

© 2022 Ranvir Rana

RESOLVING BLOCKCHAIN TRILEMMAS

BY

RANVIR RANA

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2022

Urbana, Illinois

Doctoral Committee:

Professor Pramod Viswanath, Chair
Professor Bruce Hajek
Professor Rayadurgam Srikant
Assistant Professor Andrew Miller
Assistant Professor Ling Ren

ABSTRACT

Three ideal properties characterize a blockchain: decentralization, security, and scalability. Existing blockchain designs satisfy at most two of the three properties; as examples, consider: Bitcoin is decentralized and secure, but not scalable; EOS, TRON, and several proof-of-stake blockchains are secure and scalable, but not decentralized; Conflux and IOTA are decentralized and scalable, but not secure. Unsuccessful attempts to build scalable, decentralized, *and* secure blockchains, constituting all three properties, have led to a folk theorem known as the *blockchain protocol trilemma*, which states that a blockchain can achieve only two of the three ideal properties. In this dissertation, we disprove this trilemma by proposing **Trifecta**, a blockchain architecture that achieves all three properties by being: (1) decentralized, with a limited amount of compute, storage, and communication resources per node; (2) secure against fully adaptive adversaries with hashing power up to the honest power of the entire network; and (3) throughput scaling near linearly with the number of network nodes. We implement **Trifecta** as a full-stack blockchain in Rust and demonstrate a total throughput of 250K transactions per second. A key component of the **Trifecta** architecture is **Free2Shard**, a distributed resource allocation algorithm that ensures that each shard has sufficient honest mining power under an adaptive adversary. We show the performance of **Free2Shard** to be near information-theoretically optimal by connecting to the classical work of Blackwell approachability in dynamic game theory.

In a separate line of work, a blockchain economics trilemma is posited in the literature: at most two of the following three economic properties are satisfied by any blockchain design – self-sufficient, rent-free, and resource-efficient. We argue that **Trifecta** with **Free2Shard** optimizes the rent and resource cost per transaction and propose **Advocate** to minimize the legal costs associated with bootstrapping PoW blockchains. **Advocate** is a bootstrapping gadget

that achieves optimal chain quality even under adversarial mining majority and can be expanded to work on any parallel chain type blockchains. We demonstrate the robustness of **Advocate** under a 90% adversarial majority via a full-stack implementation.

To my parents, for their love and support

ACKNOWLEDGMENTS

I want to thank my advisor, Prof. Pramod Viswanath, for his constant support and guidance. He has taught me to detect fundamental research problems in areas of practical interest and solve them using a first-principles approach leveraging intuition and design clarity. He has helped me identify the intersection of systems and theory to design better algorithms.

I want to thank my collaborators Prof. Sreeram Kannan, Prof. David Tse, Prof. Songze Li, Prof. Aggelos Kiayias, Vivek Bagaria and Dimitris Karakostas for their invaluable contributions to my research projects.

I want to thank Prof. Bruce Hajek and Prof. Andrew Miller for serving on both my doctoral committee and my qualifying exam committee, Prof. R. Srikant and Prof. Ling Ren for serving on my doctoral committee and, Prof. Yuliy Baryshnikov for serving on my qualifying exam committee. Their guidance helped me a lot during my Ph.D. journey.

I want to thank my friends, labmates and colleagues at UIUC including but not limited to Ashok V. Makkuva, Gerui Wang, Xuechao Wang, Peiyao Sheng, Hyeji Kim, Suryanarayana Sankagiri, Siheng Pan, Viraj Nadkarni, Ashwin Hebbar, Milind K. Vaddiraju, Kaushik Kulkarni, Yashraj Bhosale, Devarsh Ruparelia, Anwesa Choudhuri, Amish Goel, Sourav Das, Amit Agarwal, Mitisha Surana, and Moitreya Chatterjee. All the thought-provoking discussions indirectly helped me sharpen my analytical edge.

Lastly, I would like to thank my parents for their constant encouragement and support during my Ph.D. journey. To them I dedicate this dissertation.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS	viii
CHAPTER 1 INTRODUCTION	1
1.1 Outline and Contributions	3
CHAPTER 2 TRIFECTA UNICONSENSUS ARCHITECTURE	6
2.1 Trifecta Longest Chain	10
2.2 Trifecta	16
2.3 Ancillary modules	17
2.4 Analysis of Trifecta	20
2.5 Implementation	24
2.6 Conclusion	32
CHAPTER 3 FREE2SHARD: DYNAMIC RESOURCE ALLOCA- TION FOR TRIFECTA	34
3.1 Free2Shard DSA	35
3.2 Simulations	49
3.3 Conclusion	52
CHAPTER 4 ADVOCATE	54
4.1 Preliminaries	58
4.2 Advocate: Optimal bootstrapping of PoW Protocols	60
4.3 Advocate with BFT checkpointing	69
4.4 Advocate with BFT checkpointing and External Trust	74
4.5 Advocate: Bootstrapping for Parallel-PoW chains	78
4.6 Implementation	85
4.7 Evaluation	87
4.8 Conclusion	91
CHAPTER 5 CONCLUSION	92
REFERENCES	94

APPENDIX A	ADDITIONAL MATERIALS FOR CHAPTER 2 . . .	102
A.1	Model	102
A.2	Security Analysis	106
A.3	Data availability	110
A.4	Proofs of Scaling	115
APPENDIX B	ADDITIONAL MATERIALS FOR CHAPTER 3 . . .	118
B.1	Adversary agnostic allocations	118
B.2	Proofs	120
B.3	Extended evaluation	130
APPENDIX C	ADDITIONAL MATERIAL FOR CHAPTER 4 . . .	131
C.1	Resource minimal Advocate -BFT	131

LIST OF ABBREVIATIONS

PoW	Proof of Work
PoS	Proof of Stake
SMR	State Machine Replication
BFT	Byzantine Fault Tolerance
DoS	Denial of Service
DA	Data Availability
DSA	Dynamic Aelf Allocation
CQ	Chain Quality
UTXO	Unspent Transaction Output

CHAPTER 1

INTRODUCTION

Fundamental to the development of human society is cooperation, which is underpinned by trust. Modern platforms such as Uber and Airbnb act as trust intermediaries, providing trusted services at scale. An unwanted feature of these platforms is the complete centralization of trust into the intermediary, leading to several frictions—the natural direction forward being the democratization of trust. Attempts to provide democratized services at scale have had a long history in the form of peer-to-peer services. Peer-to-peer data-sharing networks such as BitTorrent dominated the internet traffic at some point in their development. However, they lost to centralized services due to a lack of formalized incentives and trusted outcomes. Bitcoin [1] heralded a new era of democratized trust, where a purely decentralized protocol guaranteed a secure ledger as well as formalized incentives by rewarding agents with tradable tokens.

However, several pain points have emerged with Bitcoin and other blockchain developments such as Ethereum that have led to a belief that decentralized systems work at a trade-off of certain desirable properties such as efficiency or wasteful resource usage. These perceived trade-offs have given rise to several Blockchain trilemmas that aim to highlight the limits of decentralization.

A platform should ideally satisfy the properties of Trust, Democratization, and Scalability. We observe that none of the three existing paradigms - Centralized services, peer-to-peer data sharing networks, or Bitcoin satisfy all these three properties. Given the importance of achieving these properties, designing a protocol that achieves all three properties has been the subject of recent interest. Several unsuccessful attempts to get all three properties have given rise to a folk theorem in blockchain called the Blockchain (protocol) Trilemma [2, 3, 4]. It can be stated as follows:

Blockchain protocol Trilemma: any blockchain can attain at most two of the following three properties: Decentralization, Scalability, and Security.

Decentralization: A blockchain protocol is said to be decentralized if the power to propose or accept changes to the ledger is distributed to all nodes in the network. One protocol is said to be more decentralized than another if it achieves equitable distribution over a shorter time scale. Decentralization assumes that each participant in the system has access to only $O(1)$ resources, e.g., an off-the-shelf laptop.

Scalability: A protocol is said to be scalable if it scales the throughput of the blockchain linearly with the resource per node or with the number of nodes.

Security: A protocol is said to be secure against adversaries if the safety and liveness of the ledger can be guaranteed even if nodes holding together a certain fraction of resources collude maliciously. We note that we will require the protocol to be secure against adversaries that adapt their corrupted nodes based on the public state of the protocol. Security ensures defense against an adversary with $O(N)$ resources.

In addition to satisfying the ideal properties of a trust platform, we would also like the platform to run at a low cost relative to its transaction load. The analysis of trust platforms from a cost angle has led to the Blockchain economics trilemma (proposed by [5]). The blockchain economics trilemma states that *no record-keeping mechanism is implementable without rents, social trust, or a waste of resources*. Broadly, the trilemma considers the following three types of systems to highlight this result:

- **Proof of Work:** Requires use of resources to ensure that the cost of double spend is non-negative
- **Proof of Stake:** Needs social trust to ensure that long-range attacks consisting of a change of genesis stake distribution are not feasible
- **Centralized record-keeper:** Requires rent to ensure that malicious behavior may lead to loss of future rent revenue

Our results in this dissertation can be categorized into works aiming at two broad objectives. (a) Designing a scaling blockchain architecture that

ensures security and decentralization, thereby solving the Blockchain protocol trilemma and consequently reducing the resource and rental cost of the Blockchain economics trilemma; (b) Designing a bootstrapping gadget that achieves optimal chain quality aimed at reducing the social/legal cost of the Blockchain economics trilemma.

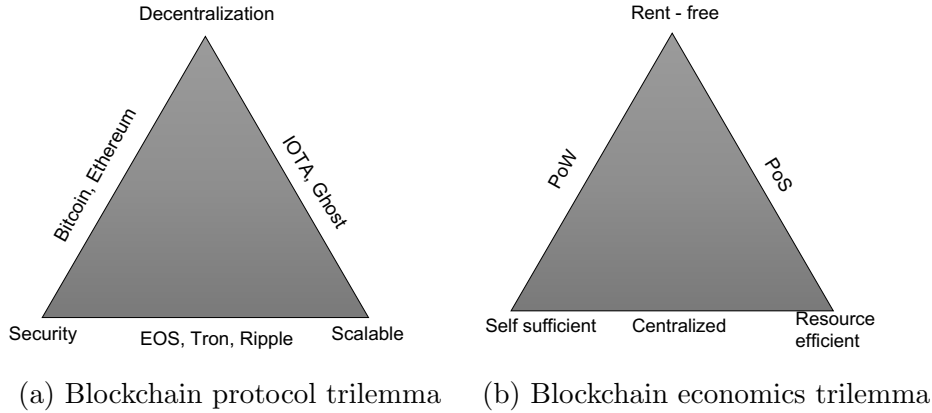


Figure 1.1: Blockchain trilemmas

1.1 Outline and Contributions

1.1.1 Trifecta with Free2Shard : Solving the Blockchain protocol trilemma

Popular blockchains such as Bitcoin and Ethereum achieve full decentralization and security while being unable to scale. Attempts to scale blockchains such as EOS and Tron have led to centralizing authority and trust through a small group of entities. Finally, attempts to get decentralization and scalability in a blockchain system such as the GHOST protocol have proven to be insecure (for example, due to the balancing attack). The inability of any of these protocols to simultaneously achieve the three requisite properties is captured in the Blockchain protocol trilemma (refer figure 1.1a).

Although no proof of the trilemma is offered in the literature, one can appreciate its provenance by considering two extreme architectures. The first architecture is a traditional *fully replicated* blockchain like Bitcoin [1], where each additional node replicates the same task as other nodes. Thus,

security increases linearly with the number of nodes (assuming equal hashing power per node), but the transaction throughput does not. The other extreme architecture is to scale the throughput by running many independent blockchains in parallel, with each blockchain being run by a subset of N/K nodes (each of the blockchains is called a *shard*). While this scales the throughput K times, each shard has only N/K nodes, thus reducing the security. Thus, this tradeoff between security and scalability seems inherent in a sharded blockchain.

This problem has attracted broad interest in the distributed systems community, with many sharding protocols being proposed [6, 7, 8, 9]. These protocols, however, fare poorly against adaptive adversaries, [10] showed that in the long run, none of these sharding protocols scale better than the blockchain substrate (base consensus) against an adaptive adversary. In Chapter 2, we review existing blockchain sharding architectures and learn from their limitations to design, analyze, and implement the **Trifecta** sharding architecture. Chapter 3 discusses the liveness limitations of **Trifecta** architecture and proposes the addition of a critical component to **Trifecta**, the **Free2Shard** DSA algorithm, to resolve the liveness limitation; we analyze the algorithm and show that it achieves the information-theoretic limit for fair, honest node allocation. **Trifecta** with **Free2Shard** ensures security against an adaptive adversary while scaling throughput by $\Theta(\frac{N}{\log N})$ under a synchronous network. In addition to solving the protocol trilemma, improving performance efficiency reduces resource cost per transaction, whereas ensuring decentralization with permissionless competitor entry ensures low rent cost. Hence **Trifecta** with **Free2Shard** also contributes to reducing the rent and resource cost axes of the economics trilemma.

The core technical result is a complete and striking solution to a dynamic Stackelberg game [11]; our approach is distinct but inspired by the classical Blackwell approachability in game theory [12]. The material in chapter 2 and 3 is partly published in [13].

1.1.2 Advocate : Optimal Bootstrapping of PoW blockchains

Trifecta with **Free2Shard** is a PoW blockchain; while legacy PoW blockchains such as Bitcoin and Ethereum have satisfied the honest majority assumption

in practice, this may not be true for a nascent blockchain. PoW blockchains are susceptible to adversarial majority mining attacks in the early stages due to incipient participation and corresponding low net hash power. Bootstrapping ensures safety and liveness during this *transient* stage by protecting against a majority mining attack, allowing a PoW chain to grow the participation base and corresponding mining hash power. Bootstrapping, however, incurs social/legal costs associated with maintaining a bootstrapping committee. The committee members are bound by the legal system to behave honestly, and maintaining such a system incurs legal costs monotonically increasing with time.

While relying on the legal cost system is the opposite of self-sufficiency, it may only be required at the bootstrapping phase of a blockchain; thus, minimizing social/legal costs implies minimizing time to bootstrap. Time to bootstrap can be minimized by retaining existing honest miners and attracting more honest miners; this requires ensuring honest miners are fairly rewarded for their contribution. Fair rewards can be guaranteed with optimal chain quality, ensuring all blocks mined by honest miners are part of the ledger. In Chapter 4, we propose **Advocate** bootstrapping gadget to ensure optimal chain quality even under an adversarial mining majority. The material in chapter 4 is partly published in [14].

CHAPTER 2

TRIFECTA UNICONSENSUS ARCHITECTURE

The fundamental consensus architecture used by most of the protocols [6, 7, 8] that scale throughput with the number of nodes can be categorized as one of *multiconsensus* sharding architecture. The multiconsensus architecture relies on each shard having a *separate consensus engine* and has a secure cryptographic allocation of nodes randomly to shards. A further periodic random reallocation of nodes to shards protects against a weakly adaptive adversary. The random reallocation is enabled by a *node to shard allocation engine* (N2S) which assigns nodes to shards using a (common) distributed randomness, generated using a consensus engine shared across all shards and commonly referred to as *beacon consensus engine*. A *state commitment engine* posts root of a Merkle Trie [15] of a shard’s execution state (the so-called “state commitment”) [16] to the beacon consensus engine at regular intervals; this facilitates fast reallocation and synchronization to new shards. Figure 2.1 illustrates a simple example with a node-to-shard allocation that allocates nodes to distinct shard, on which independent consensus is performed.

Security. Suppose a (super)majority exists in the overall set of nodes. In that case, the random node-to-shard allocation engine transfers this property to each shard (as long as each shard is large enough – this weakness will be discussed shortly). This guarantees the security of the shard consensus engine against a static adversary. Periodic reallocation of nodes to shards enhances the security of shard consensus against a slowly adaptive adversary.

Scaling. Multiconsensus architecture allows nodes to maintain only the state of the shard consensus engine and the beacon consensus engine, with

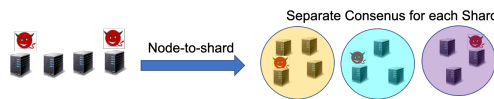


Figure 2.1: Multiconsensus architecture

the beacon consensus engine only containing state commitments and N2S allocation metadata. Splitting the set of nodes into K different shards enables parallel execution of K shard consensus engines, thus scaling transaction execution by a factor of K .

A key question is how large can K be relative to N . With a random allocation of nodes to shards, a majority fraction of honest nodes overall (eg: 60%) translates to the majority of honest nodes in a given shard with high probability (eg. 10^{-10}) only if the size of the shard is large (i.e., $G = 1100$ nodes). Since each shard should have G members, the total number of shards K has to be smaller than N/G (i.e., $N/1100$). This restricts the number of shards and, thus, the scaling capability.

Elastico [6], and Omniledger [7] pioneered the study and design of sharding methodologies for blockchain in academic literature. Omniledger, built on top of ideas from Elastico, runs N2S allocation using Randhound(a randomization protocol) on an identity blockchain. The nodes are gradually rotated between shards to decrease the synchronization load on the network. Rapid-chain [8] further proposed several improvements by reducing the communication complexity to sub-linear in the number of nodes and secure reallocation of nodes to shards building on the Cuckoo rule.

Key Vulnerability. The key property enabling the security of the multi-consensus architecture is that there is a sufficient honest (super)majority in *each* shard (this suffices for a static adversary), and the N2S allocation is periodically updated (this allows security against a weakly adaptive adversary). However, the scheme is insecure against adaptive adversaries: The N2S allocation is posted on the beacon chain. Hence, an adversary knows the list of nodes participating in any given shard. An adversary can then adaptively target all nodes allocated to a particular shard and completely corrupt it (this is well within the corruption budget since the total number of nodes in a shard is small relative to the network size). Once the adversary takes over a shard, safety (i.e., the adversary can approve invalid blocks) and liveness (the adversary can block honest transactions) can be compromised.

In permissionless settings, the possibility of the adaptive adversary threat is severe. This is reflected in the fact that permissionless sharding protocols have designed heuristic mechanisms to deal with this threat. Consider the case of Ethereum 2.0 [16] or Polkadot [17]. In these protocols, the primary mechanism for dealing with adaptive adversaries is to submit commitments of

shard state into the beacon consensus engine, and any node can contest this commitment by proving that the state transition from previous commitment includes an invalid transaction. This short proof is called fraud-proof and can be posted by anyone to invalidate a set of shard blocks [18].

While fraud proofs can be used to detect safety violations caused due to an adaptive adversary, they cannot detect liveness violations. Indeed, an adaptive adversary can corrupt the majority of any given shard and ask them to censor all honest transactions. Thus, since no invalid transactions have been included in the ledger, no fraud-proof can be created. Every time the node-to-shard allocation is rotated, the adaptive adversary corrupts the newly allocated members, thus imposing a liveness ban on that shard till the adaptive corruption budget.

While cryptographic scaling alternatives have been proposed, we note that this liveness attack persists. For example, mechanisms of verifiable computing [19] such as SNARKs [20], ZK-STARKs [21] and bullet-proofs[22] have been proposed for scaling blockchains under differing assumptions on trust and setup. These mechanisms enable scaling by letting the block proposer guarantee that the posted state accurately reflects the state after executing the transactions in the block. While these mechanisms can be used as a non-interactive alternative to fraud-proofs for scaling, they have no way of guaranteeing that transactions have been censored and are thus subject to the liveness attack by the adaptive adversary described above.

As shown by [10], none of the multiconsensus based sharding protocols scale better than the blockchain substrate (base consensus on a fully replicated ledger) against an adaptive adversary and, hence, is not the desired architecture for **Trifecta**.

An alternative architecture for sharding is the uniconsensus architecture (which is propounded to varying degrees in academic papers [23] as well as practical implementations [24]).

The requirement for node-to-shard allocation in multiconsensus architecture comes from the necessity of allocating nodes to each of the shard consensus groups. The uniconsensus architecture avoids this requirement by instead relying on a single *consensus engine* to maintain the ordering information of *all* shard blocks. Thus, the safety of each shard can be directly derived from the safety of the main consensus engine. The consensus engine only maintains a *log of a hash of shard blocks* and hence is scalable. An hon-

est consensus (super)majority is no longer needed in each shard to preserve safety; thus, each node is free to join a shard of its choice (self-allocation). Shard block execution is performed by shard nodes and is decoupled from shard block ordering performed by the consensus engine. This architecture does not require each transaction in every shard block to be valid, rather only guaranteeing the order of transactions in the shards. Since the consensus engine only maintains the hash of each block, a separate mechanism is required to ensure that the full block is available with the shard nodes - this is guaranteed by a *data availability engine*.

Security. If a majority of honest nodes exist in the overall set of nodes, the consensus engine is secure. Consider a shard with less than a majority of honest nodes, the adversarial nodes in the shard cannot change the log of shard blocks without violating the safety of the consensus engine; thus, an adaptive adversary with less than a global consensus majority cannot violate the safety of any shard. The lack of requirement of an honest majority constraint within a shard also allows for small shard size. We observe that uniconsensus architecture solves the shard ledger safety and shard size vulnerabilities of multiconsensus architecture. It, however, introduces some liveness vulnerabilities that we will discuss at the end of this chapter.

Scaling. Nodes maintaining the uniconsensus architecture only maintain the consensus engine and shard log of one shard; the consensus engine contains a log of shard block hashes, the size of which is much smaller than shard blocks. Thus, K shards can be processed in parallel with each node processing only one shard log.

Related work. The idea of scaling by a distributed system by maintaining a single consistent log with distributed data is propounded by distributed systems architectures like Corfu [25], and Tango [26]. Corfu proposed an architecture to maintain a log whose data is distributed across a cluster of flash drives. Tango built a sharded system on top of Corfu, where execution is decoupled from validity, and application nodes only need to execute the subset of entries from the log which are relevant to that application. At a high level, this fits into the uniconsensus architecture described here. However, Corfu and Tango are designed for resilience to the simpler crash faults as compared to the more complex Byzantine faults considered in this work.

To handle more complex fault models, works like Aspen [27] require consensus nodes to maintain and compute the entire ledger data. However, the

data is organized like in Tango, so that client nodes running an application only need to download the relevant data. Lazyledger [23] took this idea further to reduce storage and communication burden on the consensus nodes by allowing data availability proofs [18].

Recognizing the merits of the uniconsensus architecture, we propose Trifecta: a concrete manifestation of the uniconsensus architecture applied to Longest-chain and Prism consensus. We describe its architecture next.

2.1 Trifecta Longest Chain

We begin by describing a simple version of Trifecta, called **Trifecta Longest Chain** built on top of the classic Nakamoto’s Proof-of-Work longest chain protocol. The longest chain protocol has two functionalities: 1) produce a consensus on an *ordered* sequence of blocks by an agreement between all the nodes on the longest chain; 2) validate and add transactions to the ledger along this agreed ordered sequence. We decouple these two functionalities by having two types of blocks: *proposer blocks* and *transaction blocks* (Figure 2.2). The proposer blocks form a blockchain, and an ordered sequence is agreed upon using the longest chain protocol. The proposer blocks themselves, however, do not directly contain transactions but only references (pointers) to transaction blocks, which contain the transactions. Each transaction block belongs to a shard; hence, we refer to it as a shard block. Once an ordered sequence of proposer blocks is agreed upon, transaction validity and state execution can be performed along that sequence by nodes interested in that shard. If shard blocks and proposer blocks were mined using distinct mining processes, the adversary could focus its attention on one or the other, potentially compromising security. In **Trifecta Longest Chain**, shard blocks and the proposer blocks are generated via a random sortition of the blocks mined from the standard hash puzzle solving process (building upon the 2-for-1 mining idea [28]), so that each block, whether it is a proposer or a shard block, requires a *common* proof of work.

This decoupling provides a natural way of scaling by increasing the number of shard blocks per proposer block and dividing the shard blocks into different shards (Figure 2.3). All nodes in the network maintain the proposer chain and hence achieve global consensus on the ordering of the proposer blocks.

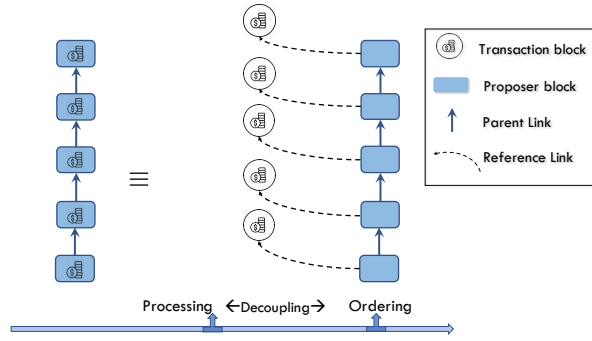


Figure 2.2: Decoupling of Nakamoto’s protocol into a proposer chain and transaction blocks.

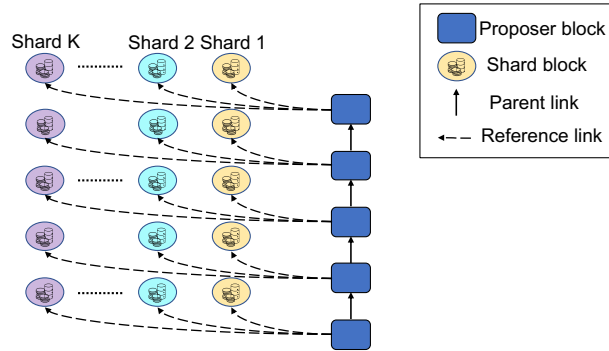


Figure 2.3: Sharding the transaction blocks to achieve throughput scaling.

However, nodes only validate and store the transactions in the shard blocks in their own shard. Hence, security is maintained by the whole network, but at the same time, the throughput can scale with the network size.

Since any node can mine proposer blocks with references to different transaction blocks, it is possible that proposer blocks can refer to transaction blocks that are unavailable with any honest node. This severe attack can completely stall the processing of the shard which contained the unavailable block. The straightforward strategy of each node downloading transaction blocks of *all* shards allows immediate checking for data availability, thwarting this attack. In this case, however, the communication resource used by each node scales linearly with N . We can adopt the idea of [18] to apply erasure coding to the transaction blocks but using a much more efficient fraud-proof called Coded Merkle tree [29] to obtain an efficient use of the communication resource.

In order for nodes to self-allocate to shards in an equitable manner, we incentivize shard transaction block mining through shard-specific transaction

fees. Hence, shards with low mining rates will have higher transaction fees, thus incentivizing nodes to shift to such shards. Finally, to support cross-shard transactions on decoupled ledgers, we build a commitment protocol inspired by TrueBit [30] and Arbitrum [31], where shard nodes can commit the state of any shard chain every G blocks, and any node can challenge it within a time-window, finalizing the correct commitment through a binary search.

2.1.1 Protocol

The data structure of **Trifecta Longest Chain** consists of two types of block-trees, a *proposer* blocktree consisting of *proposer* blocks and K shard block-trees consisting of shard blocks. We split the state (accounts/UTXOs) of the blockchain into K exclusive and exhaustive subsets (i.e., cross-shard dependencies are not modeled in this description). Clients are free to maintain their account/transactions in *any* of the K state subsets.

Proposer blocks constitute the proposer blocktree which is a Bitcoin like block chain with PoW-based Nakamoto longest chain consensus protocol. Proposer blocktree is maintained by all the nodes of the network. Proposer blocks contain pointers to shard blocks, pointer to a parent block in the blocktree, and a PoW nonce. The longest chain of the proposer blocktree is used to order the shard blocks in each shard.

Shard blocks, with K different types each corresponding to one of the K shards, are identified by a *ShardId* field (an integer ranging from 1 to K) and contain shard transactions (i.e., state updates corresponding to the associated shard) and a nonce.

Notice that a shard block does not contain a pointer to parent shard blocks; this is because shard blocks do not construct the edges/pointers of the shard blocktree; these edges/pointers are constructed by (the longest chain of) the proposer blocktree. We create any shard blocktree from the ledger of shard block pointers created by proposer blocktree. Shard block pointer ledger is constructed by starting from proposer genesis and following the longest chain.

The process of adding a shard transaction to the ledger is as follows: First, the shard block containing the shard transaction is mined; Second, a pointer

to the shard block is included in a proposer block; Third, the proposer block is finalized under the k -deep confirmation rule (similar to bitcoin).

Protocol for miner. We describe the protocol for miner in detail in algorithm 1. The environment is initialized with a genesis block for the proposer chain. Every miner mines for a proposer block and a block for one of the K shards simultaneously (to ensure security properties described later). This is ensured using *cryptographic sortition*. Each miner mines a block header which consists of data from both proposer and shard blocks. After the miner is successful in mining the block, it selectively passes the payload based on the type of block mined, i.e., if a proposer block is mined, the miner broadcasts the superblock header along with payload consisting of proposer transaction and shard block pointers while dumping the shard transaction payload, vice versa if a shard block is mined. The sortition happens as follows, $0 \leq \text{block.hash}() < \tau_1$ proposer block is mined and if $\tau_1 \leq \text{block.hash}() < \tau_2$, a shard block is mined.

Each proposer block has to pass a PoW check followed by content validity checks before being forwarded to the neighbors. A shard block pointer(content) is considered valid if the node receives the corresponding shard block (to ensure availability) and is not referred by any ancestor of the proposer block. While this network propagation of the Trifecta Longest Chain environment is similar to Bitcoin , there are four major differences in validity checks for shard blocks:

1. Shard blocks have no pointers to parent blocks; hence no parent availability check is needed.
2. Shard transactions are not checked for validity with respect to state transition function (UTXO availability).
3. Shard blocks for foreign shards are only required to pass the PoW check; there is no content validation check.
4. Shard blocks for foreign shards are only stored temporarily (example: few hours).

The mining rate of each shard is kept constant by adjusting the mining difficulty for shard blocks $(\tau_2 - \tau_1)$ at regular proposer block intervals.

Algorithm 1 Trifecta Longest Chain Miner

```

1: procedure MAIN()
2:   INITIALIZE()
3:   while true do
4:     (header, proposerCont, shardCont) = POWMINING()
5:     if header is a proposer block then
6:       block  $\leftarrow$  (header, proposerCont)
7:     else if header is a shard block then
8:       block  $\leftarrow$  (header, shardCont)
9:     BROADCASTMESSAGE(block)

10: procedure INITIALIZE()
11:   proposerTree  $\leftarrow$  proposerGenesis
12:   proposerParent  $\leftarrow$  proposerGenesis
13:   shardBlkDb  $\leftarrow$   $\phi$  ▷ shard blocks database
14:   unRefShardBlksPool  $\leftarrow$   $\phi$  ▷ proposer block content
15:   shardTxPool  $\leftarrow$   $\phi$  ▷ shard block content

16: procedure POWMINING()
17:   while true do
18:     nativeShardId  $\leftarrow$  s ▷ node chooses its own shard
19:     proposerCntMT  $\leftarrow$  MerkleTree(unRefShardBlksPool)
20:     shardCntMT  $\leftarrow$  MerkleTree(shardTxPool)
21:     nonce  $\leftarrow$  randomString
22:     header  $\leftarrow$  (nonce, nativeShardId, proposerParent,
23:                   proposerCntMT.root, shardCntMT.root)
24:     if Hash(header)  $\leq$   $\tau_1$  then ▷ proposer block mined
25:       break
26:     else if  $\tau_1 \leq$  Hash(header)  $\leq$   $\tau_2$  then ▷ shard block mined
27:       break
28:     return (header, proposerCont, shardCont)

28: procedure RECEIVEBLOCK(B)
29:   CHECKPOW(B)
30:   if B is a proposer block then
31:     if B.proposerCont  $\subseteq$  shardBlkDb then
32:       if proposerParent = B.proposerParent then ▷ mined on longest chain
33:         proposerTree.add(B)
34:         proposerParent  $\leftarrow$  B ▷ advancing the mining tip
35:         unRefShardBlksPool.remove(B.proposerCont)
36:       else if B.proposerParent  $\notin$  proposerTree then
37:         REQUESTNETWORK(B.proposerParent)
38:       else ▷ orphan block
39:         proposerTree.add(B)
40:       else if B.proposerCont  $\not\subseteq$  shardBlkDb then
41:         REQUESTNETWORK(B.proposerCont \ shardBlkDb)
42:     else if B is a shard block then
43:       shardBlkDb.add(B)
44:       if B.ShardId = nativeShardId then
45:         shardTxPool.remove(B.shardCont)
46:       unRefShardBlksPool.append(B.hash)
47:     NETWORKFORWARD(B)

48: procedure RECEIVETX(tx)
49:   if tx.type = proposer then
50:     if tx is valid then proposerTxPool.append(tx)
51:   else if tx.type = shard AND shardId = nativeShardId then
52:     if tx has valid signature then shardTxPool.append(tx)

```

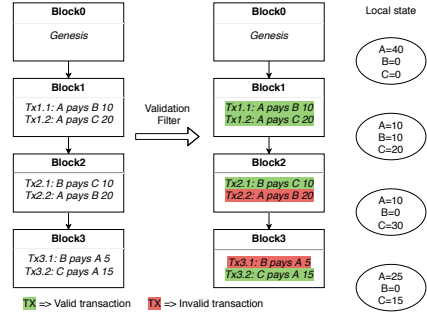


Figure 2.4: A full node filters to extract valid transactions from the blockchain.

Transaction validation. Transaction validation for proposer transactions follows the same rule as any bitcoin transaction. However, shard transactions follow decoupled validation rule described in algorithm 2.

The order of shard blocks is the same order of their pointers in the (longest chain of the) proposer blocktree. Thus, we can construct a shard block ledger by traversing along longest chain of the proposer blocktree. We then extract the shard transaction ledger from the shard block ledger. Transactions in this ledger may be valid or invalid; their validity is checked starting from genesis and traversing down the ledger, this validation filter (implemented as *shardLedger.checkValidity(tx)* in algorithm 2) checks the validity of a transaction by performing required tests, for example checking signatures, input availability, etc. This validity check is deterministic once the order of shard blocks is agreed upon globally. An example of a validation filter is shown in figure 2.4.

Algorithm 2 Tx validation

```

1: procedure IS_TX_CONFIRMED(tx, shardID)
2:   shardLedger  $\leftarrow$  GETSHARDLEDGER(shardID)
3:   return shardLedger.checkValidity(tx) ▷ Validation filter
4: procedure GETSHARDLEDGER(shardID)
5:   longestproposerChain  $\leftarrow$  LONGESTCHAIN(proposerTree)
6:   orderedShardTxs  $\leftarrow$   $\phi$  ▷ empty vector
7:   for blk in longestproposerChain do
8:     // Append txs from the native-shard blocks referred by blk.
9:     shardBlocks  $\leftarrow$  blk.shardBlocks(shardID)
10:    orderedShardTxs.append(shardBlocks)
11:   // orderedShardTxs contains ordered txs of the shard shardID
12:   return orderedShardTxs

```

2.2 Trifecta

Prism is a PoW consensus protocol that provides total ordering at the same high security as **Bitcoin** while achieving the physical limits of the underlying network: theoretic throughput at network speeds and latency at propagation delay (speed of light)[32] and practical performance of 70,000 transactions per second at 20-40s latency for confirmation [33]. It is built by deconstructing the role of blocks in **Bitcoin** and decoupling validation from transaction ordering to scale up the performance of each of the transaction, proposal, and voting functionalities. The result is a PoW consensus protocol that achieves the maximum possible *vertical* scaling performance of a blockchain. Below we systematically replace **Bitcoin** as the proposer chain protocol by **Prism** . The result is **Trifecta**, which scales optimally *both* horizontally and vertically.

