UC Santa Cruz
Journal of Systems Research

Title
Hierarchical Functionality-Oriented Layer-2 Scaling for Decentralized Resource Markets

Permalink
https://escholarship.org/uc/item/15v2q176

Journal
Journal of Systems Research, 6(1)

Authors
Najd, Mohamed E.
Almashaqbeh, Ghada

Publication Date
2026-11-15

DOI
10.5070/SR3.70464

Copyright Information
This work is made available under the terms of a Creative Commons Attribution-NonCommercial License, available at https://creativecommons.org/licenses/by-nc/4.0/

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

[SOLUTION] HIERARCHICAL FUNCTIONALITY-ORIENTED LAYER-2 SCALING FOR DECENTRALIZED RESOURCE MARKETS

Mohamed E. Najd
University of Connecticut

Ghada Almashaqbeh
University of Connecticut

Foreword by the Area Chair

Decentralized resource markets allow users to trade services without relying on a central provider. Focusing on blockchain-based markets, this paper presents marketPrism, a secure hybrid sidechain-sharding layer-2 framework that divides a resource market into functional modules and processes their transactions in parallel using separate sidechains. Through a file-storage market case study, the paper demonstrates promising performance improvements. We hope this work inspires more application-aware approaches to blockchain scalability.

– Roberto Palmieri, Lehigh University
– Lewis Tseng, University of Massachusetts Lowell

Reviewers

- Lewis Tseng, University of Massachusetts Lowell
- Joss Duff, Lehigh University
- Seo Jin Park, University of Southern California

Artifacts

The artifacts for this work were independently evaluated by the Artifact Evaluation Board (AEB) led by Eric Eide.
The AEB determined that the artifact is usable by a third party and that it allows reproducing the main results from the paper.
Artifact: https://github.com/marketprism-anon/marketprism

Reviews

Anonymized reviews are publicly available.
Reviews: https://zenodo.org/records/24550788/files/Sol_Layer_2_Scaling_Reviews.pdf

Copyright and License

Licensed under Creative Common License CC-BY-NC. Copyright retained by the authors.

HIERARCHICAL FUNCTIONALITY-ORIENTED LAYER-2 SCALING FOR DECENTRALIZED RESOURCE MARKETS

MOHAMED E. NAJD
University of Connecticut
menajd@uconn.edu

GHADA ALMASHAQBEH
University of Connecticut
ghada@uconn.edu

Abstract

Decentralized resource markets are Web 3.0 applications that build open-access platforms for trading digital resources. They promise cost reduction, transparency, and flexible service provision. However, these markets suffer from scalability problems due to their large workloads that must be processed in a timely manner.

We introduce marketPrism, a secure hybrid sidechain-sharding layer-2 solution for decentralized resource markets. marketPrism enables effective scaling by accounting for the work models and traffic patterns of these markets. It parallelizes traffic processing by leveraging *dependent sidechains* and *functionality-oriented workload splitting* that assigns each market module its own sidechain. Different from sharding, marketPrism *does not* incur any cross-sidechain transactions that tend to be costly. Moreover, it introduces *hierarchical workload sharing* that further sub-divides overloaded modules, and *weighted miner assignment* that assigns miners with vested interest to critical modules' sidechains to promote resilience. marketPrism also enables blockchain pruning to reduce the market storage footprint. Besides analyzing security, we build a proof-of-concept implementation for a distributed file storage market as a use case. Our experiments show that, compared to a single sidechain-based prior solution, marketPrism boosts throughput by 4x and reduces confirmation latency by 5x. Also, they show that marketPrism outperforms sharding by 2.5x in throughput and 3.5x in latency.

