chainBoost: A Secure Performance Booster for Blockchain-based Resource Markets

Zahra Motaqy
University of Connecticut
raha@uconn.edu

Mohamed E. Najd
University of Connecticut
menajd@uconn.edu

Ghada Almashaqbeh
University of Connecticut
ghada@uconn.edu

***Abstract*—Cryptocurrencies and blockchain technology provide an innovative model for reshaping digital services. Driven by the movement toward Web 3.0, recent systems started to provide distributed services, such as computation outsourcing or file storage, on top of the currency exchange medium. By allowing anyone to join and collect cryptocurrency payments for serving others, these systems create *decentralized markets* for trading digital resources. Yet, there is still a big gap between the promise of these markets and their practical viability. Existing initiatives are still early-stage and have already encountered security and efficiency obstacles. At the same time, existing work around promising ideas, specifically sidechains, fall short in exploiting their full potential in addressing these problems.**

To bridge this gap, we propose chainBoost, a secure performance booster for decentralized resource markets. It expedites service related operations, reduces the blockchain size, and supports flexible service-payment exchange modalities at low overhead. At its core, chainBoost employs a sidechain, that has a (security and semantic) mutual-dependence with the mainchain, to which the system offloads heavy/frequent operations. To enable it, we develop a *novel sidechain architecture* composed of temporary and permanent blocks, a *block suppression* mechanism to prune the sidechain, a *syncing protocol* to permit arbitrary data exchange between the two chains, and an *autorecovery protocol* to support robustness and resilience. We analyze the security of chainBoost, and implement a proof-of-concept prototype for a distributed file storage market as a use case. For a market handling around 2000 transactions per round, our experiments show up to 11x improvement in throughput and 94% reduction in confirmation time. They also show that chainBoost can reduce the main blockchain size by ∼90%, and that it outperforms comparable optimistic rollup solutions by reducing transaction finality by 99.7%.

1. Introduction

Cryptocurrencies and blockchains provide an innovative model that led to new research frontiers in distributed computing and cryptography [29], [60], [38]. Driven by the movement toward the decentralized Internet—Web 3.0, recent cryptocurrency systems started to provide distributed services, such as computation outsourcing, content distribution, or file storage, on top of the currency exchange medium [51], [16]. By allowing anyone to join and collect payments for serving others, these systems create *decentralized markets* for trading digital resources [30].[1]

Blockchain-based resource markets improve early designs of peer-to-peer (P2P) systems by utilizing the properties of blockchains, namely, decentralization, public verifiability, and immutability. They emerged to address the trust, cost, and transparency concerns related to centrally-managed services. For example, in content distribution networks (CDNs), studies [32], [65] showed that up to 88% of the network traffic can be offloaded to P2P-based CDNs during peak demands, which improves performance and reduces cost. Others [10] advocated for building a decentralized content-addressed web that is more robust than the current location-based one due to evidence of lost files (a Harvard-led study found that 48% of all hyperlinks cited in US Supreme Court opinions were broken [97]).

Additionally, blockchain-based resource markets create equitable ecosystems so end users can utilize their excess resources to collect revenue. Moreover, they provide new insights to improve blockchain design. Specifically, they promote the concept of useful mining, in which miners are selected to extend the blockchain based on the amount of service they provide [51], rather than based on wasteful computations as in Bitcoin. These potential benefits encouraged the development of many distributed resource markets in practice, such as Filecoin for file storage [51], Livepeer for video transcoding [16], and Golem for computation outsourcing [14], to name a few. Such systems are viewed as a basic component of Web 3.0 in which centrally-managed services that we currently use are transformed into fully-decentralized ones.

Challenges. Unfortunately, there is still a big gap between the promise of blockchain-based resource markets and their practical viability. Blockchain protocols are complex applied cryptographic protocols, and complexity usually breeds vulnerabilities. Dealing with unauthenticated and untrusted participants, and involving monetary incentives, result in several security issues and attacks [37], [43], [80]. Resource markets are even more involved due to requiring complex modules to allow these parties to trade resources with each other. Not to mention that these markets must meet particular performance requirements, depending on the service type, in order to be a successful replacement to legacy, centrally-managed digital services.

Most existing resource market initiatives are still early-stage and have already stumbled upon several security

1. We focus on open-access decentralized resource markets that employ permissionless blockchains. For brevity, we refer to these as blockchain-based resource markets going forward.

TABLE 1: Comparison of this work (chainBoost), optimistic rollups (Optimism), and ZK rollups (ZkSync Era). Batch period for Optimism and ZkSync are between 30∼60 sec, and 30∼90 sec, respectively [85], [11].

Solution*	Finality	Throughput	Decentralized	Public Verifiability	No Verifier Dilemma	No Trusted Setup	No Smart Contract Dependency
Optimism [33], [6], [19]	7 days	max 11 tx/sec	✖	✖	✖	✔	✖
ZkSync Era [72], [73]	24 hours	$8-25$ tx/sec	✖	✔	✔	✖	✖
chainBoost	$2-30$ min	$194-776$ tx/sec	✔	✔	✔	✔	✔

* For Optimism, we show the maximum observed throughput [19]. Finality means the duration needed to consider state changes induced by rollups or summaries final on the mainchain. In addition to batch processing, in Optimism, this period covers the contestation period; for ZKSync Era, it includes proof generation and verification times; and for chainBoost, it is the time needed for transactions to appear in a meta-block. Lastly, chainBoost numbers are from the experiments in Section 6 for 0.5 and 2 MB sidechain block size and average batch-equivalent duration throughput.

and efficiency obstacles. The countermeasures needed to address these threats, such as resource expenditure proofs [82], [52], dispute solving, or market matching, introduce extra overhead adding to the scalability issues of blockchains. In many occasions, this led to deployments that favor efficiency at the expense of security; system designers may choose to forgo deploying important security countermeasures to optimize performance, leading to exploitable resource markets, e.g., avoid employing service delivery proofs, which allows servers to collect payments for free [8]. Resolving core design issues at a later stage is also problematic as it leads to hard forks.

Limitations of existing solutions. Although several lines of work emerged to address the performance issues of blockchains, none of them are suitable for resource markets and, if used in this context, impose several limitations. Changing the blockchain parameters, such as increasing the block size, is problematic; it does not solve the scalability issues as all the data is still logged on the mainchain, and thus, is often unfavorable by the community (e.g., the case of Bitcoin SV hard fork [3]). Micropayment schemes and payment channels [46], [88] improve throughput of currency transfers rather than service data and events. While other layer-two solutions are either not fully decentralized due to the reliance on trusted verifiers (e.g., Optimisic and ZK Rollups [90], [22]), or slow due to the extra overhead of involving expensive zero-knowledge (ZK) proofs or repetitive computation and contesting process. We summarize the key differences between existing solutions and chainBoost in Table 1.

Others developed service-type specific techniques (e.g., lower overhead proof-of-storage [53]), or outsourced the work to third-party systems (e.g., TrueBit [25]) that are still early-stage and have their own challenges. These directions produced independent solutions that target specific use cases or systems with given semantics. Thus, there is no clear path on how to import them to other resource markets that provide different service types.

In contrast, existing deployment and research efforts around promising ideas, specifically sidechains, fall short in exploiting their full potential [35], [56], [58], [67]. They mostly focus on *currency transfer* known as two-way peg (as we explain in Section 2), many present only high-level ideas, and do not include rigorous security models or analysis. Moreover, *all* existing sidechain work has only dealt with sidechains that are *independent* of the mainchain, which limits the use cases and performance gains that can be achieved. That is, those solutions cannot be trivially generalized to allow arbitrary data exchange and workload sharing due to this independence; each chain has its own transactions, miners, and tokens, making the delegation of operations from the mainchain to a sidechain, and synchronizing the result back, difficult. Although these were posed to be doable, no concrete sidechain protocol with security and performance analysis exists for that.

An open question. Therefore, we ask the following question: *can we build a generic and secure efficiency solution for blockchain-based resource markets that has a unified architecture and interfaces, but allows for service-specific semantics?*

1.1. Our Contributions

We answer this question in the affirmative and propose chainBoost, a secure performance booster for blockchain-based resource markets. At its core, chainBoost employs a sidechain that has a (security and semantic) mutual-dependence with the market's main blockchain, to which all heavy/frequent service-related operations/transactions are offloaded. We make the following contributions.

System design. We introduce a novel sidechain architecture that shares workload processing with the mainchain. It is composed of two types of blocks: *temporary meta-blocks* and *permanent summary-blocks*. The service-related workload in the market is handled by this sidechain, while the rest is handled by the mainchain. Instead of operating like a regular blockchain and storing all transactions, we devise a *summarization and syncing protocol* and a *block suppression* mechanism that the sidechain uses. That is, the sidechain records transactions into meta-blocks that are periodically summarized into smaller summary-blocks. The summary-blocks are used to sync the mainchain by updating the relevant state variables using sync-transactions, allowing for arbitrary data exchange between the two chains. Then, the meta-blocks whose summary is confirmed on the mainchain are pruned from the sidechain. This reduces the size of the sidechain as well as the mainchain as only concise summaries are stored permanently. Furthermore, this syncing process enables the mainchain to be the single truth of the system, thus simplifying system state tracking.

To expedite transaction processing, we employ a practical Byzantine fault tolerance (PBFT) consensus, with dynamic committees elected from the mainchain miners, to run the sidechain. Thus, once a transaction appears in a meta-block it is considered confirmed. chainBoost is agnostic to the mainchain consensus protocol, and it does not assume a particular PBFT consensus or committee election mechanism. Its modular design permits using any secure protocols that realize these functionalities.

All of these design aspects form a unified architecture and interface of chainBoost that can be used with any resource market regardless of the service type it provides. In turn, the summarization process is highly customizable

allowing for expressive rules that can be tailored to suit the underlying service. These also promote flexibility as they support various modalities for service-payment exchange, market matching, etc., that can be configured based on the resource market design goals.

Resilience and robustness techniques. Due to the mutual-dependence relation between the main and side chains, interruptions on either of them may impact the other. To address that, we introduce techniques to achieve resilience and robustness. These include: (1) a mechanism to handle mainchain *rollbacks* so the sidechain can tolerate changes in the recent history of the mainchain, and (2) an *autorecovery protocol* to allow the sidechain to automatically recover from security threats related to malicious or unresponsive committees. The latter introduces the notion of *backup committees* that will step in if any of these threats are detected to restore the valid and secure operation of the sidechain.

Security analysis. We analyze the security of our system showing that the deployment of chainBoost on top of a secure resource market preserves its security in terms of service-related operations, and safety and liveness of the underlying mainchain. This is critical since we deal with dependent sidechains. As part of this analysis, we formally analyze the security of the autorecovery protocol showing bounds for its configuration parameters and the number of backup committees needed to ensure robustness and resilience against threats.

Implementation and evaluation. To assess efficiency, we build a proof-of-concept prototype of our scheme and conduct thorough benchmarks and experiments. In particular, as a use case, we show how chainBoost can substantially improve the performance of a blockchain-based file storage market. Our experiments show that, for a market handling $\sim$2000 transactions per round, chainBoost enhances throughput by $4-11$x and reduces its overall latency by $62-94\%$, depending on the sidechain configuration. Our results also show a reduction in the blockchain size by up to $\sim$90%, and demonstrate support for a variety of server payment modalities at a low cost. We also compare our solution to an optimistic rollup-based solution, and show a $\sim$99.7% reduction in finality.

Lastly, we note that, to the best of our knowledge, our work is the first to deal with dependent sidechains, and the first to show how sidechains can be used for blockchain pruning. Coupled with the performance gains achieved, we expect our work to advance the current state of the art and promote the practical viability of blockchain-based resource markets without compromising their security.

2. Background

We review the paradigm of blockchain-based resource markets and the building blocks used in our work, namely, sidechains and PBFT-based consensus.

Blockchain-based resource markets. Ethereum is among the earliest systems to provide a distributed service on top of the currency medium. It allows clients to deploy smart contracts and pay miners for the CPU cycles they spend when executing these contracts. Later models [51], [16], [14] expanded the network to involve service-specific participant roles, e.g., servers. Instead of having all miners repeat the same task, as in Ethereum, servers are matched with clients to fulfill their service requests.

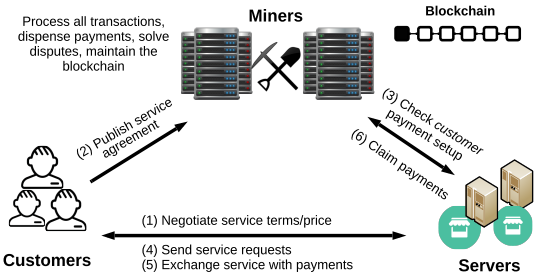


Figure 1: A generic resource market model.

A blockchain-based resource market consists of several modules (Figure 1): order matching and service term agreement, service-payment exchange, payment processing, and dispute resolution. After negotiating and committing to service terms, clients send requests to servers and pay in return. Service delivery proofs are needed to defend against colluding attackers who may collect payments without doing any work [82], [52]. Furthermore, due to the impossibility of fair exchange between untrusted parties [49], additional countermeasures are required for service-payment exchange. For example, clients create escrows for the expected total amount, and miners dispense payments to the servers—out of the escrows—in return for valid service delivery proofs.

Other solutions may divide the service into small amounts and does the same with payments (known as micropayments [88]). This reduces financial risks: if a malicious server does not deliver a service, the client loses a small payment; and if a malicious client decides not to pay, the server loses payment for a small service amount. It also promotes flexibility as clients can stop the service at anytime, e.g., as in online gaming or pay-per-minute video streaming. Lastly, solving disputes involves processing cheating claims against participants and penalizing them.

Sidechains. A sidechain is a secondary blockchain tied to a mainchain. This term was first coined by Back et al. [35], who also defined the main use cases of sidechains: supporting interoperability, enhancing scalability, and extending mainchain capability. That is, having a sidechain enables mainchain users to interact with the sidechain, allows for workload sharing, and provides an environment where new functionalities can be deployed/tested.

Despite the large body of work around sidechains (see Section 7), and that in theory the above use cases are viable, most existing research and deployment efforts focus only on currency exchange, known as two-way peg. That is, the whole goal is allowing clients to exchange two different cryptocurrency tokens between the two chains (call them token A and token B). Usually, the mainchain miners are not aware of the sidechain, and the sidechain operation involves succinct proofs proving that some amount of token A has been locked on the mainchain, so that an equivalent amount of token B can be used on the sidechain. Two-way peg is an important operation that may suffice for mainchain interactions with an independent sidechain, the only sidechain type that existing solutions consider. This is exemplified in a firewall security property [58] stating that if a sidechain gets compromised it will not impact the mainchain.

These existing models and notions do not suffice for our purposes; we require a sidechain that has a mutual-dependence relation with the mainchain allowing for arbitrary data exchange, not only currency, thus exploiting the full potential of sidechains. Again, although this is possible in theory, none of the existing research works or practical deployments demonstrate protocols for that, or how this can be used with resource markets.

PBFT-based consensus. To increase throughput and reduce confirmation delays, several consensus protocols utilize practical Byzantine fault tolerant (PBFT) [42]. We distinguish two flavors: leader-based consensus, e.g., [68], and voting-based consensus, e.g., [59]. In both cases, the assumption is that an attacker can corrupt less than one-third of the committee members running the agreement. Moreover, network operation is divided into epochs (an epoch is k consecutive rounds and a round is the time period during which a new block is mined).

In leader-based consensus, the committee leader proposes a block in a round and collects signatures from the committee members indicating their agreement. In voting-based consensus, the committee is divided into block proposers and voters, where each proposer proposes a block that voters vote on (by signing). The block with vote majority will be added to the blockchain. Having a committee, where its size is much smaller than the whole miner population, reduces communication cost and fork probability, and speeds up agreement. Due to these advantages, we adopt a PBFT-based consensus to run the sidechain in chainBoost.

Committee election is a crucial component. Obviously, having a static committee is insecure since it is vulnerable to targeted attacks. Thus, the dynamic approach is used in practice: in each epoch a new committee is elected. Since we deal with open-access blockchains, election relies on Sybil-resistant identities. For example, in proof-of-work blockchains, the probability of electing a miner is proportional to the computing power this miner owns (as in ByzCoin [68]). To address targeted attacks resulting from knowing the committee members in advance, cryptographic sortition [59] in the voting-based model has been introduced. A party reveals that it was elected after proposing a block or voting (with a proof of being elected), rendering targeted attacks ineffective. Various committee election mechanisms with different trade-offs have been studied in the literature [68], [89], [59], [66].

chainBoost's design is agnostic to the consensus type of the underlying mainchain. Also, for its sidechain, chainBoost does not assume a particular PBFT-based consensus or committee election algorithm. Any secure protocols that realize these functionalities can be used. To simplify the presentation, we use the leader-based approach when introducing our design.

3. Preliminaries

Notation. We use λ for the security parameter, and pp for the system public parameters. Each participant maintains a secret and public key, sk and pk, respectively. We use $\mathcal{L}$ to denote a ledger (or blockchain), $\mathcal{L}_{\mathsf{mc}}$, and $\mathcal{L}_{\mathsf{sc}}$ to denote the mainchain and sidechain ledgers, respectively. We use PPT as a shorthand for probabilistic polynomial time.

System model. chainBoost is designed to complement and enhance open-access distributed resource markets. Anyone can join, or leave, at any time, and these parties are known using their respective public keys. The resource market mainchain may run any consensus protocol and mining process, and miners establish Sybil-resistant identities based on their mining power. chainBoost operates in rounds and epochs (as defined earlier).