The key change is in replacing the proposer blockchain to a **Prism** blockchain with Proposer and Voter blocktrees exactly as in the description of **Prism** in [32].

The shard blocks have the same structure as **Trifecta Longest Chain**.

Protocol for miner: A formal description of the mining protocol is the following. The miner mines a superblock, which aggregates all data required to construct a Shard block, or a voter block or a Proposer block. The block mined is decided by a 3 interval sortition with thresholds (τ_1, τ_2, τ_3) , where the mined block is:

1. *Shard block* if $0 \leq \text{hash}(\text{superblock header}) < \tau_1$
2. *Voter block* if $\tau_1 \leq \text{hash}(\text{superblock header}) < \tau_2$
3. *Proposer block* if $\tau_2 \leq \text{hash}(\text{superblock header}) < \tau_3$.

This mining protocol is essentially the same as in **Prism** [32].

Transaction validation is done using the same rules as **Prism** . Ordering for shard blocks is decided by the proposer chain. Shard block ordering decides shard transaction ordering. Shard transaction validation is decoupled from transaction ordering, as with **Trifecta Longest Chain**.

DoS resistance: An adversary can mine shard blocks and withhold them across a long time span, and broadcast them all at one go, thereby flooding the shard chain with a large number of adversarial blocks. This attack is possible because honest nodes cannot detect 'old' shard blocks. We can prevent

this attack by making every shard block include a hash pointer to a ψ_1 -deep proposer block. Honest miners will only accept shard blocks that refer to proposer blocks at most ψ_2 deep, hence discarding adversarial 'old' shard blocks. Here ψ_1, ψ_2 are system parameters with $\psi_2 > \psi_1$. This approach is similar to the one used by Fruitchains [34], where it is used to guarantee short-term fairness.

2.3 Ancillary modules

This section introduces two ancillary systems needed to operate any uni-consensus architecture, including Trifecta. A data availability module to ensure that honest nodes agree on what shard blocks are available and a state commitment module to ensure that each node can run a light client on every shard for cross-shard transactions and efficient transfer of nodes to new shards. We calculate the resources required by each of these modules.

Data availability: We accomplish this availability check by employing Coded Merkle Trees (CMT) [29]. The key idea is the following: The original block is divided into K chunks and is coded to form N chunks. All nodes select P random chunks to query. The node whose shard block hash got included in the shard transaction ledger now sends forth small chunks of the block encoded as requested. The block is coded such that αN (where $\alpha < 0.5$) chunks are sufficient to decode the original blocks. If the block is incorrectly coded, short fraud proofs of the size $O(\log B)$ ($B =$ block size) can be generated, and the block will be considered invalid. A detailed description of the Data availability module can be found in appendix A.3. Data availability module removes the need for foreign shard block validations mentioned in section 2.1.1.

We calculate the communication, storage, and communication resources used by the CMT-based Data availability engine as a fraction of those used by the ledger management of a shard. Let us assume that processing a shard block utilizes B resources. The data availability engine consists of 2 resource consuming functionalities:

- **Data availability requests:** A newly mined block receives requests for $O(N)$ random chunks for the base CMT symbol of size $O(B/N)$, resource usage = $O(B \log N)$ per shard. Since each shard has $O(N/K)$

nodes, the load can be distributed equally amongst all nodes, resource usage per node = $O(\frac{BK}{N} \log N)$. Each node receives K chunks of size $O(\frac{B}{N} \log N)$. Resource usage = $O(\frac{KB}{N} \log N)$.

- **Data availability fraud proofs:** Incorrect coding proofs of size $O(\log B)$ for at most K shards. Resource usage = $O(K \log B)$.

Thus the resource usage normalized to the resource usage by shard ledger processing is: $O(\frac{K \log B}{B} + \frac{K}{N} \log N)$ which we will show in section 2.4.2 goes to $o(1)$ under a setup where B and N increase with K .

State commitment: Although self-allocation nodes in *Trifecta* do not need to rotate across shards, we still assume they do as a worst-case analysis. In the next chapter, we will see that we might want some nodes to rotate. When nodes rotate between shards or verify the correctness of a transaction in a foreign shard, they will need to fully download the new shard ledger as well as execute it from the beginning of history to synchronize to that shard. This can be extremely resource-intensive and completely drown out any gain due to sharding. A possible solution is to use verifiable computing primitives to assert that the state of the shard ledger as executed by another node is correct [19, 20, 21]; however, these results are still not yet battle-tested for general program execution, so we resort to a different mechanism for state commitments.

We employ an interactive fraud-proof-based state commit protocol inspired by Truebit [30] and Arbitrum [31] to support inter-sharding and shard synchronization in *Trifecta*. We will continue to assume that each node in the *Trifecta* network maintains a complete record of its native-shard chain and the proposer chain, and it knows nothing about any foreign shard except the shard block pointers in the proposer chain. State commitments consisting of the Merkle root of a Merkle Patricia Trie [15] of a shard’s execution state are posted at regular intervals of E proposer blocks called commitment-epochs. They serve as checkpoints for fast bootstrap and cross-shard transactions. The protocol relies on a leader and challengers, randomly chosen using mining linked to block proposals.

The interactions between the epoch leader and a challenger are illustrated in Figure 2.5 and can be briefly described as follows:

1. The epoch leader posts S_s intermediate states for the challenged state.

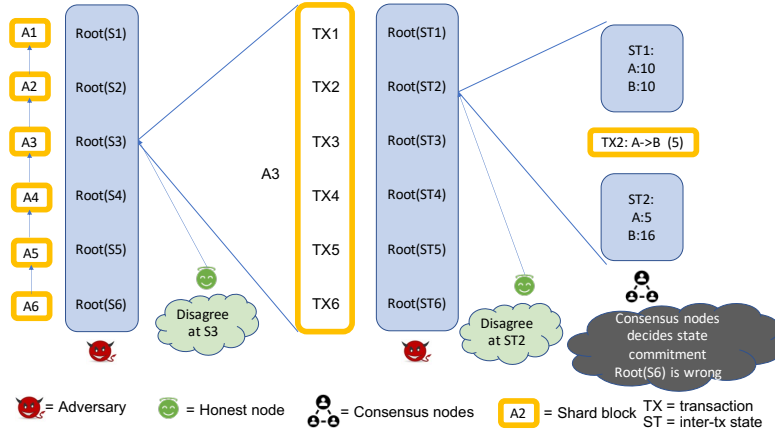


Figure 2.5: State Commitment challenge

2. A challenger responds with a number indicating the first intermediate state when the challenger’s view differs from the leader
3. The game continues to the next round with the leader posting intermediate states for the smaller challenged state and a new challenger responding according to 2.

The game ends when the conflict is resolved down to one transaction or opcode, and that transaction/opcode is posted on the SMR.

State commitments help reduce the burden on full nodes when switching shards; however, posting state commitments and running the interactive fraud-proof adds to the net resource usage. We will show that the normalized resource usage can be made $o(1)$. The state commitment engine consists of 3 main resource-consuming functionalities:

- **Shard State commitments:** There are $O(K)$ state commitments per epoch. We set the epoch duration E such that, in expectation, there is at least one shard block per epoch. Resource usage = $O(K)$.
- **State commitment challenge interactions:** Consists of challenges and replies from epoch leaders. There are $O(N)$ challenges in total which account for $R = O(\log B)$ interactions per challenge where B is the size of the shard block. Resource usage = $O(N \log B)$.
- **Shard rotation:** A node rotating to a new shard needs to synchronize to the state of the new shard. The synchronization involves downloading the state corresponding to the latest state commitment and pro-

cessing shard blocks proposed after the latest state commitment. The resource cost is $O(1 + B)$ per new allocation. We set the rotation interval for every T_r block for the `Free2Shard` DSA policy. We set $T_r = K$ to get the resource usage as $O(B/K)$.

Thus the resource usage normalized to the resource usage by shard ledger processing is: $O(\frac{N \log B}{B} + \frac{1}{K})$ which we will show in section 2.4.2 goes to $o(1)$ under a setup where B and N increase with K .

2.4 Analysis of Trifecta

Security Model. We consider a distributed system maintained by N nodes on a synchronous network. We assume that there is a natural mechanism to partition the ledger into K distinct shards of equal size (for example, think of these as distinct applications sharing a common blockchain or as distinct accounts in a payment system). The goal is to have the total throughput *scaling linearly with the number of nodes*, while each node expends only *constant amount of resources*. We will particularly be concerned about three types of resources: (1) **computation** resource - the number of transactions executed per second, (2) **storage** resource - the number of transactions / second that can be incrementally stored by the node¹ and (3) **communication** resource - the total amount of data communicated by a node. We will assume that at every node, *all three resources are sufficient* to run a non-sharded blockchain with R transactions-per-second. We will assume that, at any given time, a fraction β of nodes are controlled by the adversary - they can deviate arbitrarily from the proposed protocol. We want to prove security against a *fully adaptive adversary*, which can corrupt any subset of nodes based on the public state till that time (as long as the total number of nodes that have been corrupted by the adversary so far is less than its “budget” βN). Security encompasses two aspects: (1) *safety*: transactions once confirmed remain confirmed forever, and (2) *liveness*: new honest transactions will continue to be added within a finite amount of time.

Our main result is that `Trifecta` maintains the safety and liveness if the

¹We will make the simplifying assumption that the storage per node is growing with time. This is required for Bitcoin, for example. Ideas to relax this requirement, for example, [35], can be naturally applied to our setting too.

underlying consensus protocol (Bitcoin or Prism) is safe and live while fully scaling horizontally. In short, Trifecta achieves constant throughput per node. We describe a sketch of the security and throughput analyses below, highlighting the novel aspects over and beyond the analysis already conducted in the underlying consensus protocol. We emphasize that the security holds under an honest *overall* majority while only requiring *non-zero* honest mining power per shard – this is a very relaxed and practically appealing security requirement (accounts with significant assets in a shard are incentivized to maintain that shard). A mathematical formulation of these properties is conducted in App. A.2 within the security model formally described in App. A.1.

2.4.1 Security Analysis

Safety: Safety implies that confirmed transactions are not reversed. The essence of the argument rests on the fact that if the consensus protocol for the proposer chain satisfies safety property, then Trifecta directly inherits this safety property. This fact is based on the following ratiocination: Safety property implies that all honest nodes decide on the same ordering. If the proposer chain consensus protocol is safe, then all honest nodes agree on the same sequence of shard headers, which implies that all honest nodes agree on the same sequence shard transactions (derived from Merkle tree of shard transactions in shard headers).

Liveness: Again, if the consensus protocol for the proposer chain satisfies liveness property, then Trifecta inherits this liveness property. The intuition is based on the following: liveness means that transaction from any honest node is eventually included in the blockchain. If the proposer chain is live, then the shard header from an honest node is included in the blockchain, and an honest miner will include transactions from honest nodes.

We define three security properties: chain growth, chain quality and common prefix (standard in the analysis of longest chain protocols [28]), and show that each shard satisfies them. The relations between various security properties between the proposer and shard chains is illustrated in figure 2.6, where we observe that the proposer security properties can be directly derived from analysis of the corresponding underlying consensus algorithm

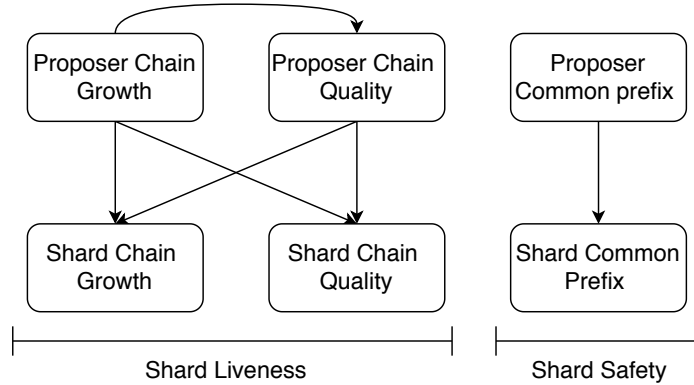


Figure 2.6: Abstraction of security properties of shard chain from the security properties of proposer chain

(longest chain from [28] or Prism from [32]). Succinct descriptions of the proof techniques are below.

Shard common prefix can be directly derived from proposer common prefix since if the proposer blockchain share the same prefix, then the shard blocks pointed by the same prefix will be common amongst the honest blocks.

Shard chain growth can be derived from proposer chain growth and proposer chain quality, since proposer chain grows and there is an honest block in the proposer chain at least once every few blocks, the honest proposer blocks will refer to multiple shard blocks, and hence lead to shard chain growth.

Shard chain quality is derived akin to shard chain growth. Since there is an honest proposer block once every few blocks in the growing proposer chain, it will at least refer to previously unREFERRED honest shard blocks and since a non-zero fraction of the shard blocks mined are honest, shard chain quality is maintained as a function of that shards honest mining fraction. Note that a shard’s chain quality can be severely throttled by reducing a target shard’s honest mining fraction; we will discuss this vulnerability in the next chapter.

Main Result: Trifecta satisfies safety as long as the majority of the total mining power is honest and liveness when additionally each shard has non-zero honest mining power.

2.4.2 Scaling

A Trifecta node will not maintain the shard ledgers of $K - 1$ foreign shards. Thus, intuitively, even though the throughput of the system is $O(K)$, the net resource usage is $O(1)$. A more in-depth analysis is needed once we integrate ancillary modules like Data availability engine and State commitment engine. We have the following results:

Proposition 2.1 (Trifecta scalability). *Trifecta ensures that the total throughput of the system scales by $O(K)$ while ensuring that the total resource usage per node is $O(1)$ if we set $N \geq \Omega(K \log K)$ and allow for a latency of $\Omega(NK)$.*

Outline of the proof: We define the *overhead ratio* as the ratio of (the resource used to maintain the proposer chain and Trifecta ancillary functionalities such as data availability) to (the resource used to maintain any one shard). We can assume that the resource rate used to maintain one shard with a throughput of $O(1)$ is $O(1)$. Ensuring that the total resource usage rate (sum of resources used to maintain the shard ledger and the overhead) is $O(1)$ ensures that the system is stable. Ensuring that the overhead ratio is $o(1)$ ensures the total resource usage rate is $O(1)$.

The leading factor in the overhead is the data availability votes. For an interval that requires the shard ledger to process one shard block of size B at a resource usage of $O(B)$, ensuring data availability requires $O(N)$ votes per shard block for K shard blocks. Hence, scaling the block size B by $\Omega(NK)$ leads to a resource overhead of $o(1)$. Since the transaction confirmation and inclusion latencies are a multiple of $\frac{B}{v}$, with $v = O(1)$ we get latency of $\Omega(NK)$. The complete proof for this proposition can be found in appendix A.4.1.

Corollary 2.1 (Trifecta Latency). *Trifecta running at any total throughput needs to allow for a latency of $\Omega(NK)$.*

Outline of the proof: Irrespective of the system's throughput, each node needs to process $O(N)$ data availability votes per shard block for every K shard blocks in the time it takes each shard to mine one shard block (Hence leading to K shard blocks in total). Latency L equals a multiple (let us call it $1/\chi$) of the time it takes to mine one shard block. Thus, each node needs to use $O(NK)$ resources in a time interval of χL . To set the resource

overhead of $o(1)$, we need $\chi L = \Omega(NK)$, giving us a latency of $\Omega(NK)$. A more detailed proof where we reduce throughput and calculate the required latency can be found in appendix A.4.2.

2.5 Implementation

We implement a full node of **Trifecta** on top of the codebase from [33] written in **Rust**. Each node belongs to a particular shard in $[K]$, referred to as its native shard. We first describe the system design and architecture of **Trifecta** and then evaluate the performance of **Trifecta** and compare it with other sharding protocols.

2.5.1 System Design

Trifecta implements a UTXO (Unspent transaction output) based state model similar to Bitcoin [1]. The transactions are similar to Pay to Public key hash (p2pkh) transactions in Bitcoin and have 4 fields:

1. *Inputs*: Unspent coins with coin identifier, value, owner address.
2. *Output*: Output coins with value, recipient address.
3. *Signatures*: by the owners of the input coins.
4. *Shard-id*: Shard id to which the transaction belongs.

A transaction is valid if all input coins are unspent and all the signatures are valid. A typical transaction is of size 168 Bytes.

Database: **Trifecta** maintains 4 databases:

1. *Block-structure database*: Maintains the blockchain structure required for consensus.
2. *Block database*: Stores all the shard blocks of the native shard and the last 100 shard blocks for other shards.
3. *Transaction memory pool*: Stores native-shard transactions that are not yet added to the ledger.

4. *UTXO database*: Maintains the state of the native shard by storing all unspent coins at the current state of the blockchain.

These databases are implemented using a key-value database [36]. *Trifecta* has 4 key modules:

Block manager: The block manager maintains the structure of proposer blocks, voter blocktrees, and shard blocks. When a new block arrives at the block manager, it adds the block metadata to the block structure database and stores the block in the block database. Shard blocks from foreign shards are stored in a fixed-size FIFO buffer assigned to that shard.

Ledger manager: Ledger manager maintains the shard transaction ledger and runs *asynchronously* with respect to the block manager. It periodically polls the block manager to check for new confirmed transaction blocks. For every transaction in a newly confirmed transaction block, it performs *sanitization*- a). If all its inputs are unspent, it updates the UTXO set by removing the input coins and adding the output coins; else, it skips over the transaction. Note that in Bitcoin, all the transactions in the block are valid, whereas *Trifecta* decouples consensus from validation, and, hence, blocks can contain invalid transactions. This is a core idea to achieve scaling.

Network manager: Network manager communicates blocks and transactions with its peers. On receiving a new block, it checks for proof of work. It then checks if all the blocks referred by this block are present in the database. If not, it adds the block in a buffer and processes it after all its referred blocks arrive. It is then sent to the block manager. The network manager also checks the signatures of shard blocks belonging to native-shard and forwards the signature verification results to the ledger manager.

Miner: At any given time, it maintains three types of blocks: a) Proposer block, b) Voter block, and c) native-shard block - which contains the latest transactions from the memory pool. It continuously ‘mines’ to find a nonce which satisfies PoW. When it successfully mines a block, it sends the block to its peers (ref lines 16:27 in Algorithm 1).

2.5.2 Evaluation

Our experiments validate that *Trifecta* scales security and throughput with respect to the number of nodes while using constant computation and state-

storage resources. Although the CMT library has been implemented in `Rust` [29], we have not fully integrated the library with cross shard transactions, and experimental results to scale communication is left for future work.

In our experiments, we specifically answer the following questions:

1. How does the throughput scale with the number of nodes?
2. How does the security scale with the number of nodes?
3. How will an increase in the number of nodes affect computation and storage requirements?

Testbed: We run our experiments on `c5d.4xlarge` AWS instances. Each instance has 16 cores, 32 GB of RAM, 400 GB of NVMe SSD storage, and network bandwidth of 10 Gbps. We connect $|\mathcal{N}|$ of these instances via a 4-degree random graph network. We limit the bandwidth of each instance to 400 Mbps and simulate an edge latency of 60 ms to reflect the real-world networks. We run the Prism proposer chain with 100 voter chains and a mining rate of 1 block/10 seconds on proposer and voter chains.

Experiment 1: Horizontal scaling

Each shard has 2 nodes² and supports 1100 transactions per second. On varying the number of shards from 1 to 99, the throughput of the whole system scales linearly in the number of shards, while resources used by each node remain the same (Figure 2.7). For 99 shards, the system achieves 100,000 transactions per second. From Theorems (A.10, A.11, A.12) we know that the security of the whole system scales with the number of nodes.

We perform data availability of non-native-shard blocks by naively downloading the whole blocks and storing them in a fixed-size FIFO buffer. This extra step increases the computation overhead due to (de)serialization and PoW check of non-native shard blocks. Also, this fixed-size buffer has a small storage overhead, increasing with the number of shards. The effects of these overheads are observed in Figure 2.9. The computation increases from 1.12 to 2.33 core utilization, and the storage increases from 1.4GB to 2.6GB, which is less than a 2x increase for a 99x increase in throughput and security. We

²This is kept small to save on AWS costs.

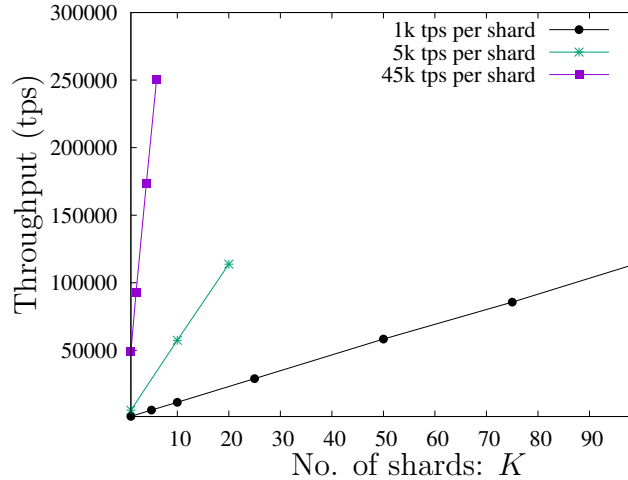


Figure 2.7: Horizontal scaling can be combined with vertical scaling of Prism to obtain higher throughput.

want to stress that these experiments show that the resource requirements of Trifecta are the same as that of Bitcoin and Ethereum; hence, it can be run on a regular laptop.

The experiments are run on a 4 degree graph network; thus, the delay across the diameter increases with the number of nodes in the system. Since the forking rate is proportional to the average delay in the network, forking increases with the number of nodes. If we maintain a constant delay along the diameter, the forking rate should remain constant for all network sizes.

Experiment 2: Horizontal + Vertical scaling

We run each shard with a throughput close to its capacity i.e, $45k$ transactions per second; such high throughput is possible because Prism [32] is vertically scalable. In this setting, too, the throughput of the whole system scales linear in the number of nodes, achieving a throughput of $250k$ tps using 6 shards. This shows that Trifecta can simultaneously achieve both horizontal and vertical scaling. Figure 2.8 shows the performance dashboard of experiment 2 run for 15 minutes with 36 nodes, with 6 nodes per shard. We observe that the total throughput of the system is $250k$ tps, with each shard contributing $42k$ tps and the confirmation latency of the system is 25 seconds.

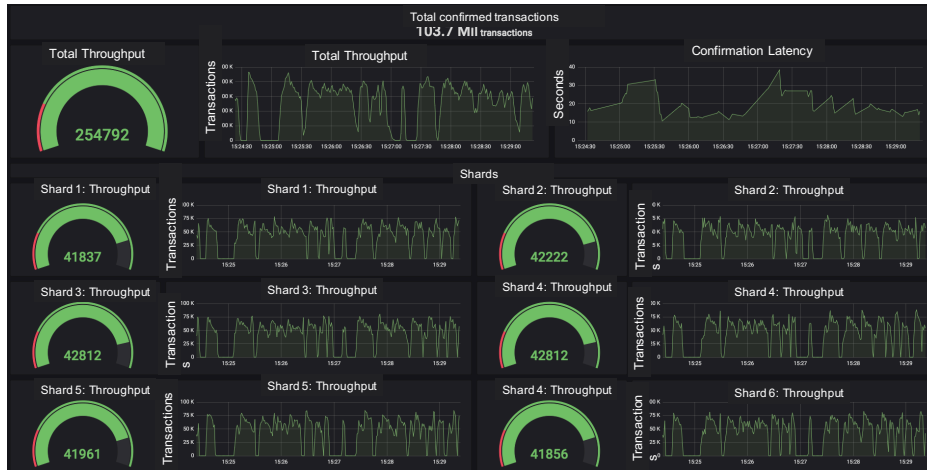


Figure 2.8: Performance dashboard for Experiment 2: horizontal+vertical scaling. The dashboard captures a time span of 5 minutes of a running experiment through the time plots of various performance metrics. The dashboard shows the total throughput, confirmation latency, and individual throughput of all 6 shards.

CPU utilization

Figure 2.9 plots the breakdown of CPU utilization across different modules of Trifecta. It confirms that the compute resources required by *block manager* are constant with respect to the number of shards. Note that the compute utilization for the network manager increases with the number of shards since the amount of communication increases; Coded Merkle Trees proposed in section 2.3 will ameliorate this effect. The integration of CMT library (written in Rust) [29] is left for future work.

Table 2.1 shows the CPU usage breakdown across different operations, and we see that it is dominated by the database read/write operations. Our implementation uses a standard off-the-shelf key-value database, and further optimization of the database can potentially reduce CPU usage.

Adversarial behavior: Performance under a liveness attack

In Section 2.4 and A.2, we prove that each shard in Trifecta is secure – safe and existentially live – as long as the majority of the nodes across all the shards are honest. Unlike other sharding protocols, each shard is secure even if a majority of miners on that shard are dishonest as long as the shard has at least one honest node and the system has honest overall majority. These

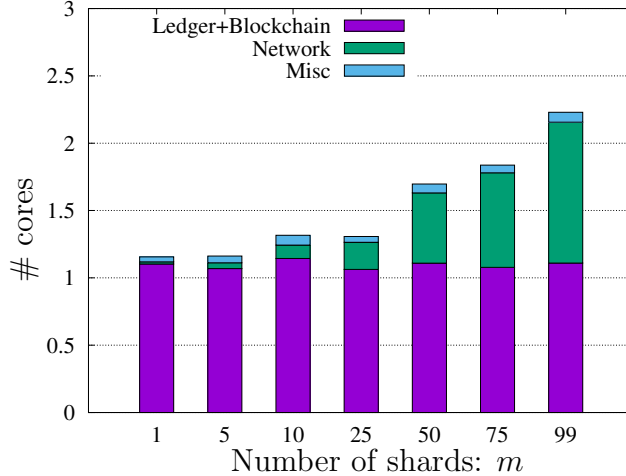


Figure 2.9: Compute usage required by Ledger and Blockchain modules remain constant with an increasing number of shards; however, compute usage by network module increases.

Module	Operation	# Cores (1 shard)	# Cores (99 shards)
Ledger+Block manager	RocksDB Read/Write	0.823	0.782
	(De)serialization	0.097	0.122
	Miscellaneous	0.183	0.206
Network	RocksDB Read/Write	0.001	0.132
	(De)serialization	0.001	0.520
	Cryptography	0.008	0.1125
	Miscellaneous	0.006	0.28
Miner	Block Assembly	0.002	0.026
Miscellaneous		0.061	0.041

Table 2.1: CPU usage breakdown of Trifecta implementation with 1 and 99 shards.

security properties protect our system against a fully adaptive adversary who can choose to concentrate and attack specific shards.

We simulate an adaptive attack in a system with 6 shards, with each shard having 6 nodes. In this attack, the adversary progressively targets and corrupt the shards by first corrupting 5 out of the 6 nodes in shard 1, then corrupting 5 out of 6 nodes in shard 1 and 2, and finally corrupting 5 out of 6 nodes in shard 1, 2 and 3. In this attack, since most of the nodes are running the honest protocol, as predicted by the theory, we observe that no transactions are reverted in any of the shards despite the adversary controlling the majority of nodes in multiple shards. Figure 2.10 shows the throughput of each shard during this attack and we observe that the attack only results in throughput reduction - representing a reduction in chain qual-

Protocol	Security(adaptive)	Throughput	Latency	# shards
Elastico	$O(1)$	40 tps	800s	16
Zilliqa	$O(N)$	2500 tps	120s	6
Omniledger	$O(1)$	3500 tps	63s	3
Rapidchain	$O(1)$	7380 tps	8.7s	16
Trifecta	$O(N)$	110,000 tps	22.7s	99

Table 2.2: Comparing Security and performance of Trifecta with other sharding protocols. Data about throughput and latency for Elastico, Omniledger, and Rapidchain is obtained from [8] and from [9] for Zilliqa

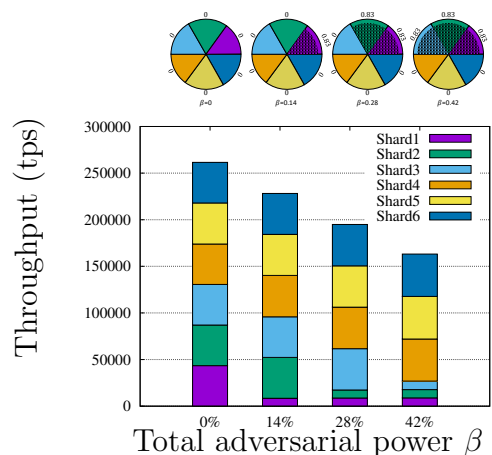


Figure 2.10: The pie chart at the top describes the adversarial distribution across shards, the numbers on the circumference of the pie charts represent a fraction of adversarial power in that shard. The histogram at the bottom shows the throughput per shard according to the given adversarial distribution

ity; the throughput of a shard reduces to $1/6$ -th when the adversary controls $5/6$ -th of the nodes in that shard.

We simulate another attack where the adversary attacks in a different manner. The 14% adversary corrupts 3 nodes in shard 1 and 1 node each shard 2 and 3; on increasing its power to 28%, it corrupts a total of 4 nodes in shard 1, 3 nodes in shard 2, 2 nodes in shard 3 and 1 nodes in shard 1. Finally, a 42% adversary corrupts 5 nodes each in shards 1,2,3. The throughput of each shard for this attack is shown in figure 2.11

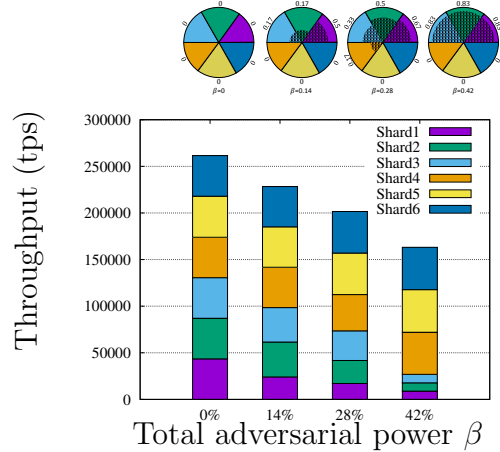


Figure 2.11: These charts depict similar adversarial experiments as described in Figure 2.10 albeit with a different adversarial distribution as depicted in the pie charts at the top

2.5.3 Performance comparison with other sharding protocols

All sharding protocols of which Elastico, OmniLedger [7], Zilliqa [9], RapidChain [8] are a few examples, are based on *node-to-shard* allocation. We note that due to their utilization of node-to-shard allocation, all these protocols, while making significant progress, suffer some issues in all three axes of the trilemma: (1) not secure under adaptive adversaries (2) scaling efficiency limited by shard with a large number of nodes and (3) requiring identity management. Furthermore, none of these protocols give a global ordering of transactions across various shards, thus making them highly susceptible to fatal synchronization attacks and message replay attacks, as recently reported in [37]. We note that due to the global ordering of all blocks, these attacks do not work on Trifecta.

Table 2.2 compares Trifecta to other sharding protocols such as Elastico [6], Omniledger [7], Zilliqa [9] and Rapidchain [8]. The performance numbers of these protocols are as reported by the respective authors; we briefly describe the testbed used in each of the protocols:

Elastico: Elastico has a C++ implementation of their full node. They run experiments on AWS EC2 `c4.large` with each instance runs two nodes. In total, they run 16 shards with 1600 nodes. As explained previously, they need a large number of nodes per shard because they use *node-to-shard* allocation. They achieve a total throughput of 1 block (size 1MB) every 50s, and, for a transaction size of 512 Bytes [8], this translates to a throughput of 40 tps.

Omniledger: Omniledger has an implementation in `Go`. Their experiments run on a network of 1800 nodes on 60 physical machines with 24GB of memory running a network with 10Gbps links throttled down to 20Mbps with a latency of 100ms. Their best-performing configuration achieves a throughput of 300K tps with 4 nodes per shard; however, since they have an honest shard majority assumption, this configuration is not secure. A more realistic experiment with 600 nodes per shard achieves a total throughput of around 3000 tps on running the first 10,000 blocks of the Bitcoin blockchain.

Rapidchain: Rapidchain implements a prototype in `Go`. A network of 4000 nodes is simulated using 32 machines, each with 64-core processors running on a network throttled to 20Mbps with 100ms latency. They achieve a throughput of 7380 tps using a configuration with 16 shards.

Zilliqa: Their testnet is implemented on Amazon EC2 instances and it comprises of 3600 nodes split across 6 shards. They achieve a throughput of 2,488 tps.

We note that these protocols are implemented in different languages, and their experiments are run on different testbeds. However, the machines and network used in these experiments are similar to that used in the `Trifecta` experiments, and we observe that `Trifecta` achieves 10 – 15 x gain in throughput and simultaneously achieves low confirmation latency.

2.6 Conclusion

In this chapter, we studied the security drawbacks of existing scaling protocols that utilize multiconsensus architecture for sharding. We proposed `Trifecta` a concrete manifestation of the uniconsensus architecture applied to Longest-chain and Prism consensus. `Trifecta` is decentralized, scales throughput almost linearly in the number of nodes, and is safe against fully adaptive adversaries. Hence it satisfies all three properties of the protocol trilemma except liveness under adaptive adversaries. We implemented `Trifecta` in `Rust` and achieved a total of 100,000 transactions per second on 200 EC2 nodes while each node only executes 1000 transactions per second, uses less than 3GB memory, and less than 3 cores on an AWS `c5d.4xlarge` instance.

`Trifecta` is existentially live, i.e., as long as there is one honest miner in a shard, it will mine a shard block that will be included in the shard ledger

eventually. However, liveness measured by chain quality can be made very low by adversaries congregating in a shard. Adversaries can reduce the chain quality of the target shard by $O(1/K)$. Although total throughput is not affected, we do not know in advance what shard can be attacked; this is similar to DOS attacks. As a result, application developers assume worst-case performance for all shards; hence, the realized throughput drops by a factor of K , and we have $O(1)$ throughput. We cannot prevent adversaries from congregating into one shard since this goes against the principle of self-allocation. It is also difficult for honest nodes to replicate the behavior of adversarial nodes to try to balance them out, since the adversary is in a Stackelberg position; it takes action AFTER observing honest behavior. In the next chapter, we propose **Free2Shard** DSA to resolve this vulnerability.

CHAPTER 3

FREE2SHARD: DYNAMIC RESOURCE ALLOCATION FOR TRIFECTA

A uniconsensus architecture such as Trifecta removes dependence on N2S allocation by relying on nodes to self-allocate. The freedom to allocate introduces a serious liveness attack on a shard where an adversary can concentrate its mining power on one shard and drown out the generation of honest shard blocks. In this attack, the fraction of honest blocks to the adversarial blocks is significantly reduced due to a large fraction of adversarial nodes in the shard. Since the shard has limited resources, it can only support a fixed number of shard blocks; hence a low honest block to total block fraction would lead to a throttling of the number of honest shard blocks added to the shard log and hence throttle chain quality and consequentially the shard’s throughput. An example of this attack is shown in Figure 3.1a.