1 Introduction

Decentralized resource markets are Web 3.0 applications that build open-access platforms for trading digital resources, e.g., computation outsourcing [10], file storage [7], and video transcoding [12]. They aim to solve the cost, trust, and governance issues of centrally-managed services. For example, conventional content distribution network (CDNs) can offload traffic during peak periods to decentralized CDNs to improve performance [25]. Moreover, they mitigate information loss due to provider misconfiguration/availability issues, e.g., Google Drive users lost their data even under Google's multi-region infrastructure [11]. Furthermore, these markets resolve incentive issues of early P2P systems that suffered from freeloading [32] or required centralized banks to handle payments [63]. In parallel, they offer insights into useful mining by tying miners' power to service contribution.

However, the high potential of these markets is impeded by several limitations. Facilitating collaboration between trustless parties in a monetary-incentivized and open-access model, introduces various design complexities and security issues. Ensuring correct and secure operation usually requires deploying new techniques, such as market management, resource expenditure proofs, and dispute resolution, which negatively impact performance. At the same time, digital services must meet particular QoS levels in terms of throughput and latency. Under the large workload they have, having a blockchain makes it harder to meet such requirements due to scalability issues; e.g., Ethereum, a popular choice for Web 3.0 applications, has an average throughput of 14 transactions/sec [6] with around 62% of the transactions have a delay of at least 5 min [46]. All these factors hinder the vision of Web 3.0 that aims to replace the centrally-managed services that we currently use with fully-decentralized ones.

1.1 Limitations of Prior Work

Layer-2 solutions are a keystone of blockchain scalability. They boost throughput and reduce delays by processing transactions off-chain and recording only state changes on-chain. Principal layer-2 paradigms that allow processing arbitrary transactions, not just currency transfers as in payment networks [30], generally include rollups and sidechains. Applying these paradigms to resource markets will exacerbate performance and security issues; they do not account for the unique design, QoS requirements, and traffic patterns of these markets, as we discuss below (a comparison can be found in Table 1). We also discuss sharding as marketPrism utilizes a sharding-like approach at the layer-2 level.

Rollups. Optimistic rollups [14, 40] process transaction batches off-chain while submitting results on-chain. Verifiers check these results and challenge incorrect ones. Optimistic rollups have long contestation periods, reaching one week [14], thus severely impact performance since results are not final until this period is over. Furthermore, they may use trusted verifiers [1, 20], while incentivizing untrusted verifiers is an open question [15, 49]—meaning that verifiers are not incentivized to vet/challenge the results, leading to adopting invalid state changes. They also suffer from DoS attacks against fraud-proofs (challenging incorrect results) that prevent their processing during the contestation period [43].

Table 1: Comparison with prior work.

Solution[a]	Type	Finality	Workload Distribution	Miner Population Based Scaling	Intra-division Scaling	Cross-division Tx Rate[b]	Pruning	On-chain Storage
Optimism [14, 21]	Optimistic rollup	7 days	N/A	No	No	N/A	No	Batch transcript and state changes
ZKSync Era [8, 23]	ZK-rollup	3-24h	N/A	No	No	N/A	Yes	State changes
Omniledger [45]	Random sharding	1 block (order of seconds)	Random	Yes	No	90%+	Optional	All transactions / State blocks[c]
Rapidchain [64]	Random sharding	1 block (order of seconds)	Random	Yes	No	90%+	No	All transactions
OptChain [56]	Localized sharding	Blockchain-dependent[d]	Score-based	Yes	No	9.28%	No	All transactions
chainBoost [55]	Dependent sidechain	1 meta-block (order of seconds)	None	No	No	N/A	Yes	State changes (mainchain) summary-blocks (sidechain)
marketPrism	Hybrid sidechain-sharding	1 meta-block (order of seconds)	Functionality-based	Yes	Yes	0 % [e]	Yes	Same as chainBoost

[a] Finality is the time needed to consider state changes final. Intra-division scaling is the ability to scale within a rollup/shard/sidechain. For chainBoost, workload distribution is only about sending all service-related traffic to a single sidechain. Optchain builds a UTXO transaction graph, with transactions that have the same PageRank score assigned to the same shard. For sharding solutions, cross-sharding rates are reported for four shards.
[b] Cross-division transactions happen when their state effects cannot be committed by a single chain's consensus decision.
[c] When pruning is not used, all transactions are logged in the shards. Otherwise, state blocks containing Merkle tree roots of all transactions are stored in the shard, while clients store Merkle proofs.
[d] Optchain's finality is dependent on the consensus algorithm used in the shards.
[e] Zero by construction under a self-contained partition. Inter-module state reads remain, but use a miner's local view of the system.