A blockchain-based resource market is run by a set of clients $\mathcal{C}$, servers $\mathcal{S}$, and miners $\mathcal{M}$.[2] It maintains a ledger $\mathcal{L}$ and provides the following functionalities:

$\mathsf{SystemSetup}(1^\lambda) \rightarrow (\mathsf{pp}, \mathcal{L}_0)$**:** Takes as input the security parameter λ, and outputs the system public parameters pp and an initial ledger state $\mathcal{L}_0$ (which is the genesis block).[3]

$\mathsf{PartySetup}(\mathsf{pp}) \rightarrow (\mathsf{state})$**:** Takes as input the public parameters pp, and outputs the initial state of the party state (which contains a keypair $(\mathsf{sk}, \mathsf{pk})$, and in case of miners, the current view of the ledger $\mathcal{L}$).

$\mathsf{CreateTx}(\mathsf{txtype}, \mathsf{aux}) \rightarrow (\mathsf{tx})$**:** Takes as input the transaction type txtype and any additional information/inputs aux, and outputs a transaction tx of one of the following types:

- $\mathsf{tx}_{\mathsf{ask}}$: Allows a client to state its service needs.
- $\mathsf{tx}_{\mathsf{offer}}$: Allows a server to state its service offering.
- $\mathsf{tx}_{\mathsf{agreement}}$: The service agreement between a client and a server (or a set of servers).
- $\mathsf{tx}_{\mathsf{serviceProof}}$: A service delivery proof submitted by a server.
- $\mathsf{tx}_{\mathsf{servicePayment}}$: Payment compensation from a client to a server for the service provided.
- $\mathsf{tx}_{\mathsf{dispute}}$: Initiates a dispute-solving process for a particular service misbehavior incident.
- $\mathsf{tx}_{\mathsf{transfer}}$: Currency transfer between participants.

$\mathsf{VerifyTx}(\mathsf{tx}) \rightarrow (0/1)$**:** Takes as input a transaction tx, and outputs 1 if tx is valid based on the syntax/semantics of its type, and 0 otherwise.

$\mathsf{VerifyBlock}(\mathcal{L}_{\mathsf{mc}}, \mathsf{B}) \rightarrow (0/1)$**:** Takes as input the current ledger state $\mathcal{L}_{\mathsf{mc}}$ and a new block B, and outputs 1 if B is valid based on the syntax/semantics of blocks, and 0 otherwise.

$\mathsf{UpdateState}(\mathcal{L}_{\mathsf{mc}}, \{\mathsf{tx}_i\}) \rightarrow (\mathcal{L}'_{\mathsf{mc}})$**:** Takes as input the current ledger state $\mathcal{L}_{\mathsf{mc}}$, and a set of pending transactions $\{\mathsf{tx}_i\}$. It reflects the changes induced by these transactions and outputs a new ledger state $\mathcal{L}'_{\mathsf{mc}}$.

Note that resource markets may offer non-service-related transactions other than $\mathsf{tx}_{\mathsf{transfer}}$. The notion above can be extended to cover these other types. Also, the set of transactions needed to operate the service can be extended; the ones in the notion above are intended to be a base set. Lastly, UpdateState is the process of mining a new block to reflect the changes induced by the executed set of transactions. Agreeing on this block and how it is mined is governed by the consensus protocol operating the mainchain of the resource market.

As mentioned before, chainBoost operates a sidechain managed by a committee elected from the mainchain miners who runs a PBFT-based consensus to mine new blocks. Furthermore, the sidechain contains two types of blocks:

2. In some models, servers play the role of miners and their mining power is represented by the amount of service they provide in the system.
3. For the rest of the algorithms, the input pp is implicit.

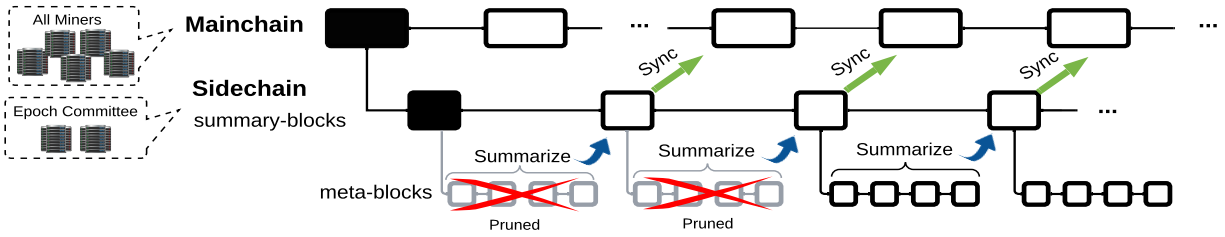


Figure 2: chainBoost diagram.

meta-blocks that record transactions, and summary-blocks that summarize meta-blocks mined in an epoch. Also, this committee issues a sync-transaction to sync the mainchain. We abstract these functionalities as follows.

$\mathsf{Setup}(\mathcal{L}^0_{\mathsf{mc}}) \to (\mathcal{L}^0_{\mathsf{sc}}, \mathcal{L}'_{\mathsf{mc}})$**:** Takes as input the mainchain genesis block $\mathcal{L}^0_{\mathsf{mc}}$. It outputs the initial sidechain ledger state $\mathcal{L}^0_{\mathsf{sc}}$ (which is the genesis block of the sidechain containing the sidechain public parameters pp' that include the traffic classification and summary rules). Also, it creates the mainchain state variables that will be synced by the sidechain, thus producing a new mainchain ledger state $\mathcal{L}'_{\mathsf{mc}}$.

$\mathsf{Elect}(\mathcal{L}_{\mathsf{mc}}) \to (\{C_i\}, \{\mathsf{leader}_i\})$**:** Takes as input the current state of the mainchain ledger $\mathcal{L}_{\mathsf{mc}}$, and outputs a set of committees $\{C_i\}$ and their leaders $\{\mathsf{leader}_i\}$.

$\mathsf{CreateSyncTx}(\mathsf{aux}) \to (\mathsf{tx}_{\mathsf{sync}})$**:** Takes as input information aux, and outputs a sync-transaction $\mathsf{tx}_{\mathsf{sync}}$.

$\mathsf{VerifySyncTx}(\mathcal{L}_{\mathsf{sc}}, \mathsf{tx}_{\mathsf{sync}}) \to (0/1)$**:** Takes as input the current sidechain ledger state $\mathcal{L}_{\mathsf{sc}}$ and sync-transaction $\mathsf{tx}_{\mathsf{sync}}$, and outputs 1 if $\mathsf{tx}_{\mathsf{sync}}$ is valid based on its syntax/semantics, and 0 otherwise.

$\mathsf{VerifyBlock}(\mathcal{L}_{\mathsf{sc}}, \mathsf{B}_{\mathsf{btype}}) \to (0/1)$**:** Takes as input the current sidechain ledger state $\mathcal{L}_{\mathsf{sc}}$, a new block B with type $\mathsf{btype} = \mathsf{meta}$ or $\mathsf{btype} = \mathsf{summary}$. It outputs 1 if B is valid based on the syntax/semantics of the particular block type, and 0 otherwise.

$\mathsf{UpdateState}(\mathcal{L}_{\mathsf{sc}}, \mathsf{aux}, \mathsf{btype}) \to (\mathcal{L}'_{\mathsf{sc}})$**:** Takes as input the current sidechain ledger state $\mathcal{L}_{\mathsf{sc}}$, and a set of pending transactions $\mathsf{aux} = \{\mathsf{tx}_i\}$ (if $\mathsf{btype} = \mathsf{meta}$) or $\perp$ (if $\mathsf{btype} = \mathsf{summary}$ since the inputs are meta-blocks from $\mathcal{L}_{\mathsf{sc}}$). It reflects the changes induced by aux and outputs a new ledger state $\mathcal{L}'_{\mathsf{sc}}$.

$\mathsf{Prune}(\mathcal{L}_{\mathsf{sc}}) \to (\mathcal{L}'_{\mathsf{sc}})$**:** Takes as input the current sidechain ledger state $\mathcal{L}_{\mathsf{sc}}$, and produces an updated state $\mathcal{L}'_{\mathsf{sc}}$ in which all stale meta-blocks are dropped.

Note that Elect may produce only one committee, the primary one that runs the sidechain, or a list of primary and backup committees as in chainBoost's design.

Security model. Our goal is to develop a secure efficiency solution—one that preserves the security of the underlying resource market (both its blockchain or ledger and the distributed service it provides). So, starting with a secure blockchain-based resource market, the security properties of this system must be preserved by chainBoost.

Ledger security. A ledger $\mathcal{L}$ is secure if it satisfies the following properties [55] (the confirmed state of $\mathcal{L}$ includes all blocks buried under at least k blocks, where k is the depth parameter):

- Safety: For any two time rounds t_1 and t_2 such that $t_1 \leq t_2$, and any two honest parties P_1 and P_2, the confirmed state of $\mathcal{L}$ maintained by P_1 at t_1 is a prefix of the confirmed state of $\mathcal{L}$ maintained by party P_2 at time t_2 with overwhelming probability.
- Liveness: If a transaction tx is broadcast at time round t, then with overwhelming probability tx will be recorded on within the confirmed state of $\mathcal{L}$ at time at most $t + u$, where u is the liveness parameter.

Based on [87], safety (or persistence) covers the consistency and future self-consistency properties, and liveness covers chain growth and quality properties.[4] A ledger protocol is parameterized by predicates to verify transaction and chain validity, ensuring that $\mathcal{L}$ records only valid transactions/blocks. For us, this also covers the validity of the additional operations/transactions introduced by the resource market so that only valid ones are accepted.

On the security of distributed resource markets. This has been thoroughly studied in [31], where that work defined threat categories related to the service and the blockchain/cryptocurrency itself. The former mainly includes denial of service, service corruption, service slacking (servers collect payments without all promised service work), and service theft (clients obtain service without paying the servers the amount they agreed on). While the latter covers violating the security properties of the blockchain defined above. A secure resource market means one that protects against all these threats.

The goal of chainBoost is to optimize the performance of a given resource market, in terms of throughput, confirmation delays, and blockchain size, using a dependent sidechain. The way the offloaded workload to this sidechain is processed is identical to the logic used by the resource market. Thus, in our security analysis, we show that chainBoost and all the techniques it introduces preserve the security of the underlying resource market.

Adversary model. chainBoost runs on top of a secure resource market that operates a secure blockchain as defined previously. We distinguish three miner behaviors; honest who follow the protocol as prescribed, malicious (controlled by an adversary) who may deviate arbitrarily, and honest-but-lazy who collaborates passively with the adversary by accepting records without validation or going unresponsive during consensus.[5] The adversary can introduce new nodes or corrupt existing ones, without going above the threshold of faulty nodes of the consensus protocol. The adversary can see all messages and transactions sent in the system (since we deal with public open-

4. Chain quality is defined as follows: in any sufficiently long list of blocks on $\mathcal{L}$, at least t of these blocks are mined by honest miners where t is a chain quality parameter.

5. We consider lazy miners on the sidechain to account for scenarios where miners are well-motivated to work honestly on the mainchain. But for the sidechain, with its heavy load, they might choose to be lazy in order to use all their resources for providing service or for the mainchain mining to collect more payments/rewards, i.e., the verifier dilemma [79].

access blockchains) and can decide their strategy based on that. This adversary can reorder messages and delay them; we assume bounded-delay message delivery, so any sent message (or transaction) will be delivered within Δ time as in [59], [87], [69]. We assume slowly-adaptive adversaries [34] that can corrupt parties (specifically miners) only at the epoch beginning. Lastly, we deal with PPT adversaries who cannot break the security of the used cryptographic primitives with non-negligible probability.

4. chainBoost Design

In this section, we present chainBoost starting with an overview of its design, followed by an elaboration on the technical details, limitations, and security.

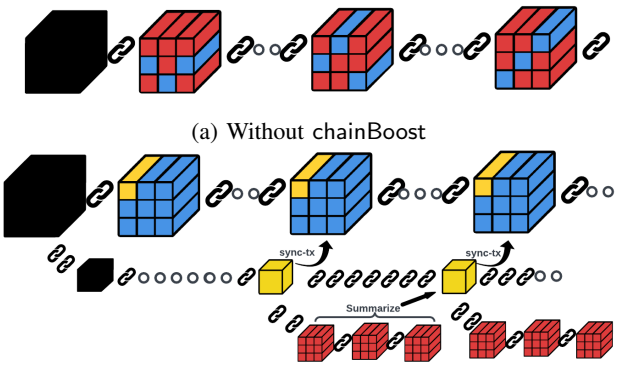
4.1. Overview

chainBoost introduces a new sidechain architecture and a set of techniques to securely boost the performance of blockchain-based resource markets. This sidechain has a mutual-dependence relationship with the mainchain as they share the market workload. In particular, all heavy/frequent service-related operations/transactions are processed by the sidechain, while only brief summaries of the resulting state changes are logged on the mainchain.

As shown in Figure 2 (and Figure 3 shows a detailed view chainBoost's impact on processing and storage), the sidechain works in parallel to the mainchain and shares its workload processing and storage. The transactions in the system are classified into sidechain and mainchain transactions. All service-related operations that can be summarized (such as service delivery proofs, service payments, and dispute-solving-related transactions) will go to the sidechain, while the rest will stay on the mainchain. Thus, in the setup phase, system designers determine this classification and the summarization rules (in the next section, we provide guidelines based on the generic resource market paradigm presented earlier). The sidechain is launched with respect to the mainchain genesis block. Both chains operate in epochs and rounds, where the epoch's length determines the frequency of the summaries.

During each epoch, a committee from the mainchain miners is elected to manage the sidechain. The rest of the miners do not track the traffic processed by the sidechain, thus reducing their workload. The sidechain is composed of *two types of blocks*: temporary meta-blocks and permanent summary-blocks. For each sidechain round, the committee runs a PBFT-based consensus to produce a meta-block containing the transactions they processed. In the last round of the epoch, this committee produces a summary-block summarizing all state changes imposed by the meta-blocks within that epoch.

To maintain a single truth of the market, represented by its mainchain, chainBoost introduces a periodic *syncing process*. Once a summary-block is published, the committee issues a sync-transaction including the summarized state changes. The mainchain miners process this transaction by updating the relevant state variables on the mainchain. To reduce the storage footprint of the sidechain, and subsequently the mainchain size, chainBoost introduces a *pruning mechanism*; when the sync-transaction



(a) Without chainBoost

(b) With chainBoost

Figure 3: Mainchain with and without chainBoost (service transactions are in red, others are in blue. Summary-blocks and sync-transactions are in yellow).

is confirmed on the mainchain, all meta-blocks used to produce the respective summary-block are discarded.

The syncing and pruning processes have positive impacts on market operation. Maintaining a single source of truth (i.e., the mainchain) simplifies tracking the system state. At the same time, having permanent summary-blocks supports public verifiability as anyone can verify the source of the state changes on the mainchain using the summaries and any other data (that might be published in full in the summary-blocks). Moreover, the pruning process reduces the storage footprint, thus allowing for faster bootstrapping of new miners.

Mutually dependent chains are challenging; if one fails so will the other. Also, network propagation delays and connectivity issues may lead to temporary forks. A question that arises here is how to securely handle rollback situations in chainBoost—when a sidechain has already synced with the mainchain but subsequently the recent mainchain blocks are rolled back (i.e., they belong to an abandoned branch of the mainchain). Another question is related to how active or passive attacks, or interruptions, on the sidechain may impact the mainchain. chainBoost introduces a *mass syncing* mechanism and an *autorecovery protocol* to handle rollbacks, enable a sidechain to recover after periods of interruption, and allow the mainchain to tolerate these interruptions. We provide more details in Section 4.2.3.

As noted, the design above is generic and not tied to a particular resource market type. It resembles a unified architecture and interface between the main and side chains that any resource market can use. Capturing service-specific semantics is accounted for in the setup of the traffic classification and summary rules. That is, based on full knowledge of the market configuration and operation, service-related transactions are specified and will be handled by the sidechain. The same applies to how this traffic should be summarized; these will generally follow a generic paradigm (that we discuss later) that can be adapted to fit the underlying market. Furthermore, chainBoost targets markets with substantial service-related traffic (we show that performance gains are minimal in case most traffic is non-service related and must reside on the mainchain). Moreover, the sidechain

configuration, in terms of round and epoch duration, and block sizes, impact the achieved performance gains.

Concrete examples. As a concrete example of the benefit of chainBoost, we look at a file storage network (or any other resource market that involves periodic service delivery proofs). A server periodically engages in a proof-of-storage protocol to prove it stores a client's file. If the proof is valid, the client pays the server for the service provided. Those proof-of-storage transactions are usually plentiful, which slows down the processing of other transactions in the system and consumes space on the blockchain. Using chainBoost, the proofs are processed by the sidechain. When generating a summary-block, the committee tallies the proofs generated by each server, and then the mainchain is synced. Thus, the mainchain maintains only a state table of proof counts for active servers. Finally, meta-blocks containing old proof-of-storage transactions whose summary got confirmed on the mainchain are pruned. This greatly reduces the chain size and speeds up transaction confirmation.

Another way chainBoost can be beneficial is for the market matching module. In any resource market, clients and servers publish their service demands (i.e., asks) and offers, respectively, and get matched after negotiating the service terms. With chainBoost, those demands and offers are recorded on the sidechain. The summary-blocks will only contain finalized service agreements, that get synced back up with the mainchain. This is a viable way to move the service matching from an out-of-band process into a component of the market, enhancing accountability, trust, and transparency. This is done without increasing the burden on the mainchain.

4.2. Technical Details

Now we delve into the technical aspects of chainBoost's setup phase, operation, architecture, and robustness.

4.2.1. Setup Phase. Introducing chainBoost changes the way the underlying resource market behaves. Transactions are separated into two categories based on their home chain. Also, a subset of the mainchain miners will be responsible of managing the sidechain on a rotating basis.