Concretely, each shard has a fixed block throughput of v_B and a transaction throughput of $Bv_B = v$. Application deployment on the blockchain assumes that a certain level of honest transaction throughput is maintained on average across time. If the average throughput is less than the anticipated transaction load, the standard queuing theory dictates that the pending transaction queue will continue growing. Thus, the deployment rests on the worst-case assumption, where application developers assume that the shard in which the application runs will be under attack by adversaries. An adversary can decrease the instantaneous honest transaction throughput $\tilde{v}_i(t)$ by not allowing any honest transaction to be included in a shard block generated by the adversary. Considering only honest shard blocks can carry honest transactions $\tilde{v}_i(t) = v * r_i(t)$, where $r_i(t)$ is the shard-normalized rate of production of honest blocks in that shard. Since the mining process ensures that the rate of production of shard blocks is proportional to mining power, $r_i(t)$ is also equal to the shard-normalized honest mining power, i.e., the ratio of the mining power of honest nodes in shard i to the total mining power in shard i .

Thus, the metric of concern for application deployments on a shard i is the *time-averaged throughput at which honest agents can post transactions on that shard* denoted by $\tilde{v}_i(t) = \frac{1}{t} \sum_{j=1}^t \tilde{v}_i(j) = v \bar{r}_i(t)$. Not knowing in advance the adversaries’ attack target shard, application deployment will assume that the selected shard will be the shard under attack, i.e. anticipated honest throughput for the selected shard is equal to $\min_i \tilde{v}_i(t) = v \min_i \bar{r}_i(t)$.

We propose **Free2Shard**, a distributed miner resource allocation algorithm enabled using a dynamic self-allocation (DSA) engine, to **Trifecta** to remove its liveness vulnerability. The following section will discuss the DSA engine, the focal point of **Free2Shard** architecture.

3.1 Free2Shard DSA

The dynamic self-allocation (DSA) engine is a sequential algorithm that guides honest nodes to (re)allocate themselves to shards in different proportions, adapting to the past adversarial allocation behavior. The DSA algorithm aims to guarantee that any shard’s long-term chain quality is above any desired level. Since the shard blocks follow the same structure as “fruits” in *Fruitchains* [34] and “input endorsers” in *Ouroboros* [38], we can expect the *chain quality* of the shard ledger to be equal to the fraction of honest mining power in that shard; we call this metric the *honest fraction* for that shard.

Security Model. We make similar security assumptions as the previous chapter, stated briefly; we consider a distributed system maintained by N nodes on a synchronous network. We assume that there is a natural mechanism to partition the ledger into K distinct shards of equal transactional load (e.g., distinct applications sharing a common blockchain or distinct accounts in a payment system). We assume that at any given time, at most β fraction of the nodes are controlled by the adversary (i.e., they can deviate arbitrarily from the DSA protocol). We consider a fully adaptive adversary, which can corrupt any subset of nodes based on the public state till that time (as long as the total number of corrupted nodes till that time is less than its “budget” βN).

The proposer chain is divided into epochs where each epoch lasts Δ proposer blocks. Honest nodes allocate themselves to a new shard at the be-

ginning of an epoch. This assumption of synchronized time for allocation is made for analytical simplicity, and we later discuss that honest nodes can be allowed to allocate themselves at any time.

Let us denote by γ the net honest mining power and γ_i and β_i as honest and adversarial mining power in shard i . For analytical brevity, we have supposed that each node has the same mining power.

The adversary's goal is to minimize the ratio of honest nodes to total nodes in any shard, thereby minimizing that shard's chain quality. The adversaries can collude, observe the honest node allocations, and then allocate themselves accordingly. Honest nodes can switch shards every epoch; however, adversaries can switch shards at a faster rate than honest nodes - almost instantaneously. We show later in this section that this agility advantage cannot reduce any shard's long-term chain quality further than reallocating themselves at the same rate as honest nodes.

We assume a timescale where one proposer chain epoch takes unit time. We estimate the chain quality in each shard at integer times t ; this fraction should correspond to the ratio of honest blocks to total blocks in any shard formed in time $[t - 1, t)$; referred to as epoch t . Let $r_i(t)$ denote the honest fraction at time t in shard i and $\bar{r}_i(t)$ denote the average.

$$r_i(t) = \frac{\gamma_i(t)}{\gamma_i(t) + \beta_i(t)}; \quad \bar{r}_i(t) = \frac{1}{t} \sum_{j=1}^t r_i(j). \quad (3.1)$$

Let $\mathbf{r}(t) = [r_1(t), r_2(t), \dots, r_K(t)]$ and $\bar{\mathbf{r}}(t) = [\bar{r}_1(t), \bar{r}_2(t), \dots, \bar{r}_K(t)]$ denote the corresponding vectors encompassing all the K shards.

We define the main optimization objective function, the solution of which is the honest strategy that can achieve a worst-case shard chain quality of $\psi(T)$ given any worst-case adversarial action:

$$\psi(T) = \max_{\{f_t\}_t} \min_{\{\beta_i(t)\}_{i,t}} \min_i \{\bar{r}_i(T)\}. \quad (3.2)$$

A DSA algorithm specifies the honest node re-allocation strategy: for each t the re-allocation is a function as follows:

$$f_t : (\boldsymbol{\beta}(1), \dots, \boldsymbol{\beta}(t - 1), \boldsymbol{\gamma}(1), \dots, \boldsymbol{\gamma}(t - 1)) \rightarrow \boldsymbol{\gamma}(t), \quad (3.3)$$

satisfying the constraint $\sum_{i=1}^K \gamma_i(t) = \gamma$.

We define $\psi_{f_t}(K)$: the honest fraction of the worst performing shard under a worst-case adversary on running the re-allocation function f_t as follows:

$$\psi_{f_t}(T) = \min_{\{\beta_i(t)\}_{i,t}} \min_i \{\bar{r}_i(T)\}. \quad (3.4)$$

Note that the adversary action in round t is allowed to depend on the honest node re-allocation policy at round t since an adversary can observe the action of honest nodes and act accordingly.

We are specifically interested in a class of DSA algorithms which *do not require* special access to the adversary fraction (since this may be unobservable). The adversary agnostic DSA algorithms will have the property that f_t is a function only of $\{\beta(j) + \gamma(j)\}_{j=1}^{t-1}$. The reason for this choice being that $\{\beta(j) + \gamma(j)\}_{j=1}^{t-1}$ correspond to the net mining power per shard, which can be easily estimated. The key question is the following: **how to convert any given f_t to be adversary agnostic?** We begin with any function f_t and arrive at an adversary agnostic allocation function g_t as follows. Suppose, conservatively, the total honest fraction $\gamma = \sum_i \gamma_i(t) = 1/2$ (in practice this fraction could be larger if the adversaries are fewer). A recursive estimate on the adversarial nodes in a shard can be obtained as follows: At round-1: we know that the $\gamma_i(1) = 1/2K$ honest mining power is uniformly allocated (since we know that $\sum_i \gamma_i(t) = 1/2$). Hence we can get the adversarial mining power as $(\gamma_i(1) + \beta_i(1) - 1/2K)$. In round 2, the honest mining power can be calculated using $\gamma(2) = f_2(\{\beta(1) + \gamma(1)\} - g_1, g_1) = g_2(\{\beta(1) + \gamma(1)\}, g_1)$, here g_1 is the adversary agnostic allocation during epoch 1.

This sets up the recursion where the data till the previous round is enough to calculate the honest power at a given round $\gamma_i(t)$ and using $\gamma_i(t) + \beta_i(t)$; we can calculate $\beta_i(t)$. We note that for any given f_t , this creates a new allocation function g_t , which only requires the total mining power per shard and the initial condition for round 1. The performance $\psi(T)$ achieved by this g_t when $\gamma \geq 1/2$ is the same as the performance achieved by f_t when $\gamma = 1/2$ (since we are using the conservative estimate in calculating g_t). A precise definition of g_t as a function of f_t is in Appendix B.1.

Information-Theoretic Limits. We see that any optimal honest strategy cannot obtain a time-averaged honest fraction greater than γ in the

worst-performing shard under an optimal adversarial strategy:

$$\psi(T) \leq \gamma. \tag{3.5}$$

The above inequality is obtained since, irrespective of the honest allocation; an adversary can simply replicate the honest node allocation and set $\beta_i(t) = \frac{\beta}{\gamma}\gamma_i(t)$, thus rendering the honest fraction of each shard as γ .

The key question is whether this upper bound is actually achievable. If so, this would be a striking and positive result: the honest nodes can re-balance the allocations optimally, i.e., the time average fraction of honest nodes in each shard is always the same as the overall fraction of honest nodes (which is γ). This result would imply that the adversary is ineffective in overcrowding any shard, even minimally. This section’s main result is that this is indeed achievable for a large enough time window T over which the average is taken. Towards building an intuition towards this result, and the nature of the optimal honest policy and the corresponding performance, we discuss next a few baseline honest node strategies and analyze their performance.

3.1.1 Simple Strategies

Static strategies. Consider a static self-allocation scheme where honest nodes allocate themselves uniformly to one of K shards, i.e., $\gamma_i(t) = \frac{\gamma}{K} \forall i \in [K]$. This strategy is vulnerable to adversarial action: an adversary simply targets shard 1 and throttles its honest fraction by allocating all its power to this shard (i.e., $\beta_1(t) = \beta$). This renders the honest fraction of shard 1 as $\frac{\gamma/K}{\gamma/K+\beta} = O(1/K)$ which approaches 0 for large K and is very sub-optimal compared to the information-theoretic limit. Thus the honest nodes have to adapt to adversarial action.

A simple dynamic strategy. A simple adaptive action by the honest nodes is the following: half the honest nodes distribute themselves uniformly randomly across K shards, and the other half follows the adversarial distribution of the past round.

$$f_t := \gamma_i(t) = \left\{ \frac{\gamma}{2\beta}\beta_i(t-1) + \frac{\gamma}{2K} \right\}. \tag{3.6}$$

The idea is that if the adversary congregates in one shard, then at least half the honest nodes react to this congregation and ameliorate any degradation in performance in that shard. We can show the following performance result.

Proposition 3.1. *For large enough T ,*

$$\Omega\left(\frac{1}{\log K}\right) \leq \psi_{ft}(T) \leq O\left(\frac{\log \log K}{\log K}\right). \quad (3.7)$$

Outline of proof: We define a g -good instance as a time instance where the instantaneous honest fraction is greater than a threshold g . We then show that the adversary cannot force τ consecutive g -bad instances; thus, every $\tau + 1$ time-instance will have at least one g -good instance. For $\beta = 0.5$ and $g = 1/4$, we get $\tau = O(\log K)$, hence average honest fraction is $\geq O(1/\log K)$. To get the upper bound, we find an adversarial strategy that achieves an average honest fraction of $O(\log \log K / \log K)$. The adversarial strategy works by gradually increasing its mining power in the target shard by a constant multiple and resetting periodically when the total mining power limit is achieved, i.e., $\beta_i(t)$ follows the sequence $l, lr, lr^2, \dots, lr^\tau, 0, l, lr, \dots$. The complete proof can be found in Appendix B.2.1.

These inequalities show that this dynamic allocation policy is much improved compared to the static allocation policy (from $O(1/K)$ to $O(1/\log K)$). However, the performance still degrades to zero as K grows, while the information-theoretic limit is a constant (independent of K). An adversarial attack on shard 1 which renders a honest fraction of $O(\frac{\log(K)}{\log(\log(K))})$ is illustrated in Figure 3.1b. We see from the proof in Appendix B.2.1 that the analysis is very specific to the structure of the dynamic honest policy in Equation (3.6) and this does not provide any intuition towards designing an improved honest policy. We make connections to game-theoretic literature next, which provides us a broader view of this problem and classical results associated with this class of max min games.

3.1.2 Online Convex optimization and Dynamic games

We make two major observations on our problem:

- The honest-fraction $\mathbf{r}(t)$ can be thought of as a reward vector with shards as dimensions. The maximization in Equation (3.2) is over a

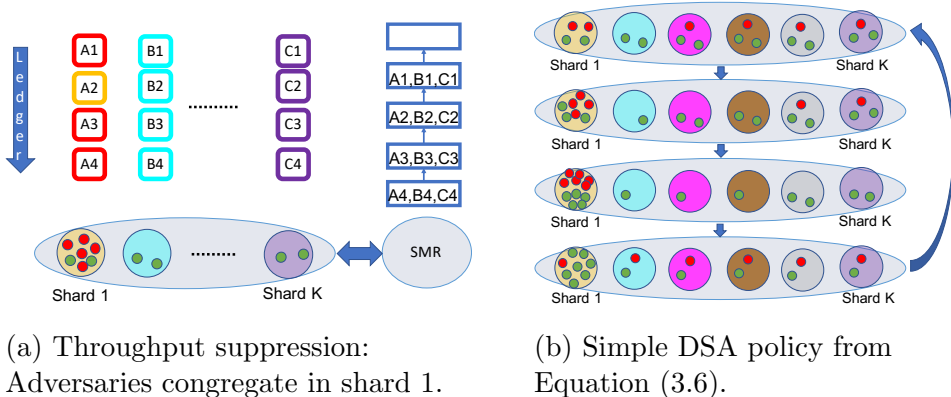


Figure 3.1: Throughput Suppression and an example of a simple DSA policy where the adversary gradually attacks shard 1, and honest nodes follow it with a delay of one round, the attack is reset when the adversary has allocated all of its power to shard 1. Red and green dots correspond to adversarial and honest nodes, respectively. Orange-cyan-...-purple circles correspond to a unique shard. Honest block in a shard has a shard-defining colored boundary, whereas adversarial blocks have a red-colored boundary

combinatorial minimum across the dimensions of the reward vector.

- Along each dimension, a game is played between honest players and adversarial player; if we consider honest parties as one, the game is a two-player zero-sum game (honest reward is $r_i(t)$ and adversarial reward is $1 - r_i(t)$).
- Adversarial player gets to see honest player's action before playing its action: a Stackelberg game.

We want to design algorithms under the following constraints:

- The two-player zero-sum game can be expanded as a distributed n-players vs. 1-player game (where the n-players setup represents the non-coordinating distributed honest nodes);
- The allocation function f_t can be *adversary agnostic*, as defined above

Our core optimization problem in Equation (3.2) looks somewhat similar to an online convex optimization (OCO) problem [39]. In this setting, there is a convex domain $\mathcal{F} \in \mathbb{R}^K$ and an arbitrary sequence of convex functions $h_t : \mathcal{H} \rightarrow \mathbb{R}$ which may be chosen adversarially. The decision-maker (honest

party here) picks an action $\mathbf{x}_t \in \mathcal{H}$ without the knowledge of the future and aims to minimize regret.

$$\sum_{t=1}^T h_t(x_t) - \min_{y \in \mathcal{F}} \sum_{t=1}^T h_t(y)$$

The second term corresponds to optimal reward in hindsight with a stationary policy. However, note that our maximization problem has a combinatorial optimization $\min_i \bar{r}_i(T)$ outside the summation. Thus, it is not possible to represent the average payoff in the form of $\sum_{t=1}^T h_t(x_t)$, which is a key aspect of departure from the online convex optimization setting, rendering that approach nowhere immediate.

One way to integrate combinatorial optimization in OCO is to implement them as sample-path constraints. Constrain $\bar{r}_i(T)$ for every shard i to be greater than a threshold (say $\gamma/2$); in this formulation, the problem will have K long-term sample-path constraints. Thus, OCO with (time-varying) long-term sample-path constraints is a possible route to solve our problem. OCO with sample path constraints was first explored by [40], which showed an impossibility result that the best reward-in-hindsight is not attainable. Later work showed positive results when considering OCO with long-term sample-path constraints but only when the constraint function is fixed [41] or stochastically chosen [42]. [43] proposed a modification to the path constraints by reducing the window length from T to $o(T)$ and showed that it is possible to minimize regret w.r.t. reward in hindsight. This result is not sufficient to yield strong throughput guarantees for our problem.

The key conceptual difference is that in our problem, the structure of the reward is known apriori as a function of the strategy; this difference between *known games* and *unknown games* turns out to be crucial. Dynamic Stackelberg games provide an alternative view [11], with zero-step rewards and a terminal reward of $\min_i \bar{r}_i(T)$, where the adversary can take action after knowing the action of the honest nodes. However, solving the optimal honest policy iteratively (dynamic programming approach) does not yield an analytical form due to our specific reward design. If the reward of $\min_i \bar{r}_i(T)$ were replaced by the vector $\bar{\mathbf{r}}$, then this game is related to the classical Blackwell approachability [12] (an extension of von Neumann’s classical minimax matrix game [44]). Along these lines [45] gave a reinforcement-learning based

strategy that leveraged Blackwell approachability in Stackelberg Stochastic games; their solution is general for known kernels but geared towards being amenable to learning for unknown transition kernels at the cost of slower than the desired rate of convergence and a policy that may be difficult to implement in a distributed manner across honest nodes. Equivalency between Blackwell approachability and online convex optimization without sample path constraints has been shown in [46] and [47].

Our problem has a known transition kernel, which allows us to design a custom allocation algorithm that can converge fast and be simple enough to be provably implemented in a distributed setting. Proofs of classical Blackwell approachability guide how the scalar reward of $\min_i \bar{r}_i(T)$ can be attained by defining the target convex set as a K -dimensional cube with lower boundaries at γ . This is the crux of this chapter’s main result, presented next.

3.1.3 Free2Shard Dynamic Self allocation

Consider the following Free2Shard policy, f_t^{F2S} where $\gamma_i(t)$ is generated as follows:

$$f_t^{F2S} : \gamma_i(t) = \gamma \frac{u_i(t-1)}{\sum_{i=1}^K u_i(t-1)}; \quad u_i(t-1) = (\gamma - \bar{r}_i(t-1))^+. \quad (3.8)$$

The main idea is to allocate honest power to shards performing the worst, i.e., shards with the highest lag from the target honest fraction γ , and not waste any honest power by allocating zero power to shards with time-averaged honest fraction greater than the target. Our main result is in the following theorem.

Theorem 3.1. *For any adversarial strategy,*

$$\psi_{f_t^{F2S}}(T) \geq \gamma \left(1 - \sqrt{\frac{K}{T}} \right). \quad (3.9)$$

We observe that $\psi_{f_t^{F2S}}(T) \geq c\gamma$ for all $T > \frac{K}{(1-c)^2\gamma^2}$, this means that we achieve $\psi_{f_t^{F2S}}(T)$ is arbitrarily close to the information-theoretic limit of γ for T large enough, thus providing tight bounds on approaching the information theoretic limit.

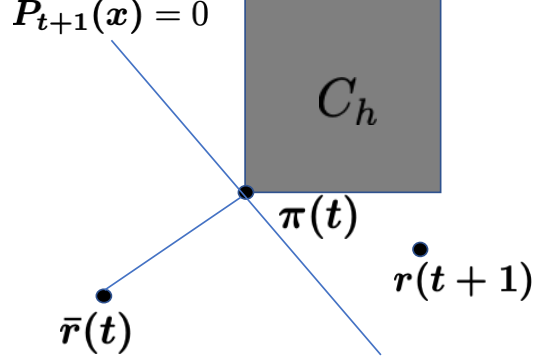


Figure 3.2: $\bar{\mathbf{r}}(\mathbf{t})$ and $\mathbf{r}(\mathbf{t} + 1)$ lie on different sides of the hyperplane P_{t+1}

Proof: We show that following f_t^{F2S} allocation, the average honest fraction vector $\bar{\mathbf{r}}(\mathbf{t})$ approaches the convex set C_γ in \mathbb{R}^K defined as follows:

$$C_\gamma = [\gamma, 1]^K \quad x \in [0, 1] \quad (3.10)$$

so $\bar{\mathbf{r}}(\mathbf{t}) \in C_\gamma$ implies $\min_i \bar{r}_i(\mathbf{t}) \geq \gamma$. Let $\boldsymbol{\pi}(\mathbf{t})$ denote the Euclidean projection of $\bar{\mathbf{r}}(\mathbf{t})$ to the convex set C_γ . Let $P_{t+1}(\mathbf{x})$ denote the hyperplane perpendicular to $\boldsymbol{\pi}(\mathbf{t}) - \bar{\mathbf{r}}(\mathbf{t})$ and containing $\boldsymbol{\pi}(\mathbf{t})$. Observe that $\boldsymbol{\pi}(\mathbf{t}) - \bar{\mathbf{r}}(\mathbf{t}) = \mathbf{u}(\mathbf{t})$:

$$P_{t+1}(\mathbf{x}) : \mathbf{u}(\mathbf{t}) \cdot \mathbf{x} - \gamma \sum_{i=1}^K u_i(\mathbf{t}) = 0. \quad (3.11)$$

We show using the strategy space inequality in Appendix B.2.2 (Equation (B.11), set $s = K$), the following:

$$\begin{aligned} \mathbf{u}(\mathbf{t}) \cdot \mathbf{r}(\mathbf{t} + 1) - \gamma \sum_{i=1}^K u_i(\mathbf{t}) &\geq 0 \quad \forall \boldsymbol{\beta}(\mathbf{t} + 1) \\ \mathbf{u}(\mathbf{t}) \cdot \bar{\mathbf{r}}(\mathbf{t}) - \gamma \sum_{i=1}^K u_i(\mathbf{t}) &\leq 0 \end{aligned} \quad (3.12)$$

The above two inequalities imply that $\mathbf{r}(\mathbf{t} + 1)$ and $\bar{\mathbf{r}}(\mathbf{t})$ lie on different sides of P_{t+1} as depicted in figure 3.2

Let us define d_t as the euclidean distance of $\bar{\mathbf{r}}(\mathbf{t})$ from the convex set C_γ ,

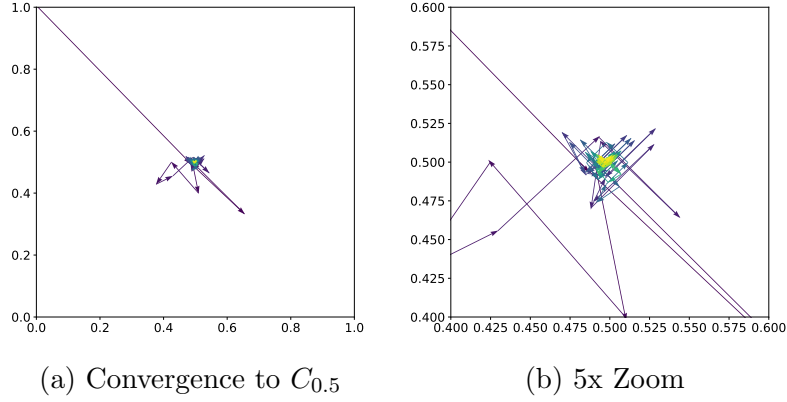


Figure 3.3: Evolution of $(r_1\bar{(t)}, r_2\bar{(t)})$, observe that the distance of $(r_1\bar{(t)}, r_2\bar{(t)})$ from $(0.5, 0.5)$ reduces with time (arrows signify causality)

i.e. $d_t = \|\bar{\mathbf{r}}(t) - \boldsymbol{\pi}(t)\|$, $d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|$ for any $\mathbf{a}, \mathbf{b} \in \mathbf{R}^K$. we show that

$$\begin{aligned}
d_{t+1}^2 &= d^2(\bar{\mathbf{r}}(\mathbf{t} + 1), \boldsymbol{\pi}(\mathbf{t} + 1)) \leq d^2(\bar{\mathbf{r}}(\mathbf{t} + 1), \boldsymbol{\pi}(\mathbf{t})) \\
&= \|\bar{\mathbf{r}}(\mathbf{t} + 1) - \boldsymbol{\pi}(\mathbf{t})\|_2^2 \\
&= \left\| \frac{t}{t+1} \bar{\mathbf{r}}(\mathbf{t}) + \frac{1}{t+1} \mathbf{r}(\mathbf{t} + 1) - \boldsymbol{\pi}(\mathbf{t}) \right\|_2^2 \\
&= \left\| \frac{t}{t+1} (\bar{\mathbf{r}}(\mathbf{t}) - \boldsymbol{\pi}(\mathbf{t})) + \frac{1}{t+1} (\mathbf{r}(\mathbf{t} + 1) - \boldsymbol{\pi}(\mathbf{t})) \right\|_2^2 \\
&= \left(\frac{t}{t+1} \right)^2 \|\bar{\mathbf{r}}(\mathbf{t}) - \boldsymbol{\pi}(\mathbf{t})\|_2^2 + \left(\frac{1}{t+1} \right)^2 \|\mathbf{r}(\mathbf{t} + 1) - \boldsymbol{\pi}(\mathbf{t})\|_2^2 \\
&\quad + \frac{2t}{(t+1)^2} (\bar{\mathbf{r}}(\mathbf{t}) - \boldsymbol{\pi}(\mathbf{t})) \cdot (\mathbf{r}(\mathbf{t} + 1) - \boldsymbol{\pi}(\mathbf{t})).
\end{aligned}$$

$$\begin{aligned}
(t+1)^2 d_{t+1}^2 - t^2 d_t^2 &\leq \|\mathbf{r}(\mathbf{t} + 1) - \boldsymbol{\pi}(\mathbf{t})\|_2^2 \\
&\quad + 2t \cdot ((\boldsymbol{\pi}(\mathbf{t}) - \bar{\mathbf{r}}(\mathbf{t})) \cdot (\boldsymbol{\pi}(\mathbf{t}) - \mathbf{r}(\mathbf{t} + 1))).
\end{aligned}$$

$\mathbf{r}(\mathbf{t} + 1)$ and $\bar{\mathbf{r}}(\mathbf{t})$ lie on different sides of \mathbf{P}_{t+1} hence,

$$2t \cdot ((\boldsymbol{\pi}_{C_\gamma}(\mathbf{t}) - \bar{\mathbf{r}}(\mathbf{t})) \cdot (\boldsymbol{\pi}_{C_\gamma}(\mathbf{t}) - \mathbf{r}(\mathbf{t} + 1))) \leq 0$$

Moreover, $\|\mathbf{r}(\mathbf{t} + 1) - \boldsymbol{\pi}_{C_\gamma}(\mathbf{t})\|_2^2 \leq \gamma^2 K$ and combining the above two inequalities, we get: $(t+1)^2 d_{t+1}^2 - t^2 d_t^2 \leq \gamma^2 K$. Summing terms over $t \in$

$\{1, \dots, T-1\}$, we get $d_T \leq \gamma \sqrt{\frac{K}{T}}$ and observe that $d_T \geq (\gamma - \min_i \bar{r}_i(T))^+$. Equivalently, $\min_i \bar{r}_i(T) \geq \gamma - d_T$ and thus $\psi_{f_t^{F2S}}(T) \leq \gamma(1 - \sqrt{\frac{K}{T}})$.

The convergence of $\bar{r}_i(t)$ for 2 worst performing shards is illustrated in figure 3.3.

3.1.4 Free2Shard Distributed Dynamic Self Allocation

Thus far, we have studied the dynamic self-allocation policy through the lens of the *mean* fraction of the honest nodes. What we really need is a (randomized) DSA policy that can be run by each node locally. A natural strategy is the following: each honest node uses the **Free2Shard** DSA policy to calculate the honest node fractions for each shard and then randomly allocates itself to one of the shards, proportional to the fractions prescribed by the **Free2Shard** policy. While this strategy is natural and performs well in experiments (cf. Section 3.2), we have not been able to evaluate its theoretical performance formally. A slightly modified strategy, described below, does enable a theoretical evaluation.

Free2Shard-dist DSA policy diverges from **Free2Shard** DSA policy in the following way: The policy aims to achieve a honest node fraction in each shard of h , strictly smaller than the information-theoretic optimal value of γ . With a slight abuse of notation, we define $u_i(t-1) = (h - \bar{r}_i(t-1))^+$. The algorithm allows honest nodes to focus on s out of K shards at a time; we order the quantities $u_1(t-1), \dots, u_K(t-1)$ and define $\tilde{u}_i(t-1) = u_i(t-1)$ if the index i is in the highest s order statistics. For other indices i , we set $\tilde{u}_i(t-1) = 0$. We follow the **Free2Shard** policy by substituting $\tilde{u}_i(t-1)$ in place of $u_i(t-1)$ in Equation (3.8), so $\tilde{\gamma}_i = \gamma \frac{\tilde{u}_i}{\sum_{i=1}^K \tilde{u}_i}$. Moreover, to ensure that each of the s prioritized shards get some honest nodes, we lower bound their prescribed honest policy to $q/(1+2q)s$, for some constant q close to 0. We ensure this by projecting the non-zero prescribed honest power $\tilde{\gamma}_i(t)$ to the set $C_{q/s}$; we use the notation $\text{Proj}(\tilde{\gamma}(\mathbf{t}), C_{q/s})$ to denote such Euclidean projection. In summary, **Free2Shard-dist** DSA policy is the following:

$$f_t^{F2S-dist}(h, q, s) : \gamma(\mathbf{t}) = \frac{1}{1 + q/\gamma} \text{Proj}(\tilde{\gamma}(\mathbf{t}), C_{q/s}). \quad (3.13)$$

Notice that the honest node fraction is no longer deterministic since the honest node allocation is randomized (and follows a multinomial distribution); each shard's marginal distribution of honest power is $\Gamma_i(t)$ is distributed as $\frac{1}{N}\text{Binomial}(n, \gamma_i(t))$ and each shard's marginal distribution of honest node fraction $r_i(t)$ is distributed as $\frac{\Gamma_i(t)}{\Gamma_i(t) + \beta_i(t)}$. Notice that $\psi_{f_t^{F2S-dist}}$ is now a random variable, and we show a concentration bound below; the proof is deferred to Appendix B.2.2. This result shows that with high probability, the information-theoretic upper bound of γ can be achieved by the appropriate honest policy.

Theorem 3.2. *For any adversarial strategy, with probability $1 - \delta$,*

$$\psi_{f_t^{F2S-dist}(h,q,s)}(T) \geq \gamma \left(\frac{h}{\gamma} - \frac{1}{\gamma} \sqrt{h^2 \frac{K}{T} + 4hs \sqrt{\frac{2}{T} \log \frac{2}{\delta}}} \right) \quad (3.14)$$

We note that $h = (1 - se^{-\frac{q}{(1+2q)s}(-c+\log c+1)}) \frac{cs}{K(1-2q)} \gamma$ can be set close to γ by choosing the variables q, s, c appropriately.

Outline of Proof. Following the same pattern as the proof of Free2Shard DSA, we show that the average honest fraction vector $\bar{\mathbf{r}}(\mathbf{t})$ approaches the convex set C_h .

We first prove a *strategy space inequality* which states the following:

$$\max_{\gamma} \min_{\beta} \sum_{i=1}^K u_i(t-1) \frac{\tilde{\gamma}_i(t)}{\tilde{\gamma}_i(t) + \beta_i(t)} \geq \gamma \frac{s}{K} \sum_{i=1}^K u_i(t-1) \quad (3.15)$$

We then modify the target allocation policy as $\boldsymbol{\gamma}$ which ensures that $\gamma_i \geq \frac{b}{s}$ as shown in Equation (3.13) where b is a constant and $q = b/(1 - 2b)$. The new allocation policy leads to a *modified strategy space inequality* given by:

$$\min_{\beta} \sum_i u_i(t-1) \frac{\gamma_i(t)}{\gamma_i(t) + \beta_i(t)} \geq \frac{s}{K} \gamma (1 - 2b) \sum_{i=1}^K u_i(t-1) \quad (3.16)$$

We now show that if every honest node chooses a shard randomly according to a choice distribution given by $\boldsymbol{\gamma}$ defined in Equation (3.13), we get a

stochastic strategy space inequality given by:

$$\mathbb{E}_{\Gamma(\mathbf{t})} \left[\min_{\beta(\mathbf{t})} \mathbf{u}(\mathbf{t} - \mathbf{1}) \cdot \mathbf{r}(\mathbf{t}) \right] \geq h \sum_{i=1}^K u_i, \quad (3.17)$$

This is similar to Equation (3.12) in Free2Shard DSA proof. We then show that following the above honest allocation strategy, time-averaged honest fraction approaches the convex set C_h with distance decreasing with time T as:

$$d_T^2 \leq h^2 \frac{K}{T} + \frac{2}{T} \sum_{t=1}^{T-1} t(Y_t) \quad (3.18)$$

where $Y_t = (\mathbb{E}_{\Gamma_i(t)} [\sum_i u_i(t-1)r_i(t)] - ((\mathbf{u}(\mathbf{t} - \mathbf{1})) \cdot \mathbf{r}(\mathbf{t})))$ is a martingale difference sequence with respect to the history at time t and $|Y_t| \leq 2hs$. Using the Azuma- Hoeffding inequality, we have

$$\mathbb{P} \left(\frac{1}{T} \left\| \sum_{t=1}^{T-1} Y_t \right\| > \epsilon_m \right) \leq 2e^{-\frac{T\epsilon_m^2}{8h^2s^2}}. \quad (3.19)$$

This allows us to conclude that with probability $(1-\delta)$, the distance to convex set converges to 0 as $d_T^2 \leq h^2 \frac{K}{T} + 4hs \sqrt{\frac{2}{T} \log(\frac{2}{\delta})}$. We observe that $d_T \geq (\gamma - \min_i \bar{r}_i(T))^+$. Equivalently, $\min_i \bar{r}_i(T) \geq \gamma - d_T$ and thus $\psi_{f_t^{F2S-dist}(h,q,s)}(T) \geq \gamma \left(\frac{h}{\gamma} - \frac{1}{\gamma} \sqrt{h^2 \frac{K}{T} + 4hs \sqrt{\frac{2}{T} \log \frac{2}{\delta}}} \right)$.

3.1.5 Number of Shards and Nodes

Conventional modeling (and the corresponding sharding literature) supposes that the number of nodes N is much larger than the number of shards K . This modeling is central to the working of node-to-shard (N2S) allocations: this way each shard has a sufficient number of honest nodes. In practice, one can imagine several shards being inactive during certain periods of time and conceivably $K > N$. In this scenario, we can derive a tighter information theoretic bound than the one in Equation (3.5) since $\psi(T) = \gamma$ implies that each shard remains active at all rounds even if there aren't sufficient honest nodes to maintain all the shards in any round; this is done next.

