad, Nicolas Gailly, and Nicolas Liochon. Handel: Practical multi-signature aggregation for large byzantine committees, 2019.

[47] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Advances in Cryptology—CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings*, pages 524–541. Springer, 2001.

[48] Christian Cachin, Jovana Mićić, Nathalie Steinhauer, and Luca Zanolini. Quick order fairness. In *International Conference on Financial Cryptography and Data Security*, pages 316–333. Springer, 2022.

[49] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 191–201. IEEE, 2005.

[50] Emmanuel J Candes and Yaniv Plan. Matrix completion with noise. *Proceedings of the IEEE*, 98(6):925–936, 2010.

[51] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[52] chainlink. chainlink.

[53] chainlink. Data oracle.

[54] Nakul Chawla, Hans Walter Behrens, Darren Tapp, Dragan Boscovic, and K Selçuk Candan. Velocity: Scalability improvements in block propagation through rateless erasure coding. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 447–454. IEEE, 2019.

[55] Sherman SM Chow, Ziliang Lai, Chris Liu, Eric Lo, and Yongjun Zhao. Sharding blockchain. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1665–1665. IEEE, 2018.

[56] coinmarketcap. coinmarketcap.

[57] Andrei Constantinescu, Diana Ghinea, Lioba Heimbach, Zilin Wang, and Roger Wattenhofer. A fair and resilient decentralized clock network for transaction ordering, 2023.

[58] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International conference on financial cryptography and data security*, pages 106–125. Springer, 2016.

[59] Imre Csiszár. The method of types [information theory]. *IEEE Transactions on Information Theory*, 44(6):2505–2523, 1998.

[60] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. *ACM SIGCOMM Computer Communication Review*, 34(4):15–26, 2004.

[61] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M Frans Kaashoek, and Robert Tappan Morris. Designing a dht for low latency and high throughput. In *NSDI*, volume 4, pages 85–98, 2004.

[62] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *International Conference on Financial Cryptography and Data Security*, pages 23–41. Springer, 2019.

[63] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges, 2019.

[64] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.

[65] Shagnik Das. A brief note on estimates of binomial coefficients. *URL: http://page. mi. fu-berlin. de/shagnik/notes/binomials. pdf*, 2016.

[66] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[67] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.

[68] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Everything is a race and nakamoto always wins. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 859–878, 2020.

[69] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Everything is a race and nakamoto always wins, 2020.

[70] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[71] David Easley, Jon Kleinberg, et al. *Networks, crowds, and markets*, volume 8. Cambridge university press Cambridge, 2010.

[72] Eden. Eden.

[73] Sameh El-Ansary, Luc Onana Alima, Per Brand, and Seif Haridi. Efficient broadcast in structured p2p networks. In *International workshop on Peer-to-Peer systems*, pages 304–314. Springer, 2003.

[74] Espresso. Espresso sequencer.

[75] EthHub. Zk-rollups.

[76] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16)*, pages 45–59, 2016.

[77] Dankrad Feist. New sharding design with tight beacon and shard block integration.

[78] Dankrad Feist and Dmitry Khovratovich. Fast amortized kate proofs.

[79] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. *IACR Cryptol. ePrint Arch.*, 2018:1119, 2018.

[80] flashbot. the-cost-of-resilience.

[81] Ardian Foti and Domenico Marino. Blockchain and charities: A systemic opportunity to create social value. In *Economic and Policy Implications of Artificial Intelligence*, pages 145–148. Springer, 2020.

[82] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1187–1201, 2022.

[83] Dankrad Feist George Kadianakis, Ansgar Dietrichs. A universal verification equation for data availability sampling.

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

[85] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous bft protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 803–818, 2020.

[86] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 129–144, 2015.

[87] Lioba Heimbach and Roger Wattenhofer. SoK: Preventing transaction reordering manipulations in decentralized finance. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*. ACM, sep 2022.

[88] James Hendricks, Gregory R Ganger, and Michael K Reiter. Verifying distributed erasure-coded data. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 139–146, 2007.