Configuration, traffic classification, and summary rules. chainBoost builds a unified architecture and interface that can be used for any resource market. At the same time, it permits tailored customization that suits the service type the market provides, which are set during the setup phase (Figure 4). chainBoost is most beneficial for resource markets with substantial service-related traffic. For configuring the public parameters pp' of the sidechain, larger sidechain block sizes, with short round duration, would be more effective. However, this size should not be larger that typical recommended sizes in practice (e.g., up to 2 MB) and the duration must be enough for the sidechain committee to agree on a block.

chainBoost provides a general framework to distribute the transactions between the mainchain and the sidechain, ensuring effective load sharing. That is, service-related transactions go into the sidechain. They are usually frequent and can be summarized—in terms of the abstract notion we defined these include $\mathsf{tx}_{\mathsf{ask}}$, $\mathsf{tx}_{\mathsf{offer}}$, $\mathsf{tx}_{\mathsf{agreement}}$, $\mathsf{tx}_{\mathsf{servicePayment}}$, $\mathsf{tx}_{\mathsf{serviceProof}}$, and $\mathsf{tx}_{\mathsf{dispute}}$. The mainchain processes and records non-service related transactions, e.g. minting new coins and currency transfers, or those that cannot be summarized, e.g., escrow creation. chainBoost also provides a flexible framework for summarizing service transactions. For example, service delivery proofs $\mathsf{tx}_{\mathsf{serviceProof}}$ can be summarized by simply counting them, and this is all that the mainchain needs to track, and the same applies for the accumulated service payments $\mathsf{tx}_{\mathsf{servicePayment}}$. For solving disputes, the proof-of-cheating transaction and the dispute outcome—validity and the imposed punishment (if any)—resemble the incident summary. We formalize these summary rules in a generic way (based on our abstract model) in Figure 5. Those rules are not fixed; they can be adapted/extended to fit the underlying resource market specifications.

$\mathsf{Setup}(\mathcal{L}^0_{\mathsf{mc}})$: Takes as input the mainchain genesis block (which includes the public parameters pp). Generates the sidechain configuration parameters:

- The epoch length ω.
- All public parameters pp' needed by the sidechain protocol.
- Traffic classification.
- Summary rules.
- Summary state variables for the mainchain $\mathsf{summary}_{\mathsf{variables}}$.

Updates $\mathcal{L}_{\mathsf{mc}}$ to produce a new state $\mathcal{L}'_{\mathsf{mc}}$ to include pp' and $\mathsf{summary}_{\mathsf{variables}}$.

Initializes:

- $\mathsf{pendingTx}_{\mathsf{mc}}$, pending transaction pool to keep track of pending mainchain traffic.
- $\mathsf{pendingTx}_{\mathsf{sc}}$, pending transaction pool to keep track of pending sidechain traffic.

Outputs: epoch length ω, sidechain genesis block $\mathcal{L}^0_{\mathsf{sc}}$ (that references $\mathcal{L}^0_{\mathsf{mc}}$), and $\mathcal{L}'_{\mathsf{mc}}$.

Figure 4: System setup.

We propose a simple approach that relies on static classification and annotation. After deciding the workload split, chainBoost adds an extra field to the transaction header indicating its type, like 00 for mainchain transactions, and 11 for sidechain transactions. This annotation is set in the network protocol and examined by the miners to determine the home chain of each transaction they receive. The summarization rules are encoded into the sidechain protocol, to be used by the committee to produce succinct state changes of the traffic they process. Those rules can be changed at later stages as long as the interpretation of summaries does not change, allowing for greater flexibility while preserving compatibility. For example, a different proof of service delivery construction can be used while the summary remains as the proof count.

Tying that to our abstract model, the transactions $\mathsf{tx}_{\mathsf{ask}}$, $\mathsf{tx}_{\mathsf{offer}}$, $\mathsf{tx}_{\mathsf{agreement}}$, $\mathsf{tx}_{\mathsf{serviceProof}}$, $\mathsf{tx}_{\mathsf{servicePayment}}$, and $\mathsf{tx}_{\mathsf{dispute}}$ will all be annotated with 11 (i.e., sidechain traffic), while the rest will be annotated with 00 as they should reside on the mainchain. Also, the mainchain now processes a

Input: meta-blocks $\mathsf{B}^1_{\mathsf{meta}}, \ldots, \mathsf{B}^n_{\mathsf{meta}}$ from an epoch.
Initialize empty summary structures $\mathsf{sum}_{\mathsf{serviceProof}}$, $\mathsf{sum}_{\mathsf{servicePayment}}$, $\mathsf{sum}_{\mathsf{dispute}}$, and $\mathsf{sum}_{\mathsf{agreement}}$.
for $i \in \{1, \ldots, n\}$ and every $\mathsf{tx} \in \mathsf{B}^i_{\mathsf{meta}}$ **do**
  if $\mathsf{tx.txtype} = \mathsf{tx}_{\mathsf{serviceProof}}$ **then**
    // cid is the service contract ID
    ++ $\mathsf{sum}_{\mathsf{serviceProof}}[\mathsf{tx.cid}]$
  elseif $\mathsf{tx.txtype} = \mathsf{tx}_{\mathsf{servicePayment}}$ **then**
    $\mathsf{sum}_{\mathsf{servicePayment}}[\mathsf{tx.cid}]$ += tx.amount
  elseif $\mathsf{tx.txtype} = \mathsf{tx}_{\mathsf{dispute}}$ **then**
    $\mathsf{sum}_{\mathsf{dispute}}[\mathsf{tx.cid}] = (\mathsf{tx.proof}, \mathsf{tx.outcome})$
  // $\mathsf{tx}_{\mathsf{agreement}}$ references $\mathsf{tx}_{\mathsf{ask}}$ and $\mathsf{tx}_{\mathsf{offer}}$, so the
  // summary covers these as well (s is the server
  // and cl is the client)
  elseif $\mathsf{tx.txtype} = \mathsf{tx}_{\mathsf{agreement}}$ **then**
    $\mathsf{sum}_{\mathsf{agreement}}[\mathsf{tx.cid}] = (\mathsf{tx.s}, \mathsf{tx.cl}, \mathsf{tx.terms})$
Output $\mathsf{sum}_{\mathsf{serviceProof}}$, $\mathsf{sum}_{\mathsf{servicePayment}}$, $\mathsf{sum}_{\mathsf{dispute}}$, and $\mathsf{sum}_{\mathsf{agreement}}$

Figure 5: Summary rules (assuming one server per a service contract. If it is a set of servers instead, then indexing should indicate the particular server using tx.cid.s).

new transaction type, namely, the sync-transaction $\mathsf{tx}_{\mathsf{sync}}$. The outcome of the sidechain setup phase, as shown in Figure 4, includes an updated state of the mainchain ledger to record the state variables used for the summary and all sidechain configuration parameters, in addition to the epoch length ω. Furthermore, as the sidechain is tied to the mainchain, its genesis block will reference the mainchain's genesis block.

Managing the sidechain. Having all miners participate in consensus on both chains is counterproductive. Furthermore, as the goal is to speed up service delivery by reducing transaction confirmation time, the sidechain requires a fast consensus protocol. Therefore, we let the sidechain be managed by a committee elected from the mainchain miners on a rotating basis, and this committee runs a PBFT-based consensus. Although we favor voting-based PBFT to mitigate targeted attacks, to simplify the discussion, we use the leader-based one. Nonetheless, chainBoost can be used with either type.

At the onset of each epoch, a new committee is elected to manage the sidechain, which distributes the sidechain workload handling among all miners (i.e., each miner will participate in this task when elected). Any secure committee election can be used, e.g., [59] (as we discuss in Section 2). To enable fast handover between committees, all miners in the network maintain a copy of the sidechain.

4.2.2. Architecture and Operation. When integrated with a resource market, chainBoost must preserve the security (safety and liveness) of the market's blockchain and the security/validity of its decentralized service. That is, the system state must be valid, publicly verifiable and available, immutable, and growing over time. We devise a new architecture that allows chainBoost to preserve those properties while reducing the overhead of service-related transactions. The operation of chainBoost's sidechain is depicted in Figure 6.

// All miners inspect incoming transactions annotation
if $\mathsf{Annot}(\mathsf{tx}) = 00$ **then**
  if $\mathsf{VerifyTx}(\mathsf{tx})$ **then**
    Add tx to $\mathsf{pendingTx}_{\mathsf{mc}}$
// Done only by sidechain miners
elseif $\mathsf{VerifyTx}(\mathsf{tx})$ **then**
  Add tx to $\mathsf{pendingTx}_{\mathsf{sc}}$

// Sidechain miners
// Check if it is the last round in the epoch
if lastRound **then**
  $\mathcal{L}'_{\mathsf{sc}} = \mathsf{UpdateState}(\mathcal{L}_{\mathsf{sc}}, \perp, \mathsf{summary})$
  $\mathsf{B}_{\mathsf{summary}} = \mathcal{L}'_{\mathsf{sc}}.\mathsf{head}$
  $\mathsf{tx}_{\mathsf{sync}} = \mathsf{CreateSyncTx}(\mathsf{B}_{\mathsf{summary}})$
else
  $\mathcal{L}'_{\mathsf{sc}} = \mathsf{UpdateState}(\mathcal{L}_{\mathsf{sc}}, \mathsf{pendingTx}_{\mathsf{sc}}, \mathsf{meta})$

// Mainchain miners
if $\mathsf{VerifySyncTx}(\mathcal{L}_{\mathsf{sc}}, \mathsf{tx}_{\mathsf{sync}})$ **then**
  Add $\mathsf{tx}_{\mathsf{sync}}$ to the head of $\mathsf{pendingTx}_{\mathsf{mc}}$

// When mining the next mainchain block, update
// $\mathsf{variables}_{\mathsf{summary}}$ based on $\mathsf{tx}_{\mathsf{sync}}$
$\mathcal{L}'_{\mathsf{mc}} = \mathsf{UpdateState}(\mathcal{L}_{\mathsf{mc}}, \mathsf{pendingTx}_{\mathsf{mc}})$

// When an epoch ends, elect new committees
$(\{C_i\}, \{\mathsf{leader}_i\}) \leftarrow \mathsf{Elect}(\mathcal{L}_{\mathsf{mc}})$

// When $\mathsf{tx}_{\mathsf{sync}}$ is confirmed on the mainchain,
// sidechain miners will prune the sidechain
$\mathcal{L}'_{\mathsf{sc}} = \mathsf{Prune}(\mathcal{L}_{\mathsf{sc}})$

Figure 6: System operation.

Sidechain blocks and growth. chainBoost's sidechain is composed of temporary meta-blocks and permanent summary-blocks. During each round in an epoch, the committee leader proposes a meta-block, containing sidechain transactions processed in that round, and initiates an agreement to collect votes from the committee. Once enough votes are collected, the block is confirmed and is added to the sidechain. Validating a meta-block proceeds as in PBFT-based consensus; it encompasses verifying block correctness (based on the same verification rules used by the underlying resource market to process service-related transactions), and that valid votes by the legitimate members are provided.

At the end of the epoch, the leader proposes a summary-block summarizing all meta-blocks in the epoch (based on the summary rules set during the setup phase). Similarly, a summary-block needs the committee agreement to be published on the sidechain. Also, validating this block involves validating the votes and verifying that the summaries are correct based on the summary rules and the content of the corresponding epoch meta-blocks.

Since chainBoost's sidechain has two block types, the way blocks refer to their predecessors, to preserve immutability, is different from regular blockchains. As shown in Figure 2, the sidechain genesis block references the mainchain genesis block, the first meta-block in an epoch references the summary-block of the previous

epoch, while any other meta-block references its predecessor meta-block in that epoch. Finally, a new summary-block references the last epoch's summary-block.

The syncing process. The sidechain committee uses the summary-blocks to sync the mainchain without requiring the mainchain miners to re-execute the offloaded operations. Every time a summary-block is published, the leader issues a sync-transaction $\mathsf{tx}_{\mathsf{sync}}$ containing the state changes. Mainchain miners will add this transaction to the head of the pending (mainchain) transaction pool, and will be used to update $\mathsf{variables}_{\mathsf{summary}}$ on the mainchain. This transaction will be accepted if it is valid based on the referenced summary-block (recall that all miners have copies of both the mainchain and the sidechain).

Pruning stale blocks. We introduce a *block suppression* technique to prune the sidechain without compromising its immutability. chainBoost can safely drop the meta-blocks once their corresponding $\mathsf{tx}_{\mathsf{sync}}$ is confirmed on the mainchain. This is done to ensure robust transition between epochs; when $\mathsf{tx}_{\mathsf{sync}}$ is published in a block buried under at least k blocks on the mainchain, the corresponding meta-blocks can be dropped (so they are not pruned within one epoch period). Given that the summary-blocks are permanent, and that they represent valid summaries produced by a committee with an honest majority (we derive a lower bound for the committee size to guarantee that in Appendix D), public verifiability of the mainchain is not impacted. Anyone can refer to the summary-blocks to verify relevant mainchain state changes.

4.2.3. Robustness and Resilience. chainBoost builds a sidechain that has a mutual-dependence relation with the mainchain. Thus, the security and validity of both chains are tied to each other. To maintain system operation, interruptions on any of these chains must be tolerated by the other. This encompasses handling: (1) block rollbacks on the mainchain, and (2) sidechain downtime from malicious attackers or even lazy honest parties. We devise mechanisms to address these cases.

Handling rollbacks. Rollbacks happen when mainchain miners switch to a longer branch, causing the most recent blocks to become obsolete.[6] This impacts chainBoost's sidechain operation on two fronts: first, the committee election if the used mechanism relies on the view of the mainchain, and second, the syncing process if the abandoned blocks contain sync-transactions.

For the first case, we restrict the view used for committee election to the confirmed blocks on the mainchain. For example, for the sliding window-based approach [68], the committee is composed of the miners of the last y confirmed blocks, and for stake-based election [59], the amount of stake owned by each miner is computed based on the confirmed mainchain history. By doing so, even if a rollback happens, it will not change the sidechain committees elected during the rollback period.

For the second case, we introduce a *mass-syncing* technique where a $\mathsf{tx}_{\mathsf{sync}}$ captures x summary-blocks instead of one. If a rollback is detected, the current sidechain committee issues a $\mathsf{tx}_{\mathsf{sync}}$ based on the summary-block not only from the current epoch, but also from previous epochs impacted by the rollback. We set this rollback period to be the time needed to confirm a mainchain block.

6. Since we assume a secure mainchain, miners may disagree only on the recent history of the blockchain.

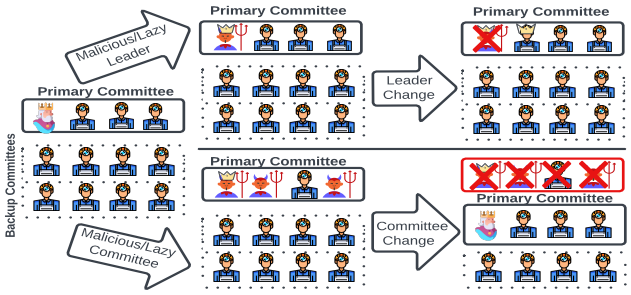


Figure 7: Autorecovery cases and mechanisms

Autorecovery protocol. We introduce an autorecovery protocol to detect and recover from any sidechain interruptions.[7] We provide an overview of this protocol in this section, while a detailed threat modeling and protocol description can be found in Appendix A.

Interruption cases. To be on the conservative side, our threat model considers that a sidechain committee may contain honest miners, malicious miners (their number is denoted as m), and lazy (honest) ones (their number is denoted as l). As we explain in Appendix A, having $m + l \geq 2f + 2$ (where f is the maximum number of faulty nodes in the committee) violates safety—the committee might agree on invalid blocks, and having $m + l \geq f + 1$ violates liveness—no agreement can be reached. Furthermore, a malicious leader may propose invalid blocks/$\mathsf{tx}_{\mathsf{sync}}$ (note that an invalid $\mathsf{tx}_{\mathsf{sync}}$ can be detected it is verified), and a lazy/unresponsive one may not propose a block or issue a $\mathsf{tx}_{\mathsf{sync}}$. These interruption cases are shown in Figure 7, and our protocol involves two techniques to detect and recover from them.

(1) *Leader change.* We use the view-change technique used in PBFT protocols [42] parameterized by a timeout ζ. Each committee member monitors the behavior of the leader. If an invalid block/$\mathsf{tx}_{\mathsf{sync}}$ is proposed, each member sends a leader-change message. Once an agreement is reached, the next leader candidate will take over (thus, the committee election algorithm not only elects the leader but also the candidates for leader change). The outcome of the process, i.e., the agreed-upon leader-change message, will be published in a meta-block (and consequently in a summary-block) to announce the leader change.

Furthermore, and to account for the case that committee members themselves could be unresponsive, and hence, no agreement on the new leader is reached, another timeout parameter η is used, such that $\eta > \zeta$. If the leader change does not conclude in η period, the committee is identified as unresponsive and will be handled by the backup committee technique (discussed below). Moreover, η is also used to deal with the case when leader candidates are lazy/malicious, leading to a long sequence of leader changes—putting the committee on hold.

7. Some forms of autorecovery can be found in the literature. For example, in Algorand [59], miners mine empty blocks in the rounds when no agreement is reached to preserve liveness. However, [36] showed that under a small adversarial spike (over the honest majority assumption) Algorand's network fails to self-heal indefinitely. In Spacemesh [21], self-healing is discussed based on a mesh (layer DAG) blockchain, whereas chainBoost does not restrict the mainchain structure.

At the beginning of any round in any epoch, each C_i does the following:

case 1—lack of progress:
Set $\mathcal{L} = \mathcal{L}_{\mathsf{sc}}$
if $\mathcal{L} = \mathcal{L}_{\mathsf{sc}}$ at the end of $i \cdot \eta$ duration **then**
Each member in C_i issues an unresponsive-committee message.

case 2—invalid operation:
Receive a new $\mathsf{B}_{\mathsf{meta}}$ or $\mathsf{B}_{\mathsf{summary}}$.
// btype is either meta or summary
if $\mathsf{VerifyBlock}(\mathcal{L}_{\mathsf{sc}}, \mathsf{B}_{\mathsf{btype}}) = 0$ **then**
Each member in C_i issues a misbehaving-committee message.

case 3—no syncing:
if no $\mathsf{tx}_{\mathsf{sync}}$ in $\mathcal{L}_{\mathsf{mc}}$ at the end of the epoch **then**
Resort to mass-sync in the next epoch.