Information Theoretic Limit The sum of honest nodes of all shards is limited by the total number of honest nodes in the system since, at every round, there will be at most N shards which can be maintained by honest nodes, and the adversary can set its policy: $B_i(t, \Gamma_i(t)) = \frac{\beta}{\gamma} \Gamma_i(t)$. Thus, the honest fraction of the N out of K shards which are non-zero is γ , yielding the following bounds:

$$\max_{\{f_t\}_t} \min_{\{\beta_i(t)\}_{i,t}} \sum_i \{\bar{r}_i(T)\} \leq \gamma N \quad (3.20)$$

$$\begin{aligned} \max_{\{f_t\}_t} \min_{\{\beta_i(t)\}_{i,t}} \min_i \bar{r}_i(T) &\leq \frac{1}{K} \sum_i \bar{r}_i(T) \\ \psi(T) &\leq \gamma N / K. \end{aligned} \quad (3.21)$$

We note that the Free2Shard-dist self allocation strategy smoothly meets this new upper bound; this is done via the honest nodes focusing only on a subset of shards in a round to achieve $\psi_{f_t^{F2S-dist}(h,q,s)}(T)$ which is within a $O(\frac{1}{\log N})$ multiplicative factor of the improved information theoretic upper bound: Theorem 3.2 states that for large enough T , $\psi_{f_t^{F2S-dist}(h,q,s)}(T) \geq 0.5h$. We observe that $h \geq \frac{a}{\log N} \frac{N}{K}$ where a depends on the choice of c, q and $s = \frac{N}{4 \log N}$. Thus, $\psi_{f_t^{F2S-dist}(h,q,s)}(T) \geq \frac{a}{2 \log N} \frac{N}{K} \gamma$ which is within $O(\frac{1}{\log N})$ multiplicative factor of the information theoretic limit in Equation (3.21).

3.1.6 Discussion

A small minority can ensure sufficient throughput for each shard. For Free2Shard-dist DSA results to hold, not all honest nodes need to follow Free2Shard-dist policy: Theorem 3.2 remains true even if a small (but constant) fraction of the honest nodes follow the policy. For example, the Free2Shard-dist policy allows us to set $\gamma = 0.1$. We note that this is very valuable in a practical setting since many blockchains have their own foundations that own sub-majority stakes. In contrast to existing protocols, even a small minority following the rotation policy can ensure sufficient throughput across all shards in Free2Shard. Note that we still need an honest majority of nodes running the full node for Trifecta to ensure safety; however, liveness is guaranteed even if only a small fraction of those honest nodes follow Free2Shard-dist DSA.

Slow rotation. Free2Shard-dist DSA results hold true even if honest nodes rotate slowly at a rate of Δ and adversaries rotate instantly. The results hold since an optimal adversary will modify β at the same rate as γ . We will now prove this argument. Let $\gamma_i(t) = \gamma_i[n] \forall t \in \{(n-1)\Delta+1, \dots, n\Delta\}$ and $\beta_i[n] = \sum_{t=(n-1)\Delta+1}^{n\Delta} \frac{1}{\Delta} \beta_i(t)$. We observe that $\frac{\gamma_i(t)}{\gamma_i(t)+\beta_i(t)}$ is convex in $\beta_i(t)$, thus, the time-average of a given round is lower bounded as follows:

$$\sum_{t=(n-1)\Delta+1}^{n\Delta} \frac{\gamma_i[n]}{\gamma_i[n] + \beta_i(t)} \geq \sum_{t=(n-1)\Delta+1}^{n\Delta} \frac{\gamma_i[n]}{\gamma_i[n] + \beta_i[n]}. \quad (3.22)$$

Thus, the optimal adversarial strategy is to modify β at the same rate as γ ; Moreover, the results hold when nodes randomize their rotation choice (choosing to rotate with probability $\frac{1}{\Delta}$). This makes for a distributed implementation and has the added benefit that asynchronous rotation alleviates network load by preventing all state synchronization communication load from being focussed simultaneously.

3.2 Simulations

We run a simulation of DSA based on the parameters observed from the system experiments. The code is available at [48]. The theoretical analysis for Theorem 3.2 takes into account various randomness observed in practice; we empirically verify the robustness stated analytically and add in further realistic effects to estimate the performance of DSA in the real world.

Simulated adversary: The adversary controls 50% of the nodes in the network and undertakes censorship attacks where it tries to throttle the throughput of the worst-performing shard. It can switch shards instantaneously as opposed to honest nodes, which takes five inter-proposer-block times to reallocate to a new shard. The adversary has a global view of the honest node allocation at any time and uses that information to take its next step according to an adversarial strategy aimed at introducing a liveness attack by decreasing the honest fraction of a target shard. The space of such adversarial adaptations is infinite (thus cannot be empirically evaluated); we show mathematically that the worst-case throttling of the honest node fraction in any shard can be at most as small as $\frac{\log K}{K}$; here $\tau = \frac{\log K}{\log \log K}$. The

formal statement is below

Proposition 3.2. *Consider an adversary that works in periods of length τ . In any period at round t , the adversary uniformly allocates its power on the $K \left(\frac{1}{\log K}\right)^t$ worst-performing shards. The period ends at $t = \tau$ and the adversary restarts with $t = 1$.*

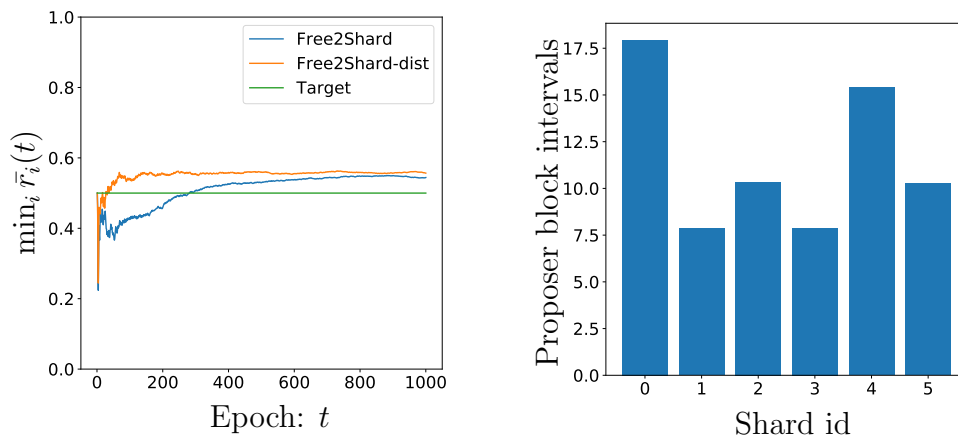
Then for any honest policy, the worst-case shard's honest node fraction is less than $O\left(\frac{\log K}{K}\right)$ for all rounds $t \leq \tau$.

The proof can be found in Appendix B.2.3. The adversary follows the above strategy to attack the worst-performing shards.

Simulated Realistic effects

- *Link latency:* Information propagation is assumed to happen over a synchronous network. However, the adversary can control the delays between nodes within the synchronous network assumptions to ensure that half of the nodes receive a message from the adversary just before the end of an epoch and the other half receives the message just after the end of an epoch. The adversary can use this advantage to mount an attack where it releases a proposer block containing a lot of adversarial shard blocks at the boundary of an epoch, such that half of the nodes receive it by the end of the epoch and the other half do not. This attack ensures that the mining power estimates of these two cohorts of honest nodes are different hence deviating from the analytical assumptions.
- *Native-shard shift latency:* When a node changes its native-shard at the end of an epoch, it needs to reset various transaction and state databases and receive verified latest state checkpoint (Section 2.3) for the new native-shard from its peers; moreover it needs to catch up to the latest state by validating the transactions confirmed after the state checkpoint. The transfer of state and state verification involves updating the UTXO database and verifying the state Merkle trees. Based on the resource usage breakdown from table 2.1 and other profiling data, we estimate that under a state size of 18M accounts and a proposer block interval time of 10s, such transfer should take approximately 50s or 5 proposer blocks. We set an epoch length of 100 proposer blocks to reduce the fraction of the time needed for state synchronization.

- *Mining difficulty adjustment delay:* In our analysis, we assumed that the mining difficulty is always adjusted instantly such that at any time a constant rate of blocks (from honest and adversarial miners) are added to the shard ledger. Instant difficulty adjustment cannot be implemented in practice since there is a delay in observing the increased mining power and setting the difficulty. For our experiments with Triecta proposer blocks, each proposer block refers to 3000 shard blocks per shard; given such a large number of shard blocks referred, we update the shard block difficulty at the interval of every 10 proposer block (equivalent to 30K shard blocks; contrast with Bitcoin where this is done every 2016 blocks).



(a) Worst-case average honest fraction (b) Mean transaction inclusion latency

Figure 3.4: *Left:* We observe that the chain quality of the worst performing shard improves to the desired level (0.56) around epoch 50 while running **Free2Shard-dist**, whereas **Free2Shard** performs similarly and stabilizes at 0.54 at epoch 400. *Right:* Transaction inclusion latency for **Free2Shard-dist** is uniform across shards and with no shard experiencing unreasonable latency

Homogeneous sharding with $N > K$: We simulate the dynamic adversarial strategy above and the distributed (randomized) versions of **Free2Shard** and **Free2Shard-dist** as the honest policy. We aim to maintain the same honest fraction in each shard (equal to $0.5 = \gamma$ here). We plot the time average honest fraction of the worst-performing shard as a function of time in Figure 3.4a and make the following observations. (a) **Free2Shard-dist** performs slightly better than **Free2Shard** – this is because **Free2Shard-dist** sets a lower

bound to the honest power allocation on the allocated shards, this ensures that the increase in mining power across epochs is by a smaller factor, (b) `Free2Shard` and `Free2Shard-dist` achieve an honest fraction of 0.56 and 0.54 as compared to the worst-case $1/K$. We also plot the average latency of transaction inclusion defined as the time taken between transaction generation and transaction inclusion in the shard ledger in Figure 3.4b. We notice that the mean transaction inclusion delay between the worst and best-performing shards differs only by 100s, demonstrating uniformity across shards. We show that the performance of `Free2Shard-dist` is consistent even with 100 shards in Appendix B.3.

Heterogeneous sharding with $N < K$: With $N < K$, under a homogeneous setting, all shards will attain an honest fraction around $\gamma \frac{N}{K}$ as guided by the information-theoretic limit in equation 3.21, however, we might want some shards to attain higher honest fractions. In practice, different shards have different activity levels, demanding different target levels of participation from the nodes. We propose the target honest fraction of each shard to decrease as $1/(\lceil i/5 \rceil + 1)$, with $N = 25, K = 100, \beta = 0.5$; where i is the index of the shard. We simulate DSA with the adversary stated in 3.2 with instant state transfers and mining estimation. The plot of the time average honest node fraction across the shards at the end of the simulation is shown in Figure 3.5. We make the following observations. (a) `Free2Shard` supports heterogeneous honest fraction allocation across shards; (b) even though N is sufficiently smaller than K , we can set some shards to achieve a target honest-factor of γ , which is the best we can achieve even with $N > K$. We see that the performance in meeting the targets is strong; a theoretical justification for this strong performance for heterogeneous sharding is of great interest and is deferred to future work.

3.3 Conclusion

In this chapter, we proposed `Free2Shard` DSA to resolve the liveness vulnerability of `Trifecta`. `Free2Shard` ensures that the long-term honest fraction of the worst performing shard is $O(1)$, hence achieving performance arbitrarily close to the information-theoretic bound. It achieves this while ensuring that honest nodes are free to self-allocate, maintaining identity-free partici-

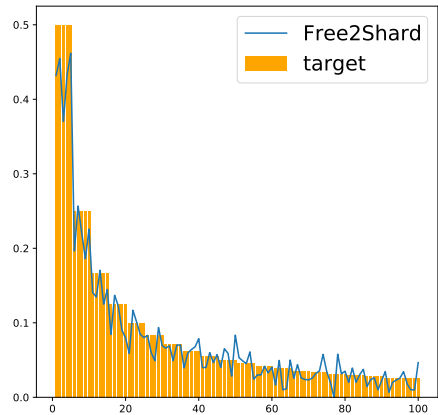


Figure 3.5: Comparison between target honest fraction set for heterogeneous sharding and average honest fraction achieved at the end of the experiment

pation of consensus nodes, and ensuring dynamic availability. Additionally, **Free2Shard** works even if a small fraction of the nodes participate in DSA and rotate at a much slower rate than the adversaries. Simulations with parameters inspired by systems experiments show that **Free2Shard** converges fast against a latency-optimal adversary even under some practical effects not modeled during mathematical analysis. Thus, **Trifecta** with **Free2Shard** satisfies all three properties of the protocol trilemma.

Additionally, achieving near-optimal throughput in the number of nodes implies that the resource cost per transaction of **Free2Shard** is very low, thus minimizing the resource cost axis of the economics trilemma. Moreover, the identity-free, self-allocation principle ensures that shards seeking high rents through transaction fees can see the rents competed out by additional miners joining that shard. This competition reduces the rental cost per transaction, minimizing the rental cost axis of the economics trilemma.

Trifecta with **Free2Shard** is a PoW blockchain; PoW blockchains are difficult to bootstrap since the initial mining power is low enough for adversaries to gain a mining majority. In the next chapter, we discuss **Advocate**, which is an optimal bootstrapping gadget.

CHAPTER 4

ADVOCATE

The focus of this dissertation so far has been on PoW blockchains since as epitomized by Bitcoin, they have proven themselves to be a successfully secure blockchain design. The security has been shown in theory [49] as well as in practice (Bitcoin has not seen any safety or liveness incidents in more than 13 years of being online). An important, and less studied, aspect of PoW blockchains is that they are very hard to bootstrap: at the early stages of a PoW blockchain, there is not much participation (measured in terms of mining hash power); making it relatively easy for an adversary to procure mining power and overpower the honest majority.

If the PoW blockchain can avoid the dangers of such attacks in its infancy, eventually significant hash power is attracted to participate in the mining process and the security is correspondingly strengthened. Thus secure *bootstrapping* is crucial to the successful development of any PoW blockchain. Principled approaches to bootstrapping PoW blockchains is the focus of this paper.

Related Works

Checkpointing. Checkpointing is a standard technique used in state machine replication protocols [50], where a centralized server issues checkpoints attesting to the recent state of the protocol. In blockchains, checkpoints attest to the hash of well-embedded blocks every so often that new users can securely bootstrap using a recent execution state of the protocol. A key benefit of such checkpointing is that an adversary, even with super-majority mining power, cannot create a long-range reversion attack (i.e., forking from a block created long ago). Since such a long-range fork will deviate from the stated checkpoint, clients will reject them. Practical blockchains utilizing checkpointing include Bitcoin [1, 51], Peercoin [52], Feathercoin [53], and RSK [54]. Satoshi Nakamoto himself (presumably honest) maintained a cen-

tralized checkpointing mechanism until late 2014. Additionally, checkpoints of some form have been central in the context of Proof-of-Stake (PoS) protocols, including Ouroboros [55], Snow White [56], and Ouroboros Praos [57], as well as e.g. in hybrid consensus [58], Thunderella [59], ByzCoin [60], and Algorand [61]. In a related context, Fantomette [62] employs distributed checkpoints to secure a blockDAG-based ledger.

Finality Gadgets: A crucial problem in checkpointing is ensuring the safe delivery of checkpoints to the blockchain client. A new generation of blockchain algorithms has sought to decentralize this process by designing a separate distributed consensus protocol to issue checkpoints. We will refer to this class of solutions as finality gadgets, which are comprised of a Byzantine Fault Tolerant (BFT) protocol for finalizing blocks created by a Proof-of-work (PoW) or Proof-of-stake (PoS) chain protocol. They have become prevalent methods for combining the best features of the BFT and PoW protocols. They are proposed for deployment in many major blockchains, including Ethereum 2.0 [16, 63]. Depending on the context, the checkpoint committee can be comprised of a fixed committee (for example, run by independent community leaders), or the committee itself can be elected using stake deposits.

Rationale for Finality Gadgets. There are multiple reasons for utilizing finality gadgets, and different protocols emphasize different properties. We enumerate the properties that motivate the development of finality gadgets as follows.

1. Safety against long-range attacks;
2. Economic finality;
3. Availability vs. Finality tradeoff capability;
4. Responsiveness: low-latency block confirmation;

The *raison d'être* of finality-gadgets (including centralized checkpoints) is to provide property (1): safety against long-range attacks by an adversarial majority. In a Proof-of-Work system, this is a crucial safeguard since an adversary can temporarily control a super-majority mining power, for example, by renting cloud mining equipment. Finality gadgets and checkpoints can prevent such attacks.

Beyond this basic reason, different protocols optimize for different criteria. Casper [64, 65] focuses on (2) that ensures that if safety is violated, the

malicious action is detected, and at least one-third of the staking nodes will lose their stake, a similar approach is taken by GRANDPA [63] and AFGJORT [66]. Recent works [67, 68] have identified general BFT protocols which have such detectability. Some protocols like [69, 70] focus on (3) by designing gadgets that let users prioritize adaptivity or availability by implementing different confirmation rules. Some protocols like WINKLE [71] guarantee (1) by utilizing transaction traffic for voting. Finally, other finality gadgets (e.g. AFGJORT [66]) are optimized for property (4): responsiveness, the ability of the BFT to confirm blocks produced by the PoW chain near-instantaneously.

Bootstrapping gadget: A bootstrapping gadget is a finality gadget with one additional property: *Liveness despite adversarial majority*. During the initial stages of a novel PoW protocol, liveness is required to ensure that honest miners are rewarded for their effort even under an adversarial majority. Lack of a live checkpointing protocol creates an undesired spiral: Low honest participation \rightarrow Low honest miner rewards \rightarrow honest miners leaving (lower participation).

Motivation and contributions

Key shortcoming of existing solutions. While all the aforementioned protocols satisfy several properties of finality gadgets, none of them can work as bootstrapping gadgets. This was observed in a recent paper [72] for even the simple centralized checkpointing protocol. An adversary controlling a majority mining power can issue a long private adversarial chain that does not contain honest transactions. When such a chain is checkpointed repeatedly, honest clients lose liveness in the system. The paper proposed the inclusion of a random nonce as well as a checkpoint certificate (issued by the central checkpointer or BFT) to reduce the impact of this attack. The key idea is that since this random nonce needs to be included in the next block, this creates a renewal event where the adversarial blocks stored prior to the event have to be disregarded, creating a new race between the honest and adversarial chains at each checkpoint. While this approach can ensure a non-zero chance that the honest chain can win, thus giving asymptotic liveness, the latency of transaction inclusion as well as the chain quality (fraction of honest blocks in the final ledger) and the corresponding mining rewards for honest miners decrease exponentially as mining power increases beyond 50% or as

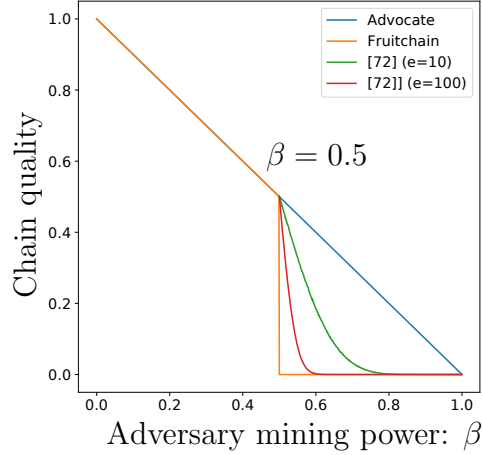


Figure 4.1: Chain quality (CQ) of fruitchains deteriorates to 0 when $\beta > 0.5$; an upper bound on CQ of previous work [72] deteriorates rapidly with epoch length and β ; CQ of **Advocate** is optimal

the inter-checkpoint interval increases.

Main Contributions. This chapter focuses on building a bootstrapping gadget that achieves safety *and* liveness under an adversarial majority. We propose **Advocate**, a new scheme that achieves optimal chain quality. The core idea is the inclusion of appropriate reference links to checkpoint blocks. Variations of this idea have been proposed in different contexts in the literature: in [49] for achieving a 1/2 threshold Byzantine Agreement, in Fruitchains [34] for designing incentives, in inclusive protocols [73] for minimizing block wastage due to forking and in general DAG (directed acyclic graph) protocols, such as Conflux [74], for improving throughput. We prove that **Advocate** achieves optimal chain quality while ensuring transaction inclusion for all honest transactions within two epochs even under a 99% mining adversary. The plots for chain quality of related works are in Figure 4.1; an upper bound on the chain quality of [72] diminishes rapidly with adversarial power (β) and epoch size (e), whereas **Advocate** achieves the *optimal chain quality* equal to the honest mining power ($1 - \beta$).

General applicability. We create appropriate blackbox interfaces through which our protocol can employ any BFT protocol for checkpointing, thus making our construction widely applicable. We also demonstrate that our bootstrapping gadget **Advocate** can work with various PoW protocols beyond the Nakamoto consensus protocol.

System Implementation. We perform extensive experiments on a distributed testbed to demonstrate the robustness of our protocol under and up to 90% adversarial mining majority and compare performance gains with prior state-of-the-art. To demonstrate the compatibility of **Advocate** with high throughput parallel-chain architectures, we implemented **Advocate** on **Prism** and demonstrated an honest throughput of 8,200 tx/s under a 70% adversarial majority. The system implementation code is available at [75].

Organization

The rest of the chapter is organized as follows. Section 4.1 provides an overview of the preliminaries used in our work, including the threat model and the distributed ledger’s properties and block production mechanisms. Section 4.2 describes **Advocate** under a single checkpointing node and provides its security analysis to show safety and liveness with optimal chain quality. Section 4.3 extends **Advocate** to ensure similar performance and security guarantees under a committee based BFT-SMR protocol by providing a unified network functionality. Section 4.5 integrates **Advocate** into parallel-chain architectures by providing a *meta-protocol* that can be readily integrated into **Prism**, **OHIE** and ledger-combiner to achieve high throughput under an adversarial majority.

4.1 Preliminaries

4.1.1 The Distributed Ledger Model

The distributed ledgers analyzed in this chapter are constructed as blockchains. A ledger is formed as a hash chain (or tree) of blocks containing transactions that alter the ledger’s state. New blocks, which extend the chain, are created by mining parties at regular intervals. Given a tree of blocks, each party chooses a single branch as the *main chain*; blocks that are not part of the main chain are called *uncles*.

We assume that the ledger protocol is set in the synchronous setting just as in the previous chapters, with a delay upper bound of Δ . Specifically, the execution proceeds in rounds. On each round, every party is activated to participate in the protocol. Communication is performed via a “diffuse”

functionality, i.e., a gossip protocol, such that no point-to-point communication channels exist, but rather a peer-to-peer network is formed. Therefore, every message produced at round r is received by all other parties by round $r + 1$. We also assume that the number of participating parties is fixed for the duration of the execution.

The ledger’s core properties are described in detail by the Bitcoin Backbone model [49].

Definition 4.1 (Stable Block and Transaction). *A block is stable if it is k -deep in the main chain. A transaction published in a stable block is also stable.*

Definition 4.2 (Safety). *A transaction reported as stable by an honest party on round r is reported as stable by all honest parties on round $r + 1$, at the same position in the ledger.*

Definition 4.3 (Liveness). *A transaction which is provided continuously as input to the parties becomes stable after u rounds.*

An important property of interest is *chain quality* (Definition 4.4) [49, 76]. Briefly, this property ensures that the number of blocks that each party contributes to the chain is bounded by the party’s mining power μ .

Definition 4.4 (Chain Quality (q, l)). *Let q be the proportional mining power of \mathcal{A} . Chain quality with parameter l states that for any honest party \mathcal{P} with chain \mathcal{C} , it holds that, for any l consecutive blocks of \mathcal{C} , the ratio of adversarial blocks is at most $1 - q$.*

4.1.2 Threat Model

Our work considers polynomial-time executions, such that all parties, including the adversary \mathcal{A} , are locally polynomial-bounded. On each execution round, the adversary may “corrupt” a party, at which point it accesses the party’s internal state; when the corrupted party is supposed to be activated, the adversary is activated instead. Additionally, \mathcal{A} is “adaptive”, i.e., corrupts parties on the fly (till the total corruption budget of β), and “rushing”, i.e., retrieves all honest parties’ messages before deciding its strategy at each round.

\mathcal{A} controls β of the network’s mining power and tries to break safety and liveness. To break safety, \mathcal{A} forces two non-corrupted nodes to accept different chains as stable, i.e., to report different transactions as stable in the same position in their respective ledger. To break liveness, \mathcal{A} attempts to prevent a transaction from becoming stable within u rounds. We explore settings where the honest majority assumption is violated, i.e., when the adversary may control more than $1/2$ of the net mining power. In those settings, the ledger cannot be secure in a standalone fashion, hence the need for the checkpointing protocols presented in this work.

4.2 Advocate: Optimal bootstrapping of PoW Protocols

Our main contribution is a novel protocol, **Advocate**, that ensures safety and liveness against a super-majority mining adversary on the PoW chain. This section considers a single (honest) checkpointing node in order to clearly present the main innovations, while the following sections relax this assumption by proposing a distributed checkpointing federation.

Checkpointing in **Advocate** is performed via *certificates*. Specifically, at regular intervals of e blocks on the main chain, the checkpointing service issues a signed certificate, which is published on the chain within c blocks on the main chain. The certificate defines the canonical chain that parties should adopt. **Advocate** is parameterized by the two values c and e .

4.2.1 Checkpointing Party Behavior

The checkpointing party is connected to the blockchain network, so at each round t it holds a view of the PoW chain. Therefore, on any round, the checkpointing party maintains a list of leaves $\mathcal{L}(t)$ of its local block-tree.

The party issues a checkpoint certificate at regular intervals (i.e., every e blocks). The i -th checkpoint certificate issued by the party is denoted C_i . A checkpoint certificate is constructed as follows: $C_i = \{B_i, \mathcal{R}_i, S_i\}$; B_i is the checkpointed block, i.e., the block of the main PoW chain that the party checkpoints; \mathcal{R}_i is a list of references of blocks that are not part of the main chain, i.e., leaves of the block tree which are not checkpointed; S_i

is the signature of the certificate. The initial, bootstrapping certificate is $C_0 = \{0, \{0\}, S_0\}$. For the rest of the paper, a *checkpointed* block is a block which is either referenced in a checkpointing certificate or exists in a branch, a block of which is referenced in a certificate.

In summary, the checkpointing party behaves as follows:

- It runs a full node on the PoW network.
- When it receives a block B_i from the PoW node, which is least c blocks downstream of the previous checkpoint *and* the chain contains the previous checkpoint certificate, it issues a new checkpoint certificate $C_i = \{B_i, \mathcal{R}_i, S_i\}$, which finalizes B_i .
- The list $\mathcal{R}_i \subseteq \mathcal{L}(t)$ contains references to all the leaves of the block tree. In practice, it suffices to reference only $\mathcal{R}_i \setminus \mathcal{R}_{i-1}$, i.e., the incremental information since the previous checkpoint; we ignore this distinction for conceptual clarity and set $\mathcal{R}_i = \mathcal{L}(t)$.

4.2.2 Main Chain behavior

With the introduction of checkpoints, the PoW node behavior needs to change appropriately. The *key change* is that, once a new certificate checkpoints block B_i , it should be published in at least one of the c blocks that immediately extend B_i ; the first block that includes the certificate is called the *referring block*. The nodes follow the *longest checkpointed* chain. In summary, *Advocate* modifies the main-chain rule as follows:

- Go to B_i in the blocktree.
- If there exists a descendant block B_i^r within c blocks of B_i that contains C_i , pick the longest chain which contains B_i^r as the main chain. (*Note: A block B_i^r which contains C_i but is more than c blocks after B_i is not acceptable.*)
- If no such B_i^r exists, then either of the following holds:
 1. B_i is *not* c -deep in the longest chain: pick the longest chain containing B_i as the main-chain.

2. B_i is c -deep in the longest chain: pick one of the chains which is $(c - 1)$ -deep and contains B_i as the main chain (breaking ties arbitrarily).

Mining behavior Miners follow the main chain rules described above. Additionally, w.r.t. a checkpoint certificate C_i , two cases exist:

1. the main chain contains C_i in some block B_i^r : proceed mining as usual.
2. the main chain does not contain C_i : include C_i alongside the list of transactions to be mined.

We suppose that when a miner creates a new block, the block contains all transactions in the miner’s mempool (in practice, this requires sufficiently incentivized transaction fees). With hindsight, this assumption will prove useful to argue that a transaction is published in the first honestly-generated block that is produced after the transaction’s creation.

Let main chain oracle F_{mco} represent the view in the execution of the underlying consensus algorithm (Nakamoto) and the public global tree G_t at time t is received by time $t + \Delta$ (by the synchronous network assumption). The main chain oracle gets the additional checkpointing information from **Advocate**. The interaction of a main chain oracle (F_{mco}) with the **Advocate** functionality can be formalized using a functionality F_{Advocate} described next.

Advocate F_{Advocate}

F_{Advocate} and F_{mco} interact using various push-pull messages as described below; message delay is considered zero since both the modules use the same machine

1. *potentialCandidate*: F_{mco} sends this message as soon as it receives a new block B_n that satisfies the checkpoint criteria
2. *candidateFinalized*: F_{Advocate} sends this message as soon as it receives a new checkpointing certificate C_n
3. *sendReferences*: On receiving a a potential candidate, F_{Advocate} immediately checks with the F_{mco} to see if there are any unreferred uncle blocks; F_{mco} replies immediately with *unreferredBlocks*

4. *isCertValid*: On receiving a checkpoint certificate \mathcal{C}_n on the main chain, F_{mco} immediately requests $F_{Advocate}$ to check if it's valid (correct signatures, etc.); $F_{Advocate}$ immediately replies in boolean using *certValidity*.
5. *isValidBlock*: Triggered when $F_{Advocate}$ receives a new checkpoint certificate; it sends the above message with block hashes to the main chain oracle, to see if those blocks are valid. The main chain oracle replies with *validBlock*
6. *validBlock*: Response to the above; if F_{mco} has not received the block yet, it waits for Δ before the reply. If the block is not yet received or is invalid due to main chain consensus protocol, it replies *False*, else it replies *True*.
7. *certRequest* A query to the certificate database stored by $F_{Advocate}$, the response *requestedCert* is immediate.

Note that all the interactions between $F_{advocate}$ and F_{mco} are immediate except for *isValidBlock* which has a maximum delay of Δ ; delay bound of the synchronous network. The checkpointing party maintains the same functionalities F_{mco} and $F_{Advocate}$ with an additional checkpointing service functionality F_{cps} .

Checkpointing module F_{cps}

F_{cps} receives the *checkpointCandidate* and *unreferredBlocks* in the form of *inputValue* message from $F_{Advocate}$. This message takes zero delay once $F_{Advocate}$ receives *potentialCandidate* from F_{mco} . F_{cps} queries $F_{Advocate}$ regarding the validity of this input using the *isInputValid* and *inputValidity* messages and gets a reply after a delay t_{cps} . If the input is valid, F_{cps} immediately certifies the input and sends the message to $F_{Advocate}$.

Note that the above process will happen within a time t_{cps} with $t_{cps}/\Delta \ll 1$, since the process is in the same machine. The message *isInputValid* and *inputValidity* seems redundant for now; however, we will see in section 4.3, that this message classification is critical.

4.2.3 Decoupled Validity: Ledger Creation

Without loss of generality, we assume that the execution completes with the issuing of a final checkpoint. To construct the aggregate ledger at any point of the execution, the blocks of the main chain are concatenated with the blocks of the non-main branches. In the aggregate ordering, the main chain blocks, up to and including the referring block (i.e., the block which includes the checkpointing certificate), are prioritized over the blocks which are not part of the main chain (i.e., the “uncle” blocks).

Formally, let $T(\mathcal{L}_i)$ be the tree corresponding to the leaves \mathcal{L}_i . Let the forest F_i be the difference between $T(\mathcal{L}_i)$ and the previously checkpointed tree: $F_i := T(\mathcal{L}_i) \setminus \{T(\mathcal{L}_{i-1}) \cup \text{Chain}_i\}$, where Chain_i is the main chain up to (and including) the referring block for checkpoint C_i . $\pi(\cdot)$ denotes a topological sort of the blocks in a forest, with ties broken in a universal manner (e.g. via block hashes). The aggregate ledger is constructed by concatenating the referring block (i.e., Chain_i) with $\pi(F_i)$. Therefore, when the referring block for certificate C_i is read, the blocks referred by C_i (i.e., the blocks in $\pi(F_i)$) are also read in the order defined in C_i . We assume that the (honest) checkpointing node follows the universal topological sorting π when constructing the reference list of certificate C_i . Note that we can follow other universal ordering approaches for blocks in F_i (e.g. sort by hash) without affecting the security of our protocol.

Figure 4.2 depicts the ledger construction, s.t. the sanitized ledger is obtained by parsing the main chain and referenced blocks and removing invalid (e.g. double spending) entries.

4.2.4 Security Properties of Advocate

In this section we show that **Advocate**, under the centralized setting of a single honest checkpointing party, satisfies a host of desirable properties. First and foremost, Theorems 4.1 and 4.2 prove that **Advocate** satisfies safety and liveness (cf. Section 4.1.1).

Theorem 4.1 (Safety). *Let B be a block that is checkpointed by **Advocate** via certificate C . If B is part of the main chain, then B is stable (cf. Definition 4.1). If B is an uncle, then B is stable if C is published in a block which is k -deep, with $k = e - c$.*

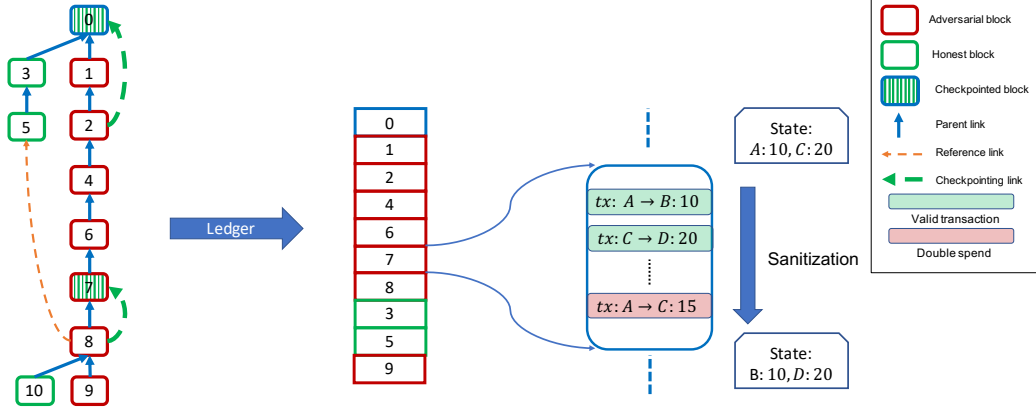


Figure 4.2: **Advocate** protocol: 30% honest mining power ensures 30% of the blocks in the ledger are mined by honest miners. Ledger sanitization ignores/removes invalid transactions post ordering.