[89] Toke Høiland-Jørgensen, Bengt Ahlgren, Per Hurtig, and Anna Brunstrom. Measuring latency variation in the internet. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 473–480, 2016.

[90] Ming Jin, Xiaojiao Chen, and Sian-Jheng Lin. Reducing the bandwidth of block propagation in bitcoin network with erasure coding. *IEEE Access*, 7:175606–175613, 2019.

[91] Merija Jirgensons and Jānis Kapenieks. Blockchain and the future of digital learning credential assessment and management. *Journal of Teacher Education for Sustainability*, 20(1):145–156, 2018.

[92] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1353–1370, 2018.

[93] Anil Kamath, Rajeev Motwani, Krishna Palem, and Paul Spirakis. Tail bounds for occupancy and the satisfiability threshold conjecture. *Random Structures & Algorithms*, 7(1):59–80, 1995.

[94] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010.

[95] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 165–175, 2021.

[96] Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-fair consensus in the permissionless setting. In *Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop*, pages 3–14, 2022.

[97] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus. *Cryptology ePrint Archive*, 2021.

[98] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. Cryptology ePrint Archive, Paper 2020/269, 2020. https://eprint.iacr.org/2020/269.

[99] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[100] Uri Klarman, Soumya Basu, Aleksandar Kuzmanovic, and Emin Gün Sirer. bloxroute: A scalable trustless blockchain distribution network whitepaper. *IEEE Internet of Things Journal*, 2018.

[101] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.

[102] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[103] Lens. Lens social network.

[104] Kfir Lev-Ari, Alexander Spiegelman, Idit Keidar, and Dahlia Malkhi. Fairledger: A fair blockchain protocol for financial institutions. *arXiv preprint arXiv:1906.03819*, 2019.

[105] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. Cryptology ePrint Archive, Paper 2020/842, 2020. https://eprint.iacr.org/2020/842.

[106] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30, 2016.

[107] Dahlia Malkhi and Kartik Nayak. Extended abstract: Hotstuff-2: Optimal two-phase responsive bft. Cryptology ePrint Archive, Paper 2023/397, 2023. https://eprint.iacr.org/2023/397.

[108] Dahlia Malkhi and Kartik Nayak. Extended abstract: Hotstuff-2: Optimal two-phase responsive bft. Cryptology ePrint Archive, Paper 2023/397, 2023. https://eprint.iacr.org/2023/397.

[109] Yifan Mao, Soubhik Deb, Shaileshh Bojja Venkatakrishnan, Sreeram Kannan, and Kannan Srinivasan. Perigee: Efficient peer-to-peer network design for blockchains. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 428–437, 2020.

[110] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[111] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. Sprites: Payment channels that go faster than lightning. *CoRR abs/1702.05812*, 306, 2017.

[112] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 31–42, 2016.

[113] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous byzantine consensus with t¡ n/3 and o (n2) messages. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*, pages 2–9, 2014.

[114] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[115] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[116] Kamilla Nazirkhanova, Joachim Neu, and David Tse. Information dispersal with provable retrievability for rollups. *arXiv preprint arXiv:2111.12323*, 2021.

[117] Shutter network. Shutter network.

[118] Till Neudecker and Hannes Hartenstein. Short paper: An empirical analysis of blockchain forks in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 84–92. Springer, 2019.

[119] Luong Trung Nguyen, Junhan Kim, and Byonghyo Shim. Low-rank matrix completion: A contemporary survey. *IEEE Access*, 7:94215–94237, 2019.

[120] Obol. Obol.

[121] A Pinar Ozisik, Gavin Andresen, George Bissias, Amir Houmansadr, and Brian Levine. Graphene: A new protocol for block propagation using set reconciliation. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 420–428. Springer, 2017.

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

[123] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology—CRYPTO'91: Proceedings*, pages 129–140. Springer, 2001.

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

[125] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[126] Michael O Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM (JACM)*, 36(2):335–348, 1989.

[127] Fatemeh Rahimian, Sarunas Girdzijauskas, Amir H Payberah, and Seif Haridi. Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks. In *2011 IEEE International Parallel & Distributed Processing Symposium*, pages 746–757. IEEE, 2011.