// Committee take over
For case 1 and 2, once an agreement is reached:
for $i \in \{1, \ldots, \kappa\}$ Set $C_{i-1} = C_i$

Figure 8: Autorecovery protocol—backup committees.

(2) *Backup committees.* To deal with an unresponsive committee or one that may publish invalid blocks, we introduce the concept of backup committees. A backup committee has the same size as the primary committee, and it includes miners elected based on the older history of the mainchain, excluding the primary members.

This committee monitors the primary committee's behavior and maintain all valid sidechain transactions they receive. When the backup committee detects an invalid block being published on the sidechain, or unresponsive primary committee (i.e., no activity during a timeout η), it initiates the process of taking over. The backup committee is subject to the same threat model of the primary committee, and could be susceptible to the same threat cases discussed above. Thus, we designate κ backup committees, denoted as $\{C_i\}_{i=1}^{\kappa}$, each one monitors the one ahead of it in the line and takes over in case of misbehavior. Our protocol is captured in Figure 8.

As shown in the figure, the monitoring period for each backup committee is larger than the one used by the committee ahead of it. That is, the first backup committee C_1 is watching the primacy committee C_0, while other backup committees will step in if the previous backup committee(s) and the primary committee are unresponsive/misbehaving. When the backup committee becomes the primary committee, it resumes operation (after discarding invalid blocks, if any). Moreover, in case syncing did not take place at the end of the epoch, but there is a valid summary-block, the next epoch committee will issue a mass sync transaction for its current epoch and the previous one. Full details can be found in Appendix A.

Such autorecovery capability cannot be unconditional; the security is guaranteed under an appropriate protocol configuration (such as the number of backup committees and the committee size). We formally analyze that showing how to configure these parameters in a way that makes the failure probability of our autorecovery protocol negligible (full analysis can be found in Appendix D).

4.2.4. Limitations and potential extensions. We discuss limitations and potential extensions to our system. These are related to the applicability of chainBoost, the overhead of its autorecovery protocol, the miner incentives to maintain both chains, and the issue of transaction fees based on the temporary storage aspect of the meta-blocks. We discuss these issues in Appendix B.

4.3. Security

As chainBoost introduces a dependent-sidechain that shares the workload with the mainchain, and a block pruning mechanism, we have to show that, under this new architecture, the security of the resource market is preserved. In Appendix C, we prove the following theorem.

Theorem 1. chainBoost *preserves the safety and liveness (cf. Section 3) of the underlying resource market.*

5. Implementation

To assess the performance gains of chainBoost, we implement a proof-of-concept and conduct experiments testing several impactful factors. In doing so, we chose a file storage resource market as a use case inspired by many real-world Web 3.0 systems building such service, e.g., Storj [23], Sia [20], and Filecoin [12]. We discuss the implementation details in this section, and the experiment setup and evaluation results in the next section. Our implementation can be found at [7].

Sidechain implementation. We implemented our sidechain architecture in Go, including meta-blocks, summary-blocks, the summary rules, the syncing process, chain pruning, and the transaction annotation indicating the traffic split between the chains. In our implementation, each miner maintains a copy of the mainchain and the sidechain. Miners on the sidechain committee maintain a separate queue for each chain pending transactions (mainchain miners follow only the mainchain traffic).

For the PBFT protocol run by the sidechain, we use the collective signing (CoSi)-based PBFT-based consensus using BLSCoSi from Cothority [70]. For the communication between miners, we employ the Cothority Overlay Network Library (Onet) [71]. For committee election, we use the sliding window mechanism from [68]. We chose this simple approach as it suffices to show the performance gains that chainBoost can achieve. In the bootstrapping phase of the sliding window, our implementation picks the members of the committee at random. Later on, each potential committee member is added to the committee once, even if it mined multiple mainchain blocks within the sliding window. Finally, the most recently added committee member will be the committee leader for the epoch.

Each sidechain round proceeds as follows. The leader creates a block (a meta-block, or if it is the epoch end, a summary-block) and initiates an agreement using the CoSi-based PBFT. When an agreement is reached, the leader publishes the block, along with the aggregated committee signature produced via the CoSi process, on the sidechain. At the end of the epoch, the leader issues a sync-transaction, based on the summary-block, and broadcasts it to the mainchain miners. This transaction has the

highest priority, thus it will be placed first in the mainchain pending transaction queue, for fast processing.

Use case: file storage resource market. We use a decentralized file storage network (inspired by Filecoin [51]) as a use case exemplifying a resource market to show the benefits that chainBoost can achieve. In this market, there are two types of participants: the storage providers or servers (who also serve as miners since the amount of storage service they provide represents their mining power), and clients who want to use the service. In our implementation, this market is composed of two modules:

Market matching module: A client asks for service for a specific duration, file size and price using a propose-contract transaction (which also creates an escrow to pay for the service). The server answers with a commit-contract transaction matching a specific client offer. Both of these transactions are published on the mainchain. In our implementation, offer and demand generation is automatically done, and whenever a contract expires, a new set of propose and commit transactions are issued.

Service-payment exchange module: A server proves that it is storing a file by issuing a proof-of-storage every mainchain round. In our proof-of-concept, we use compact (non-interactive) proof-of-retrievability (PoR) [94] as a proof-of-storage (for completeness, we discuss how these proofs work in Appendix E). For implementing this protocol, we use the bilinear BN256 [83] group of elliptic curves for BLS signatures from the Kyber library [15]. As in Filecoin, by default, the payment for all valid PoRs is dispensed once the storage contract ends (we test different server payment modalities in the next section).

Mainchain implementation. The mainchain has one block type; mainchain blocks. Without chainBoost, all transactions are logged in these blocks, but with chainBoost, these blocks contain mainchain transactions along with concise summaries of the sidechain workload. Inspired by Filecoin, mining on the mainchain relies on the storage amount the servers provide, i.e., mining power. Thus, for each mainchain round, a miner will be selected to mine the next block in proportion to the amount of service it provides. This selection is done using cryptographic sortition similar to [59]. That is, each miner evaluates a Verifiable Random Function (VRF) using a random seed computed from the previous round and compares it to their mining power. If the VRF output is lower than the mining power, the miner is a potential round leader. It is possible to have multiple leaders in the same round. To resolve this issue, the miner with the first announcement will be the round leader.[8] We use Algorand's implementation of VRFs [54]. Recall that chainBoost's performance is agnostic to the mainchain consensus, thus this simplified version is enough for our purposes.

Traffic generation. We apply the same traffic distribution observed in Filecoin using [13]. That is, 98% of the traffic is service-related transactions and 2% are payment (currency transfer) transactions. The majority of the service-related traffic is composed of proof-of-storage transactions (the rest are for contract-propose/commit transactions), which are proof-of-retrievability transactions in our case. Transactions are added at the beginning of each mainchain round. A detailed explanation of our traffic distribution and generation, as well as transaction structure and sizes, can be found in Appendix F.

Optimistic rollups comparison. To provide a baseline comparison with a layer-two solution, we implement an optimistic rollup solution inspired by optimism [18], that we dub OPBoost and apply it to the base resource market. In each round, service-related transactions are batched up and processed off-chain. Then the state changes induced by a finalized batch are logged on the mainchain.

6. Performance Evaluation

We now discuss our experiment setup and results showing the performance gains of chainBoost.

Experiment setup. We deploy our system on a computing cluster composed of 10 hypervisors, each running a 12-Core, 130 GiB RAM, VM, connected with 1 Gbps network link. Our experiments, unless stated otherwise, are conducted over a network of 8000 nodes, with each node serving two contracts. An experiment length is 61 mainchain rounds, with an epoch length $k = 10$ mainchain rounds—equivalent to 30 sidechain rounds (we use mc-round and sc-round to denote mainchain and sidechain round, respectively), with a committee size of 500 miners. As discussed earlier, 2% of the traffic is payment transactions while the rest are service-related transactions, as in Filecoin. Our mainchain and sidechain block sizes are 1MB, and we collect experiment data using SQLite [64].

We test the impact of several parameters, including traffic distribution, block size, the ratio of sidechain rounds to mainchain rounds in an epoch, and various server payment modalities. For performance comparison purposes, we show results for the file storage network performance with and without chainBoost (this will allow us to quantify the performance gains that our system can achieve).[9]

In reporting our results, we measure the following metrics (some experiments also report blockchain size):

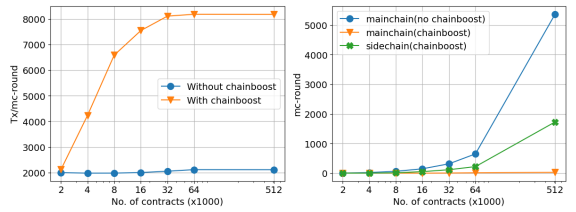
- *Throughput:* This metric corresponds to the number of transactions processed per a mainchain round. It is computed as the average number of transactions that appear in a mainchain block (or round) plus the average number of transactions in all sidechain blocks published during the same mainchain round.
- *Confirmation time:* This is the time (in mainchain rounds) that takes a transaction to be confirmed from the moment it enters the queue. As we use PBFT on both chains, once a transaction is published in a block it is considered confirmed.

We note that after an experiment run ends, we continue processing any traffic left in the queues until those queues are empty, and measure the metrics above accordingly.

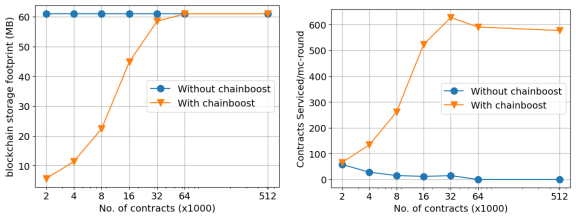
Scalability. In this experiment, we investigate the behavior of chainBoost as we increase the network load by increasing the number of service contracts. In each run, we set the contracts to be {2K, 4K, 8K, 16K, 32K, 64K, 512K}. Each contract results in a PoR transaction per mainchain

8. This is a simplifying assumption that can be comparable to Algorand's priority-based work distribution [59]. Here, the priority is given to the first leader with a valid block.

9. To the best of our knowledge, our system is the first solution that target resource markets, and the first to introduce a dependent sidechain. Thus, it is incomparable to existing sidechain solutions that restrict themselves to two-way peg and independent sidechains.



(a) Throughput (b) Confirmation time



(c) Storage footprint (d) Serviced contracts

Figure 9: Scalability experiment results (for 1 MB block, and 3 sc-rounds per mc-round for chainBoost).

round, a propose and commit contract transactions (when the contract is created), and a service payment transaction when the contract expires. We report throughput, latency, and blockchain size of the file storage market with and without chainBoost. We also report the number of newly serviced (or active) contracts per round (note that a new contract becomes active once its contract-commit transaction is confirmed). The results are shown in Figure 9.

As shown, chainBoost significantly improves the performance of the file storage. Throughput doubles as we double the number of contracts from 2K to 4K until it peaks at ∼4x the throughput of the base file storage market (i.e., the one without chainBoost) for higher numbers of contracts. Improvements are even more noticeable when we examine the confirmation time as seen in Figure 9b. Besides the almost-zero[10] confirmation time on the mainchain we get by moving the frequent transactions to the sidechain, the confirmation time growth on the sidechain is not as steep as on the base file storage. Thus, even if the number of transactions increases significantly, they are processed significantly faster with chainBoost.

For the blockchain size (Figure 9c), we observe that for networks with less than 32K contracts, chainBoost reduces the amount of data stored on the blockchain, going from a ∼90% reduction for 2K contracts to 0.3% reduction for 32K contracts (for the latter, the huge load causes the network to be saturated with non-sidechain traffic, still with chainBoost these transactions are confirmed faster than the base file system with chainBoost). chainBoost also allows more contracts to be serviced each round (Figure 9d), since chainBoost allows the transactions that activate expired contracts to be confirmed at a fast rate. Overall, with chainBoost, a file storage network can achieve higher throughput, lower confirmation time, and provide a faster service with a smaller blockchain storage footprint, thus greatly improving the market's scalability.

Impact of block size. We examine the impact of the mainchain and sidechain block sizes on performance; we set the size to $\{1, 2, 4, 8\}$ MB, and $\{0.5, 1, 1.5, 2\}$ MB for the mainchain and sidechain, respectively.

10. In our implementation the mainchain and sidechain traffic is added at the beginning of the round. So an instant transaction publication, or almost zero confirmation delay, means a transaction that was queued and processed in the same round.

TABLE 2: Average throughput (in $\times 1000$ tx/mc-round) with and without chainBoost for different block sizes.

MC Block Size (MB)	w/o chainBoost	w/ chainBoost—SC Block Size (MB)			
		0.5	1	1.5	2
1	2.00	4.73	7.54	10.36	13.17
2	3.96	4.73	7.54	10.36	13.17
4	7.910	4.73	7.54	10.36	13.17
8	15.98	4.73	7.54	10.36	13.17

TABLE 3: Average confirmation time (in mc-rounds) for mainchain/sidechain traffic under different block sizes.

MC Block Size (MB)	w/o chainBoost	w/ chainBoost—SC Block Size (MB)				
		Chain	0.5	1	1.5	2
1	145.88	MC	0.03	0.03	0.03	0.03
		SC	132.59	51.41	24.46	10.92
2	61.81	MC	0	0	0	0
		SC	132.59	51.41	24.46	10.92
4	20.74	MC	0	0	0	0
		SC	132.59	51.41	24.46	10.92
8	0.92	MC	0	0	0	0
		SC	132.59	51.41	24.46	10.92

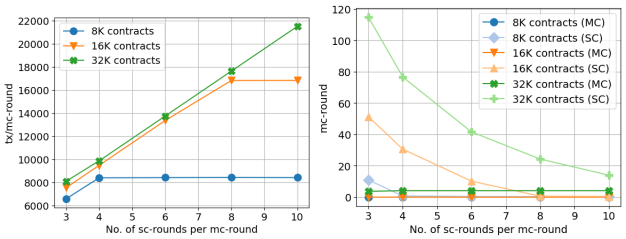
Throughput-wise, in Table 2, we observe that applying chainBoost, even at the smallest block size setting, improves the throughput of the base system for small mainchain block sizes (1 and 2 MB). For a mainchain with 1MB block size, throughput is improved by a factor of 2.36x - 6.6x, which increases as the sidechain block size grows. For larger mainchain block sizes, we notice that chainBoost with small block sizes has a smaller impact on throughput. This is because that traffic amount is fixed for all setups, the mainchain with a large block size can handle all this traffic. However, in practice, a mainchain block size of less than 8MB is usually used, and the results indicate that system designers are better off with using a large block size with the chainBoost's sidechain.

From confirmation time, in Table 3, we find that chainBoost improves confirmation time significantly. This is because the queues are not clogged with PoR transactions so other transactions are confirmed almost instantly on the mainchain, and the latency on the sidechain is improved as its block size increases as it allows for processing a larger number of transactions per sc-round.

When we apply the results of Table 3 to blockchains that have a similar block rate as Bitcoin (i.e. 1 block every 10 minutes) and a block size of 1MB, the confirmation time will go from around 1.02 days to an almost instant confirmation for mainchain transactions, to around 8, 4 and 1.8 hours for sidechain transactions, when using a 1, 1.5, 2 MB sidechain block sizes, respectively. If we apply the same analysis to chains using the Ethereum block rate (1 block every 12 seconds), the confirmation time on the sidechain drops from around 30 min to around 10, 5, and 2 minutes, for the same sidechain block sizes.

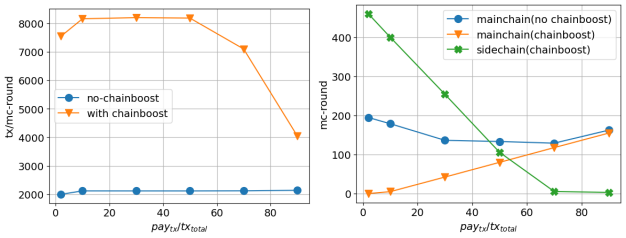
Overall, by using chainBoost to process frequent and heavy service-related traffic, throughput and latency depend on the sidechain configuration. Thus, chainBoost adopters should pick a value for the sidechain block size based on the expected traffic load.

Number of sidechain rounds per epoch. In all previous experiments, an epoch lasted for 10 mainchain rounds, or 30 sidechain rounds. In this experiment, we use the same



(a) throughput (b) confirmation time

Figure 10: chainBoost under various sidechain-to-mainchain round ratios



(a) throughput (b) confirmation time

Figure 11: chainBoost under various traffic distributions.

number of mainchain rounds per epoch but we change the number of sidechain rounds in an epoch to 40, 60, 80, and 100. (Note that a large number of sidechain rounds per epoch is possible with blockchains that have long rounds.) We measure performance for each configuration, repeating the experiment for different traffic loads.

As shown in Figure 10a, throughput increases until the network is no longer saturated with transactions. For example, for a network with 8K contracts, throughput maxes out at around 8.4K transactions/mc-round, when the sidechain epoch size/mainchain epoch size (in rounds) ratio is 4. After that, we start observing empty blocks on the sidechain for higher ratios. We can extrapolate that for 32K, where throughput maxes out at a ratio of 16.