Proof: Let $C_i = \{B_i, \mathcal{R}_i, S_i\}$ be the i -th certificate and B a block checkpointed by C_i . By the definition of the protocol, all honest parties eventually accept a chain that contains a referring block B_C , which contains C_i . Observe that, once C_i is created, the ledger position of B_i is finalized, given the ledger construction description in Section 4.2.3. Therefore, if B is part of the main chain, its ledger position is also fixed as soon as C_i is created. If B is an uncle, its ledger position depends on the referring block B_C . Specifically, the ledger construction rules enforce that the uncle blocks, which are checkpointed by C_i are appended in the final ledger *after* the referring block for C_i . However, a referring block *can* be reverted if a chain appears which is both valid (i.e., contains a correct referring block) and long enough. Therefore, the position of uncle blocks, which are checkpointed by C_i , is finalized only when the certificate C_{i+1} is issued, which occurs after, at most $e - c$ main chain blocks. \square

Theorem 4.2 (Liveness). *Let h be the probability that at least one honest block is created per round; **Advocate** satisfies liveness (cf. Definition 4.3) with parameter $u = \lceil \frac{2}{h} \rceil \cdot e$.*

Proof: The proof follows directly from the ledger construction (cf. Section 4.1.1) and Theorem 4.1. Specifically, let t be the round when block B is created. If B is a main chain block, then it becomes stable with the issuing of the first checkpoint after t which, by definition of the checkpointing behavior (Section 4.2.1), occurs at most $\frac{e}{h}$ rounds after t . If B is an uncle

block, then, as shown in Theorem 4.1, it becomes stable with the issuing of the first checkpoint *after* B becomes checkpointed; in other words, B becomes stable when 2 checkpoints are issued after it is created. However, the chain growth depends at worst on the honest miners’ mining power (e.g. if the adversary abstains), therefore two checkpoints are issued on expectation at most $\lceil \frac{2}{h} \rceil \cdot e$ rounds after t . Finally, as mentioned in Section 4.2.2, a transaction is published in the first honestly-generated block, which is produced after the transaction’s creation. In turn, this block is checkpointed, either as part of the main chain or as an uncle block, by the upcoming checkpoint. \square

The next property that we explore is chain quality (cf. Definition 4.4). First, we observe that **Advocate** cannot guarantee chain quality over any fixed window of l consecutive blocks of the final ledger. Briefly, the adversary can produce blocks in private and release them, such that the checkpoint certificate refers to all of them at once, hence temporarily flooding the ledger with adversarial blocks. However, as Theorem 4.3 shows **Advocate** *does* guarantee chain quality over the entire ledger. This is a direct improvement of the checkpoint protocol in [72], which guarantees safety and liveness but not chain quality.

Theorem 4.3 (Chain Quality). *Let β be the adversarial power. For every execution, during which l blocks are created in aggregate by all parties, **Advocate** satisfies chain quality (cf. Definition 4.4) with parameters $l, (1 - \beta)$.*

Proof: During the entire execution, the honest parties collectively create (on expectation) at least $(1 - \beta) \cdot l$ blocks. At the end of the execution, a checkpoint is issued, referencing all main chain and uncle blocks that are not checkpointed. After the issuing of the last checkpoint, the aggregate ledger at the end of the execution contains all blocks created by all parties; hence the ratio of honest blocks in the final aggregate ledger is at least $1 - \beta$. \square

Advocate with hooks. To achieve chain quality for smaller windows of blocks, we propose a slightly modified version of **Advocate**. Each block now contains a reference to the latest checkpoint certificate C_j when it was mined. Next, such block can be referenced by a certificate C_i only if $i - j \leq t$, i.e., it can be referenced only by one of the t certificates that immediately follow C_j . This constraint, called a *hook*, prevents \mathcal{A} from releasing old blocks.

Theorem 4.4 shows that **Advocate** with hooks ensures chain quality for any window of blocks containing t consecutive checkpoints.

Theorem 4.4 (Short Term Chain Quality). *Under Advocate with hooks, the ratio of honest blocks in any window of l consecutive blocks, which includes t checkpoints, is at least $\frac{(1-\beta)\cdot(t-1)}{t+\beta+t\cdot\beta-1}$, where β is the adversarial power.*

Proof: Let Υ be the maximum number of blocks produced on expectation by *all* parties (honest and adversarial) between two consecutive checkpoints. Without loss of generality, assume a window of blocks which begins with the checkpointed block of certificate C_i and ends with the checkpointed block of certificate C_{i+t} . C_i can reference at most $t\cdot\beta\cdot\Upsilon$ adversarial blocks (i.e., which have been created after certificate C_{i-t}) and at minimum 0 honest blocks (i.e., if all honest blocks created between certificates C_{i-1} and C_i are part of the main chain). Also, certificate C_{i+t} can reference at most $(1-\beta)\cdot\Upsilon$ honest blocks and at minimum 0 adversarial blocks (i.e., if all honest blocks created between certificates C_{i+t-1} and C_{i+t} are uncle blocks and all such adversarial blocks are part of the main chain). In this case, the above window of blocks contains $2\cdot t\cdot\beta\cdot\Upsilon$ adversarial blocks and $(t-1)\cdot(1-\beta)\cdot\Upsilon$ honest blocks, hence the ratio of honest blocks is $\frac{(1-\beta)\cdot(t-1)}{t+\beta+t\cdot\beta-1}$. \square

Finally, we introduce two performance metrics accentuating the functionality of *Advocate*. First, the *chain inclusion gap* (Definition 4.5) expresses the expected number of blocks until a new block is stable. Corollary 4.2 shows that plain *Advocate* cannot ensure a chain inclusion gap, whereas *Advocate* with hooks guarantees a chain inclusion gap of $(\beta\cdot t - \beta + 1)\cdot\Upsilon$ blocks, where Υ is the maximum number of blocks that all parties produce on expectation between two consecutive checkpoints; the proof follows directly from Theorems 4.3 and 4.4. We know that the chain grows at the normalized rate $1-\beta$; thus, the maximum number of blocks mined between two checkpoints is $\frac{e}{1-\beta} = \Upsilon$. Observe that the chain inclusion gap increases linearly with Υ and, consequently, with the epoch length, i.e., the time interval between two consecutive checkpoints; as shown in Figure 4.3, this is a direct improvement on the result of [72], where it increases exponentially under adversarial mining majority.

Definition 4.5 (Chain Inclusion Gap). *Let party P with a main chain C of length l , which creates a new block B . The chain inclusion gap with parameter g states that, when B becomes stable, its position in the aggregate ledger is at most $l + g$.*

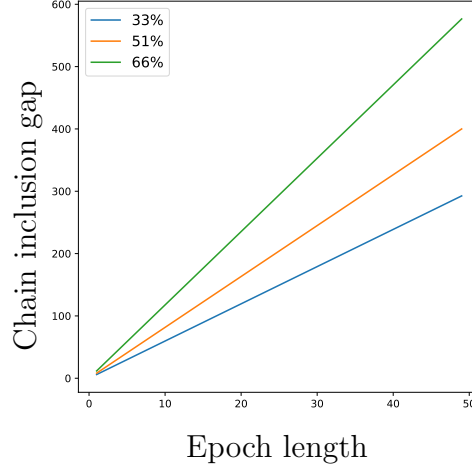


Figure 4.3: Chain inclusion gap increases linearly with epoch length; compare with Figure 4 in [72] where it increases exponentially in epoch length.

Corollary 4.1 (Advocate Chain Inclusion Gap). *Advocate guarantees chain inclusion gap (cf. Definition 4.5) with parameter $g = \infty$. Advocate with hooks guarantees chain inclusion gap with parameter $g = (\beta \cdot t - \beta + 1) \cdot \frac{e}{1-\beta}$, where β is the adversarial power, t is the hook parameter, and e is the checkpoint epoch length.*

Second, *optimistic serializability* (Definition 4.6) ensures that the checkpointing service does not trivialize the ledger maintenance. Specifically, under fully honest conditions, i.e., $\beta = 0$, transactions are ordered in the ledger in the order of their arrival, if such arrival order exists. Permissionless protocols like Nakamoto longest chain ensures optimistic serializability and the checkpointing service does not affect the block ordering; playing a supplementary role. Evidently, *Advocate* satisfies optimistic serializability by design.

Definition 4.6 (Optimistic Serializability). *For two transactions tx, tx' such that tx was given as an input to all honest nodes at a round r and is valid w.r.t. ledger $L_P(r)$ at round r and tx' was given as an input to all honest parties after round r , then it holds that for any $r' > r$, the ledger $L_P(r')$ of any honest party P cannot include tx', tx in this order, given that the network consists of all honest node.*

Corollary 4.2 (Advocate optimistic serializability). *Advocate guarantees optimistic serializability (cf. Definition 4.6).*

Consider a Nakamoto longest chain protocol; it is easy to show that it guarantees optimistic serializability. Since the honest nodes received tx before tx' , all miners will mine ledger with tx before tx' . Even when the ledger is forked, within each fork, the parent is known and, hence, the order is maintained for all parties.

In *Advocate*, the certificate may refer to a transaction tx again; hence the ledger $L_P(r)$ might have transactions tx, tx' in that order. However, since the base consensus is Nakamoto, it will ensure that tx, tx' exists in that order before checkpointing. Thus the tx' referred by the checkpoint C_r will be a second occurrence and will be removed by ledger sanitization.

4.2.5 Contrast with Fruitchains and Conflux

In terms of safety, the transaction inclusion from F_i has similarity to fruits in Fruitchains [34] and DAG references in the pivot chain of Conflux [74]. However, we note that if the adversarial mining power is greater than 50%, the adversary can always beat honest nodes by creating a mainchain in Fruitchain and a conflicting pivot chain in Conflux, thus violating safety.

In terms of liveness, Fruitchains has a chain quality of $1 - \beta$, which is ensured because all blocks created by honest miners are eventually included as fruits. This chain quality however is reduced to 0 for $\beta > 0.5$. To understand this abrupt loss of chain quality, let us consider an attack by a 51% adversary. The adversary creates a longer blockchain consisting of only adversarial blocks, and the adversarial blocks do not include references to fruits mined by honest miners. Since the longest blockchain is chosen to create the fruit ledger, it will not consist of any honest fruits, rendering the chain quality 0. Similar arguments can be made for a heavier pivot chain generated by adversaries in Conflux. This abrupt loss of chain quality for $\beta > 0.5$ is depicted in Figure 4.1.

4.3 *Advocate* with BFT checkpointing

Although *Advocate*, as described above, satisfies the desired security properties, it assumes a single checkpointing node. This centralized design is problematic, especially in systems like distributed ledgers, whose primary purpose

is decentralization. In this section, we present **Advocate** with Byzantine Fault Tolerant (BFT) checkpointing, which extends the single checkpointing node with a committee of n nodes. Although this extension might seem trivial, certain fine points (for example, running a BFT-SMR with external state validation) need to be analyzed to establish this equivalency. In contrast to other checkpointing protocols, **Advocate**-BFT allows multiple checkpointing candidates since a candidate includes both the checkpointed block and the reference links, thus increasing the input space for the BFT committee nodes. The committee achieves consensus on the contents of the checkpoint certificate, which is then published on the main chain.

The adversary \mathcal{A} controls up to f of the committee nodes, such that $n \geq 3f + 1$, and a fraction $\beta \in [0, 1)$ of the PoW mining power. Note that the committee running BFT is independent from the committee of miners. Hence, it is probable for the committee of miners (i.e., a small community for novel PoW chains) to be in an adversarial majority, in contrast to the BFT committee (consisting of well-established and legally bound validators) to will be in an honest supermajority.

4.3.1 Advocate-BFT

The committee nodes act as full nodes for the PoW main chain and run a separate SMR (BFT-based State Machine Replication) protocol; any generic BFT-SMR protocol should suffice. On receiving a valid PoW block, the committee node posts a transaction $\langle \text{Blockhash}, \text{Depth} \rangle$ on the SMR chain, which is finalized after some rounds as per the BFT-SMR's rules.

The SMR chain announces a new checkpoint when a transaction containing a block with depth e more than the previous checkpoint is posted on the SMR chain. A checkpoint transaction $tC_i = \{H(B_i), M(R_i)\}$ is posted on the SMR chain, where R_i consists of all the main chain blocks referenced on the SMR chain between the references for B_{i-1} and B_i , $M(R_i)$ denotes its Merkle root and $H(B_i)$ denotes hash of block B_i .

The checkpoint certificate $C_i = \{tC_i, R_i, w_i\}$ consists of the checkpoint transaction $tC_i = B_i, M(R_i)$, a witness w_i stating that it is finalized on the SMR chain, and the list of references R_i . C_i should be posted on the SMR chain before the depth of $d(B_i) + c$, where $d(B_i)$ is the depth of checkpoint

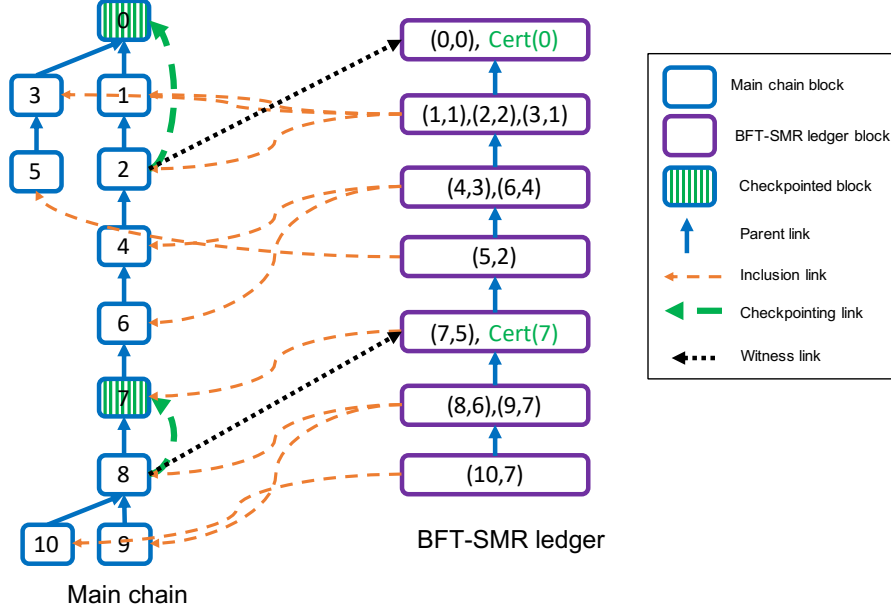


Figure 4.4: Advocate -BFT: SMR chain maintains inclusion reference links for main chain blocks

B_i .

The validity rules for block inclusion on the SMR chain are as follows:

- Data availability: The block should be available.
- Block validity: The block should have valid PoW; note that full transaction validity is not required.
- No checkpoint conflict: The block should not be at the height of $d(B_i) + e$ and extend a chain that does not contain B_i .

The BFT checkpointing service realizes in a distributed manner the checkpointing node of Section 4.2. Specifically, the BFT service collects all leaves which are not checkpointed, including the main chain and uncle blocks, and issues a certificate which references them. Assuming the BFT protocol is secure, the committee will i) issue a certificate which ii) references all non-checkpointed blocks, so the analysis of Section 4.2 also applies here.

Main chain behavior The main chain miners act as light nodes for the SMR chain. They include C_i in the main chain as soon as tC_i is finalized on the SMR chain. We assume that the main chain nodes are connected to at least one honest SMR chain node, to get the references from $M(R_i)$. The

mining behavior and validity rules, including the c constraint on checkpoint inclusion, remain the same as described in Section 4.2.

Latency To compute the latency for a transaction, let τ_m be the time until a block B containing the transaction is mined. Let τ_t be the time until a transaction containing the hash of B is posted on the SMR chain and τ_f be the time until that transaction is finalized on the SMR chain. The total time until an honest transaction is considered for checkpointing is $\tau_i = \tau_m + \tau_t + \tau_f$. Observe that the value τ_t is not affected by the adversarial mining fraction. Finally, the transaction is confirmed when the next checkpoint is posted on the SMR chain, i.e., after time τ_c until the checkpoint is finalized. Therefore, the overall latency of a transaction is $\tau = \tau_i + \tau_c$. We note that, in accordance to the discussion in Section 4.2, the parameter c depends on the BFT latency, i.e., Δ_{BFT} . We propose another variant of Advocate-BFT in Appendix C.1 that uses lower BFT resources at the cost of higher confirmation latency.

4.3.2 BFT integration

We abstract the BFT functionality F_{BFT} and show equivalence with the checkpointing service F_{cps} described in Section 4.2.

BFT-SMR service F_{BFT}

F_{BFT} is a part of a network of P replicas participating in BFT-SMR. F_{BFT} takes an input I_{BFT} and outputs O_{BFT} after a delay bounded by Δ_{BFT} . F_{BFT} may take no input and still output O_{BFT} depending on the state S_{BFT} and I_{IBFT} : an input received by some replica. F_{BFT} checks validity of I_{BFT} and/or I_{IBFT} with respect to S_{BFT} stored locally using V_{BFT} , i.e., the input validity predicate. A message from one replica implementing F_{BFT} to another takes a maximum delay of Δ .

We now establish an equivalence between F_{BFT} and F_{cps} . *inputValue* in F_{cps} is equivalent to I_{BFT} in F_{BFT} , while *commitDecision* is equivalent to O_{BFT} . However, a major difference is that the delay between the two is Δ_{BFT} in F_{BFT} and Δ_{cps} in F_{cps} , s.t. $\Delta_{BFT}/\Delta_{cps} > 1$. A major deviation is in regards to the implementation of V_{BFT} , which corresponds to the *isInputValid* and *inputValidity* messages in F_{cps} . Since the state is not in the same module and the input may be indirectly received from a different replica (created

by other node), the data needed for validating I_{BFT} may not be available to $F_{advocate}$, thus returning *inputValidity* may take unknown time. This is resolved via a unified network functionality \mathcal{N}_{uni} (Figure 4.5) to which each BFT replica connects.

Unified network functionality \mathcal{N}_{uni}

\mathcal{N}_{uni} gets messages from the nodes connected to it. The message handling is split in 3 levels. The first level is the *Network Handler*, which manages network functions like message downloading and forwarding. Once the message is downloaded, it is passed to the *Validity Handler*, which verifies the message w.r.t. a well-defined validity predicate V_{BFT} , which utilizes *isInputValid* and *inputValidity*. Once the checks pass, the validity handler forwards the message to F_{BFT} , marking the message as *received*.

Note that \mathcal{N}_{uni} ensures that, once a message is received by F_{BFT} , checking V_{BFT} is instantaneous, thus replicating a local state. Moreover, \mathcal{N}_{uni} does not change the synchronous setting delay. Specifically, let i be the first honest node to receive a message m at time t . By definition, $F_{Advocate}$ sends *inputValidity* to node i at some time $t' \leq t$. Now, the message m propagates across all nodes in a synchronous manner, hence it is downloaded by each honest node j at the latest at time $t + \Delta$. Therefore, $F_{advocate}$ is queried by node j regarding *isInputValid* of m at time $t + \Delta$. Since $F_{Advocate}$ replied *inputValidity* to node i at time t , it will also reply *inputValidity* to j at $t + \Delta$. Thus, m is marked as *received* by node j by time $t + \Delta$, ensuring that the network is Δ -synchronous under \mathcal{N}_{uni} . Note that \mathcal{N}_{uni} only requires the input to be received by node j (validity is deterministic once input is received) hence constraining the system by maximum delay in broadcast Δ .

Properties of the BFT service

As discussed above, as long as the BFT-SMR protocol is secure, i.e., satisfies safety and liveness, it securely realizes the single checkpointing node in a distributed manner. However, our integration of the federation BFT into checkpoints affects the values of the parameters c, e . Specifically, c should be at least τ_f/τ_r larger than in Section 4.2, τ_f being the time required for a transaction to be finalized by the BFT and τ_r being the size of each round of the ledger protocol. Therefore, faster BFT protocols are preferable in order to minimize the time until the checkpoint certificate is finalized.

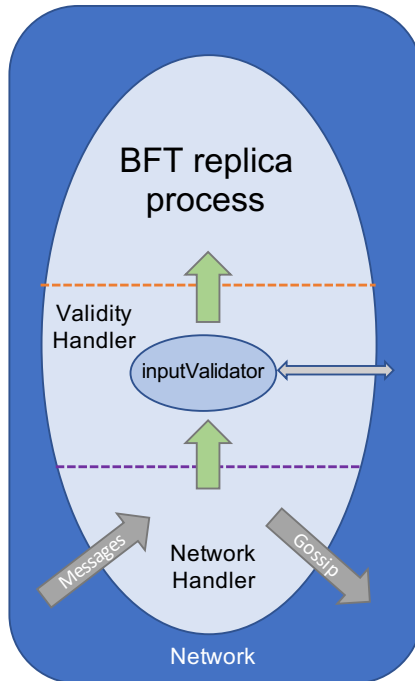


Figure 4.5: The network layer of BFT is modified to accommodate the `isInputValid` functionality; the end impact on BFT replica process is equivalent to being connected to a synchronous network without $F_{Advocate}$

So far, we assumed that enough committee members are honest, such that BFT protocol is secure; in the following section 4.4 we relax this assumption by considering cases when the BFT security may be violated. In that direction, we explore trust that is brought from established blockchains, e.g., Bitcoin or Ethereum.

4.4 Advocate with BFT checkpointing and External Trust

In the previous section, we presented a federated variant of **Advocate**, which assumes a secure execution of a BFT protocol. We relax this assumption by considering cases when the BFT threshold may be violated. In that direction, external trust is brought from established blockchains, e.g. Bitcoin or Ethereum.

To this goal, [72] proposes a timestamped ledger where main chain nodes use an established ledger to timestamp main chain blocks, hence using the

timestamps as checkpoints. Deriving checkpoints from timestamps allows one to fully decentralize the checkpointing mechanism. However, as we show next, the proposed mechanism can be susceptible to data availability attacks, even when a small fraction of the main chain nodes are adversarial.

4.4.1 Timestamped ledger

The decentralized implementation of a timestamped ledger requires main chain miners to post a blockhash on the Bitcoin chain to generate a logical timestamp which comprises of the block’s depth. The main chain follows the oldest timestamped chain rule summarized as follows: *“Start the chain from genesis, at a fork, parse both the chains until a timestamped position is found, select the chain with the oldest timestamped block; if no such timestamp is found select the longest chain”*. Thus the timestamped blocks can be treated as checkpoints, with timestamps being used for deterministic fork resolution.

Data Availability Attack Consider an adversary \mathcal{A} that mines a block B_a and posts $H(B_a)$ via a transaction on Bitcoin, though without revealing its contents. The blockhash gets timestamped, so honest miners should try to build on top of it. Specifically, forking away from B_a will lead to a DoS attack at a later point, i.e., when the adversary releases B_a ’s content and the timestamped chain rule prioritizes it. However, since B_a ’s contents are currently unavailable, honest miners cannot validate any pending transactions, which may depend on B_a ’s transactions. Therefore, the ledger is effectively stalled; the attack is depicted in Figure 4.6. Note that this attack does not require a 51% mining adversary since only one adversarial and unavailable block can stall the blockchain, which can be produced even with low mining power.

4.4.2 Advocate-external

We now present **Advocate-external**, which is resistant to the data availability attack described above. To construct it, we utilize a committee in addition to an external, secure ledger. The former guarantees liveness and safety, as long as the honest threshold is guaranteed, while the latter ensures safety even when the adversary controls a super-majority both in the PoW chain

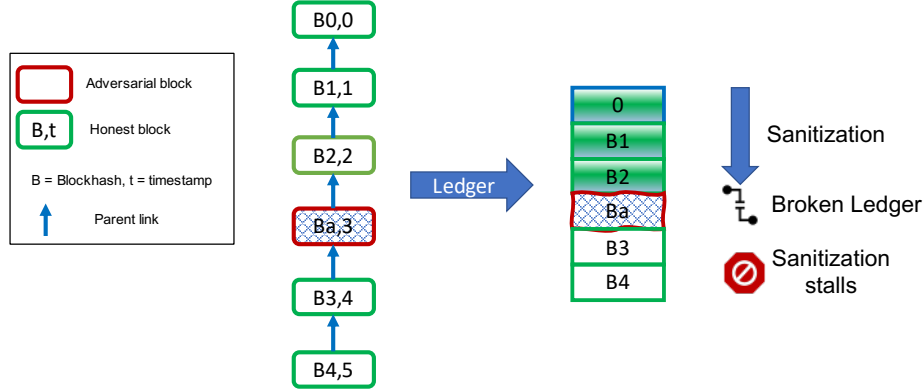


Figure 4.6: DA on timestamped ledger: Ledger sanitization stalls since transactions in B_a are not available

and the committee.

Advocate-external is similar to **Advocate-BFT** of Section 4.3, with one notable difference: a commitment of each checkpoint is first published on the external ledger before being considered.

As before, a new checkpoint is signed by enough committee nodes, i.e., above the threshold required by the BFT protocol, which produces a threshold signature tC_i . Observe that, in the case of an adversarial majority among the committee nodes, multiple signatures may be posted on the external ledger for conflicting yet valid checkpoint candidates. Once the transaction which contains the threshold signature is finalized in the external ledger, a certificate to this (timestamping) transaction is created and posted on the main chain as $C_i = \{tC_i, W_i\}$, following the c constraint (cf. Section 4.2). If multiple valid signatures exist for the same block depth, the one that was timestamped first is picked. Finally, in addition to following the described checkpoint and chain validity rules, each honest node, i.e., both committee members and main chain miners, adopts a block only if it has access to its full content. Figures 4.7 and 4.8 provide further intuition on the design of **Advocate-external**.

Security Properties of Advocate-external We now show that **Advocate-external** satisfies the desired ledger properties, namely safety and liveness. Specifically, the former is always guaranteed, while the latter is ensured only when a majority of federation nodes are honest. Additionally, since only a hash and a signature are posted on the external ledger, the state that is required is minimal. However, the PoW and committee nodes are required

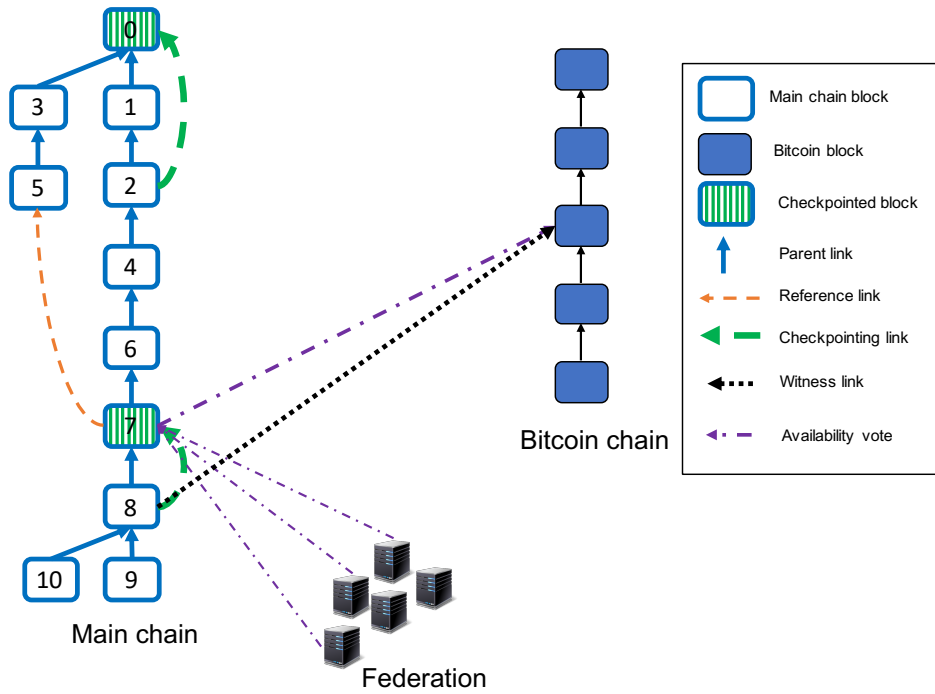


Figure 4.7: Advocate-external: Availability votes are aggregated and posted on external ledger.

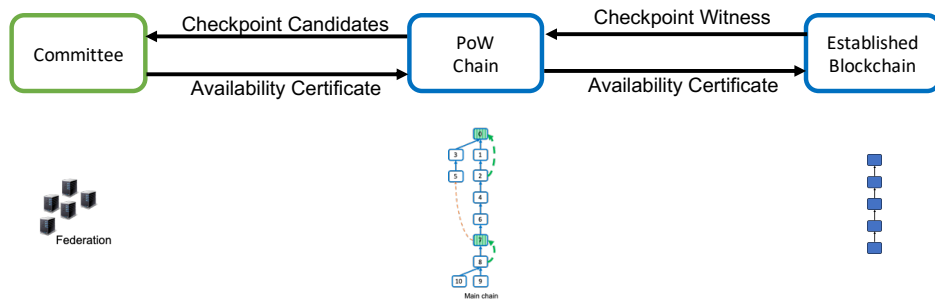


Figure 4.8: Generalized Advocate-external: Committee provides availability certificate, witness from established blockchain provides safety

to act at least as light nodes to the external blockchain so as to be able to verify the checkpoint certificate (timestamping) proofs.

Theorem 4.5. *Let B be a block which is checkpointed by Advocate-external. If B is reported as finalized by at least one honest party, then every honest party reports B as finalized after at most one round.*

Proof: Let B be a block which is checkpointed by Advocate-external and is reported as finalized by at least one honest party. Consequently, first, the block is checkpointed via a certificate signed by a majority of committee nodes and posted on the external blockchain. Second, since an honest party P reports it as finalized, then, by definition of how mining nodes behave, the B 's content is available to P . However, P can send B 's content to all other honest parties who, upon receiving it, will also report B as finalized. Finally, since the external ledger is secure, the certificate ordering will remain safe; thus, a checkpoint that is confirmed by honest PoW nodes will remain confirmed.

Theorem 4.6. *Let T be the security threshold required by the BFT protocol employed by the checkpointing committee of Advocate-external; if more than T committee members are honest, then Advocate-external provides the same liveness guarantees as Advocate-BFT (cf. Section 4.3).*

Proof: The proof follows directly from the security properties of Advocate-BFT; since Advocate-BFT satisfies liveness (without the need of an external ledger), then Advocate-external does so too if both the BFT protocol and the external ledger are secure.

So far, we have designed Advocate to Bootstrap a PoW chain running on a single longest chain consensus. Advocate can be easily extended to other PoW protocols. The following section presents a variant of Advocate for multi-chain systems.

4.5 Advocate: Bootstrapping for Parallel-PoW chains

Many emerging PoW blockchains rely on a “parallel-chain” architecture for scaling, where multiple chains run in parallel and are aggregated. Two successful parallel-chain architectures are Prism [32] and OHIE [77]. Although

these two protocols are significantly different, we demonstrate the generalizability of *Advocate* by extending *Advocate* to both these settings by proposing a *meta-protocol* *Advocate-PC* for integrating *Advocate* to parallel-chain architectures. For simplicity, we design *Advocate-PC* using a single (honest) checkpointing node, which can be readily extended to a BFT federation as described in Section 4.3.

4.5.1 *Advocate-PC*: Meta-protocol

Consider M parallel chains, with mining power *sortition* across them. A block is labelled as B_{m,j,f_b} if it belongs to branch f_b of chain m and has rank j . f_b is a function of parent of B_{m,j,f_b} . The rank $\mathcal{R}(B_{m,j,f_b}) = j$ of a block is determined by the parallel-chain protocol’s specifics and is deterministic when a block is mined. We highlight the first important meta-principle:

Rank criterion: *Blocks mined by honest miners have monotonically increasing rank within a chain.*

Block ranks are used to determine epoch intervals with a checkpointing epoch spanning blocks with a rank difference of e . We denote a chain as *payload-carrying*, denoted by $Y(i) = 1$ if its blocks are designed to carry a transaction payload. We set one chain (chain 0) as a *base-chain*, which may or may not be payload-carrying.

Checkpointing party behavior.

- Upon receiving Block B_{0,j,f_b} at rank $\mathcal{R}(\tilde{C}_{i-1})+e$, where \tilde{C}_{i-1} is the latest checkpointed base-chain block, it creates a new checkpoint certificate C_i , the block B_{0,j,f_b} is the checkpointed block \tilde{C}_i ;
- The certificate $C_i = \{R(\tilde{C}_i), R_i, \mathbf{B}_i\}$ defines: (a) a vector of $M - 1$ parallel-chain blocks \mathbf{B}_i , i.e., one tip block from each parallel chain except the base-chain, (b) the reference list R_i of all payload-carrying blocks not referenced by any checkpoint until Rank $\mathcal{R}(\tilde{C}_i)$, (c) a reference $R(\tilde{C}_i)$ to the checkpointed base-chain block \tilde{C}_i .

Validity rules.

- All chains extend the latest checkpoint;

- A base-chain block is invalid if it extends the chain past rank $\mathcal{R}(\tilde{C}_i) + c$ and does not contain C_i ;
- A non-base-chain is invalid if none of its blocks refer to the base-chain block containing the certificate C_i by rank $\mathcal{R}(\tilde{C}_i) + c$;
- A non base-chain tip block B_{m,j,f_b} is valid for inclusion in \mathbf{B}_i only if chain m has referred to C_{i-1} .

The ledger creation rules are similar to **Advocate**; the checkpoint certificate brings in all the referred blocks R_i in the respective payload-carrying chain's ledger. We now use the above meta-protocol to integrate **Advocate** to **OHIE** consensus.

4.5.2 Advocate-OHIE

OHIE composes M parallel instances of Bitcoin longest chains. Miners mine on all M chains in parallel by utilizing mining sortition. Under an honest majority assumption, each honest chain has the same security properties as the longest chain protocol. Total ordering between the chains is necessary in order to create one ledger. **OHIE** achieves this by requiring blocks to have a tuple (**rank**, **next-rank**). We note that **next-rank** is always higher than **rank** and **rank** of a block is at least as high as the **next-rank** of its parent. Therefore, we can conclude that the **rank** monotonically increases along a chain. We can see that this rank satisfies the *Rank criterion* for \mathcal{R} defined above.

OHIE achieves total ordering by generating a sequence of confirmed blocks amongst the parallel chains. Confirmation rank is decided based on the k -deep rule of the individual chains, and blocks are ordered according to their **ranks** for **ranks** below confirmation rank, breaking ties by chain-id. The protocol security breaks down if we have an adversarial mining majority; a confirmation rank can no longer be defined. Observe that **rank** in **OHIE** satisfies the rank criterion and thus can be treated as \mathcal{R} .

We set chain 0 as the base-chain arbitrarily and implement **Advocate-PC**. C_i is generated with a **rank** difference of e on chain 0. We can directly use the checkpointing party behavior and validity conditions from **Advocate-PC**. C_i sets the checkpointed confirmation rank. The references will have the

same rank as the block B_{m,j,f_b} in chain m referring/including the checkpoint certificate. The protocol is safe since the chain order beyond the confirmation rank cannot be changed; live with optimal chain quality since all the uncle blocks in all M chains will be eventually included in their respective ledger.

Although OHIE achieves high throughput, it also presents high latency. Hence, next we consider Prism, a parallel-chain architecture-based consensus protocol that achieves optimal throughput and latency.