[128] Sandi Rahmadika, Siwan Noh, Kyeongmo Lee, Bruno Joachim Kweka, and Kyung-Hyune Rhee. The dilemma of parameterizing propagation time in blockchain p2p network. *Journal of Information Processing Systems*, 16(3):699–717, 2020.

[129] Ranvir Rana, Sreeram Kannan, David Tse, and Pramod Viswanath. Free2shard: Adaptive-adversary-resistant sharding via dynamic self allocation. *arXiv preprint arXiv:2005.09610*, 2020.

[130] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[131] Tom Richardson and Ruediger Urbanke. *Modern coding theory*. Cambridge university press, 2008.

[132] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proceedings first international conference on peer-to-peer computing*, pages 99–100. IEEE, 2001.

[133] Elias Rohrer and Florian Tschorsch. Kadcast: A structured approach to broadcast in blockchain networks. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 199–213, 2019.

[134] Michael Rothschild and Joseph Stiglitz. Equilibrium in competitive insurance markets: An essay on the economics of imperfect information. In *Uncertainty in economics*, pages 257–280. Elsevier, 1978.

[135] Tim Roughgarden. Transaction fee mechanism design for the ethereum blockchain: An economic analysis of eip-1559. *arXiv preprint arXiv:2012.00854*, 2020.

[136] Jerome H Saltzer, David P Reed, and David D Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288, 1984.

[137] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. Bft protocol forensics. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, pages 1722–1743, 2021.

[138] Peiyao Sheng, Bowen Xue, Sreeram Kannan, and Pramod Viswanath. Aced: Scalable data availability oracle. *arXiv preprint arXiv:2011.00102*, 2020.

[139] Sikka. Sikka.

[140] skip. skip.

[141] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022.

[142] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009, 2009.

[143] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. *arXiv preprint arXiv:1908.04756*, 2019.

[144] thedefiant. flashbots-tornado-sanctions-mev.

[145] Dimitris Vyzovitis, Yusef Napora, Dirk McCormick, David Dias, and Yiannis Psaras. Gossipsub: Attack-resilient message propagation in the filecoin and eth2.0 networks, 2020.

[146] Sarisht Wadhwa, Luca Zanolini, Francesco D'Amato, Aditya Asgaonkar, Fan Zhang, and Kartik Nayak. Breaking the chains of rationality: Understanding the limitations to and obtaining order policy enforcement. Cryptology ePrint Archive, Paper 2023/868, 2023. `https://eprint.iacr.org/2023/868`.

[147] wikipedia. softmax.

[148] wondernetwork. wondernetwork.

[149] Bowen Xue, Soubhik Deb, and Sreeram Kannan. Bigdipper: A hyperscale bft system with short term censorship resistance, 2023.

[150] Lei Yang, Vivek Bagaria, Gerui Wang, Mohammad Alizadeh, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Scaling bitcoin by 10,000 x. *arXiv preprint arXiv:1909.11261*, 2019.

[151] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. {DispersedLedger}:{High-Throughput} byzantine consensus on variable bandwidth networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 493–512, 2022.

[152] Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. Sok: Mev countermeasures: Theory and practice, 2022.

[153] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

[154] Haifeng Yu, Ivica Nikolić, Ruomu Hou, and Prateek Saxena. Ohie: Blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 90–105. IEEE, 2020.

[155] Mingchao Yu, Saeid Sahraei, Songze Li, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. Coded merkle tree: Solving data availability attacks in blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 114–134. Springer, 2020.

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

[157] Pouriya Zarbafian and Vincent Gramoli. Lyra: Fast and scalable resilience to reordering attacks in blockchains. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 929–939, 2023.

[158] ZCash. Halo2.

[159] Lihao Zhang, Taotao Wang, and Soung Chang Liew. Speeding up block propagation in blockchain network: Uncoded and coded designs. *arXiv preprint arXiv:2101.00378*, 2021.

[160] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without byzantine oligarchy. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 633–649, 2020.