The sidechain confirmation time decreases as the ratio of sc-rounds to mc-rounds increases (Figure 10b). As the number of sidechain rounds per epoch increases, a larger number of transactions can be published, reducing the waiting time (in terms of mc-rounds) of service transactions in the sidechain queue. If we compare these numbers to the base market without chainBoost, under the same configuration, from previous experiments, we observe that we can achieve around 4x - 11x improvement in throughput and 62.5% - 94% decrease in confirmation time. Thus, the ratio of sidechain rounds per mainchain round should be defined in a way that captures the expected load to achieve high-performance gains when chainBoost is used.

Traffic distribution. In this experiment, we fix the number of total transactions we generate but we vary the distribution of payment transactions, making them $\{2, 10, 30, 50, 70, 90\}\%$ of the total transactions while the rest are service-related ones. Throughput-wise, we observe in Figure 11a that chainBoost achieves maximum throughput when payment transactions are within 10% - 50%, after which throughput drops. However, chainBoost is still able to achieve 2x throughput improvement when payment transactions reach 90% of the total traffic. This is due to the difference in size between payment transactions and service transactions; space occupied on the mainchain by service transactions without chainBoost can fit more payment transactions, thus increasing throughput.

TABLE 4: Throughput and confirmation time for various server payment models with/without chainBoost.

	Payment Modality Setup	Throughput	MC Latency	SC Latency
without	contract end	1978.70	22.23	N/A
	epoch end	2001.63	22.56	N/A
	each mc-round	2202.20	24.65	N/A
with	contract end	6583.20	0	8.39
	epoch end	6964.49	0.84	8.23

For confirmation time, we observe in Figure 11b that as the ratio of payment transactions increases, confirmation time on the mainchain increases, while the one on the sidechain decreases. This is due to the increase in the mainchain workload while the sidechain one decreases. Hence, for blockchain systems where non-service-related traffic is dominant, chainBoost provides marginal improvements. Nonetheless, in practice, distributed resource market have a large workload of service transactions, which is where chainBoost should be used.

Server payment modalities. Paying for each proof-of-service delivery results in a huge number of service payment transactions, which overwhelm the system. Thus, markets aggregate these payments before processing (either pay at the end of a contract, as in Filecoin, or resort to deploying additional machinery). In this experiment, we show how chainBoost can support payment aggregation at low overhead. In particular, we test three service payment modalities: pay at the end of the contract duration, at the end of each mainchain round, and pay at the end of each epoch (note the last two are equivalent in the chainBoost as summaries are produced at the end of an epoch). So in the case without chainBoost, and although there are no epochs, we enforce them by paying every k mainchain rounds (lower frequency than every round and higher than at the end of a contract as servers would like to receive their partial payments earlier than at the end of, especially, long contracts). We run this experiment with 8000 miners (or servers) with each miner handling one contract.

Our results (Table 4) show that chainBoost improves the performance of the base file storage market. It achieves better throughput, with similar confirmation time if we pay for service at the end of each epoch, rather than the end of the contract. Thus, even if the base system aggregates payments as part of its network protocol, chainBoost can further improve this performance and allow for flexible setup as this frequency can be changed over time by merely changing the summary rules rather than the market network protocol. Hence, chainBoost supports (flexible) micropayment aggregation by design.

Optimistic rollups comparison. We compare chainBoost to a resource market that uses an Optimism-inspired rollup solution (OPBoost). We configure the rollup to process 1.5 MB of transactions (an Optimism batch is 1.8 MB [26], we use 1.5 MB for a fair comparison with chainBoost), and to take 3 mc-rounds for processing (Optimism batch processing takes between 2 and 4 Ethereum rounds to be finalized [85]; we use an average of 3 Ethereum rounds). chainBoost is configured with 3 sc-rounds per mc-round, and each sidechain block is 0.5 MB. We report throughput, latency—the time needed for

TABLE 5: Comparison of chainBoost vs. OPBoost

	Throughput (tx/mc-round)	Latency (mc-round)	Finality (mc-round)
chainBoost	4730	132.59	132.59
OPBoost	4682	129.81	50529

a transaction to appear in a sidechain block in chainBoost, or a non-finalized processed rollup in OPBoost, and finality—the time needed for a rollup or sidechain summary to be considered final.

Our results (Table 5) show that, given the equivalent configurations, throughput and latency are similar. However, chainBoost provides significantly shorter finality than OPBoost since a transaction is final once it appears in a meta-block. OPBoost has a one-week contestation period, which is equivalent to 50400 mc-round in the experiment setup; users has to wait for that long before taking any action based on the rollup-induced state changes. Furthermore, OPBoost has the risk of adopting invalid results due to incentive incompatibility of verifiers who may not contest the rollup results as discussed earlier.

7. Related Work

In this section, we review prior work on sidechains and relevant blockchain scalability solutions.

Sidechain-based solutions. As mentioned before, the term sidechain was first introduced by [35]; they outline the main use cases of sidechains and present a (high-level) protocol for two-way peg. Since then, a long line of work around sidechains has emerged.

Among these works, [35], [58], [67], [84], [57], [56] focus on two-way peg protocols, where [67] allow the sidechain to monitor events on the mainchain (a restricted form of data exchange). [44] briefly note the challenges of employing a sidechain, and [92], [91] tackle permissioned sidechains for Ethereum. [75] utilize sidechains to build a mutable blockchain, where a sidechain has the solo purpose of mining a modified block to replace an older one. The works [74], [93] use sidechains to improve scalability: [74] combine sharding and sidechains to allow parallel processing of smart contracts, and [93] use a tree of sidechains to scale a distributed manufacturing service. Besides requiring smart contracts in both schemes, [74] rely on the users to distribute the load among the shards, and [93] improve scalability by compromising security (by operating the sidechain at lower mining difficulty than the mainchain). Other works, mainly [96], [58], use a committee selected at random from a pool of sidechain miners to advance the sidechain and synchronize its state in the mainchain, but they both are limited to asset exchanging protocols and use independent chains. [45] also employ independent sidechains in synchronizing the asset transfer.

In parallel, many industrial initiatives explored sidechains [90], [9], [95], [5], [48], [77], [27]. However, these suffer from several limitations: they work only with specific blockchain types, mainly Bitcoin and Ethereum; focus only on two-way peg; rely on optimistic-rollups; target permissioned blockchains or federated ones; or have incomplete specifications, and the majority requires smart contracts. Also, all previous solutions, except for [58], [67], [96], do not provide rigorous security analysis. Even those who included formal security notions, they were limited to two-way pegs and independent sidechains.

None of these (independent) sidechain solutions can be trivially extended to permit arbitrary data exchange, workload sharing, and chain pruning. Independent chains have their own ecosystems in terms of miners/users, network protocol, transactions, and tokens, which makes it hard (if not unfeasible) to split the workload between them while keeping the mainchain as the single source of system state; clearly having only a two-way peg operation does not support that. At the same time, chain dependency means that their valid and secure operations are tied to each other; thus, additional countermeasures are needed to ensure that both chains satisfy safety and liveness, and that they can recover from interruptions on any of these chains.

Relevant blockchain scalability solutions. Existing solutions can be classified into three categories [62]: layer-one or on-chain solutions, layer-two or off-chain solutions, and sidechain solutions (reviewed earlier). On-chain solutions improve performance by changing the blockchain parameters (e.g. block size and block time) or modifying its design (e.g. changing the consensus mechanism or blockchain structure) such as sharding [69]. Changing the blockchain parameters is problematic. For example, increasing the block size may increase throughput, but it increases verification and propagation delays (and still all transactions are logged on-chain), leading to higher network partitioning probability—a solution that is not favorable by the community and leads to hard forks (e.g., the case of BSV [3]). Sharding divides the miners and the workload among shards to allow parallel processing, and they usually focus on how to handle cross-shard transactions and load balancing among the shards. None of these schemes devised a way to prune the mainchain

Off-chain solutions include two classes: state channels and rollups. In both cases transactions are processed off-chain and only state changes are recorded on-chain. Payment channel networks [46], [61], [81] are an example of the state channel approach, but they target only payment aggregation. ZK rollups [22], [56], [41], [40] extend this model by performing computations off-chain with ZK proofs submitted on-chain to prove output validity. ZK proofs are costly and may require a trusted setup. Optimistic rollups [90], [17] assume all submitted results are valid unless proven otherwise. A set of computing parties perform the computation, one of them submit the results (or state changes), while the rest of the parties (aka verifiers) should dispute incorrect results using an interactive challenge-response protocol. Despite its efficiency, this paradigm requires long dispute periods, slowing down system operation and service delivery. Furthermore, it may involve centralized trusted verifiers [6], [24], while incentivizing non-trusted verifiers to perform their job is still an open question [78], [50], [86] which may lead to adopting incorrect ledger state changes.

chainBoost aims to exploit the full potential of sidechains. It considers dependent sidechains and allows for arbitrary data exchange with the mainchain, rather than just two-way peg. It achieves validity of state changes resulting from the sidechain using fast consensus. Thus, it voids the use of expensive ZK proofs or slow and interactive contestation process. chainBoost offers a unified interface and architecture that can be used with any

resource market regardless of the service it provides. Also, it is agnostic for the consensus and mining type used by the underlying resource market mainchain. Lastly, chainBoost is the first system to show how sidechains can be used for blockchain pruning, thanks to its two-type block architecture.

8. Conclusion

In this paper, we presented chainBoost, a secure performance booster for blockchain-based resource markets. It introduces a new sidechain architecture that has a mutual-dependence relation with the mainchain. chainBoost allows these chains to exchange arbitrary data—all heavy/frequent service-related transactions are offloaded to the sidechain, while logging only concise summaries on the mainchain. Furthermore, chainBoost supports blockchain pruning and automatic recovery from interruptions. We analyze the security of our system and conduct thorough experiments to evaluate its performance. The results show the great potential of chainBoost in improving performance of resource markets in practice.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. CNS-2226932.

References

[1] Akamai. https://www.akamai.com/.

[2] Algorand. https://www.algorand.com/.

[3] Bitcoin sv. https://bitcoinsv.com/.

[4] Bitcoin transactions. https://developer.bitcoin.org/reference/transactions.html.

[5] Btcrelay. http://btcrelay.org/.

[6] Can i run a verifier - optimism docs. https://help.optimism.io/hc/en-us/articles/4413155125403-Can-I-run-a-verifier. [Accessed 26-02-2024].

[7] chainboost source code. https://github.com/CSSL-UConn/chainboost-release.

[8] Confirm activity flaw in nucypher. https://github.com/nucypher/nucypher/issues/1272.

[9] Cosmos. https://cosmos.network/.

[10] Decentralised file storage for a better web. https://filecoin.io/blog/posts/why-is-decentralized-and-distributed-file-storage-critical-for-a-better-web/.

[11] Discussion #87 - zksync developers - upgrade plan for block.timestamp, block.number and blockhash. https://github.com/zkSync-Community-Hub/zksync-developers/discussions/87. [Accessed 26-02-2024].

[12] Filecoin: A decentralized storage network for humanity's most important information. https://filecoin.io/.

[13] Gas statistics - filfox. https://filfox.info/en/stats/gas.

[14] Golem network. https://www.golem.network/.

[15] The kyber cryptography library. https://github.com/dedis/kyber.

[16] Livepeer. https://livepeer.com/.

[17] Matic network. https://matic.network/.

[18] Optimism — optimism.io. https://www.optimism.io/. [Accessed 26-02-2024].

[19] Optimism transaction chart. https://optimistic.etherscan.io/chart/tx. [Accessed 26-02-2024].

[20] Sia: Decentralized data storage. https://sia.tech/.

[21] Spacemesh. https://spacemesh.io/.

[22] Starkware solutions. https://starkware.co/.

[23] Storj: Fast, secure cloud storage at a fraction of the cost. https://www.storj.io/.

[24] The state of Arbitrum's progressive decentralization — Arbitrum DAO - Governance docs — docs.arbitrum.foundation. https://docs.arbitrum.foundation/state-of-progressive-decentralization. [Accessed 26-02-2024].

[25] Truebit. https://truebit.io/.

[26] Why not have more transactions per batch - optimism docs. https://help.optimism.io/hc/en-us/articles/4522562007195-Why-not-have-more-transaction-per-batch-. [Accessed 26-02-2024].

[27] Xdai. https://www.xdaichain.com/.

[28] Chernoff bounds for binomial and hypergeometric distributions, 2012. https://www.hariharan-ramesh.com/ppts/chernoff.pdf.

[29] Aydin Abadi, Michele Ciampi, Aggelos Kiayias, and Vassilis Zikas. Timed signatures and zero-knowledge proofs—timestamping in the blockchain era—. In *ACNS*, 2020.

[30] Ghada Almashaqbeh. Rethinking service systems: A path towards secure and equitable resource markets. *USENIX ;login: Magazine*, 2021.

[31] Ghada Almashaqbeh, Allison Bishop, and Justin Cappos. Abc: A cryptocurrency-focused threat modeling framework. In *INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019.

[32] Nasreen Anjum, Dmytro Karamshuk, Mohammad Shikh-Bahaei, and Nishanth Sastry. Survey on peer-assisted content delivery networks. *Computer Networks*, 116:79–95, 2017.

[33] Matthew Armstrong. Ethereum, smart contracts and the optimistic roll-up. 2021.

[34] Georgia Avarikioti, Eleftherios Kokoris-Kogias, and Roger Wattenhofer. Divide and scale: Formalization of distributed ledger sharding protocols. *arXiv preprint arXiv:1910.10434*, 2019.

[35] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. 2014.

[36] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Consensus redux: distributed ledgers in the face of adversarial supremacy. *Cryptology ePrint Archive*, 2020.

[37] Samiran Bag, Sushmita Ruj, and Kouichi Sakurai. Bitcoin block withholding attack: Analysis and mitigation. *IEEE Transactions on Information Forensics and Security*, 12(8):1967–1978, 2016.

[38] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *CRYPTO*, 2014.

[39] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *EUROCRYPT*, 2001.

[40] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. *IACR Cryptol. ePrint Arch.*, 2020.

[41] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *IEEE SP*, 2020.

[42] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[43] Mauro Conti, E Sandeep Kumar, Chhagan Lal, and Sushmita Ruj. A survey on security and privacy issues of bitcoin. *IEEE Communications Surveys & Tutorials*, 20(4):3416–3452, 2018.

[44] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *Financial Crypto*, 2016.

[45] Alfonso De la Rocha, Lefteris Kokoris-Kogias, Jorge M Soares, and Marko Vukolić. Hierarchical consensus: A horizontal scaling framework for blockchains. In *IEEE ICDCSW*, 2022.

[46] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.

[47] Theo Diamandis, Alex Evans, Tarun Chitra, and Guillermo Angeris. Designing multidimensional blockchain fee markets. In *5th Conference on Advances in Financial Technologies (AFT 2023)*, 2023.

[48] Johnny Dilley, Andrew Poelstra, Jonathan Wilkins, Marta Piekarska, Ben Gorlick, and Mark Friedenbach. Strong federations: An interoperable blockchain solution to centralized third-party risks. *arXiv preprint arXiv:1612.05491*, 2016.

[49] Shimon Even and Yacov Yacobi. Relations among public key signature systems. Technical report, Computer Science Department, Technion, 1980.

[50] Ed Felten. The Cheater Checking Problem: Why the Verifier's Dilemma is Harder Than You Think — medium.com. https://medium.com/offchainlabs/the-cheater-checking-problem-why-the-verifiers-dilemma-is-harder-than-you-think-9c7156505ca1. [Accessed 26-02-2024].

[51] filecoin.io. Filecoin: A cryptocurrency operated file storage network. 2017.

[52] Ben Fisch. Tight proofs of space and replication. In *EUROCRYPT*, 2019.

[53] Ben Fisch, Joseph Bonneau, Nicola Greco, and Juan Benet. Scaling proof-of-replication for filecoin mining. *Benet//Technical report, Stanford University*, 2018.

[54] Algorand Foundation. go-algorand. https://github.com/algorand/go-algorand/blob/master/crypto/vrf.go.

[55] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, 2015.

[56] Alberto Garoffolo, Dmytro Kaidalov, and Roman Oliynykov. Zendoo: A zk-snark verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains. In *IEEE ICDCS*, 2020.

[57] Alberto Garoffolo and Robert Viglione. Sidechains: Decoupled consensus between chains. *arXiv preprint arXiv:1812.05441*, 2018.

[58] Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *IEEE SP*, 2019.

[59] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *ACM SOSP*, 2017.

[60] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. *PKC*, 2020.

[61] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *ACM CCS*, 2017.

[62] Abdelatif Hafid, Abdelhakim Senhaji Hafid, and Mustapha Samih. Scaling blockchains: A comprehensive survey. *IEEE Access*, 8:125244–125262, 2020.

[63] Abdelatif Hafid, Abdelhakim Senhaji Hafid, and Mustapha Samih. A tractable probabilistic approach to analyze sybil attacks in sharding-based blockchain protocols. *IEEE Transactions on Emerging Topics in Computing*, 2022.

[64] Richard D Hipp. Sqlite, 2020.

[65] Dmytro Karamshuk, Nishanth Sastry, Andrew Secker, and Jigna Chandaria. Isp-friendly peer-assisted on-demand streaming of long duration content in bbc iplayer. In *IEEE INFOCOM*, 2015.

[66] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO*, 2017.

[67] Aggelos Kiayias and Dionysis Zindros. Proof-of-work sidechains. In *Financial Crypto*, 2019.

[68] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *Usenix Security*, 2016.

[69] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *IEEE SP*, 2018.

[70] DEDIS Lab. cothority/blscosi. https://github.com/dedis/cothority/tree/main/blscosi/blscosi.

[71] DEDIS Lab. onet. https://github.com/dedis/onet.

[72] Matter Labs. Finality - zksync era docs. https://era.zksync.io/docs/reference/concepts/finality.html. [Accessed 26-02-2024].

[73] Matter Labs. Zksync era overview - zksync era docs. https://docs.zksync.io/build/developer-reference/zkSync.html. [Accessed 26-02-2024].