4.5.3 Advocate-Prism

Prism consists of 3 categories of blocks: *Proposer*, *Voter*, and *Transaction* blocks. Transaction blocks assemble transactions and do not have an inherent order. Proposer blocks refer to transaction blocks and indirectly assemble transactions; they do not form a chain natively but rather are arranged in levels L ; each level can contain multiple proposer blocks, of which one will be chosen as a leader. Voter blocks form a bitcoin-like longest chain. There are $M - 1$ such voter chains. Votes from the voter chain decide the leader proposer block at each level, thus forming the proposer chain. A fundamental design decision is which type(s) of Prism blocks to checkpoint, i.e., transaction, proposal, or voter blocks. Since transaction blocks have no inherent order, checkpointing these blocks does not directly result in a well-defined ordering of transactions. Therefore, our protocol checkpoints voter blocks and proposal blocks instead and considers transaction blocks as a part of proposer blocks. Thus, we are left with M parallel chains with 1 payload-carrying chain: the proposer chain.

Proposer levels follow the *Rank criterion* since honest miners always mine blocks with increasing proposer levels. Moreover, voting blocks can derive their level from the level of the latest proposer block they vote on (or the parent voter block level if they vote for none). The proposer level (or proposer level voted) can be used as \mathcal{R} . Thus, we can integrate Advocate-PC by setting the Proposer chain as the base-chain. Since the Proposer chain is the only payload-carrying chain, R_i refers to only proposer blocks.

The checkpoint certificate C_i consists of a checkpointed proposer block P_i , a list of referred proposer blocks R_i , and a $M - 1$ dimensional vector of voter blocks. The ledger is created from the list of references in checkpoint

certificates—probabilistic confirmation (for $\beta < 0.5$) is done for blocks not checkpointed yet. Using the Prism confirmation rule guarantees low latency. The protocol is safe since the checkpoint cannot be altered and live since honest proposer blocks will be included in the ledger via references in the checkpoint certificate.

4.5.4 Checkpointing Party Behavior

As before, the checkpointing party connects to the Prism main chain network and maintains the tree of blocks. On regular intervals, i.e., every e proposal blocks, the node creates and signs a checkpoint certificate, which includes references of all non-checkpointed proposal blocks, as well as the latest valid voter blocks for each voter chain. In summary, the checkpointing party behaves as follows:

- It runs a full node on the Prism network.
- Upon receiving a proposer block P_i at level $L(P_{i-1}) + e$, where P_{i-1} is the latest checkpoint, it creates a new checkpoint certificate C_i ; the block P_i is called the *checkpointed* block.
- The certificate $C_i = \{R(P_i), R_i, \mathbf{V}_i\}$ defines: i) a vector of voter blocks \mathbf{V}_i , i.e., one voter block from each voter chain (following the longest chain rule for each), ii) the reference list R_i of all proposal blocks not referenced by any checkpoint until level $L(P_i)$, i.e., the level of block P_i , iii) a reference $R(P_i)$ to the checkpointed proposal block P_i .
- It maintains the list of blocks confirmed by the voter chains till \mathbf{V}_i .

4.5.5 Prism Main Chain Behavior

Intuitively, the checkpoint certificate for a proposer block P_i should be included in at least one proposer blocks in the range of levels $(L(P_i), L(P_i) + c]$, which should then be voted by all voter chains. Specifically, regarding block validity:

- A proposer block, which is created by an honest miner at some level in range $(L(P_i), L(P_i) + c]$, either refers to a proposer block that contains

the checkpoint certificate for block P_i or, if no such block exists, it contains the certificate itself.

- A proposer block at level $L(P_i) + c$ is valid if and only if it refers to a proposer block that contains the checkpoint certificate for block P_i or contains the certificate itself.

Voter chain validity rules are modified as follows:

- Honest miners vote for a valid proposal block at level $L(P_i) + c$, if such block exists.
- A voter chain is valid if it extends the latest checkpointed voter block for that voter chain.
- A voter chain can be checkpointed at level $L(P_i)$ if and only if it is valid and it has voted for a valid proposal block at level $L(P_{i-1}) + c$.

4.5.6 Ledger Creation

Given the checkpointing process described above, the final ledger is created as follows (Figure 4.9):

- For the proposal blocks at levels up until the latest checkpointed block, the ledger is derived from the checkpoint reference list R_i .
- For proposer blocks at levels beyond the latest checkpointed block, the ledger is derived from the confirmation rules of Prism.
- The checkpointing node follows the confirmation rule of Prism for blocks confirmed by \mathbf{V}_i ; thus, R_i will not conflict with the blocks confirmed by \mathbf{V}_i .

4.5.7 Security Properties of Advocate-Prism

The guarantees for **Advocate-Prism** are similar to **Advocate**. The additional guarantee that **Prism** offers is that the normal mode latency (when the adversarial ratio is zero or negligible) is near-optimal (a constant factor of network latency Δ). The main claim here is that the guarantee continues to hold for **Advocate-Prism** for appropriate parameter choices.

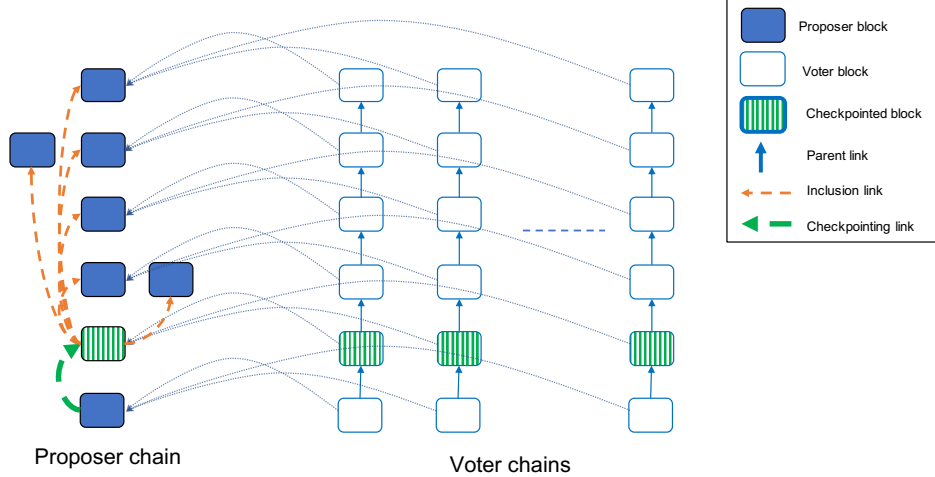


Figure 4.9: **Advocate-Prism**: Checkpoints proposer and voter blocks, reference to all proposer blocks not referenced before

Theorem 4.7 (Safety). *Every transaction which is referenced by a proposal block B , which is checkpointed by **Advocate-Prism** via certificate C , is stable.*

Proof: It suffices to show that, as soon as a certificate is issued, the block B cannot be reordered. Hence any transaction referenced by B is finalized in the sanitized ledger. If B is referenced by certificate C , then its ledger position can change only if i) B is referenced by a future checkpoint certificate C' or ii) C is replaced by a different certificate. However, since the checkpointing node is honest, it does not create conflicting certificates. At the same time, by definition of the **Prism** main chain behavior, C is eventually included in a proposal block. Therefore, the position of a proposal block B (and its referenced transactions) is finalized as soon as a checkpoint certificate references to it.

Theorem 4.8 (Liveness). *Let h be the probability that at least one honest proposal block is created per round; **Advocate-Prism** satisfies liveness (cf. Definition 4.3) with parameter $u = \lceil \frac{1}{h} \rceil \cdot e$.*

Proof: The proof follows similarly to Theorem 4.2. Specifically, a transaction becomes stable after an honest block B is created and gets checkpointed, which happens on expectation after $\lceil \frac{1}{h} \rceil \cdot e$ rounds. Additionally, the checkpoint certificate should also be live, i.e., it should be included in the proposal chain and get voted on. By definition of the main chain node behavior, a valid voter chain contains a reference to a certificate C_i by proposal level

$L(P_i) + c$. Additionally, if at least one honest voter block is created, then it will reference C_i ; however, since the honest mining power percentage is $1 - \beta > 0$, the honest miners will eventually mine a voting chain that contains C_i and, in the absence of any other voting chain containing C_i (even a longer one mined by the adversary), this chain is considered valid.

- **Normal Mode latency:** Checkpoint certificate C_i maintains the proposal block order till the voting blocks \mathbf{V}_i , thus if the proposal block is confirmed by prism voter chains, the checkpointing node will confirm it. Under normal mode, the proposal blocks are confirmed with optimal latency using prism voter rules and the checkpointing rules will not violate that confirmation.
- **Optimistic serializability:** Prism ensures optimistic serializability since any honest miner mining tx' will have tx or will refer to a proposer block (reference ordered first) containing tx indirectly. **Advocate-Prism** can refer to tx after tx' . However, optimistic serializability of Prism ensures that this won't be the first occurrence of tx and will be removed from the ledger by sanitization.

4.6 Implementation

We implement **Advocate** on a codebase in **Rust** and compare its performance with various existing checkpointing techniques. To test the performance of **Advocate** to the limit, we integrate **Advocate-Prism** along with the high performance implementation of **Prism** written in **Rust** [78]. We describe the system design and architecture of **Advocate** and **Advocate-Prism** and then evaluate the performance of **Advocate** by comparing it with other checkpointing protocols. The code is available at [75].

4.6.1 System Design

Database Advocate maintains the following databases: (1) Checkpoint database, (2) Block-structure database, (3) Ledger-database, (4) Transaction memory pool (Prism), and (5) UTXO database (Prism) in rocksDB key-value store.

The transactions in **Advocate-Prism** have a similar structure as the pay-to-public-key-hash(p2pkh) transactions in Bitcoin.

Block manager. Blocks have an additional option field containing a checkpoint certificate. When a new block arrives at the block manager, its validity is checked according to the validity conditions in Sections 4.2 and 4.5. For **Advocate-Prism**, the manager maintains a separate structure for proposer, voter, and transaction blocks.

Network Manager. The network manager communicates blocks, transactions(Prism), and checkpoint certificates with its peers. On receiving a block, it performs PoW and parent checks. It also performs data-availability checks for block parents and checkpoint references. If the reference data is not available, it stores the block in an orphan buffer and processes it when the dependencies arrive. The orphan management is derived from the Unified network functionality mentioned \mathcal{N}_{uni} in Section 4.3.

Integrated checkpointing module. The integrated checkpointing module runs F_{cps} along with $F_{advocate}$. The checkpointing party is implemented as a single full node described in section 4.2. For all nodes barring that one, F_{cps} is inactivated. The checkpointing module is alerted with a *potential-Candidate* when a block at the requisite height is received; it generates a certificate by signing the candidate and references. Effects of a BFT-SMR based checkpointing committee on our performance metrics are emulated by delaying the broadcast of checkpoint certificate by Δ_{BFT} .

Ledger manager. Ledger manager in **Advocate** implementation integrates the checkpoint reference in the block ledger. Since we do not aim to check performance limits in **Advocate** implementation, we do not implement transaction execution. **Advocate-Prism** implementation aims to check performance limits and hence implements full-stack transaction execution. The ledger manager runs asynchronously with respect to the block manager. It runs on a separate thread and periodically polls the block manager for newly confirmed transactions. When a checkpoint certificate is received, the block manager sends an update to the ledger manager to include the checkpoint references at a determined location. The ledger is assembled, and transac-

tions are sanitized.

Miner receives ledger tip updates from the block manager (including checkpoint updates). Miner in **Advocate** mines a single block at a time, whereas a miner in **Advocate-Prism** mines three types of blocks Proposer, Voter, and Transaction using mining sortition described in [32]. If the miner has not received a new certificate when its mining tip reaches level $e + c - 1$ past the previous checkpoint, it pauses mining and waits for the checkpoint certificate.

4.6.2 Comparison baselines

We implement two other checkpointing protocols as baselines to compare performance metrics of **Advocate**. Below, we briefly describe these baselines and their integration with **Prism**.

Stochastic-checkpointing. Derived from [72], the checkpoint certificates referring to a single Block-hash(checkpoint) are introduced in the ledger. The certificates add randomness at every epoch, ensuring the adversary cannot implement a front-running attack described in [72]. Stochastic-checkpointing is implemented by modifying the **Advocate** F_{cps} to generate checkpoint without references.

Nakamoto-checkpointing. Derived from the off-chain checkpoints published by Nakamoto in the early days of bitcoin (checkpointing via GitHub). The checkpoint certificates are posted off-chain and consist of a block's hash. The full node codebase recognizes these checkpoints and only considers chains extending these checkpoints as valid. Nakamoto-checkpointing is implemented by modifying code for Stochastic-checkpointing not to include the certificate on-chain. The Block manager and miner are notified of the latest checkpoint when the Network manager receives a checkpoint-certificate message.

4.7 Evaluation

Our evaluation answers the following questions:

1. How do performance and security metrics of **Advocate** compare to state-of-the-art checkpointing and finality gadgets?

2. How does the performance of **Advocate** react to a slow checkpointing service?
3. How does **Advocate** perform with large epoch sizes?
4. How does **Advocate** integrate with very high throughput PoW blockchains, e.g., **Prism**? How is the performance overhead?

Testbed: We run our **Advocate** experiments on c5d.large AWS instances. These instances have 2 cores, 4 GB of RAM and 50 GB of NVMe SSD storage, and a network bandwidth of 10 Gbps. We run our high-performance **Advocate-Prism** experiments on more powerful c5d.4xlarge instances with 16 cores, 32GB of RAM, 400GB NVMe SSD, and 10 Gbps total bandwidth. We connect these N instances using a random-regular overlay network with degree 4. For **Advocate** experiments, we set the block arrival rate as 1 block/second (very low size blocks and low network latency) due to time restrictions.

Adversary model: The main advantage of **Advocate** is to ensure security under an adversarial mining majority. Hence we run all our experiments with $\beta \geq 0.5$. To model perfect coordination between adversaries, we implement the adversary as a single node with appropriately high mining power. We implement private mining adversaries for all three protocols under contention. The adversary mines a private chain and broadcasts private blocks in bursts of epoch length when a specific trigger is initiated. The private chain resets if an honest block is checkpointed. The differences between triggers for different protocols are described below:

- *Trigger for Nakamoto-checkpointing:* The honest public chain is about to mine a checkpoint candidate.
- *Trigger for Stochastic-checkpointing and Advocate:* adversary’s private chain has a block at the checkpoint level.

Performance metrics: We use three metrics defined below to measure performance:

- *Fractional Goodput (FG):* Let Goodput (\mathcal{G}) be the number of honest transactions confirmed per unit time and optimal throughput (\mathcal{T}) be

Parameters	$e = 5, \Delta_{BFT}=0$			$e = 5, \Delta_{BFT}=2$			$e = 10, \Delta_{BFT}=0$		
Metrics	<i>FG</i>	<i>IL</i>	<i>HW</i>	<i>FG</i>	<i>IL</i>	<i>HW</i>	<i>FG</i>	<i>IL</i>	<i>HW</i>
Nakamoto-cp	0.148	67.14	0.7032	-	-	-	-	-	-
Stochastic-cp	0.323	6.688	0.3539	0.204	13.76	0.594	0.227	21.53	0.546
Advocate	0.588	3.611	0	0.514	2.546	0	0.475	6.712	0.048

(a) $\beta = 0.5$

Parameters	$e = 5, \Delta_{BFT}=0$			$e = 5, \Delta_{BFT}=2$			$e = 10, \Delta_{BFT}=0$		
Metrics	<i>FG</i>	<i>IL</i>	<i>HW</i>	<i>FG</i>	<i>IL</i>	<i>HW</i>	<i>FG</i>	<i>IL</i>	<i>HW</i>
Nakamoto-cp	0	∞	1	-	-	-	-	-	-
Stochastic-cp	0.101	43.11	0.696	0.033	24.43	0.9	0	∞	1
Advocate	0.389	3.491	0	0.330	3.217	0	0.311	6.512	0.056

(b) $\beta = 0.67$

Parameters	$e = 5, \Delta_{BFT}=0$			$e = 5, \Delta_{BFT}=2$			$e = 10, \Delta_{BFT}=0$		
Metrics	<i>FG</i>	<i>IL</i>	<i>HW</i>	<i>FG</i>	<i>IL</i>	<i>HW</i>	<i>FG</i>	<i>IL</i>	<i>HW</i>
Nakamoto-cp	0	∞	1	-	-	-	-	-	-
Stochastic-cp	0	∞	1	0	∞	1	0	∞	1
Advocate	0.102	2.849	0	0.072	4.319	0.278	0.087	6.467	0.13

(c) $\beta = 0.9$

Table 4.1: Advocate evaluation for varying adversarial power

the maximum throughput in the absence of an adversary. We define fractional goodput as \mathcal{G}/\mathcal{T} .

- *Ledger Inclusion latency (IL)*: for an honest party P is the time taken (measured in terms of mean block arrival time Δ_A) between transaction generation and inclusion in the ledger of P .
- *Honest block wastage (HW)*: Fraction of honest blocks that are not part of the ledger.

The performance metrics are tabulated in Tables 4.1a, 4.1b, 4.1c for a variety of experimental settings: varying adversary mining power, BFT network latency, epoch size. Each experiment was conducted over a range of 50-100 epochs. We observe that **Advocate** is far superior to its competitors in all settings and metrics. We make the following more nuanced observations from the data.

Favorable parameters ($e = 5, \Delta_{BFT} = 0$). Low epoch size and zero certificate generation delay are favorable parameters for any checkpointing

protocol. We observe that Nakamoto-cp performs well when $\beta = 0.5$; however, the performance deteriorates rapidly with β . Stochastic-cp performs better than Nakamoto-cp; however, the gap between Stochastic-cp and **Advocate** increases rapidly as β increases. Next, we analyze the performance penalty on introducing unfavorable parameters such as large epoch size and large checkpoint certification time.

Slow BFT-SMR ($e = 5, \Delta_{BFT} = 0$). The delay for producing a certificate is set to be around 40% of the best inter-epoch time. As expected, Stochastic-cp and **Advocate** perform worse than the previous setting; however, the performance hit in terms of *FG* and *HW* is higher for Stochastic-cp as compared to **Advocate** since all blocks in **Advocate** will eventually be inserted on the chain. In **Advocate** the difference in performance between the two settings can be attributed to the blocks at the end of the experiment.

Large epoch size ($e = 10, \Delta_{BFT} = 0$): Fig. 4.1 explains the effect of epoch size on Stochastic-cp. Intuitively, larger epoch size leads to the honest minority losing the race to the next checkpoint more often. Observing the performance of Stochastic-cp on increasing e from 5 to 10 with $\beta = 0.5$, its *IL* increases from 6.688 to 21.53; whereas *FG* of **Advocate** only increases from 3.611 to 6.712; an increase proportional to epoch lengths' as discussed in Section 4.2.

Unexpected behavior We observe that *IL* for **Advocate** decreases on increasing β . Counter-intuitively, it happens because our adversarial strategy (for Stochastic-cp and **Advocate**) releases the private chain as soon as it reaches the depth of an epoch; hence, a more powerful adversary will mine the private chain faster hence lowering the *IL* in the experiment. This is because, in **Advocate**, any checkpointing event will bring in all references to honest blocks. A smarter adversary can wait till the honest chain is about to complete the epoch and increase latency by a factor $1/(1 - \beta)$ as shown in Theorem 4.2.

Prism The Prism full-stack implementation can achieve a throughput of 70K tx/s; coupled with **Advocate** an optimized implementation should achieve a throughput of $(1 - \beta) \cdot 70K$ tx/s. However, checkpointing adds to the validity rules, leading to loss of throughput. Our implementation of **Advocate-Prism** aims to develop a proof-of-concept, s.t. checkpointing can be integrated into

high throughput parallel chains without much expense of throughput. To this end, we design an adversary \mathcal{A} for Prism who censors honest transaction blocks. For such \mathcal{A} with $\beta = 0.7$, our full-stack implementation of Advocate-Prism (with UTXO at the application layer and p2p networking at the networking layer) can achieve a goodput of 8200 tx/s with a confirmation latency of 120s. While 8200 tx/s with a 70% adversary is much higher than any existing protocol can achieve, we believe it can be further improved by optimizing the interaction between the ledger manager and the integrated checkpointing module. This direction can be explored in future research.

4.8 Conclusion

In this chapter, we proposed **Advocate** - a bootstrapping gadget that achieves three main results: (a) optimal liveness under a super-majority adversary for the Nakamoto longest chain protocol with a BFT checkpointing committee, (b) extension of Advocate to use existing external consensus such as Bitcoin for bootstrapping, and (c) immediate black-box generalization to a variety of parallel-chain based scaling architectures, including OHIE [77] and Prism [32]. We demonstrate the robustness of **Advocate** via a full-stack implementation tested under a 90% adversarial majority. Optimal liveness ensures social/legal cost of bootstrapping is minimized.

CHAPTER 5

CONCLUSION

In this dissertation, we propose **Trifecta** uniconsensus sharding architecture with **Free2Shard** DSA to solve the Blockchain protocol trilemma. We use **Trifecta** with **Free2Shard** along with **Advocate** with a design objective to minimize the three properties of the Blockchain economics trilemma.

In chapter 2 we proposed **Trifecta** sharding architecture. This architecture is motivated by the drawbacks of Node-to-shard allocation in the more prevalent multiconsensus sharding architectures. **Trifecta** is a concrete manifestation of the uniconsensus architecture on Longest-chain and Prism consensus. It is decentralized since it supports identity-free consensus nodes and dynamic availability; it is scalable since its throughput scales almost linearly with the number of nodes, and it is partially secure - it is safe and existentially live.

In chapter 3, we proposed **Free2Shard** DSA to resolve the liveness vulnerability of **Trifecta**. **Free2Shard** achieves performance arbitrarily close to the information-theoretic bounds and retains the decentralization, scalability, and safety properties of **Trifecta**, hence solving the blockchain protocol trilemma.

In chapter 4, we proposed **Advocate** as a bootstrapping gadget for PoW protocols. It achieves optimal chain quality even under an adversarial mining majority. We show that **Advocate** can be easily extended to parallel-chain-based scaling architectures, thereby showing compatibility with Prism and **Trifecta** .

We used the Blockchain economics trilemma’s cost dimensions as a motivation to select the layer 1 problems to work on, thereby aiming to design systems that optimize each of these properties. To that end, we briefly discuss each of these costs and how our protocols optimize it:

Optimizing resource cost with Trifecta. The resource-efficiency axis of the economics trilemma considers the cost an adversary under a PoW system

has to bear to double-spend a transaction whose value has been transferred. Enabling the attack requires the attacker to spend mining costs that might have been otherwise used to gain additional transaction and mining fees. A model incorporating characteristics such as the number of ancillary transactions on the record-keeping system will capture the optimization in resource cost per transaction by optimizing throughput. **Trifecta** with **Free2Shard** optimizes throughput, minimizing the resource cost per transaction.

Free market for rent with Trifecta. The rent analysis lends itself to the result that the rent, i.e., transaction fees, has to be high enough so that the record-keeper is incentivized not to cheat lest it may lose future rent revenue. This model thus sets a lower bound on the rent. This rent model can be expanded to incorporate competition between different record-keeping systems (shards) and the ease of entry for rent-seekers in a record-keeping system as demonstrated by [79]. **Trifecta** enables multiple record keepers competing for accounts and a low friction way of transferring accounts from one record-keeper to another. It allows rent-seekers (miners) the freedom to join a shard of their choice. This competition creates a marketplace for rent. Such a marketplace will help lower the rent cost per transaction.

Optimizing social/legal costs via a bootstrapping committee with Advocate. The self-sufficiency axis considers the cost of losing a social connection when an agent lies to another about the genesis stake distribution on a PoS chain. This cost can be viewed from a legal lens where committee members bound under the legal system behave in a byzantine manner and incur legal costs. While relying on the legal cost system is the opposite of self-sufficiency, it may only be required at the bootstrapping phase of a blockchain. **Advocate** reduces legal costs by reducing the time needed for bootstrapping. Distributing rewards fairly to honest miners is achieved through optimal chain quality, ensuring a high retention rate and inflow rate of honest miners into the PoW protocol.

This dissertation uses the Blockchain economics trilemma as a guiding compass to design blockchain protocols and, as shown informally above, optimizes each of its cost axes. A single unified economic model that unifies these costs and formally shows that the **Trifecta** bootstrapped with **Advocate** achieves minimal net cost is left for future work.

REFERENCES

- [1] S. Nakamoto et al., “Bitcoin: A peer-to-peer electronic cash system,” 2008.
- [2] T. McConaghy, “The DCS triangle,” <https://blog.bigchaindb.com/the-dcs-triangle-5ce0e9e0f1dc>.
- [3] G. Slepak and A. Petrova, “The DCS theorem,” <https://arxiv.org/abs/1801.04335>.
- [4] “On sharding blockchain,” <https://github.com/ethereum/wiki/wiki/Sharding-FAQ>.
- [5] J. Abadi and M. Brunnermeier, “Blockchain economics,” 2019.
- [6] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, “A secure sharding protocol for open blockchains,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 17–30.
- [7] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, “Omniledger: A secure, scale-out, decentralized ledger via sharding,” in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 583–598.
- [8] M. Zamani, M. Movahedi, and M. Raykova, “Rapidchain: Scaling blockchain via full sharding,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018.
- [9] Zilliqa, “The ZILLIQA technical whitepaper,” <https://docs.zilliqa.com/whitepaper.pdf>, 2017.
- [10] G. Avarikioti, E. Kokoris-Kogias, and R. Wattenhofer, “Divide and scale: Formalization of distributed ledger sharding protocols,” *arXiv preprint arXiv:1910.10434*, 2019.
- [11] H. von Stackelberg, *Market Structure and Equilibrium*. Springer, 2011.
- [12] D. Blackwell et al., “An analog of the minimax theorem for vector payoffs.” *Pacific Journal of Mathematics*, vol. 6, no. 1, pp. 1–8, 1956.

- [13] R. Rana, S. Kannan, D. Tse, and P. Viswanath, “Free2share: Adversary-resistant distributed resource allocation for blockchains,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 1, pp. 1–38, 2022. [Online]. Available: <https://doi.org/10.1145/3547353.3522651>
- [14] R. Rana, D. Karakostas, S. Kannan, A. Kiayias, and P. Viswanath, “Optimal bootstrapping of pow blockchains,” in *Proceedings of the Twenty-Third International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2022. [Online]. Available: <https://doi.org/10.1145/3492866.3549731> pp. 231–240.
- [15] L. Luu and N. Rush, “Peacerelay merkle-patricia trie proof verification,” 2017.
- [16] E. Research, “Ethereum 2.0,” <https://github.com/ethereum/eth2.0-specs>, 2020.
- [17] G. Wood, “Polkadot: Vision for a heterogeneous multi-chain framework,” 2016.
- [18] M. Al-Bassam, A. Sonnino, and V. Buterin, “Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities,” *arXiv preprint arXiv:1809.09044*, 2018.
- [19] B. Parno, J. Howell, C. Gentry, and M. Raykova, “Pinocchio: Nearly practical verifiable computation,” in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 238–252.
- [20] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, “Succinct non-interactive zero knowledge for a von neumann architecture,” in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 781–796.
- [21] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, “Scalable, transparent, and post-quantum secure computational integrity.” *IACR Cryptology ePrint Archive*, vol. 2018, p. 46, 2018.
- [22] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, “Bulletproofs: Short proofs for confidential transactions and more,” in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 315–334.
- [23] M. Al-Bassam, “Lazyledger: A distributed data availability ledger with client-side smart contracts,” 2019.

- [24] A. Skidanov and I. Polosukhin, “Nightshade: Near protocol sharding design,” 2019.
- [25] M. Balakrishnan, D. Malkhi, V. Prabhakaran, T. Wobbler, M. Wei, and J. D. Davis, “Corfu: A shared log design for flash clusters,” in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 1–14.
- [26] M. Balakrishnan, D. Malkhi, T. Wobber, M. Wu, V. Prabhakaran, M. Wei, J. D. Davis, S. Rao, T. Zou, and A. Zuck, “Tango: Distributed data structures over a shared log,” in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013, pp. 325–340.
- [27] A. E. Gencer, R. van Renesse, and E. G. Sirer, “Short paper: Service-oriented sharding for blockchains,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 393–401.
- [28] J. Garay, A. Kiayias, and N. Leonardos, “The bitcoin backbone protocol: Analysis and applications,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 281–310.
- [29] M. Yu, S. Sahraei, S. Li, S. Avestimehr, S. Kannan, and P. Viswanath, “Coded merkle tree: Solving data availability attacks in blockchains,” Cryptology ePrint Archive, Report 2019/1139, 2019, <https://eprint.iacr.org/2019/1139>.
- [30] J. Teutsch and C. Reitwießner, “A scalable verification solution for blockchains,” *arXiv preprint arXiv:1908.04756*, 2019.
- [31] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, “Arbitrum: Scalable, private smart contracts,” in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1353–1370.
- [32] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, “Prism: deconstructing the blockchain to approach physical limits,” *ACM Computer and Communications Security Conference*, 2019.
- [33] L. Yang, V. Bagaria, G. Wang, M. Alizadeh, D. Tse, G. Fanti, and P. Viswanath, “Prism: Scaling bitcoin by 10,000 x,” *arXiv preprint arXiv:1909.11261*, 2019.
- [34] R. Pass and E. Shi, “Fruitchains: A fair blockchain,” in *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 2017.
- [35] T. E. Jedsor, “Mimblewimble,” 2016.

- [36] “Rocksdb a persistent key-value store,” <https://rocksdb.org>.
- [37] A. Sonnino, S. Bano, M. Al-Bassam, and G. Danezis, “Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers,” *arXiv preprint arXiv:1901.11218*, 2019.
- [38] A. Kiayias, A. Russell, B. David, and R. Oliynykov, “Ouroboros: A provably secure proof-of-stake blockchain protocol,” in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.
- [39] S. Arora, E. Hazan, and S. Kale, “The multiplicative weights update method: a meta-algorithm and applications,” *Theory of Computing*, vol. 8, no. 1, pp. 121–164, 2012.
- [40] S. Mannor, J. N. Tsitsiklis, and J. Y. Yu, “Online learning with sample path constraints,” *Journal of Machine Learning Research*, vol. 10, no. 3, 2009.
- [41] H. Yu and M. J. Neely, “A low complexity algorithm with $o(\sqrt{T})$ regret and $o(1)$ constraint violations for online convex optimization with long term constraints,” *arXiv preprint arXiv:1604.02218*, 2016.
- [42] H. Yu, M. J. Neely, and X. Wei, “Online convex optimization with stochastic constraints,” in *31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.*, 2017.
- [43] N. Liakopoulos, A. Destounis, G. Paschos, T. Spyropoulos, and P. Mertikopoulos, “Cautious regret minimization: Online optimization with long-term budget constraints,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 3944–3952.
- [44] J. Von Neumann, “A certain zero-sum two-person game equivalent to the optimal assignment problem,” *Contributions to the Theory of Games*, vol. 2, no. 0, pp. 5–12, 1953.
- [45] D. Kalathil, V. S. Borkar, and R. Jain, “Approachability in stackelberg stochastic games with vector costs,” *Dynamic games and applications*, vol. 7, no. 3, pp. 422–442, 2017.
- [46] J. Abernethy, P. L. Bartlett, and E. Hazan, “Blackwell approachability and no-regret learning are equivalent,” in *Proceedings of the 24th Annual Conference on Learning Theory*. JMLR Workshop and Conference Proceedings, 2011, pp. 27–46.
- [47] N. Shimkin, “An online convex optimization approach to blackwell’s approachability,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 4434–4456, 2016.

- [48] *Free2Shard DSA simulation*. [Online]. Available: https://github.com/ranvirranaiitb/free2Shard_DSA_sim
- [49] J. A. Garay, A. Kiayias, and N. Leonardos, “The bitcoin backbone protocol: Analysis and applications,” in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, ser. Lecture Notes in Computer Science, E. Oswald and M. Fischlin, Eds., vol. 9057. Springer, 2015. [Online]. Available: https://doi.org/10.1007/978-3-662-46803-6_10 pp. 281–310.
- [50] M. Castro, B. Liskov et al., “Practical byzantine fault tolerance,” in *OSDI*, vol. 99, no. 1999, 1999, pp. 173–186.
- [51] S. Nakamoto, “Bitcoin 0.3.2 released,” 2010, <https://bitcointalk.org/index.php?topic=437.msg3807>.
- [52] S. King and S. Nadal, “Ppcoin: Peer-to-peer crypto-currency with proof-of-stake,” 2012.
- [53] D. Gilson, “Feathercoin secures its block chain with advanced checkpointing,” 2013, <https://www.coindesk.com/feathercoin-secures-block-chain-advanced-check-pointing>.
- [54] S. D. Lerner, “Rsk white paper overview,” 2015, https://docs.rsk.co/RSK_White_Paper-Overview.pdf.
- [55] A. Kiayias, A. Russell, B. David, and R. Oliynykov, “Ouroboros: A provably secure proof-of-stake blockchain protocol,” in *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, ser. Lecture Notes in Computer Science, J. Katz and H. Shacham, Eds., vol. 10401. Springer, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-63688-7_12 pp. 357–388.
- [56] P. Daian, R. Pass, and E. Shi, “Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake,” in *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, ser. Lecture Notes in Computer Science, I. Goldberg and T. Moore, Eds., vol. 11598. Springer, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-32101-7_2 pp. 23–41.