[74] Nam-Yong Lee. Hierarchical multi-blockchain system for parallel computation in cryptocurrency transfers and smart contracts. *Applied Sciences*, 11(21):10173, 2021.

[75] Nam-Yong Lee, Jinhong Yang, Md Mehedi Hassan Onik, and Chul-Soo Kim. Modifiable public blockchains using truncated hashing and sidechains. *IEEE Access*, 7:173571–173582, 2019.

[76] Eric Lehman, Tom Leighton, and Albert R Meyer. Mathematics for computer science. Technical report, Technical report, 2006. Lecture notes, 2010.

[77] Sergio Lerner. Drivechains, sidechains, and hybrid 2-way peg designs. 2016.

[78] Jiasun Li. On the security of optimistic blockchain mechanisms. *Available at SSRN 4499357*, 2023.

[79] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *ACM CCS*, 2015.

[80] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. Smart contracts for bribing miners. In *Financial Crypto*, 2018.

[81] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. Sprites: Payment channels that go faster than lightning. In *Financial Crypto*, 2019.

[82] Tal Moran and Ilan Orlov. Simple proofs of space-time and rational proofs of storage. In *CRYPTO*, 2019.

[83] Michael Naehrig, Ruben Niederhagen, and Peter Schwabe. New software speed records for cryptographic pairings. In *Latincrypt*, 2010.

[84] Russell O'Connor and Marta Piekarska. Enhancing bitcoin transactions with covenants. In *Financial Crypto*, 2017.

[85] Jenny Pan. Bridging and Finality: Optimism and Arbitrum — jumpcrypto.com. https://jumpcrypto.com/writing/bridging-and-finality-op-and-arb/. [Accessed 26-02-2024].

[86] Aiden Park. Optimistic Rollup is Not Secure Enough Than You Think[EN] — medium.com. https://medium.com/onther-tech/optimistic-rollup-is-not-secure-enough-than-you-think-cb23e6e6f11c. [Accessed 26-02-2024].

[87] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT*, 2017.

[88] Rafael Pass and Abhi Shelat. Micropayments for decentralized currencies. In *ACM CCS*, 2015.

[89] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, 2017.

[90] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, pages 1–47, 2017.

[91] Peter Robinson. The merits of using ethereum mainnet as a coordination blockchain for ethereum private sidechains. *The Knowledge Engineering Review*, 35, 2020.

[92] Peter Robinson, Raghavendra Ramesh, and Sandra Johnson. Atomic crosschain transactions for ethereum private sidechains. *Blockchain: Research and Applications*, 3(1):100030, 2022.

[93] Nejc Rožman, Janez Diaci, and Marko Corn. Scalable framework for blockchain-based shared manufacturing. *Robotics and Computer-Integrated Manufacturing*, 71:102139, 2021.

[94] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *EUROCRYPT*, 2008.

[95] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper*, 2016.

[96] Lingyuan Yin, Jing Xu, and Qiang Tang. Sidechains with fast cross-chain transfers. *IEEE Transactions on Dependable and Secure Computing*, 19(6):3925–3940, 2021.

[97] Jonathan Zittrain, Kendra Albert, and Lawrence Lessig. Perma: Scoping and addressing the problem of link and reference rot in legal citations. *LIM*, 14:88, 2014.

A. Autorecovery Protocol: A Detailed Description

In this section, we elaborate on the design of chainBoost's autorecovery protocol. We define all sidechain interruption cases (using the ABC threat modeling framework [31]), then we present our protocol that addresses them.

A.1. Interruption Cases

We begin with modeling the tolerance of PBFT-based consensus. Then, we list the possible threats categorized by their underlying causes. Finally, we remove threats that are neutralized by system assumptions and elaborate on those that require detection and recovery.

PBFT consensus tolerance. In PBFT [42], up to f faulty nodes can be tolerated (i.e., while preserving the safety and liveness properties) with a committee size, denoted as S_c, of $3f + 1$. Thus, reaching an agreement requires at least $2f + 1$ supporting votes (to guarantee safety—agreeing on valid records—these votes are from honest nodes). On the other hand, if $f + 1$ votes are absent, liveness will be violated as no agreement can be reached. The original PBFT [42] is not scalable for large consensus groups, and so follow-up works build their consensus protocols on modified versions. ByzCoin [68], for example, is designed to scale better and provides a near-optimal tolerance with $S_c = 3f + 2$; so it requires at least $2f+2$ supporting votes to reach agreement. We adopt this version since we deal with large-scale distributed systems. Also, we adopt the leader-based approach, so a leader proposes a block for the committee to agree on, and once an agreement is reached, the block is appended to the sidechain.

Potential threat cases. Interruptions on the sidechain are related to absence of meta or summary blocks (due to not initiating the agreement or not reaching an agreement), absence of sync-transactions, or publishing invalid blocks/sync-transactions. These are threats to the liveness and safety of the sidechain, and consequently the mainchain since service-related traffic is not being processed properly. These could be due to malicious or lazy leader or committee members—those who may go unresponsive, behave arbitrarily, or simply passively collaborate with the adversary by accepting records without validation.[11]

We denote the committee elected in an epoch to run the sidechain as primary committee C_0 of size S_c. We also denote the number of malicious members in a committee as m and the number of lazy nodes as l. Accordingly, we have the following threat cases:

- Lazy/malicious committee: if the majority of the committee members are lazy or malicious, the following threats may take place: becoming an unresponsive committee, publishing invalid blocks or invalid sync-transactions, or pretending that the leader is unresponsive to force a leader change.
- Lazy/malicious leader: who proposes invalid (meta or summary) blocks/sync-transactions or is unresponsive (i.e., does not initiate agreement on block proposals or does not issue a sync-transaction).

The unresponsive lazy committee threat happens when lazy and malicious members are unresponsive, so that when $m+l \geq f+1$. This prevents committee agreement, and hence, violates liveness. The threat of publishing invalid blocks happens when lazy/malicious members agree on an invalid block proposal. That is, the leader is malicious and proposes an invalid (meta or summary) block; malicious (perhaps colluding) committee members would agree, and lazy honest members (who do not validate the block) would also agree. This threat happens when $m + l \geq 2f + 2$, thus violating safety. Lastly, a malicious leader may attempt to issue an invalid sync-transaction—one that is not consistent with its corresponding summary-block. Note that for this threat case, the invalid sync-transaction will be rejected by mainchain miners since only valid transactions are accepted. Thus, this threat is detected by the absence of $\mathsf{tx}_{\mathsf{sync}}$ from the mainchain ledger state $\mathcal{L}_{\mathsf{mc}}$ (as shown in Figure 8).

For changing the leader, as we describe later, chainBoost uses the view change mechanism of PBFT [42]. That is, committee members agree that the leader is unresponsive/malicious and force a leader change. This agreement also requires $2f + 2$ votes to take place. Thus, framing a leader may happen when $m + l \geq 2f + 2$, which is considered a safety violation (i.e., merged with the safety case above). At the same time, not reaching an agreement on changing the leader puts the system on hold. This takes place when there are unresponsive $m+l \geq f+1$ committee members, which is a liveness violation (covered by the liveness case above).

A.2. Protocol Design

We introduce an autorecovery protocol to address the previous threat cases, guaranteeing that the sidechain can recover from any interruptions, thus preserving the security of the mainchain and the market as a whole. At a high level, our protocol is composed of two techniques to handle malicious/unresponsive leader or committee. The former is detected by the primary committee members themselves, and dealt with by initiating a *leader change*. While for the latter, we introduce the concept of *backup committees* who will monitor the primary committee and take over if a misbehavior is detected.

(1) Leader change. This mechanism is composed of two phases: detection of a malicious/unresponsive leader, and recovery from this situation. Leader change is parameterized by a timeout ζ and it is repeated for every round in every epoch.

Detection. In the primary committee, each member monitors the leader's behavior and can independently detect the following:

11. Note that being unresponsive due to bad network conditions is ruled out in our system since we assume bounded-delay message delivery.

- If no activity is detected for a ζ duration, then the leader is identified as unresponsive.
- If an invalid block proposal is received, or an invalid sync-transaction is broadcast, the leader is identified as malicious.

For the latter, recall that all committee members process the sidechain traffic, and hence, can verify the validity of any proposed meta-block. Also, all of them have copies of the meta-blocks published so far in an epoch, so they can validate a proposed summary-block. Similarly, they can validate the sync-transaction since they have its corresponding summary-block (same applied for mainchain miners). Recall also that an invalid sync-transaction will be rejected by mainchain miners.

Recovery. Upon the detection of either of the above cases, each (honest) committee member sends a *leader-change* message to the rest of the committee. Once the next candidate leader receives $2f + 1$ leader-change messages (including its own), it takes over as the new leader in that epoch. The leader-change messages, which in case of invalid behavior will be accompanied by a copy of the invalid block proposal/sync-transaction, constitute the *leader-change certificate* that is published in the next meta-block (and also the summary-block of the epoch) as a proof on the validity of the leadership change.

Note that, if $f + 1 \leq m + l \leq 2f + 1$, an agreement on changing a leader will not be reached. This is reduced to the threat of unresponsive-committee (handled by the backup committee technique discussed next). Furthermore, it could be the case that a large number of future leader candidates are malicious or lazy. This will result in a long process of leader change that will put the system on hold (violates liveness). This situation is also reduced to the unresponsive committee case. Furthermore, absence of a valid sync-transaction from the mainchain, due to continuous leader-change, will be reduced to the unresponsive committee case and addressed based on whether there is a valid summary-block in the epoch.

(2) Backup committees. In this technique, besides the primary committee handling the workload of an epoch, there will be a set of backup committees standing by. A backup committee has the same size as the primary committee and monitors the sidechain traffic. The primary committee and all backup committees (we denote their number by κ) are composed of disjoint set of miners. Thus, for each epoch the election mechanism produces $\kappa+1$ committees $\{C_i\}_{i=0}^{\kappa+1}$ with a publicly known order—C_0 is the primary committee, C_1 is the first backup committee, and so on. Note that when a backup committee takes over, it becomes the primary committee, and the next committee in line becomes the first backup committee, and so on. This mechanism is also composed of two phases: detection and recovery, as we explain below.

Detection. A backup committee is responsible for detecting an unresponsive primary committee or one that publishes invalid blocks. Detecting the former is based on a timeout η such that $\eta > \zeta$; that is, no blocks or sync-transactions are published during η period. Detecting the latter is done when verifying a newly published block. The backup committee monitors the sidechain traffic and published blocks, so it can identify if a block is invalid.

Recovery. For the case of lack of progress, the leader of the backup committee initiates an agreement on a *unresponsive-committee* message. While if invalid blocks are detected, it issues a *misbehaving-committee* message containing a reference to the invalid block(s). Once the agreement is complete, by having at least $2f + 2$ signatures or votes over any of these messages, the backup committee will take over. All relevant information about the committee takeover will be published in the next meta-block (which will be created by this new committee), and eventually it will be published in the summary-block of that epoch. All invalid blocks are dropped: if it is a summary-block (and so all meta-blocks are valid), the new primary committee only publishes a summary-block [12], otherwise, the work done by the faulty committee is considered invalid and the backup committee starts all over. In case of the absence of a sync-transaction at the end of an epoch, but there is a valid summary-block, no takeover will take place. Instead, the new epoch committee will issue a mass-sync-transaction covering both epochs.

Since a backup committee itself could be unresponsive or malicious, we require having κ backup committees. Thus, the protocol above is repeated but with a larger timeout. That is, backup committee C_i will monitor for $i \cdot \eta$ period (as shown in Figure 8) to account for the timeout needed for the primary committee and all backup committees ahead of C_i.

The success of detection and recovery is conditioned on having the backup committee reach agreement on the misbehavior, and having this committee work properly when taking over. Thus, the success probability of the backup committee mechanism determines the success probability of our autorecovery protocol (recall that the unsuccessful leader-change threat is reduced to the unresponsive committee threat). In Appendix D, we analyze the success probability of the autorecovery protocol, which allows us to set up the number of backup committees κ needed to make this probability overwhelming, thus preserving liveness and safety of the sidechain, and consequently the mainchain.

B. Limitations, Tradeoffs, and Future Work

Applicability. In designing chainBoost, we target a particular system category; blockchain-based resource markets. The level of performance gains depends on the underlying system design and functionality. chainBoost is more suitable for systems that exhibit high volumes of frequent service-related traffic that can be summarized, and thus can be off-loaded to the sidechain. Systems that do not exhibit this behavior will not benefit much from the use of our scheme (as we show empirically in Section 6). Thus, outside the blockchain-based resource markets, the main motivation for this work, a viability study is needed in order to decide whether chainBoost is applicable/beneficial. As part of our future work, we aim to identify

12. In this case, there may be a race condition between the the transaction published by the faulty committee and the one from the backup committee. This can be remediated by delaying the processing of sync transactions on the mainchain by Δ (the delay bound on messages, as specified in Section 3), to check for the eventuality of race conditions. The sync transaction from the committee who took over will attest to the takeover—i.e., it will report the incident and show the agreement on its occurrence, and hence, will automatically invalidate the other sync-transaction

other categories of blockchain-based systems where we can employ chainBoost. In particular, we will investigate decentralized finance (DeFi) systems, e.g., decentralized exchanges and money lending applications, and tokens on top of Ethereum.

Furthermore, our system model targets permissionless blockchains. This begs the question of whether chainBoost can be used in the context of permissioned blockchains. For example, a hybrid digital service has its own centrally-managed infrastructure, e.g., Akamai [1] for content delivery network (CDNs), and may rely on a peer-assisted network to extend its coverage and help in handling peak demands. Here, it makes more sense to have a permissioned blockchain to manage the peer-assisted network by creating a resource market for the service provided by end users (or peers). chainBoost can be applied here, and in fact, at a lower overhead than what is incurred in a permissionless setting. That is, our autorecovery protocol is not needed; the miners managing the main and side chains are composed of a small set of reliable and fully-identified machines. This also means that committee election is not needed where the PBFT consensus is run by this set of registered miners.

The autorecovery protocol. Based on the analysis in Appendix D, there is a tradeoff between the committee size and the number of backup committees required to have a negligible failure probability of autorecovery. Naturally, larger committee size will lead to higher probability of having (active) honest majority, which leads to a smaller κ (i.e., smaller number of backup committees, and thus shorter worst case autorecovery time). However, a larger committee size will introduce additional overhead in terms of increased agreement time. Thus, system designers should select a tardeoff based on the round length in the targeted resource market and any performance constraints it has with respect to service delivery.

Miner incentives. Incentive compatibility is an important parameter when studying blockchain security. An issue that arises in our setup is incentivizing miners to work faithfully on both the main and side chains. This is outside the scope of this work; we focus on the system design and its robustness/security aspects. As part of our future work, we plan to investigate the following direction: tying the incentives of maintaining both chains together. In detail, a miner (who is a member of the sidechain committee) will collect the full rewards of the recent blocks it has mined on the mainchain only if this miner is active on the sidechain. If it does not participate in the voting process on meta- or summary-blocks, or does not propose a block when selected as a leader, the mining rewards will be deducted based on the number and weight of missing votes (i.e., a vote on a summary-block may have a larger weight than one on a meta-block, also a leader's punishment will be larger than a member's punishment).

Storage pricing and transaction fees. Another issue that arises under the setup of chainBoost is configuring the transaction fees. Note that not all transactions occupy permanent space on the mainchain, and even not all of them occupy a permanent space on the sidechain but instead their summaries do. Thus, sidechain transactions—these that are discarded—may have lower fees than mainchain ones, or these that are recorded in full in a summary-block, as they do not occupy a permanent storage space. At the same time, the fees of sidechain transactions can be used to solve the issue of miner incentives above; that is, sidechain traffic may have larger fees than these of mainchain traffic to compensate the miners for the work on the sidechain. Studying this tradeoff for resource pricing and transaction fees falls under the recent notion of multidimensional fees [47], and we leave it as part of our future work.

C. Proof of Theorem 1

In order to prove Theorem 1, we prove the following lemmas showing that chainBoost is secure and so it preserves safety and liveness of the underlying resource market.

Lemma 1 (Preserving safety)**.** chainBoost *preserves the safety property of the underlying resource market.*

Proof. Due to meta-block pruning and the mainchain syncing process that chainBoost introduces, the following threats may arise in the resource market:

- Service slacking and theft: A server, who did not provide a service, claims it did so but the proofs were removed during sidechain pruning. Also, a customer, who received the service, claims it did not receive anything as no proofs exists (since they were pruned) and avoids paying the server.
- Out-of-sync mainchain: A committee leader does not issue a $\mathsf{tx}_{\mathsf{sync}}$ at the end of the epoch, causing the sidechain and mainchain to be out of sync.
- Invalid syncing: A committee leader issues an invalid $\mathsf{tx}_{\mathsf{sync}}$, thus causing the mainchain to be updated with invalid state changes.
- Violating sidechain quality: A sidechain committee mines invalid meta- or summary-blocks, resulting in invalid mainchain state updates during syncing.

Note that chainBoost does not impact currency transfers or any other transactions that are processed on the mainchain. These are secure by the security of the underlying resource market and its mainchain.

We now show how chainBoost addresses these threats. This mainly relies on the use of a secure PBFT with a secure/unbiased committee election mechanism, and the security of the autorecovery protocol of our scheme. Furthermore, in Appendix D, we derive a lower bound for the committee size S_c to guarantee honest majority.

Service slacking and theft. These threats are mitigated by chainBoost's design. First, recall that the sidechain processes the service-related traffic using the same specifications/logic of the underlying resource market. And second, meta-blocks are not dropped until after their summary-block is mined and its corresponding $\mathsf{tx}_{\mathsf{sync}}$ is confirmed on the mainchain, and that summary-blocks are stored permanently on the sidechain. Therefore, sidechain miners can verify that a $\mathsf{tx}_{\mathsf{sync}}$ is consistent with its corresponding summary-block. Given that the sidechain uses a secure PBFT consensus, the committee size guarantees honest majority, and the autorecovery protocol addresses any interruptions on the sidechain, published summary-blocks are valid based on their corresponding meta-blocks. Thus, only valid summaries, that a committee with honest majority has produced through consensus, will be used to

sync the mainchain. This is enough to verify any service claims despite old meta-block pruning.

Out-of-sync mainchain. These threats are mitigated using chainBoost's autorecovery protocol and mass-syncing. When the sidechain committee detects that no $\mathsf{tx}_{\mathsf{sync}}$ is issued, they initiate a leader-change and the new leader issues the $\mathsf{tx}_{\mathsf{sync}}$. Also, if this primary committee does not do that, it is considered unresponsive and the backup committee technique will be activated. Mass-syncing handles cases when there are rollbacks on the mainchain, allowing the mainchain to capture all summarized traffic that was missed during the rollback period.

Invalid $\mathsf{tx}_{\mathsf{sync}}$ *and violating sidechain quality*. Invalid $\mathsf{tx}_{\mathsf{sync}}$ is detected immediately when mainchain miners verify its consistency with the corresponding summary-block, and so it will be rejected. The autorecovery protocol will deal with the leader who issues such an invalid transaction, as well as a leader that proposes an invalid summary- or meta-block, or a committee that agrees on such invalid proposals as explained in Section 4.2.3 and Appendix A.

As a result, chainBoost satisfies safety for the sidechain, and hence, does not result in any security threats to the safety of the mainchain and the resource market as a whole. □

Lemma 2 (Preserving liveness)**.** chainBoost *preserves the liveness property of the underlying resource market.*

Proof. Having a committee managing the sidechain and responsible for processing and summarizing service-related traffic may impose threats to the liveness and public verifiability of the resource market as follows:

- DoS attacks: The sidechain committee targets particular clients or servers and excludes their transactions from being published.
- Violating sidechain liveness: A misbehaving sidechain committee does not add new blocks to the sidechain, or misbehaving leader does not issue a $\mathsf{tx}_{\mathsf{sync}}$. This will disrupt the entire resource market as service-related transactions are not (fully) processed.
- Violating sidechain/mainchain public verifiability: This threat encompasses any impacts of the summary/syncing/pruning processes on the public verifiability of the side and the main chains.

Again, since chainBoost uses a secure PBFT consensus to run the sidechain with a rotating committee, and a secure autorecovery protocol, it addresses the threats above as follows.

DoS: This is alleviated by the security of the PBFT consensus and its rotating committee. As the sidechain committee changes every epoch, maintaining a position to perpetually deny particular transactions is unfeasible.

Violating sidechain liveness: These threats are mitigated using chainBoost's autorecovery protocol. At the round level, when the committee does not hear anything from the leader, they issue a leader change. At the epoch level, when a backup committee does not see growth on the sidechain after a timeout, it engages in a committee-change and takes over as described in Section 4.2.3 and Appendix A.

Violating sidechain/mainchain public verifiability: chainBoost uses a PBFT-based consensus for the sidechain, and relies on autorecovery to ensure the correct operation of this sidechain. Also, any rollbacks on the mainchain will be handled using the mass-syncing mechanism, so no summaries will be lost. As summary-blocks are accurate snapshots of the state of the sidechain for their epochs, by the security of the PBFT consensus, and that meta-blocks are kept until the $\mathsf{tx}_{\mathsf{sync}}$ is confirmed (and even for the rollback period), public verifiability of both the sidechain and mainchain state is preserved.

Accordingly, chainBoost satisfies liveness of its sidechain, and hence, preserves liveness of the mainchain and the resource market. □

Security of the autorecovery protocol. For the threats solved through autorecovery, the mitigation is not unconditional; autorecovery fails if all backup committees selected for an epoch fail, and a backup committee fails if it has over θ_l deviant members (the liveness threshold θ_l is the threshold for absent votes that will violate liveness as defined in Appendix D). Through a probabilistic analysis of the autorecovery failure probability, we derive a formula for failure probability as a function of the committee size S_c and the number of backup committees κ. An appropriate configuration keeps the failure probability negligible.

Lemma 3. *The probability of the event that* chainBoost*'s autorecovery fails (denoted as AF) can be expressed as:*

$$\Pr[AF] = \sum_{i=(\kappa+1)\theta_l}^{(\kappa+1)S_c} \frac{\binom{\mathscr{M}}{i}\binom{N-\mathscr{M}}{(\kappa+1)S_c-i}}{\binom{N}{(\kappa+1)S_c}} \left(\frac{[y^i]\Psi(y)}{\binom{(\kappa+1)S_c}{i}} \right)$$

where $\mathscr{M}$ is the number of misbehaving miners (so $\mathscr{M} = m + l$ from the previous section), N is the number of all miners, $\Psi(y) = \left(\sum_{i=\theta_l}^{S_c} \binom{S_c}{i} y^i\right)^{\kappa+1}$, and $[y^i]\Psi(y)$ denotes the coefficient of y^i in Ψ, calculated using $[y^i]\Psi(y) = \frac{1}{i!}\frac{\mathrm{d}^i}{\mathrm{d}y^i}\Psi(0)$.

We provide a detailed security analysis and formal proof of Lemma 3 in Appendix D.

D. Autorecovery Security Analysis

chainBoost's autorecovery protocol must ensure that the market can recover from any of the sidechain interruptions (defined in Appendix A) with overwhelming probability. In other words, the autorecovery failure probability must be negligible. In this section, we formally model the components related to the autorecovery protocol, define its failure probability, and derive a formula for this probability that allows system designers to configure the protocol parameters in a way that makes this probability negligible to achieve secure operation. This section also includes a formal analysis of the committee size to guarantee honest majority (i.e., meeting the safety threshold for PBFT).

D.1. Modeling and Definitions

Modeling PBFT consensus tolerance. Recall that chainBoost uses a PBFT to run the sidechain as in [68]. Also, as discussed in Section 4.2.3, chainBoost uses the original PBFT's leader change mechanism [42] to replace

a misbehaving or unresponsive leader. We define the liveness and safety tolerance (or thresholds) for both of these PBFT protocols along the lines as in [42], [63].

The original PBFT provides optimal safety and liveness thresholds, which we define as follows (where S_c denotes the committee size).

Definition 1 (Optimal Safety Threshold)**.** *The optimal safety threshold, denoted by θ'_s, is the minimum number of supporting votes required to agree in a PBFT consensus, where $\theta'_s = \lceil \frac{2S_c+1}{3} \rceil$.*

Note that the optimal safety threshold indicates the minimum number of nodes that can approve a leader's proposal, whether it is a malicious agreement or an honest one. That is, if the number of misbehaving nodes is larger than θ'_s, then a malicious agreement (e.g., on an invalid proposal that does not comply with the designated protocol) can be made. To achieve security, and so have an honest agreement, the number of honest responsive nodes must be at least θ'_s.

Definition 2 (Optimal Liveness Threshold)**.** *The optimal liveness threshold, denoted by θ'_l, is the minimum number of absent votes that will prevent the committee from agreeing in a PBFT consensus, where $\theta'_l = S_c - \theta'_s + 1 = \lfloor \frac{S_c+2}{3} \rfloor$.*

For the sidechain PBFT, and without loss of generality, we assume that system designers adopt a PBFT-based consensus protocol that requires at least θ_s ($\theta_s \in [\theta'_s, S_c)$) supporting votes to consider a leader's proposal approved. We define the general safety and liveness thresholds for this PBFT-based consensus protocol as follows. Note that the thresholds related to the adopted PBFT-based consensus are bounded by the optimal thresholds but can be different with respect to the chosen consensus.

Definition 3 (Safety Threshold)**.** *The safety threshold, denoted by θ_s, is the minimum number of supporting votes required to agree in a PBFT-based consensus, where $\theta_s \in [\theta'_s, S_c)$.*

Definition 4 (Liveness Threshold)**.** *The liveness threshold, denoted by θ_l, is the minimum number of absent votes that will prevent agreement in a PBFT-based consensus, where $\theta_l = S_c - \theta_s + 1$.*

Note that a higher safety-threshold means larger part of the committee should vote to have an agreement, so an optimal PBFT needs less threshold than other PBFT-based consensus mechanisms. On the liveness side, the higher a consensus is resilient against liveness attacks, the more members need to be non-responding to violate liveness, so optimal for liveness means a higher threshold. Thus, we have the following ordering:

$$\theta_l \leq \theta'_l \leq \theta'_s \leq \theta_s \tag{1}$$

This ordering is used in the rest of this section to merge or reduce threat cases and to extract the threat scenarios and committee election outcomes that violate security.

Modeling committee election outcome. We consider a network of size N, each node (aka miner) can be either honest, lazy, or malicious with probability p_h, p_l, p_m, respectively, with $p_h + p_l + p_m = 1$. By assumption, each (primary and backup) committee member is elected independently. We have one primary committee and κ backup committees, all denoted by C_i, where C_0 is the primary committee, and C_i for $i \in \{1, \ldots, \kappa\}$ are the backup committees, for an epoch. Let h_i, l_i, and m_i be the number of honest, lazy, and malicious members in a committee C_i of size S_c for $i \in \{0, \ldots, \kappa\}$, such that $h_i + l_i + m_i = S_c$.

Let $\mathcal{O}_i = (h_i, l_i, m_i)$ denote the random variable corresponding to committee election outcome for C_i. The total number of malicious or lazy nodes, referred to as misbehaving going forward, in C_i is denoted by x_i. The overall committee election outcome, denoted by $\mathcal{O}$, is a set of committee election outcomes for $\{C\}_{i=0}^{\kappa}$:

$$\mathcal{O} = \big\{(h_i, l_i, m_i)\big\}_{i=0}^{\kappa} = \{\mathcal{O}_i\}_{i=0}^{\kappa} \tag{2}$$

Let $\Pr[\mathcal{O}]$ denote the probability of having $\mathcal{O}$ as the overall committee election outcome and the probability that the committee election outcome for C_i is such that it contains X_i malicious or lazy members is shown by $\Pr[O_i | X_i = m_i + l_i]$.

Committee and autorecovery failure definitions. A committee C_i fails when it cannot reach an agreement or agree on malicious proposals. In turn, autorecovery fails when all (primary and backup) committees fail. That is, (1) when the primary committee is unresponsive (does not reach an agreement) or agrees on malicious proposals, happens when Equation 3 holds, (2) all backup committees fail to detect the misbehaving primary committee, happens when Equation 4 holds, or (3) after it takes over the primary committee, this backup committee becomes a misbehaving one itself, happens when Equation 5 holds.[13]

$$\mathcal{O} = \big\{\mathcal{O}_0 | \theta_l \leq x_0 < \theta_s \vee x_0 \geq \theta_s\big\} \tag{3}$$

$$\mathcal{O} = \big\{\{\mathcal{O}_i\}_{i=1}^{\kappa} | x_i \geq \theta'_l\big\} \tag{4}$$

$$\mathcal{O} = \big\{\{\mathcal{O}_i\}_{i=1}^{\kappa} | \theta_l \leq x_i < \theta_s \vee x_i \geq \theta_s\big\} \tag{5}$$

Therefore, based on the threshold ordering in Equation 1, the auto-recovery fails when the overall committee election outcome equals equation 6 (so the lower bound for the number of misbehaving nodes x_i is θ_l as defined below).

$$\mathcal{O}_F = \big\{\{\mathcal{O}\}_{i=0}^{\kappa} | x_i \geq \theta_l\big\} \tag{6}$$

Based on that, we define a committee failure and the autorecovery failure (AF) as follows.

Definition 5 (Committee Failure)**.** *A (primary or backup) committee C_i fails if in its committee election outcome, $\mathcal{O}_i = (h_i, l_i, m_i)$ the number of misbehaving nodes violates the liveness threshold, i.e., $x_i \geq \theta_l$.*

Definition 6 (Autorecovery Failure (AF))**.** chainBoost*'s autorecovery protocol fails if all primary and backup committees, C_i for $i \in \{0, \ldots, \kappa\}$, fail (cf. Definition 5), and the probability of autorecovery failure, $\Pr(AF)$, is*

13. Note that any malicious strategic behavior is covered in this combination, including malicious nodes which constitute the backup committee's majority but do not prevent it from the initial detection of the threat, but after taking over the primary committee behave maliciously—by confirming invalid state updates or not responding

defined as $\Pr(AF) = \Pr(\mathcal{O}_F)$ *where* $\mathcal{O}_F$ *is as defined in Equation 6.*

Accordingly, a secure autorecovery protocol is one that has a negligible $\Pr(AF)$. In the next section, we derive an expression for $\Pr(AF)$ that allows system designers to configure the protocol parameters in a way that makes this quantity negligible, thus achieving security.

D.2. Autorecovery Failure (AF) Probability

As defined in the previous section, autorecovery fails when all (primary and backup) committees fail. Since these committees are disjoint, the probability of failure in one committee can be generalized to other committees. That is, the population and probabilities change with each draw, and so we model our committee election as a sampling without replacement. We follow two approaches in our analysis of the failure probability: the joint Hypergoemtric distribution approach, and the generating function approach, which we describe below (the former is easier to understand, while the latter is easier to compute).

Joint Hypergeometric distribution approach. Let N be the total number of miners in the network, and $\mathcal{M}$ be the total number of misbehaving miners among N. We model the distribution of misbehaving nodes in each committee as a hypergeometric distribution, as each draw decreases the population and changes the probability of selecting a misbehaving node to the committee.

Theorem 2. chainBoost*'s autorecovery failure probability can be expressed as:*

$$\Pr(AF) = \sum_{x_\kappa=\theta_l}^{S_c} \cdots \sum_{x_0=\theta_l}^{S_c} \prod_{i=0}^{\kappa} \frac{\binom{\mathcal{M}-\sum_{j=0}^{j<i} x_j}{x_i}\binom{(N-iS_c)-(\mathcal{M}-\sum_{j=0}^{j<i} x_j)}{S_c-x_i}}{\binom{N-iS_c}{S_c}}$$

Proof. The number of malicious or lazy nodes in the first committee is modeled by the hypergeometric distribution with the parameters N, $\mathcal{M}$ and S_c. The probability of having an election outcome in which the primary committee has x_0 misbehaving nodes is

$$\Pr(\mathcal{O}_0|X_0 = x_0) = \frac{\binom{\mathcal{M}}{x_0}\binom{N-\mathcal{M}}{S_c-x_0}}{\binom{N}{S_c}}$$

For the backup committees, we need to account for the nodes that were selected in the committees preceding them as well as for changes in the misbehaving nodes population $\mathcal{M}$; in the i^{th} committee, the overall population is $N - iS_c$ and the malicious nodes are $\mathcal{M} - \sum_{j=0}^{j=i-1} x_j$. Thus, the probability of the i^{th} committee containing x_i misbehaving members would be:

$$\Pr(\mathcal{O}_i|X_i = x_i) = \frac{\binom{\mathcal{M}-\sum_{j=0}^{j=i-1} x_j}{x_i}\binom{(N-iS_c)-(\mathcal{M}-\sum_{j=0}^{j=i-1} x_j)}{S_c-x_i}}{\binom{N-iS_c}{S_c}}$$

The overall election outcome which causes autorecovery failure is the one in which all committees fail. Recall that a committee fails when the total number of malicious or lazy nodes in that committee becomes equal to or greater than the committee liveness-threshold, θ_l (see Definition 5), so we have $\theta_l \leq x_i \leq S_c$. Thus:

$$\Pr(AF) = \sum_{x_\kappa=\theta_l}^{S_c} \cdots \sum_{x_0=\theta_l}^{S_c} \prod_{i=0}^{\kappa} \Pr(\mathcal{O}_i|X_i = x_i)$$

By replacing $\Pr(\mathcal{O}_i|X_i = x_i)$ with its expression computed using the Hypergeometric distribution from above, we obtain the formula in the theorem, which completes the proof. □

Generating function approach. The previous formula is complicated and hard to evaluate (needs extensive computations). Hence, we do the same analysis using the generating function approach. In [63], the generating function approach is used to calculate the probability of a sharding take-over attack in a sharding-based protocol in which the attack is successful once at least one shard becomes insecure.

The goal of this analysis is to calculate the number of ways of having all committees fail in order to compute the failure probability of autorecovery.

First, we calculate the probability of having x malicious or lazy elected committee members, in all $\kappa + 1$ committees (so $x = x_0 + \cdots + x_\kappa$). Then we calculate the ways we can distribute them among $\kappa+1$ committees such that all of them fail. Finally, we compute the auto-recovery failures by multiplying the two previous probabilities.

The generating function that represents a failure in one committee is the following:

$$\Phi(y) = \sum_{x_i=\theta_l}^{S_c} \binom{S_c}{x_i} \times y^{x_i} \tag{7}$$

where each coefficient $\binom{S_c}{x_i}$ indicates the number of ways x_i malicious or lazy nodes can be distributed in that committee. Note that Equation 7 represents a committee's failure because its number of malicious or lazy nodes is more than the liveness-threshold threshold $x_i \geq \theta_l$.