- [57] B. David, P. Gazi, A. Kiayias, and A. Russell, “Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain,” in *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. B. Nielsen and V. Rijmen, Eds., vol. 10821. Springer, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-78375-8_3 pp. 66–98.
- [58] R. Pass and E. Shi, “Hybrid consensus: Efficient consensus in the permissionless model,” in *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, ser. LIPIcs, A. W. Richa, Ed., vol. 91. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. [Online]. Available: <https://doi.org/10.4230/LIPIcs.DISC.2017.39> pp. 39:1–39:16.
- [59] R. Pass and E. Shi, “Thunderella: Blockchains with optimistic instant confirmation,” in *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. B. Nielsen and V. Rijmen, Eds., vol. 10821. Springer, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-78375-8_1 pp. 3–33.
- [60] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, “Enhancing bitcoin security and performance with strong consistency via collective signing,” in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, T. Holz and S. Savage, Eds. USENIX Association, 2016. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kogias> pp. 279–296.
- [61] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, “Algorand: Scaling byzantine agreements for cryptocurrencies,” in *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 2017. [Online]. Available: <https://doi.org/10.1145/3132747.3132757> pp. 51–68.
- [62] S. Azouvi, P. McCorry, and S. Meiklejohn, “Betting on blockchain consensus with fantomette,” *CoRR*, vol. abs/1805.06786, 2018. [Online]. Available: <http://arxiv.org/abs/1805.06786>
- [63] A. Stewart and E. Kokoris-Kogia, “GRANDPA: a byzantine finality gadget,” *CoRR*, vol. abs/2007.01560, 2020. [Online]. Available: <https://arxiv.org/abs/2007.01560>

- [64] V. Buterin and V. Griffith, “Casper the friendly finality gadget,” *arXiv preprint arXiv:1710.09437*, 2017.
- [65] V. Buterin, D. Reijnders, S. Leonardos, and G. Piliouras, “Incentives in ethereum’s hybrid casper protocol,” in *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2019, Seoul, Korea (South), May 14-17, 2019*. IEEE, 2019. [Online]. Available: <https://doi.org/10.1109/BLOC.2019.8751241> pp. 236–244.
- [66] T. Dinsdale-Young, B. Magri, C. Matt, J. B. Nielsen, and D. Tschudi, “Afgjort: A partially synchronous finality layer for blockchains,” in *Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*, ser. Lecture Notes in Computer Science, C. Galdi and V. Kolesnikov, Eds., vol. 12238. Springer, 2020. [Online]. Available: https://doi.org/10.1007/978-3-030-57990-6_2 pp. 24–44.
- [67] P. Civit, S. Gilbert, and V. Gramoli, “Polygraph: Accountable byzantine agreement,” Cryptology ePrint Archive, Report 2019/587, 2019, <https://eprint.iacr.org/2019/587>.
- [68] P. Sheng, G. Wang, K. Nayak, S. Kannan, and P. Viswanath, “Bft protocol forensics,” 2020.
- [69] J. Neu, E. N. Tas, and D. Tse, “Ebb-and-flow protocols: A resolution of the availability-finality dilemma,” Cryptology ePrint Archive, Report 2020/1091, 2020, <https://eprint.iacr.org/2020/1091>.
- [70] S. Sankagiri, X. Wang, S. Kannan, and P. Viswanath, “The checkpointed longest chain: User-dependent adaptivity and finality,” 2020.
- [71] S. Azouvi, G. Danezis, and V. Nikolaenko, “Winkle: Foiling long-range attacks in proof-of-stake systems,” Cryptology ePrint Archive, Report 2019/1440, 2019, <https://eprint.iacr.org/2019/1440>.
- [72] D. Karakostas and A. Kiayias, “Securing proof-of-work ledgers via checkpointing,” Cryptology ePrint Archive, Report 2020/173, 2020, <https://eprint.iacr.org/2020/173>.
- [73] Y. Lewenberg, Y. Sompolinsky, and A. Zohar, “Inclusive block chain protocols,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 528–547.
- [74] C. Li, P. Li, W. Xu, F. Long, and A. C.-c. Yao, “Scaling nakamoto consensus to thousands of transactions per second,” *arXiv preprint arXiv:1805.03870*, 2018.

- [75] anon, “Advocate system implementation,” <https://github.com/advocate-checkpoint/advocate-systems>, 2021.
- [76] A. Kiayias and G. Panagiotakos, “On trees, chains and fast transactions in the blockchain,” in *Progress in Cryptology - LATINCRYPT 2017 - 5th International Conference on Cryptology and Information Security in Latin America, Havana, Cuba, September 20-22, 2017, Revised Selected Papers*, ser. Lecture Notes in Computer Science, T. Lange and O. Dunkelman, Eds., vol. 11368. Springer, 2017. [Online]. Available: https://doi.org/10.1007/978-3-030-25283-0_18 pp. 327–351.
- [77] H. Yu, I. Nikolic, R. Hou, and P. Saxena, “Ohie: Blockchain scaling made simple,” 2019.
- [78] L. Yang, V. Bagaria, G. Wang, M. Alizadeh, D. M. Tse, G. Fanti, and P. Viswanath, “Prism: Scaling bitcoin by 10,000x,” *ArXiv*, vol. abs/1909.11261, 2019.
- [79] G. Huberman, J. D. Leshno, and C. Moallemi, “Monopoly without a monopolist: An economic analysis of the bitcoin payment system,” *The Review of Economic Studies*, vol. 88, no. 6, pp. 3011–3040, 2021.
- [80] J. Garay, A. Kiayias, and N. Leonardos, “The bitcoin backbone protocol: Analysis and applications,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015.
- [81] S. Lin and D. J. Costello, *Error control coding*. Pearson, 2004.
- [82] A. Orlicsky, K. Viswanathan, and J. Zhang, “Stopping set distribution of LDPC code ensembles,” *IEEE Trans. on Information Theory*, vol. 51, no. 3, 2005.

APPENDIX A

ADDITIONAL MATERIALS FOR CHAPTER 2

A.1 Model

We consider a synchronous, round-based network model similar to that of [80] and [32]. Blockchain is defined as a pair (Π, g) , where Π is an algorithm that maintains a blockchain data structure \mathcal{C} consisting of a set of *blocks*. The function $g(\mathbf{tx}, \mathcal{C})$ encodes a *ledger inclusion rule*; it takes in a transaction \mathbf{tx} and a blockchain \mathcal{C} , and outputs $g(\mathbf{tx}, \mathcal{C}) = 1$ if \mathbf{tx} is contained in the ledger, \mathcal{L} , defined by \mathcal{C} and 0 otherwise. For example, in Bitcoin, $g(\mathbf{tx}, \mathcal{C}) = 1$ iff \mathbf{tx} appears in any block on the longest chain.

Let \mathbb{N} denote the set of participating nodes. The set of *honest* nodes $\mathcal{H} \subset \mathbb{N}$ strictly follow the blockchain protocol Π . Each honest node has a fixed compute, memory and communication resources. *Malicious* nodes $\mathbb{N} \setminus \mathcal{H}$ are collectively controlled by an adversarial party \mathcal{A} which is modelled as byzantine and thus can arbitrarily deviate from the protocol Π . The fraction of adversarial hash power is denoted by $\beta = 1 - \frac{|\mathcal{H}|}{|\mathbb{N}|}$ ¹. In a sharded blockchain, the nodes are grouped into m shards, and the set of nodes in shard $s \in [m]$ is denoted by \mathbb{N}_s , where the set of honest and malicious nodes is given by $\mathcal{H}_s \subset \mathbb{N}_s$ and $\mathbb{N}_s \setminus \mathcal{H}_s$. We have $\bigcup_s \mathbb{N}_s = \mathbb{N}$, $\bigcup_s \mathcal{H}_s = \mathcal{H}$. The blockchain ledger \mathcal{L} is internally composed of multiple sub-ledgers corresponding to different shards.

¹ β for **bad**. Like [80], we have assumed all nodes have the same hash power, but this model can easily be generalized to arbitrary hash power distributions.

Letting κ denote a security parameter, the *environment* $\mathbb{Z}(1^\kappa)$ captures all aspects external to the protocol itself, such as inputs to the protocol (i.e., new transactions) or interaction with outputs. In an execution of the blockchain protocol, the environment \mathbb{Z} first initializes all nodes as either honest or malicious; like [80]. Thereafter, protocol proceeds in rounds of Δ seconds each and the environment can adaptively change the set \mathcal{H}_s between rounds, as long as the adversary’s total hash power remains bounded by β and $|\mathcal{H}_s| \geq 1 \forall s \in [m]$. This models a *completely adaptive* adversarial setting.

In each round, the environment first delivers inputs to the appropriate nodes (e.g., new transactions), and the adversary delivers any messages to be delivered in the current round. Here, delivery means that the message appears on the recipient node’s input tape. Nodes incorporate the inputs and any messages (e.g., new blocks) into their local blockchain data structure according to protocol Π .

Both honest and malicious nodes interact with a random function $H : \{0, 1\}^* \rightarrow \{0, 1\}^\kappa$ through an oracle $\mathbf{H}(x)$, which outputs $H(x)$. In each round, a node $n \in \mathbb{N}$ is allowed to access the oracle $\mathbf{H}(\cdot)$ at most q times with different nonces s in an attempt to find a valid proof of work. These restrictions model the limited hash rate in the system. If an oracle call produces a proof of work, then the node can deliver a new block to the environment.

All messages broadcast to the environment are delivered by the adversary. The adversary has various capabilities and restrictions. (1) Any message broadcast by an honest node in the previous round must be delivered by the adversary at the beginning of the current round to all remaining honest nodes. However, during delivery, the adversary can present these messages to each honest node in whatever order it chooses. (2) The adversary cannot forge or alter any message sent by an honest node. (3) The adversary can control the actions of malicious nodes. For example, the adversary can choose how malicious nodes allocate their hash power, decide block content, and release mined blocks. Notably, although honest blocks publish mined blocks immediately, the adversary may choose to keep blocks they mined private and release in future round. (4) The adversary can deliver malicious nodes' messages to some honest nodes in one round, and the remaining honest nodes in the next round. We consider a “rushing” adversary that observes the honest nodes' actions before taking its own action for a given round. Notice that we do not model rational users who are not necessarily adversarial but nevertheless may have incentives to deviate from protocol.

Metrics.

Our primary concern will be throughput and security scaling with respect to number of nodes $|\mathbb{N}|$. We now define metrics as done in [32]. Let random variable $\text{VIEW}_{\Pi, \mathcal{A}, \mathbb{Z}}$ denote the joint view of all parties over all rounds; here we have suppressed the dependency on security parameter κ . The randomness is defined over the choice of function $H(\cdot)$, as well as any randomness in the adversary \mathcal{A} or environment \mathbb{Z} . Our goal is to reason about the joint view for all possible adversaries \mathcal{A} and environments \mathbb{Z} . In particular, we want to study the evolution of \mathcal{C}_n^r , or the blockchain of each honest node $n \in \mathcal{H}$ during round r . Following the Bitcoin backbone protocol model [80], we consider protocols that execute for a finite execution horizon r_{\max} , polynomial in κ .

Definition A.1 ([32]). *A transaction tx is $(\epsilon, \mathcal{A}, \mathbb{Z}, r_0, \kappa)$ -cleared iff under an adversary \mathcal{A} , environment \mathbb{Z} , and security parameter κ ,*

$$\mathbb{P}_{\text{VIEW}_{\Pi, \mathcal{A}, \mathbb{Z}}} \left(\bigcap_{\substack{r \in \{r_0, \dots, r_{\max}\} \\ i \in \mathcal{H}}} \{g(\text{tx}, \mathcal{C}_i^r) = b\} \right) \geq 1 - \epsilon - \text{negl}(\kappa),$$

where $b \in \{0, 1\}$; $b = 1$ corresponds to confirming the transactions and $b = 0$ corresponds to rejecting the transaction.

That is, a transaction is considered confirmed (resp. rejected) if all honest party will include (resp. exclude) it from the ledger with probability more than ϵ plus a term negligible in κ resulting from hash collisions, which we ignore in our analysis. We suppress the notation κ from here on.

Our objective is to optimize throughput scaling with respect to number of shards m . We let $|S|$ denote the number of elements in set S . We let \mathbb{T}^r denote all transactions delivered up to and including round r .

Definition A.2 (Throughput, [32]). *Blockchain protocol Π supports a throughput of λ transactions per round if there exists U_ϵ , linear in $\log(1/\epsilon)$, such that for all environments \mathbb{Z} that produce at most λ transactions per round, and for $\forall r \in [1, r_{\max}]$,*

$$\max_{\mathcal{A}} |\{\mathbf{tx} \in \mathbb{T}^r : \mathbf{tx} \text{ is not } (\epsilon, \mathcal{A}, \mathbb{Z}, r)\text{-cleared}\}| < \lambda U_\epsilon.$$

The system throughput is the largest throughput that a blockchain protocol can support.

Notice that although $|\mathbb{T}^r|$ grows with r , whereas the right-hand side of A.2 is constant in r ; this implies that the system throughput λ is the expected *rate* at which we can clear transactions maintaining a bounded transaction queue, taken worst-case over adversary \mathcal{A} and environments \mathbb{Z} producing at most λ transactions per round.

A.2 Security Analysis

A.2.1 Proposer chain properties

In this section we consider Trifecta with longest chain protocol as its proposer chain. Garay *et. al* in [28] define three important properties of the longest chain protocol: common-prefix, chain-quality and chain-growth. It was shown that, under a certain typical execution of the mining process, these properties hold, and the properties are then used to prove the persistence and liveness of the longest chain protocol. Here we use these properties of the longest chain to prove liveness and consistency of shards. We borrow the definition of the longest chain properties [32] which are very similar to from [28].

Definition A.3 (Chain-Growth). *The Proposer chain-growth property with parameters ϕ_g and s_g states that for any s_g rounds there are at least $\phi_g s_g$ blocks added to the proposer chain during this time interval.*

Definition A.4 (Chain-Quality). *The proposer (μ_g, u_g) -chain-quality property holds at round r if at most μ_g fraction of the last v_g consecutive blocks on the proposer chain \mathcal{C} at round r are mined by the adversary.*

Definition A.5 (Common-prefix). *The k_g -deep common-prefix property holds at round r if the k_g -deep prefix of the longest proposer chain at round r remains a prefix of any longest chain in any future round.*

We now state three lemmas are under events $E[r - r', r]^2$. Its exact definition is not required for the proof and except for the fact the it occurs with probability $1 - \epsilon(r')$ where $\epsilon(r')$ decreases exponentially in r' . The proposer chain is mined at rate f and let $k = 2fr'$. Since the proposer chain of Trifecta follows the longest chain, the proposer chain properties has the following three properties:

Lemma A.1 (Lemma C.7 in [32]). *(Proposer chain growth) Under event $E[r - r', r]$, proposer chain grows by at least $\frac{k}{6}$ in the interval $[r - r', r]$.*

Lemma A.2 (Lemma C.8 in [32]). *(Proposer common prefix) Under event $E[r - r', r]$, the k -deep common prefix property holds at round r .*

²Equation (18) C.1 [32]

Lemma A.3 (Lemma C.9 in [32]). *(Proposer chain quality) Under event $E[r - r', r]$, the (μ_g, k) -chain quality holds at round r for $\mu_g = \frac{7+2\beta}{8}$*

The proposer chain quality leads to the following corollary.

Corollary A.6. *Any $k = 2r'f$ consecutive blocks in the proposer chain of an honest party has at least one honest block under the event $E[r - r', r]$.*

A.2.2 Shard chain properties

Let us denote the proposer chain by \mathcal{C} and shard chains by \mathcal{C}_s for $s \in [m]$. Given two chains \mathcal{C} and \mathcal{C}' , we say that $\mathcal{C}' \preceq \mathcal{C}$, if \mathcal{C}' is a common prefix of \mathcal{C} . Let $\mathcal{C}^{\lceil k}$ denote the proposer chain \mathcal{C} with last k blocks truncated. Function M_s maps the proposer chain to shard chain s i.e $\mathcal{C}_s = M_s(\mathcal{C})$ and we define the k -deep prefix of shard chain s as $\mathcal{C}_s^{\lceil k} := M_s(\mathcal{C})^{\lceil k}$. We now define the three shard properties along the lines of Definitions A.3, A.4, and A.5:

Definition A.7 (Shard Chain-Growth). *The shard chain growth property with parameters v and ϕ_s states that for any v rounds there are at least ϕ_s blocks added to the shard chain during this time interval.*

Definition A.8 (Shard Chain-Quality). *The Shard μ_s -Chain quality property holds at round r if at most μ_s fraction of all the blocks in the longest shard chain s at round r are mined by the adversary.*

Definition A.9 (Shard chain Common-Prefix). *The shard chain Common-Prefix property holds at round r if k_s -deep prefix of the longest shard chain at round r remains a prefix of any longest shard chain in any future round (for an honest party).*

All the nodes mine on the proposer chain and hence the adversary controls β fraction mining power on proposer chain. Each node simultaneously mines on a shard via PoW sortition discussed in Section ?? . For $s \in [m]$ let the fraction of honest and adversarial mining on shard s at round r be denoted by $\alpha_s[r]$ and $\beta_s[r]$ respectively s.t. $\sum_{s=1}^m \beta_s[r] = \beta$ and $\sum_{s=1}^m \alpha_s[r] = \alpha = 1 - \beta$. The relative adversarial fraction in shard $s \in [m]$ is $\tilde{\beta}_s[r] = \frac{\beta_s[r]}{\beta_s[r] + \alpha_s[r]}$ and let $\tilde{\beta}_s^{[r_1, r_2]} := \frac{1}{r_2 - r_1 + 1} \sum_{r=r_1}^{r_2} \tilde{\beta}_s[r]$ denote the average of the adversarial fraction power from rounds r_1 to r_2 . The mining rate of shard s blocks is f_s . We now define random variables which will be used to prove the shard properties. Let $H_s[r]$ denote the number of honest blocks mined in the shard s in round r , $Z_s[r]$ is the number of shard blocks mined by adversary in round r in shard s . By the sortition process $H_s[r]$ and $Z_s[r]$ are Poisson processes with mean $f_s \Delta(1 - \tilde{\beta}_s[r])$ and $f_b \Delta \tilde{\beta}_s[r]$ respectively. We denote $H_s[r_1, r_2] := \sum_{r=r_1+1}^{r_2} H_s[r]$, similarly for Z_s .

Events: Let $n_1 > 6$, $r' = \frac{k}{2f}$ and $n_2 := \max(\frac{r-1}{r'}, 6)$. We now define the events which will be used to prove the shard properties.

$$\begin{aligned} \mathbf{E}_1 [1, r - 6r'] &:= \{ H_s [1, r - 6r'] > \\ &\quad 0.5 f_s \Delta (n_2 - 6) r' (1 - \tilde{\beta}_s^{[1, r-6r']}) \} \\ \mathbf{E}_2 [1, r] &:= \{ Z_s [1, r - r'] < 2 f_s \Delta r \tilde{\beta}_s^{[1, r]} \} \\ \mathbf{E}_3 [r - n_1 r', r - 6r'] &:= \{ H_s [r - n_1 r', r - 6r'] > \\ &\quad 0.5 f_s \Delta r' (1 - \tilde{\beta}_s^{[r-n_1 r', r-6r']}) (n_1 - 6) \} \\ \mathbf{E}_s (r, n_1) &:= \mathbf{E}[r - 6r', r] \cap \mathbf{E}_1 [1, r - 6r'] \\ &\quad \cap \mathbf{E}_2 [1, r] \cap \mathbf{E}_3 [r - n_1 r', r - 6r'] \end{aligned}$$

Events \mathbf{E}_1 and \mathbf{E}_3 lower bound that the number of honest shard s blocks mined in a certain intervals whereas event \mathbf{E}_2 upper bounds the number of adversarial shard s blocks in certain interval. \mathbf{E}_s is the intersection of all the events.

Theorem A.10 (Shard Chain-Growth). *Under event $\mathbf{E}_s(r, n_1)$, the shard chain grows by at least $0.5 f_s \Delta r' (1 - \tilde{\beta}_s^{[r-n_1 r', r-6r']}) (n_1 - 6)$ blocks in the interval $[r - n_1 r', r]$.*

Proof: Partition the interval $[r - n_1 r', r]$ into two sub intervals: $[r - n_1 r', r - 6r']$ and $[r - 6r', r]$. Since $\mathbf{E}_3[r - yr', r - 6r'] \supseteq \mathbf{E}_s(x, y, r)$, the number honest shard s blocks mined in the first sub-interval is at least $0.5f_s \Delta r' (1 - \tilde{\beta}_s^{[r - n_1 r', r - n_1 r']})(n_1 - 6)$. From Lemma A.1, we know that there is at least one honest proposer block mined in the second sub-interval $[r - 6r', r]$ and denote it by B . Therefore all the honest shard blocks mined in the first sub-interval is referred by either B or previous honest proposer blocks.

Theorem A.11 (Shard Common-prefix). *Under the event $\mathbf{E}[r - r', r]$, where $r' = \frac{k}{2f}$, the k -deep common-prefix property holds at round r .*

Proof: Let $C[r]$ and $C_s[r]$ denote the longest proposer chain at round r and its corresponding shard s chain at round r . The mapping of Trifecta Longest Chain of satisfies the following property:

$$M_s(\mathcal{C}) = M_s(\mathcal{C}^{[k]} || M_s(\mathcal{C}^{k|}))$$

where $\mathcal{C}^{[k]}$ is the k -suffix of the chain \mathcal{C} , $||$ is the chain concatenation operator. Lemma A.2 shows that $\mathcal{C}(r_1)^{[k]} \preceq \mathcal{C}(r_2)$ under $E[r - r', r]$. Applying the above mapping property, we obtain $M_s(\mathcal{C}(r_1)^{[k]}) \preceq M_s(\mathcal{C}(r_2)) \implies \mathcal{C}_s(r_1)^{[k]} \preceq \mathcal{C}_s(r_2)$.

Theorem A.12 (Shard Chain-quality). *Under the event $\mathbf{E}_s(r, 1)$, the μ_s -chain quality property holds at round r for $\mu_s = \frac{2\tilde{\beta}_s^{[1, r]} n_2}{0.5(1 - \tilde{\beta}_s^{[1, r - 6r']})(n_2 - 6) + 2\tilde{\beta}_s^{[1, r]} n_2}$.*

Proof. Since $\mathbf{E}_1[1, r - 6r'] \supseteq \mathbf{E}_s(1, r)$ and $\mathbf{E}_2[1, r] \supseteq \mathbf{E}_s(1, r)$ and from Lemma A.10, the number of honest and adversarial shard blocks in shard chain s for interval $[1, r]$ respectively satisfies:

$$\begin{aligned} H_s[1, r - 6r'] &> 0.5f_s \Delta r' (1 - \tilde{\beta}_s^{[1, r - 6r']})(n_2 - 6) \\ Z_s[1, r] &< 2f_s \Delta r \tilde{\beta}_s^{[1, r]} \end{aligned}$$

Therefore shard chain quality thus satisfies

$$\begin{aligned} \mu_s &= \frac{Z_s[1, r]}{Z_s[1, r] + H_s[1, r - 6r']} \\ &> \frac{2\tilde{\beta}_s^{[1, r]} n_2}{0.5(1 - \tilde{\beta}_s^{[1, r - 6r']})(n_2 - 6) + 2\tilde{\beta}_s^{[1, r]} n_2}. \quad \square \end{aligned}$$

A.2.3 Extension of security properties to Trifecta

Lemma C.6, D.6 and D.2 from [32] prove that Proposer chain of Prism follows Chain Growth, Chain-Quality and Common-Prefix properties respectively as stated by Lemma A.1, A.3, A.2 albeit under a different event $\mathbf{E}'[r - r']$ with a probability $1 - \epsilon'(r')$.

We use this to slightly modify $\mathbf{E}_s(r, n_2)$ to accommodate for $\mathbf{E}'[r - r']$.

$$\begin{aligned} \mathbf{E}'_s(r, n_1) := & \mathbf{E}'[r - 6r', r] \cap \mathbf{E}_1[1, r - 6r'] \\ & \cap \mathbf{E}_2[1, r] \cap \mathbf{E}_3[r - n_1r', r - 6r'] \end{aligned}$$

The relevant theorems for the security of each shard chain can be proven in similar manner as Theorems A.10, A.12, A.11 by substituting $\mathbf{E}_s(r, n_1)$ with $\mathbf{E}'_s(r, n_1)$ in theorem definitions and respective proofs.

A.3 Data availability

We phrase the following *data availability* problem: Given only the header of some transaction block \mathbf{B} in shard j , how can a node in shard i verify that the data is available with at least one honest node in shard j in a *communication efficient* manner?

More specifically, any protocol to verify data availability should satisfy the following security requirements.

Soundness: If a node in shard i has concluded that a transaction block in shard j is fully available, then at least one honest node in shard j will be able to fully recover this block within a constant delay of Δ_c .

Agreement: If a node in shard i has concluded that a transaction block in shard j is fully available, then all the other nodes in the system will conclude that the block is fully available within a constant delay of Δ_c .

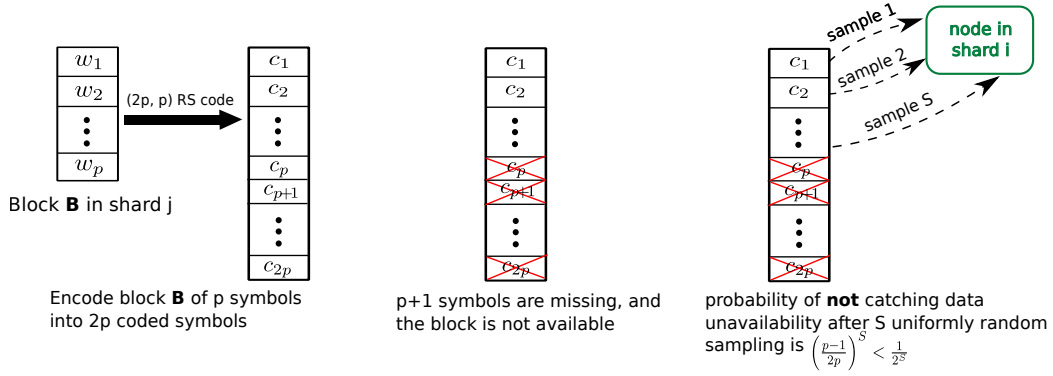


Figure A.1: Random sampling to check data availability of a block **B**, which is encoded by a $(2p, p)$ Reed-Solomon code.

We solve the above data availability problem in *Trifecta* by utilizing coded Merkle tree (CMT) [29], a technique for a light node that only stores block headers to verify the data availability in a communication-efficient manner. Specifically, to apply CMT, we make a minimal assumption that each node is connected to *at least one honest node in every shard*,³ and a node in shard i is viewed as a light node to shard j for every (i, j) pair.

A.3.1 Efficient random sampling via erasure coding

Instead of downloading the entire block, a node could randomly sample a small portion of \mathbf{B} from some node in shard j who claims to have the entire \mathbf{B} . It was proposed in [18] to do this efficiently using erasure codes (see, e.g., [81]). For the purpose of data availability, we can view \mathbf{B} as a stream of b bytes. For some design parameter p , we can evenly and arbitrarily partition the block into p data symbols w_1, \dots, w_p . The block producer encodes \mathbf{B} into $n \geq p$ coded symbols c_1, \dots, c_n using e.g., an (n, p) Reed-Solomon (RS) code. In this case, \mathbf{B} can be recovered using *any* p out of n coded symbols, and it is available unless $n - p + 1$ coded symbols are missing. When, for instance, $n = 2p$, as illustrated in Fig. A.1, after successfully sampling S coded symbols, a node in shard i gains confidence that with probability $1 - (\frac{p-1}{2p})^S$, some node in shard j has at least 50% of the coded symbols, so it can completely recover \mathbf{B} .

While a node in shard i knows that some node in shard j will have sufficiently many symbols to decode, it cannot verify that the coding was done correctly and that the decoded block is indeed the original \mathbf{B} . To defend against such “incorrect-coding” fraud, the nodes outside the shard will forward their sampled symbols to their neighboring nodes inside the shard and rely on them to detect coding errors.

Coding-fraud proof. An honest node in shard j decodes the block \mathbf{B} and verifies the correctness of coding by comparing the Merkle root committed to the decoded block with the one stored in the header. Upon a mismatch, the node constructs a coding-fraud proof to inform the other nodes in shard j . This proof, as large as the original block, contains p coded symbols using which \mathbf{B} can be decoded and verified. It was proposed in [18] to use 2-dimensional Reed-Solomon (2D-RS) codes to reduce the proof size from b to \sqrt{b} , at the cost of storing $2\sqrt{n}$ additional hashes in the header.

In *Trifecta*, a coding-fraud proof is published as a special transaction on the proposer chain. Since each proposer transaction is downloaded by every node in the network, to make *Trifecta* communication efficient, we need a mechanism to verify data availability with 1) constant sampling size for foreign transaction blocks and 2) constant coding-fraud proof size.

³Note that this assumption is much weaker than assuming that majority of each shard is honest.

A.3.2 Constant communication via coded Merkle tree

An (n, p) linear code is characterized by a $(n - p) \times n$ parity check matrix \mathbf{H} , such that a transaction block is correctly encoded iff the coded symbols c_1, \dots, c_n satisfies $\mathbf{H}[c_1, \dots, c_n]^\top = \mathbf{0}$. In other words, the coded block has to pass $n - p$ parity check equations, each of which is an inner product with a row of \mathbf{H} . Since an incorrectly encoded block fails at least one equation, we propose to use a *single* failed equation as the coding-fraud proof.

To make this proof succinct, we leverage low-density parity-check (LDPC) codes for which each parity check equation involves computing XOR of a *constant* number of d symbols. When one of the equations is violated, the decoder generates a coding-fraud proof consisting of $d - 1$ symbols and d membership proofs for all the symbols in that equation. Another node can easily verify the proof by 1) checking that the $d - 1$ symbols pass their membership proofs and 2) decoding the missing symbol and verifying that it fails its membership proof. To enable such a proof, Trifecta adopts a hash-aware peeling decoder, which decodes a new symbol from an equation with $d - 1$ hash-verified symbols at each time.

We use LDPC ensembles that are known to perform well under the peeling decoder (see, e.g., [82]), i.e., the ones with large “stopping distance” which are guaranteed to be decodable even if only a constant fraction of the coded symbols are available. As an example, an $(n, p) = (32000, 8000)$ regular LDPC code whose parity check equation involves $d = 8$ coded symbols can be decoded by a peeling decoder from any $0.876n = 28032$ symbols, with probability of at least 99.97%. This also makes the probability that the block is not available but not detected by a node after $S = 30$ samples as small as 0.00063%. Therefore, a node outside the shard takes a constant number of samples to check data availability.

To be able to identify which equation is violated during peeling decoding, we require the hash commitments of *all* coded symbols available at the decoder. Storing all of them in the block header is prohibitively expensive since the number of hashes increases with block size n . To resolve this issue, we choose to recover the hashes on-the-fly, by utilizing Coded Merkle tree (CMT) [29], which iteratively encodes the data with an LDPC code, computes hash commitments of the coded symbols, and concatenates these hashes into the data for the next level until the size of the hash commitments is small enough to be stored in the block header.

Brief description of CMT. The first level of the tree consists of n coded symbols created from p data symbols constituting the original block via a rate $r = p/n$ systematic LDPC code. For the second level, we first compute the hashes of all the symbols in the first level, then aggregate $q \geq 2$ hashes together to form a total of n/q new data symbols, and then use another $(\frac{n}{qr}, \frac{n}{q})$ LDPC code to generate $\frac{n}{qr}$ coded symbols. We repeat this process to grow higher levels of the tree until reaching a constant target number of t hashes, which are stored in the header.⁴ The membership proof of a base-level symbol in CMT consists of all the sibling hashes between this symbol and the tree root. The proof automatically provides membership proofs for all the symbols on the higher levels in the proof. Hence, sampling a symbol on the base level samples one membership-proved symbol on each level of CMT for free.

To decode a block, a node decodes the corresponding CMT starting from the top level and all the way down to the base level. The decoded data at each level, which is the hash commitments of the symbols on the level below, is used to verify the symbols that are newly decoded from the peeling decoder. This allows a node to construct a coding-fraud proof at any level with a constant size of $d - 1$ symbols.

⁴A standard Merkle tree is a special case of CMT with $r = 1$, $q = 2$, and $t = 1$.

Security guarantees We can prove, using the similar arguments for security as in [29], soundness and agreement of this approach using CMT are violated with a probability that decreases exponentially with S when the total number of nodes N scales linearly with the block size. Since soundness and agreement are satisfied within a constant delay Δ_c , we can argue that the blockchain system now has the same security properties as proved in section 2.4 albeit with a network delay bound of Δ_c .

A.4 Proofs of Scaling

A.4.1 Proof of Proposition 2.1

Let us define the *overhead ratio* as the ratio of (resource used to maintain the consensus engine and Trifecta ancillary functionalities) to (resource used to maintain any one shard). We define resource usage as a 3-dimensional vector: (1) Computation (2) Communication and (3) Storage. We will compute the overhead as the maximum value of the 3-dimensions. Here, we account for resources consumed by all data structures involved in maintaining the single consistent log; this is discussed in detail next.

Single consistent log: The ordered log is maintained by the order manager and consists of the following entries:

- **Shard block pointers:** Consists of the tuple (hash of the shard block, shard-id, mining proof). K such shard block pointers are appended to the log for every shard block and each pointer is of size $\log K$. Resource usage = $O(K \log K)$.
- **Shard State commitments:** There are $O(K)$ state commitments per epoch. We set the epoch duration E such that in expectation, there is one shard block per epoch. Resource usage = $O(K)$.
- **State commitment challenge interactions:** Consists of challenges and replies from epoch leaders. There are $O(N)$ challenges in total which accounts for $R = O(\log B)$ interactions. Resource usage = $O(N \log B)$.

- **Data availability votes:** *Signed* (Availability, shard block hash). Constant size posted for K shard blocks and needs $O(N)$ votes per shard block. Resource usage = $O(NK)$.
- **Data availability fraud proofs:** Incorrect coding proofs of size $O(\log B)$ for at most K shards. Resource usage = $O(K \log B)$.

Shard rotation: A node rotating to a new shard needs to synchronize to the state of the new shard. The synchronization involves downloading the state corresponding to the latest state commitment and processing shard blocks proposed after the latest state commitment. The resource cost is $O(1 + B)$ per new allocation. We assume that all nodes may need to rotate every Δ blocks. A node also needs to synchronize to another shard if it is chosen as its leader for state commitment purposes. The epoch leader election will elect a node with probability $O(K/N)$ per epoch. We set $\Delta = K$ to get the resource usage as $O(B/K + BK/N)$.

Data availability requests: A newly mined block receives requests for $O(N)$ random chunks for the base CMT symbol of size $O(B/N)$, resource usage = $O(B \log N)$ per shard. Since each shard has $O(N/K)$ nodes, the load can be distributed equally amongst all nodes, resource usage per node = $O(\frac{BK}{N} \log N)$. Each node receives K chunks of size $O(\frac{B}{N} \log N)$. Resource usage = $O(\frac{KB}{N} \log N)$.

Mining: The miner has to mine shard blocks for the shard of its choice (native shard) and proposer blocks. The cost of maintaining state to mine shard blocks is $O(1)$. The cost of maintaining state to mine proposer blocks is included in the cost of maintaining the ordered log calculated above.

Total resource usage

The total resource usage per shard block is given by $O(K \log K + N \log B + NK + B/K + BK/N)$ in all 3 resource dimensions, Thus *Overhead-ratio* = $O\left(\frac{K \log K}{B} + \frac{N \log B}{B} + \frac{NK}{B} + \frac{K \log N}{N}\right) = o(1)$, as B becomes large ($\Omega(NK)$) and $N > \Omega(K \log K)$.

A.4.2 Proof of Corollary 2.1

Let us extend the proof from appendix A.4.1 by allowing for a total throughput lower than $O(K)$. We set per shard throughput of $v = f(N, K)$, where $f(N, K)$ is an $o(1)$ function. Note that in all previous discussions, v is $O(1)$. This setting allows us to set a total throughput of $Kf(N, K) = o(K)$. The resource usage rate for shard ledger is $O(f(N, K))$ and the resource overhead being

$$O\left(\frac{f(N, K)K \log K}{B} + \frac{f(N, K)N \log B}{B} + \frac{f(N, K)NK}{B} + \frac{f(N, K)K \log N}{N}\right) \quad (\text{A.1})$$

Note that average time to add a block to a shard is given by

$$\chi L = \frac{B}{v} = \frac{B}{f(N, K)} \quad (\text{A.2})$$

Substituting Equation (A.2) in Equation (A.1), we get a resource overhead of $O\left(\frac{K \log K}{L} + \frac{N \log B}{L} + \frac{NK}{L} + \frac{f(N, K)K \log N}{N}\right) = o(1)$ as latency L becomes $\Omega(NK)$ and $N > \Omega(K \log K)$. The result is contrary to the intuition that a latency-throughput tradeoff can be achieved by varying block size B . For example, a block size of $O(N\sqrt{K})$ should result in a total throughput lower than $O(\sqrt{K})$. However, a \sqrt{K} factor reduction in block size does not lead to lowering of the inter block time since the transaction throughput per shard v is reduced by a factor of \sqrt{K} as well.