Since the committees do not share members, they can be represented as a union of disjoint sets. The convolution rule [76] can be used to calculate the generating function of all committees $\Psi(y)$ as the product of their generating functions, i.e., $\Psi(y) = (\Phi(y))^{\kappa+1}$. The coefficient of y^x for $(\kappa + 1) \times \theta_l \leq x \leq (\kappa + 1)S_c$ represents all the ways through which x misbehaving nodes can be distributed among $\kappa + 1$ committees such that all those committees fail. Hence, we have:

$$[y^x]\Psi(y) = \frac{1}{x!}\frac{\mathrm{d}^x}{\mathrm{d}y^x}\Psi(0) \tag{8}$$

Theorem 3. chainBoost*'s autorecovery failure probability can be expressed as follows:*

$$\Pr(AF) = \sum_{x=(\kappa+1)\theta_l}^{(\kappa+1)S_c} \frac{\binom{\mathcal{M}}{x}\binom{N-\mathcal{M}}{(\kappa+1)S_c-x}}{\binom{N}{(\kappa+1)S_c}} \cdot \frac{[y^x]\Psi(x)}{\binom{(\kappa+1)S_c}{x}}$$

Proof. The probability of all committees' failure, knowing that $(\kappa+1)\theta_l \leq x$ misbehaving are elected from the network, is the following.[14]

$$\alpha = \Pr(S_c, \theta_l, \kappa, x) = \frac{[y^x]\Psi(y)}{\binom{(\kappa+1)S_c}{x}}$$

Also, the probability of having a total of x malicious or lazy members elected from the network for the total of $\kappa+1$ committees, each with the size of S_c is

$$\beta = \Pr(S_c, \kappa, x) = \frac{\binom{\mathscr{M}}{x}\binom{N-\mathscr{M}}{(\kappa+1)\times S_c - x}}{\binom{N}{(\kappa+1)S_c}}$$

Finally, the autorecovery failure probability, considering the probability of having a total of $x \in \{(\kappa+1)\theta_l, \ldots, (\kappa+1)S_c\}$ misbehaving miners elected from the network, and the probability of having all committees failed beacause of these miners, is the following:

$$\Pr(AF) = \Pr(S_c, \theta_l, \kappa) = \sum_{x=(\kappa+1)\theta_l}^{(\kappa+1)S_c} \alpha\beta$$

Substituting for α and β defined earlier produces the formula in the theorem, which completes the proof. □

D.3. Recovery Time

In this section, we analyze the time needed for the autorecovery protocol to detect and recover from worst-case interruptions. This happens when the primary committee produces all the meta-blocks during an epoch, with the last meta-block being invalid. Each meta-block takes $\eta - \epsilon$ (for some infinitesimally small $\epsilon > 0$), either because the leader is withholding it for that duration, or the committees are engaged in leader change operations. The following $\kappa - 1$ backup committees behave similarly since they start over when an invalid block is detected.

The total delay imposed by the adversarial behavior on the primary committee or any of the first $\kappa - 1$ backup committees would be:

$$\mathcal{D}_c = (\eta - \epsilon)(k-1) + T_{agr} \tag{9}$$

where k is the number of sidechain blocks in an epoch, and T_{agr} is the time needed for the backup committee to agree on an *misbehaving-committee* message.

In the worst-case scenario, the total delay in the first κ committees would be:

$$\{\mathcal{D}\}_0^{\kappa-1} = \kappa\mathcal{D}_c \tag{10}$$

The final backup committee will perform the entire work (mining metablocks and a summary-block), each block taking $\eta - \epsilon$ to produce. Thus its delay would be:

$$\mathcal{D}_\kappa = (\eta - \epsilon)k \tag{11}$$

As a result, in the worst case, the time it takes for the autorecovery to recover from is (expressed as a function of the parameters defined above):

$$\begin{aligned}\mathcal{D}(k, \eta, \kappa) &= \{\mathcal{D}\}_0^{\kappa-1} + \mathcal{D}_\kappa \\ &= (\kappa+1)(\eta-\epsilon)(k-1) \\ &\quad + \kappa(T_{agr}) + (\eta - \epsilon)\end{aligned} \tag{12}$$

14. Note that the probability of all committees' failure, knowing that $x < (\kappa+1)\theta_l$ malicious or lazy members are elected from the network, is zero.

η, being the timeout for a committee change operation, is bounded by the sidechain round duration r_{sc}. Also, T_{agr} is significantly smaller than r_{sc}. Thus Equation 12 becomes:

$$\begin{aligned}\mathcal{D}(k, \eta, \kappa) &= (\kappa+1)(\eta-\epsilon)(k-1) + \kappa(T_{agr}) \\ &\quad + (\eta - \epsilon) \\ &\leq (\kappa+1)r_{sc}(k-1) + \kappa r_{sc} + r_{sc} \\ &\leq r_{sc}\big((\kappa+1)(k-1) + \kappa + 1\big) \\ &\leq r_{sc}(\kappa+1)k = (\kappa+1)\Delta_{epoch}\end{aligned}$$

where Δ_{epoch} is the epoch duration. Considering that the adopted committee election outcome is resilient against adaptive adversaries in the τ-agile corruption mode (or slowly adaptive adversary in which τ steps are needed to corrupt parties) [89], [34], this means that $\tau \approx \kappa + 1$ epoch duration to prevent the adversary from targeting the committee members after being elected.

D.4. Committee Size Analysis

In this section, we analyze the committee size S_c with the goal of limiting the probability of having a misbehaving majority in the primary committee. In particular, we formulate the probability of this event and bound it to an acceptable failure F, allowing us to determine S_c that satisfies F (recall that the primary committee and all backup committees have the same size S_c).

We draw the members of the committee from a finite population of N miners, such that N contains $\mathscr{M}$ misbehaving miners and a miner cannot be elected twice. Thus, the committee election can be modeled as a sampling without replacement from the population N. The probability of a primary committee containing x_0 misbehaving nodes would follow the Hypergeometric distribution:

$$\Pr(X = x) = \frac{\binom{\mathscr{M}}{x}\binom{N-\mathscr{M}}{S_c - x}}{\binom{N}{S_c}} \tag{13}$$

The primary committee fails when $x > \theta_l$ (as defined in Definition 5). The probability of this event can be computed as:

$$\Pr(X \geq \theta_l) = \sum_{x=\theta_l}^{S_c} \frac{\binom{\mathscr{M}}{x}\binom{N-\mathscr{M}}{S_c - x}}{\binom{N}{S_c}} \tag{14}$$

In a network with a large node population N, as assumed in [2] and [68], committee election can be modeled as a sampling with replacement, thus can be modeled using a Binomial distribution. As such, the probability of a failed committee can be approximated as:

$$\Pr(X \geq \theta_l) = \sum_{x=\theta_l}^{S_c} \binom{S_c}{x} p^x (1-p)^{S_c - x} \tag{15}$$

Both Binomial and Hypergeometric distributions can be bound using a Chernoff bound [28]. Let p be the probability of a miner to be a misbehaving one, so $p = p_m + p_l$ from before, we define $\mu = pS_c$ to be the expected number of misbehaving nodes and so we can express $\theta_l = \omega + \mu$ for $\omega > 0$ (where we use ω instead of Δ from [28] to avoid overloading) leading to $\Pr(X \geq \theta_l) = \Pr(X \geq \omega + \mu)$, which can be bounded as:

$$\Pr(X \geq \omega + \mu) \leq \begin{cases} \exp(\frac{-\omega^2}{3\mu}) & if \frac{\omega}{\mu} < 1 \\ \exp(\frac{-\omega}{3}) & otherwise \end{cases} \tag{16}$$

We define $\gamma = \frac{\theta_l}{S_c}$, so we have $\omega + \mu = \gamma S_c$, and hence Equation 16 becomes:

$$\Pr(X \geq \omega + \mu) = \Pr(X \geq \gamma S_c) \leq \begin{cases} \exp(\frac{-(\gamma-p)^2 S_c}{3p}) & if \frac{\gamma}{p} < 2 \\ \exp(\frac{-(\gamma-p) S_c}{3}) & otherwise \end{cases} \quad (17)$$

For $\gamma = \frac{1}{3}$, and an adversarial power between 25% and 30%, $\frac{\gamma}{p}$ is less than 2 ($\frac{4}{3}$ and $\frac{10}{9}$) respectively. We use $\exp(\frac{-(\gamma-p)^2 S_c}{3p})$ to bound the failure probability that we set to be $F = 10^{-10}$. $\exp(\frac{-(\gamma-p)^2 S_c}{3p}) \leq 10^{-10}$ means that $(\gamma - p)^2 S_c \geq 30p\ln(10)$ and $S_c \geq \frac{30p\ln(10)}{(\gamma-p)^2}$. Thus, the committee size, in this case, needs to be at least between 2487 and 18651. For failure probabilities $F_1 = 10^{-5}$ and $F_2 = 10^{-3}$, the committee size needs to be at least between 1244 and 9326, and at least between 747 and 5595, respectively.

E. Compact Proofs of Retrievability

The compact proof of retrievability (PoR) introduced in [94] is based on BLS signatures [39]. It is a challenge-response protocol; the verifier challenges the prover to compute a proof based on a randomly selected parts (or blocks) of the stored file. Without these blocks, the prover cannot generate a valid proof. This scheme is proven to be secure against any PPT adversary under the Computational Diffie–Hellman (CDH) assumption over bilinear groups in the random oracle model.

Notation. We denote the natural numbers by $\mathbb{N}$, the integers by $\mathbb{Z}$, and the integers modulo some prime p by $\mathbb{Z}_p$. A cyclic multiplicative group $\mathbb{G}$ has an order p generated by generator $g \in \mathbb{G}$. We let $\lambda \in \mathbb{N}$ denote the security parameter. A bilinear group is given by a description $(e, p, \mathbb{G}, \mathbb{G}_T)$ such that $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order p and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear asymmetric map (pairing), which means that $\forall a, b \in \mathbb{Z}_p : e(g^a, g^b) = e(g, g)^{ab}$. Thus, we implicitly have that $e(g, g)$ generates $\mathbb{G}_T$. The membership in $\mathbb{G}$ and $\mathbb{G}_T$ can be efficiently decided, group operations and the pairing $e(\cdot, \cdot)$ are efficiently computable, generators can be sampled efficiently, and the descriptions of the groups and group elements each have linear size. Lastly, we have the hash function $H : \{0, 1\}^* \to \mathbb{G}$, which is modeled as a random oracle.

Preprocessing phase. In this scheme, and before the protocol can start, the client does the following: It breaks the erasure encoded file M into n blocks, each containing s sectors. The file sectors are denoted by $\{m_{ij}\}$ for $1 \leq i \leq n$ and $1 \leq j \leq s$ (each sector $m_{ij} \in \mathbb{Z}_p$). So if the processed file is b bits long, then it will be divided into $n = \lceil b/(s \log p) \rceil$ blocks. Second, the client generates a signature keypair $(\mathsf{sk}, \mathsf{pk})$, chooses a random $\alpha \in \mathbb{Z}_p$, and computes $v = g^\alpha$. The client sets the secret key as (α, sk) and the public key as (v, pk).

After that, the client computes the *authenticator values* σ_i for each block $i = \{1, \ldots, n\}$ as:

$$\sigma_i = \left(H\left(name \parallel i\right) . \prod_{j=1}^{s} u_j^{m_{ij}} \right)^{\alpha} \quad (18)$$

where $name \in \mathbb{Z}_p$ is a random file name, and public generators $\{u_j\}_{1 \leq j \leq s}$ are randomly sampled from $\mathbb{G}$ for each file. The client then computes a *file tag* $\tau = name \parallel n \parallel u_1 \parallel \cdots \parallel u_s$ together with a signature on τ using her private key sk. Finally, the client sends the file sectors $\{m_{ij}\}$ and the authenticators σ_i to a storage server.

PoR protocol operation. It proceeds as follows:

1) Generate challenge: A verifier verifies the signature on the file tag τ using the client's public key pk, and if valid, parses it to recover $name$ and $\{u_j\}_{1 \leq j \leq s}$. Then, it chooses a random challenge set Q of index–coefficient pairs $\{(i,\ v_i)\}$ (where $i \in \{1, \ldots, n\}$ and $v_i \in \mathbb{Z}_p$), and sends this set to the prover.
2) Generate response: The prover calculates the response (σ, μ) where $\mu = \{\mu_j\}_{1 \leq j \leq s}$, and $\mu_j = \sum_{(i,\ v_i) \in Q} v_i m_{ij}$. That is, it combines the blocks in Q sector-wise, each with its multiplier v_i. The prover then computes the aggregated authenticator for the blocks in the challenge $\sigma = \prod_{(i,v_i) \in Q} \sigma_i^{v_i} \in G$, and sends (μ, σ) to the verifier.
3) Verify response: The verifier computes the pairing and accepts the proof if $e(\sigma, g) = y$:

$$y = e\left(\prod_{(i,\ v_i) \in Q} H(name \parallel i)^{v_i} . \prod_{j=1}^{s} u_j^{\mu_j},\ v \right) \quad (19)$$

Non-interactive PoR protocol in our proof-of-concept implementation. The protocol described above is interactive, which does not suit the blockcahin setup. We can transform into an non-interactive one as follows: the prover computes the random challenge set using a random seed (for a security parameter of $\lambda = 128$ the seed should be 256 bits). The source of randomness for the seed can be pulled from the hash of latest block in the blockchain (since we model hash functions as random oracles). So, instead of having the verifier transmit the challenge set, for each round, the prover computes this set using the hash of the block mined in the previous round. Thus, in our setup, the client is the file owner who preprocesses the file as mentioned above, the blockchain miners are the verifiers, and the storage servers are the provers. The client put the value τ in the contract-propose transaction.

Parameter selection. As described above, one authenticator value for each file block is a storage overhead for the storage server. Using the bilinear BN256 group of elliptic curves in our implementation, authenticator values are points on this curve and $|\sigma_i| = \log p$. Therefore, having s as the number of sectors, the total storage overhead is $\frac{b}{s \log p} \cdot \log p = \lceil \frac{b}{s} \rceil$. Choosing a larger s value reduces the one-time overhead of servers' storage at the cost of higher communication. This communication is the PoR transactions that the server repeatedly sends to the blockchain miners in order to prove that it still storing the file. Each PoR proof includes one point and s scalars, so the proof size is $(1 + s) \log p$. Thus, a PoR transaction contains the PoR proof, along with the round number for which it was generated. In our implementation, we set $s = 2$, thus the size of the PoR transaction is 515 bytes.

For example, for a 10 GB file, with $s = 1$, the server storage overhead is 10 GB (for the authenticator, so total storage is 20 GB for both), but the server response (PoR) contains one scalar in $\mathbb{Z}_p$ (256 bits) plus a point (64 bytes)

on the curve $\mathbb{G}$. Whereas, when $s = 10$, with the same file size, the server storage overhead would be 1 GB for the authenticator (so total storage is 11 GB) but the server response contains 10 scalars in $\mathbb{Z}_p$ (2560 bits) and a point on the curve $\mathbb{G}$. Note that the PoR transaction size is a function of s, not the file size. Also, the query set size does not impact PoR transaction size or the storage overhead. This set is generated locally by the server and anyone can verify it using the round seed, the contract file-tag τ, and the client public key, which are all publicly available.

F. Transaction Types and Traffic Generation

In this section, we provide details on the structure of the transactions and the way traffic is generated and processed by the miners in our proof-of-concept implementation.

Transaction types. We have six transaction types as follows (their sizes are found in Table 6):

Contract-propose**:** Corresponds to a service agreement proposed by a client. Its structure contains the contract ID, a pointer to a payment structure (the escrow), a duration, a file size, a start round for the service, service fee (which is the payment value per mainchain round), and a file tag for the file to be stored.

Contract-commit**:** A server issues a commit transaction when it commits to providing a service to a client; a response to a Contract-propose, so it contains the contract ID.

Payment**:** The payment (currency transfer) transactions in our system follow the UTXO model and have the same structure as in Bitcoin [4].

Proof-of-retrievability**:** It is issued by a server every round for each file it stores. It contains the round number during which a PoR is generated, and a cryptographic proof based on the scheme from [94].

Storage-payment**:** This transaction pays the server from the client's escrow. It has the same structure as payment transaction, but adds the contract ID.

Sync-transaction**:** Its structure follows the summary rules adopted in chainBoost. In our proof-of-concept implementation, we summarize PoRs by counting them. Thus, this transaction contains an array of contract IDs (where we assume each contract is handled by one server) and the number of PoRs generated for that contract (by the corresponding server) for the epoch. Thus, its size varies based on the number of active contracts in the epoch.

Traffic generation. When configuring our experiments, we assign each server a contract(s) with a specific duration of service. While the contract is active, the server issues one PoR transaction per mainchain round. The contract service duration (in mainchain rounds) is selected at random from a normal distribution $\mathcal{N}(\mu = 40, \sigma^2 = 20)$ with a mean μ and standard deviation σ. Once a contract expires, mainchain miners issue a Storage-Payment transaction to pay the server for the service. Since we automatically match clients and servers, contract-propose and contract-commit are issued at the same time. This means that the contract is activated once again.

For traffic distribution, we follow the one of Filecoin [51]. Using the the Filfox Explorer [13], we find that around 2% of all transactions in the system are currency transfer (called Send in the explorer). The remainder is split into 96% representing service proof transactions (e.g., in the explorer these include TerminateSectors, ProveCommitSector, etc.) and 2% for other operational transactions (e.g., ChangeOwnerAddress). So, in terms of our system, these translate into 2% payment transactions, and 98% PoR and propose/commit contract transactions.

TABLE 6: Transaction Types and Sizes

Transaction Type	Size (Bytes)
contract-propose	645
contract-commit	79
Payment	398
Proof-of-Retrievability	515
Storage-Payment	406
Sync-transaction	Varies based on number of active contracts

Traffic processing. As discussed in Section 5, in our proof-of-concept implementation miners maintain separate queues for the traffic they process. Miners on the mainchain maintain two queues; one is for payment transactions, as they are generated as quota of the overall traffic, and are allocated a quota of each block size (in our experiment, it's 30%). The other is for all other mainchain traffic (i.e., all other transactions, except Proofs-of-Retrievability when running with chainBoost). The second queue enforces a higher priority for sync-transactions, and all the other ones are FIFO. Miners on the sidechain committee have another queue for sidechain traffic (i.e., Proofs-of-Retrievability transactions). In each round, the round leader forms a block of the transactions in the queue corresponding to the respective mainchain or sidechain. Each block is filled with transactions up to its capacity, or until queues of a specific chain are empty.