APPENDIX B

ADDITIONAL MATERIALS FOR CHAPTER 3

B.1 Adversary agnostic allocations

The allocation function g_t takes the total mining power per shard and the previous allocations $(\{g_{t-j}\}_{j=1}^{t-1})$ as an input,

$$g_t : (\{\boldsymbol{\beta}(1) + \boldsymbol{\gamma}(1)\}, \dots, \{\boldsymbol{\beta}(t-1) + \boldsymbol{\gamma}(t-1)\}, g_1, \dots, g_{t-1}) \rightarrow \boldsymbol{\gamma}(t), \quad (\text{B.1})$$

We show that any deterministic f_t with $\gamma = 1/2$ can be implemented as a deterministic g_t . We assume $g_1 = f_1 = \boldsymbol{\gamma}(1) = [1/2K, \dots, 1/2K]$ since honest nodes can uniformly allocate themselves at the start. At $t = 2$ we apply the mapping, $g_2(x_1, y_1) = f_2(x_1 - y_1, y_1)$, i.e. $g_2(\{\boldsymbol{\beta}(1) + \boldsymbol{\gamma}(1)\}, g_1) = g_2(\{\boldsymbol{\beta}(1) + \boldsymbol{\gamma}(1)\} - g_1, g_1)$ since $\{\boldsymbol{\beta}(1) + \boldsymbol{\gamma}(1)\} - g_1 = \{\boldsymbol{\beta}(1) + \boldsymbol{\gamma}(1)\} - \boldsymbol{\gamma}(1) = \boldsymbol{\beta}(1)$. We inductively extend this mapping to a general t where $g_t(x_1, \dots, x_{t-1}, y_1, \dots, y_{t-1}) = f_t(x_1 - y_1, \dots, x_{t-1} - y_{t-1}, y_1, \dots, y_{t-1})$. A key property of our DSA is that nodes do not know $\boldsymbol{\beta}(j)$, they estimate $\{\boldsymbol{\beta}(j) + \boldsymbol{\gamma}(j)\}$ from the total mining power in each shard and use it as an input to g_t .

Let us define $\psi_{f_t}(K, \gamma, T)$ and $\psi_{g_t}(K, \gamma, T)$ the honest fraction of the worst performing shard under a worst case adversary on running the re-allocation function f_t and g_t respectively as follows:

$$\psi_{f_t}(K, \gamma, T) = \min_{\{\beta_i(t)\}_{i,t}} \min_i \{\bar{r}_i(T)\} \quad s.t. \sum_{i=1}^K \gamma_i(t) = \gamma \quad (\text{B.2})$$

$$\psi_{g_t}(K, \gamma, T) = \min_{\{\beta_i(t)\}_{i,t}} \min_i \{\bar{r}_i(T)\} \quad s.t. \sum_{i=1}^K \gamma_i(t) = \gamma \quad (\text{B.3})$$

we prove the following lemma:

Lemma B.1. *For deterministic allocation functions f_t and g_t ; $\psi_{g_t}(K, \gamma, T) \geq \psi_{f_t}(K, 1/2, T) \forall \gamma \geq 1/2$*

Proof:

We argue that:

$$\psi_{g_t}(K, \gamma, T) \geq \psi_{g_t}(K, 1/2, T) \tag{B.4}$$

In the above inequality; the additional honest miners in RHS can be thought of as behaving as adversaries (whose strategy space is exhaustive) in the worst case; if the additional honest miners behave as adversaries, we get $\psi_{g_t}(K, \gamma, T) \geq \psi_{g_t}(K, 1/2, T)$. Moreover since there always exists a bidirectional mapping from f_t to g_t : $g_t(x_1, \dots, x_{t-1}, y_1, \dots, y_{t-1}) = f_t(x_1 - y_1, \dots, x_{t-1} - y_{t-1}, y_1, \dots, y_{t-1})$, we can state the following equality.

$$\psi_{g_t}(K, 1/2, T) = \psi_{f_t}(K, 1/2, T) \tag{B.5}$$

Combining Equation (B.4) and Equation (B.5) we get:

$$\begin{aligned} \psi_{g_t}(K, \gamma, T) &\geq \psi_{g_t}(K, 1/2, T) \quad \forall \gamma \geq 1/2 \\ &= \psi_{f_t}(K, 1/2, T) \end{aligned} \tag{B.6}$$

B.2 Proofs

B.2.1 Proof of Proposition 3.1

Proof: Lower bound:

We start by focusing on a single shard being targeted by the adversary (denoted by shard-id=1 w.l.o.g.) and show that adversary cannot sustain an attack for a long enough time instance for that shard even if the adversary focuses all its resource in targeting that shard. Let a_t denote the adversarial allocation of a at epoch t and b_{t-1} denote the honest allocation to the worst performing shard at epoch t , l denote the honest allocation a shard can have and u denote the highest honest allocation possible for any shard. We have $a_t := \beta_1(t)$, $b_t = \frac{\gamma}{2\beta}a_t + \frac{\gamma}{2K}$, $l = \frac{\gamma}{2K}$, $u = \frac{\gamma}{2} + \frac{\gamma}{2K}$. We have $\frac{\gamma}{2K} \leq b_t \leq \frac{\gamma}{2} + \frac{\gamma}{2K}$.

$$\begin{aligned} \psi(K) &= \min_{\{\frac{\gamma}{2K} \leq b_t \leq \frac{\gamma}{2} + \frac{\gamma}{2K}\}} \left\{ \frac{1}{T} \sum_{t=1}^T \frac{b_{t-1}}{b_{t-1} + a_t} \right\} \\ &= \min_{\{\frac{\gamma}{2K} \leq b_t \leq \frac{\gamma}{2} + \frac{\gamma}{2K}\}} \left\{ \frac{1}{T} \sum_{t=1}^T \frac{b_{t-1}}{b_{t-1} + (\frac{2\beta}{\gamma}b_t - \frac{\beta}{K})} \right\} \\ &\geq \min_{\{\frac{\gamma}{2K} \leq b_t \leq \frac{\gamma}{2} + \frac{\gamma}{2K}\}} \left\{ \frac{1}{T} \sum_{t=1}^T \frac{b_{t-1}}{b_{t-1} + \frac{2\beta}{\gamma}b_t} \right\} =: \phi(K) \end{aligned} \quad (\text{B.7})$$

$$\phi(K) = \min_{\{l \leq b_t \leq u\}} \left\{ \frac{1}{T} \sum_{t=1}^T \frac{b_{t-1}}{b_{t-1} + \omega b_t} \right\} \quad (\text{B.8})$$

where $\omega = \frac{2\beta}{\gamma}$. Now, we call a epoch t g -good if the honest-fraction at that time is greater than g . We will show that the adversary does not have the ability to have a consecutive run of g -bad instances for more than τ -periods for any shard. Suppose an instant t is not g -good. Then

$$\frac{b_{t-1}}{b_{t-1} + \omega b_t} \leq g$$

which implies $b_t \geq b_{t-1}r$ where $r = \frac{1-g}{\omega g}$. Note $g \leq \frac{1}{1+\omega}$ implies $r \geq 1$. Suppose there are τ consecutive g -good instances at time t . Then $b_t \geq b_{t-\tau}r^\tau$. Given $b_t \leq u$ and $b_{t-\tau} \geq l$, we have $u \geq lr^\tau$. This implies $\tau \leq \log_r \left(\frac{u}{l}\right)$. Thus there is a g -good instant every $\tau + 1$ instances. Intuitively, this is true since the resources of the adversary will be exhausted and needs to be reset in order to incrementally increase it's allocation to that shard. This implies that the honest-fraction is at least $\frac{g}{\tau+1}$.

$$\phi(K) \geq \frac{g \log r}{\log \left(\frac{ur}{l}\right)} \geq \frac{g \log \left(\frac{1-g}{\omega g}\right)}{\log \left(r^{\frac{\gamma}{2} + \frac{\gamma}{2K}}\right)}.$$

Consider a worst case scenario where $\beta = 0.5$, $\gamma = 0.5$, for all $g \leq \frac{1}{1+\omega} = 1/3$, choosing $g = 1/4$, and $1/2 + 1/2K \leq 1$, we get:

$$\phi(K) \geq \frac{0.14}{\log_2(3K)}. \quad (\text{B.9})$$

Upper Bound: We demonstrate an adversarial strategy which holds to the following claim: $\psi(K) \leq O\left(\frac{\log \log K}{\log K}\right)$. Compare the denominators of $\psi(K)$ and $\phi(K)$:

$$\begin{aligned} \frac{\gamma}{6\beta}a_{t-1} + \frac{\gamma}{6K} + \frac{\gamma a_t}{3\beta} + \frac{\gamma}{3K} &\leq \left\{ \frac{\gamma}{2\beta}a_{t-1} + \frac{\gamma}{2K} \right\} + a_t \\ &\leq \left\{ \frac{\gamma}{2\beta}a_{t-1} + \frac{\gamma}{2K} \right\} + a_t + \frac{\gamma}{K} \end{aligned}$$

We observe $\phi(K) \leq \psi(K) \leq 3\phi(K)$ and hence can rewrite our claim as : $\phi(K) \leq O\left(\frac{\log \log K}{\log K}\right)$.

We show an adversarial sequence to establish the upper bound.

Let the sequence b_t be $\ell, lr, \dots, lr^\tau, 0, \ell, lr, \dots, lr^\tau$. Note that the reset is necessary since the adversarial power is limited to β . We will calculate $\phi(K)$ based on a single period. Recall $\ell = \frac{\gamma}{2K}$, $u = \frac{\gamma}{2} + \frac{\gamma}{2K}$. Note $lr^\tau = u$, so $\tau = \log_r(\frac{u}{\ell})$.

$$\begin{aligned} \phi(K) &\leq \frac{\tau}{\tau+1} \frac{1}{1+2r} + \frac{1}{\tau+1} \\ &= \frac{\tau}{\tau+1} \frac{1}{1+2r} + \frac{1}{\tau+1} \left[\frac{1}{1+2r} + \frac{2r}{1+2r} \right] \\ &= \frac{1}{1+2r} + \frac{2r}{2r+1} \frac{1}{\tau+1} \\ &= \frac{1}{1+2r} + \frac{2r}{2r+1} \cdot \frac{\log r}{\log(cr)} \end{aligned}$$

where $c = \frac{u}{\ell} = K + 1$. Choosing $r = \log K$, we get

$$\begin{aligned} \phi(K) &\leq \frac{1}{1+2\log K} + \frac{2\log K}{2\log K+1} \cdot \frac{\log \log K}{\log(K+1)} \\ &= O\left(\frac{\log \log K}{\log K}\right) \end{aligned} \tag{B.10}$$

The convergence of $\bar{r}_i(t)$ for 2 worst performing shards of experiments corresponding to figure 3.4a is illustrated in figure 3.3.

B.2.2 Proof of Theorem 3.2

Strategy space inequality

Let us define $f(\gamma)$ as follows:

$$f(\gamma) = \min_{\beta} \sum_{i=1}^K u_i \frac{\gamma_i}{\gamma_i + \beta_i}$$

We now show that $f(\gamma) = (\sum_j \sqrt{u_j \gamma_j})^2$, using constrained Lagrange optimization.

$$\begin{aligned}
f(\gamma) &= \min_{\beta} \sum_{i=1}^K u_i \frac{\gamma_i}{\gamma_i + \beta_i} \text{ s.t. } \sum_{i=1}^K \beta_i = \beta \\
obj &= \sum_i u_i \frac{\gamma_i}{\gamma_i + \beta_i} + \lambda (\sum_{i=1}^K \beta_i - \beta) \\
\frac{\partial obj}{\partial \beta_i} &= \frac{-u_i \gamma_i}{(\gamma_i + \beta_i)^2} + \lambda = 0 \\
\gamma_i + \beta_i &= \frac{\sqrt{u_i \gamma_i}}{\sqrt{\lambda}} \text{ and } \sqrt{\lambda} = \sum_{i=1}^K \sqrt{u_i \gamma_i}
\end{aligned}$$

substituting the value of λ in $f(\gamma)$, we get:

$$f(\gamma) = \sum_{i=1}^K (\sqrt{u_i \gamma_i} \sum_{i=1}^K \sqrt{u_i \gamma_i}) = (\sum_{i=1}^K \sqrt{u_i \gamma_i})^2$$

Let \mathbf{v} denote a sorted list of \mathbf{u} sorted in a descending order and $pos(u_i)$ denote the position of u_i in \mathbf{v} with $pos(\max u_i) = 1$.

We define \tilde{u}_i as follows:

$$\tilde{u}_i = \begin{cases} u_i, & \text{if } pos(u_i) \leq s \\ 0, & \text{otherwise} \end{cases}$$

substituting $\gamma_i = \gamma \frac{\tilde{u}_i}{\sum_{i=1}^K \tilde{u}_i}$, we get:

$$\begin{aligned}
f(\gamma) &= \frac{\gamma}{\sum_{i=1}^K \tilde{u}_i} (\sum_{i=1}^K \tilde{u}_i)^2 = \gamma \sum_{i=1}^K \tilde{u}_i \geq \gamma \frac{s}{K} \sum_{i=1}^K u_i \\
\max_{\gamma} \min_{\beta} \sum_{i=1}^K u_i \frac{\gamma_i}{\gamma_i + \beta_i} &\geq f(\gamma) \geq \gamma \frac{s}{K} \sum_{i=1}^K u_i \tag{B.11}
\end{aligned}$$

Modified strategy space inequality

Let us now modify the policy γ to ensure that $\gamma_i \geq \frac{b}{s} i f \tilde{\gamma}_i > 0$. Let $\tilde{\gamma}_i = \gamma \frac{\tilde{u}_i}{\sum_{i=1}^K \tilde{u}_i}$, we have $f(\tilde{\gamma}) \geq \frac{s}{K} \gamma \sum_{i=1}^K u_i$ let us define γ as follows:

$$\gamma = \frac{1}{1 + q/\gamma} \text{Proj}(C_{q/s}, \tilde{\gamma}),$$

where $\text{Proj}(C_{q/s}, \cdot)$ is a projection of the s non-zero values of $\tilde{\gamma}_i$ to the set $C_{q/s} = [q/s, 1]^s$, the projection will cause at most s values to grow by $\frac{q}{s}$. Thus, we normalize all values by $1 + q/\gamma$. We need to ensure that $\gamma_i \geq \frac{b}{s} \forall i \in [K]$, hence we get the inequality:

$$\frac{q/s}{1 + q/\gamma} \geq \frac{q/s}{1 + 2q} \geq \frac{b}{s} \Rightarrow q \geq \frac{b}{1 - 2b}$$

we set $q = \frac{b}{1-2b}$, our modified γ now satisfies $\gamma_i \geq b/s$.

We now have the following inequality

$$f(\gamma) \geq \frac{1}{1 + q/\gamma} f(\tilde{\gamma}) \geq (1 - 2b) f(\tilde{\gamma})$$

$$\min_{\beta} \sum_i u_i \frac{\gamma_i}{\gamma_i + \beta_i} \geq \frac{s}{K} \gamma (1 - 2b) \sum_{i=1}^K u_i$$

Stochastic strategy space inequality

We define the event $E_i(t) : \Gamma_i(t) \geq cN\gamma_i(t)$, and $\mathbf{E}(t) : \bigcap_{i=1}^K E_i(t)$

Let us compute the tail bound on $\Gamma_i(t)$ using the divergence bound on binomial distributions, we get

$$\begin{aligned}
P(\Gamma_i(t) \leq cN\gamma_i(t)) &\leq e^{-nD(c\frac{\gamma_i(t)}{\gamma} \parallel \frac{\gamma_i(t)}{\gamma})} \\
D(c\frac{\gamma_i(t)}{\gamma} \parallel \frac{\gamma_i(t)}{\gamma}) &= c\frac{\gamma_i(t)}{\gamma} \log(c) + (1 - c\frac{\gamma_i(t)}{\gamma}) \log\left(\frac{1 - c\frac{\gamma_i(t)}{\gamma}}{1 - \frac{\gamma_i(t)}{\gamma}}\right) \\
&\geq (-c + c \log c + 1)\frac{\gamma_i(t)}{\gamma} \\
&\geq (-c + c \log c + 1)\gamma_i(t) \\
&\geq (-c + c \log c + 1)\frac{b}{K} \\
P(\Gamma_i(t) \leq cN\gamma_i(t)) &\leq e^{-n\frac{b}{K}(-c+c \log c+1)}. \tag{B.12}
\end{aligned}$$

We use the divergence bounds derived above to lower bound the event probability

$$P(\mathbf{E}(t)) = 1 - P(\mathbf{E}(t)^c) \geq 1 - \sum_{i=1}^K P(E_i(t)^c) \tag{B.13}$$

$$\geq 1 - se^{-n\frac{b}{s}(-c+c \log c+1)} \tag{B.14}$$

where equation B.13 uses a union bound and the fact that $P(E_i(t)^c) = 0$ for $K-s$ shards which have $\gamma_i = 0$ and equation B.14 is derived from equation B.12.

$$\begin{aligned}
\mathbb{E}_{\mathbf{r}(t)} & \left[\min_{\beta(t)} \sum_i u_i(t-1) \frac{\Gamma_i(t)}{\Gamma_i(t) + N\beta_i(t)} \right] \\
& \geq P(\mathbf{E}(t)) \min_{\beta(t), \mathbf{E}(t)} \sum_i u_i(t-1) \frac{\Gamma_i(t)}{\Gamma_i(t) + N\beta_i(t)} \\
& \geq P(\mathbf{E}(t)) \left(\sum_{i=1}^K \sqrt{u_i c \gamma_i} \right)^2 \\
& \geq P(\mathbf{E}(t)) c(1-2b)\gamma \sum_{i=1}^K u_i \\
& \geq (1 - se^{-n \frac{b}{s}(-c+c \log c+1)}) c(1-2b)\gamma \sum_{i=1}^K \tilde{u}_i \\
& \geq (1 - se^{-n \frac{b}{s}(-c+c \log c+1)}) c(1-2b) \frac{s}{K} \gamma \sum_{i=1}^K u_i \\
\mathbb{E}_{\mathbf{r}(t)} & \left[\min_{\beta(t)} \mathbf{u}(t-1) \cdot \mathbf{r}(t) \right] \geq h \sum_{i=1}^K u_i
\end{aligned}$$

We want to lower bound $(1 - se^{-n \frac{b}{s}(-c+c \log c+1)}) \geq 1 - \epsilon$, thus we need

$$\begin{aligned}
se^{-n \frac{b}{s}(-c+c \log c+1)} \leq \epsilon & \Rightarrow n \geq \frac{s \log(\frac{s}{\epsilon})}{b(-c+c \log c+1)} \\
N & \geq \frac{s \log(\frac{s}{\epsilon})}{\gamma b(-c+c \log c+1)} \tag{B.15}
\end{aligned}$$

Approach to the convex set

We will now show that $\bar{\mathbf{r}}(t)$ approaches C_h with $h = (1 - se^{-n \frac{b}{s}(-c+c \log c+1)}) c(1 - 2b) \frac{s}{K} \gamma$.

Let $\pi_{C_h}(t)$ be the projection of $\bar{\mathbf{r}}(t)$ on C_h . Let us define a halfspace \mathbf{H}_{t+1} formed by the hyperplane \mathbf{P}_{t+1} , such that \mathbf{P}_{t+1} is normal to $(\bar{\mathbf{r}}(t) - \pi_{C_h}(t))$ and passes through π_{C_h} and \mathbf{H}_{t+1} contains C_h . The variables defined above satisfy the following equations:

$$\begin{aligned}
\pi_{C_h}(\mathbf{t}) &= \arg \min_{\mathbf{y} \in C_h} \|\mathbf{y} - \bar{\mathbf{r}}(t)\| \\
\pi_{C_h}(t)_i &= hI_{\{\bar{r}_i(t) < h\}} + \bar{r}_i(t)I_{\{\bar{r}_i(t) \geq h\}} \\
\pi_{C_h}(\mathbf{t}) - \bar{\mathbf{r}}(t) &= \mathbf{u}(t) \quad u_i(t) = (h - \bar{r}_i(t)) \\
\mathbf{P}_{t+1}(\mathbf{x}) &: \sum_{i=1}^K (h - \bar{r}_i(t))_+ x_i - h \sum_{i=1}^K (h - \bar{r}_i(t))_+ = 0 \\
\text{such that} \quad \mathbf{u} \cdot \mathbf{x} &- h \sum_{i=1}^K u_i = 0 \\
\mathbf{H}_{t+1}(\mathbf{x}) &: \sum_{i=1}^K (h - \bar{r}_i(t))_+ x_i - h \sum_{i=1}^K (h - \bar{r}_i(t))_+ \geq 0 \\
\text{such that} \quad \mathbf{u} \cdot \mathbf{x} &- h \sum_{i=1}^K u_i \geq 0
\end{aligned}$$

Let us define d_t as the distance of $\bar{\mathbf{r}}(t)$ from the convex set C_h , i.e. $d_t = \|\bar{\mathbf{r}}(t) - \pi_{C_h}(t)\|$, $d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|$ for any $\mathbf{a}, \mathbf{b} \in \mathbf{R}^K$.

$$\begin{aligned}
d_{t+1}^2 &= d^2(\bar{\mathbf{r}}(\mathbf{t} + \mathbf{1}), \pi_{C_h}(\mathbf{t} + \mathbf{1})) \leq d^2(\bar{\mathbf{r}}(\mathbf{t} + \mathbf{1}), \pi_{C_h}(\mathbf{t})) \\
&= \|\bar{\mathbf{r}}(\mathbf{t} + \mathbf{1}) - \pi_{C_h}(\mathbf{t})\|_2^2 \\
&= \left\| \frac{t}{t+1} \bar{\mathbf{r}}(\mathbf{t}) + \frac{1}{t+1} \mathbf{r}(\mathbf{t} + \mathbf{1}) - \pi_{C_h}(\mathbf{t}) \right\|_2^2 \\
&= \left\| \frac{t}{t+1} (\bar{\mathbf{r}}(\mathbf{t}) - \pi_{C_h}(\mathbf{t})) + \frac{1}{t+1} (\mathbf{r}(\mathbf{t} + \mathbf{1}) - \pi_{C_h}(\mathbf{t})) \right\|_2^2 \\
&= \left(\frac{t}{t+1} \right)^2 \|\bar{\mathbf{r}}(\mathbf{t}) - \pi_{C_h}(\mathbf{t})\|_2^2 + \left(\frac{1}{t+1} \right)^2 \|\mathbf{r}(\mathbf{t} + \mathbf{1}) - \pi_{C_h}(\mathbf{t})\|_2^2 \\
&\quad + \frac{2t}{(t+1)^2} (\bar{\mathbf{r}}(\mathbf{t}) - \pi_{C_h}(\mathbf{t})) \cdot (\mathbf{r}(\mathbf{t} + \mathbf{1}) - \pi_{C_h}(\mathbf{t}))
\end{aligned}$$

$$(t+1)^2 d_{t+1}^2 - t^2 d_t^2 \leq \|\mathbf{r}(\mathbf{t} + \mathbf{1}) - \boldsymbol{\pi}_{C_h}(\mathbf{t})\|_2^2 + 2t * ((\boldsymbol{\pi}_{C_h}(\mathbf{t}) - \bar{\mathbf{r}}(\mathbf{t})) \cdot (\boldsymbol{\pi}_{C_h}(\mathbf{t}) - \mathbf{r}(\mathbf{t} + \mathbf{1})))$$

we know the following equations:

$$\begin{aligned} \|\mathbf{r}(\mathbf{t} + \mathbf{1}) - \boldsymbol{\pi}_{C_h}(\mathbf{t})\|_2^2 &\leq h^2 K \\ (\boldsymbol{\pi}_{C_h}(\mathbf{t}) - \bar{\mathbf{r}}(\mathbf{t})) \cdot \boldsymbol{\pi}_{C_h}(\mathbf{t}) &= h \sum_{i=1}^K u_i \\ (\boldsymbol{\pi}_{C_h}(\mathbf{t}) - \bar{\mathbf{r}}(\mathbf{t})) \cdot \boldsymbol{\pi}_{C_h}(\mathbf{t}) &\leq \mathbb{E}_{\Gamma_i(t+1)} \left[\sum_i u_i(t) r_i(t+1) \right] \end{aligned}$$

Summing terms for $t \in [T]$, we get

$$d_T^2 \leq h^2 \frac{K}{T} + \frac{2}{T} \sum_{t=1}^{T-1} \frac{t}{T} (\mathbb{E}_{\Gamma_i(t)} \left[\sum_i u_i(t-1) r_i(t) \right] - (\mathbf{u}(\mathbf{t} - \mathbf{1})) \cdot \mathbf{r}(\mathbf{t}))$$

The term $Y_t = (\mathbb{E}_{\Gamma_i(t)} [\sum_i u_i(t-1) r_i(t)] - ((\mathbf{u}(\mathbf{t} - \mathbf{1})) \cdot \mathbf{r}(\mathbf{t})))$ is a martingale difference sequence w.r.t. history at time t and $|Y_t| \leq 2hs$

Given $\epsilon_m > 0$, by the Azuma- Hoeffding inequality we have:

$$\mathbb{P} \left(\frac{1}{T} \left\| \sum_{t=1}^{T-1} Y_t \right\| > \epsilon_m \right) \leq 2e^{-\frac{T\epsilon_m^2}{8h^2s^2}}$$

Let us set $\epsilon_m = 2hs\sqrt{\frac{2}{T} \log(\frac{2}{\delta})}$, we get

$$\mathbb{P} \left(\frac{1}{T} \left\| \sum_{t=1}^{T-1} Y_t \right\| > \epsilon_m \right) \leq \delta$$

Thus with a high probability of $1 - \delta$, we have distance to the convex set reducing as:

$$d_T^2 \leq h^2 \frac{K}{T} + 4hs\sqrt{\frac{2}{T} \log(\frac{2}{\delta})}$$

B.2.3 Proof of Proposition 3.2

Proof: At $t = 1$ the adversary is focused on $\frac{K}{\log K}$ shards, let those shards form a set S_1 . Let the honest node allocate themselves as $\gamma(\mathbf{2})$

$$\frac{1}{|S_1|} \sum_{i \in S_1} \gamma_i(\mathbf{2}) \leq \frac{1 - \beta}{|S_1|}$$

We will now show that the median value $\gamma(\mathbf{2})$ is less than twice its mean. Let us denote mean by ε and median be denoted by κ . We know that half of the values are greater than κ , thus mean is greater than average calculated by setting the lower half terms as 0 and the greater half terms as κ , i.e. $\varepsilon \geq \frac{1}{|S_1|}[0] + \frac{1}{|S_1|}[|S_1|\kappa/2]$ implying $\varepsilon \geq \frac{1}{2} * \kappa$. Thus median $\leq \frac{2(1-\beta)(\log K)}{K}$. This implies that the last $\frac{K}{2 \log K}$ shards (arranged in descending order of their honest fraction) will have honest fraction less than $\frac{2(1-\beta)(\log K)}{K}$.

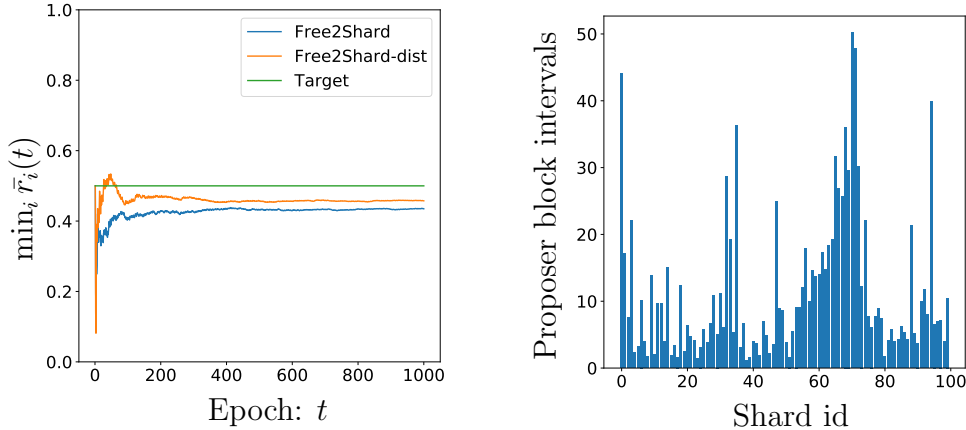
The adversary now spreads only to $\frac{K}{(\log K)^2}$ lowest performing shards, this set is made up of the subset of the $\frac{K}{2 \log K}$ shards discussed above since $\frac{K}{2 \log K} \leq \frac{K}{(\log K)^2}$ (for $K > e^2$).

We can again show that the adversary allocates as $\gamma(\mathbf{3})$ and show that the last $\frac{K}{2^2 \log K}$ shards (arranged in descending order of their honest fraction) will have honest fraction less than $\frac{2(1-\beta)(\log K)}{K}$ and the adversary now spreads only to $\frac{K}{(\log K)^3}$ lowest performing shards, this set is made up of the subset of the $\frac{K}{2^2 \log K}$ shards discussed above since $\frac{K}{2 \log K} \leq \frac{K}{(\log K)^2}$ (for $K > e^2$).

The attack continues till $\frac{K}{(\log K)^\tau} = 1$, the adversary cannot concentrate further. Solving the above equation, we get $\tau = \frac{\log K}{\log \log K}$, completing the proof.

B.3 Extended evaluation

We evaluated a system parameterized simulation of DSA with 36 nodes and 6 shards and a 50% adversary (worst case scenario) in Section 4.6. We perform additional simulations with 1000 nodes and 100 shards to show that DSA achieves the desired results with large number of shards. We plot the time average honest fraction of the worst performing of the 100 shards in Figure B.1a. On comparing with Figure 3.4a, we observe that both `Free2Shard-dist` and `Free2Shard` perform on par with the expectations, achieving a honest fraction of 0.47 and 0.42 respectively (near the desired level of 0.5) fairly quickly. We notice that the mean transaction inclusion delay for the worst performing shard is around 500s which is withing practical limits.



(a) Worst-case average honest fraction (b) Mean transaction inclusion latency

Figure B.1: *Left:* We observe that the chain quality of the worst performing shard improves to the desired level (0.47) around epoch 50 while running `Free2Shard-dist`, whereas `Free2Shard` performs worse and stabilizes at 0.42 at epoch 200. *Right:* Transaction inclusion latency for `Free2Shard-dist` is uniform across shards and with no shard experiencing unreasonable latency

APPENDIX C

ADDITIONAL MATERIAL FOR CHAPTER

4

C.1 Resource minimal **Advocate** -BFT

The **Advocate**-BFT protocol in section 4.3 requires all block references to be posted on the SMR chain. Although this design offers reduced latency, it is heavy in terms of usage on the SMR chain. In this section, we offer an alternative protocol which focuses on minimizing resource usage (Figure C.1). We show that it suffices to only post on the SMR chain the reference to a checkpoint block, thus decreasing resource usage to $O(1)$ from $O(e)$, e being the inter-epoch interval, though at the cost of decreased liveness.

The committee nodes now act as full nodes for the main chain. On receiving a block B_i , at depth e more than the latest checkpoint, the committee nodes post a transaction tC_i containing the hash $H(B_i)$ on the SMR chain. As before, the checkpoint is finalized after Δ_{BFT} , with the following validity rules for block B_i :

- Data availability: The block, as well as its ancestors, should be available.
- Block validity: The block should have valid PoW and the transactions within the block should have valid signatures.
- The block must be at a height e more than the last checkpoint.

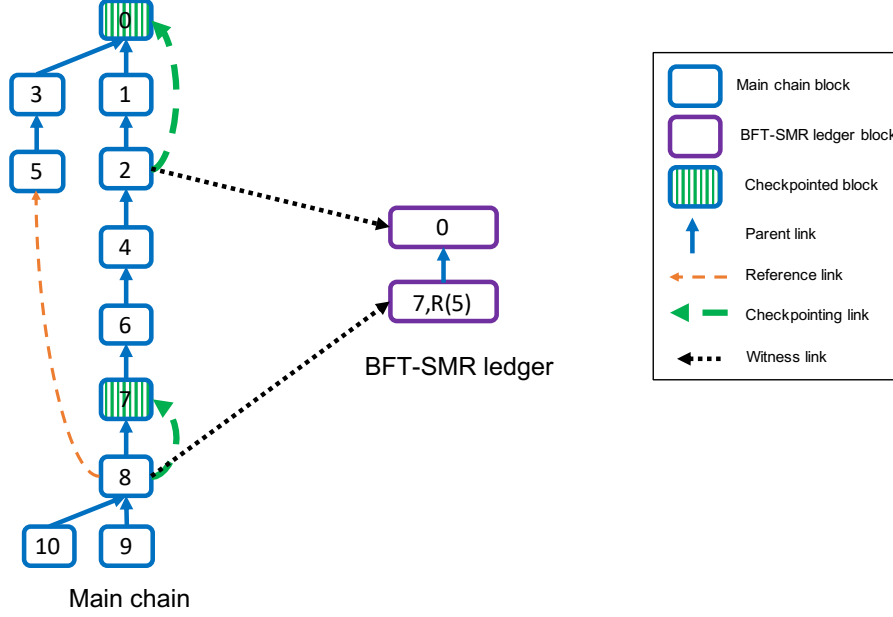


Figure C.1: Resource minimizing Advocate -BFT: SMR chain only maintains blockhash of checkpoint candidates

Main chain behavior Main chain nodes act as light nodes on the SMR chain. The miner of block B_i , at depth e more than the latest checkpoint, includes a reference list R_i of all the leaves of $L(t)$ at the time of mining. The validation rule should be updated, to accept blocks at such a height only if they define such reference list (which can be null). Following, when tC_i is finalized on the SMR chain, the new certificate $C_i = \{tC_i, w_i\}$ should be posted on the main chain, under the c constraint. All other mining and main chain ledger rules remain the same as in Section 4.2.

Latency As before, let τ_m be the time until a transaction is mined by a main chain block B . Following, the transaction is finalized after a checkpoint candidate referencing B is finalized on the SMR chain, i.e., after time τ_{cf} . Therefore, the overall latency is $\tau = \tau_m + \tau_{cf}$. However, observe that the adversary might not reference B in its checkpoint candidates. Therefore, τ_{cf} might span multiple epochs, i.e., until an honest checkpoint candidate is finalized. In this case, the analysis of [72] applies, i.e., an honest checkpoint is guaranteed to be published eventually, although latency increases exponentially with the adversarial mining power.