Zero-knowledge (ZK) rollups [23, 51] generate ZK proofs attesting for batch processing correctness. Proof generation is costly, takes on the order of minutes, which becomes worse when attesting to complex transactions [19]. ZK-rollups also have high confirmation delays that may reach up to 24 hours [22, 23], and require high-performance hardware—operating a node can cost 1365 USD per month [28].

Data availability in rollups requires recording the full transaction batch on-chain, thus rollups do not reduce the mainchain storage cost.[1] Some ZK-rollups avoid that at the expense of more expensive ZK proofs [4, 28]. All these factors make rollups unsuitable for resource markets as they impact QoS support.

Sidechains. Sidechains [27, 34, 37, 41, 47, 57] operate a secondary blockchain in parallel to the mainchain. Despite their potential in improving scalability, most prior works focused on currency transfer, or two-way peg, between the side and main chains. Moreover, all of them (including industrial deployments, e.g., Cosmos [3] and parachains such as Polkadot [16]) employ independent sidechains, where each chain has its own miners, transactions, and network protocol. This limits the utility of sidechains and prevents workload sharing and arbitrary data exchange between the two chains. Cosmos [3] allows some form of data exchange using data duplicates, but this amplifies storage costs. Furthermore, none of these sidechain solutions support blockchain pruning.

Dependent sidechains. To address these limitations, [55] proposed a new sidechain architecture, called chainBoost, that has mutual-dependency relationship with the mainchain. All service-related traffic is offloaded to the sidechain, which processes this traffic (into temporary blocks) and summarize it to produce concise state changes. These summaries are used to sync the mainchain, so once syncing is confirmed, the temporary blocks are pruned. By sharing the workload with the sidechain, chainBoost significantly boosts throughput, and reduces latency and blockchain size. However, it operates a *single sidechain*; under heavy market workload, this sidechain will be overwhelmed. Also, chainBoost *does not scale with the number of miners*; for every epoch a committee of the mainchain miners manages the sidechain while the rest of the miners do not contribute to workload processing.

Sharding. Sharding aims to achieve parallelism by scaling with the number of miners. It splits the blockchain into shards and distributes the miners and system workload among these shards to enable parallel processing. Earlier sharding schemes are either centralized [29], have fixed shard membership [62], target permissioned blockchains [65], or suffer from security issues—shards are assigned small miner committees that can be taken over by the adversary [52]. The works [45, 64] address these issues via decentralized large enough shard assignment. However, they randomly assign transactions to shards leading to a high volume of cross-shard transactions (more than 90% of the transactions [45, 54, 56, 64]). Cross-shard transactions are complex, introduce long delays, and often require end users to handle them [45], making these solutions unsuitable for resource markets. Solutions that improve transaction distribution locality [35, 39, 48, 56, 61] create imbalanced shards, with overloaded shards become performance bottlenecks. Lastly, [50] discuss workload-based subsharding, but the design is high level and requires a highly-interactive protocol to handle conflicted transactions.

An open question. Thus, we ask the following question: *Can we build a layer-2 scalability framework that enables parallel processing in resource markets while accounting for their design and traffic patterns, and without impacting the correct and secure operation of these markets?*

[1] EIP-4844 [5] in Ethereum aims to place transaction batches in blobs that are pruned after a few weeks once the contestation period is over.

1.2 Our Contributions

We answer this question in the affirmative and introduce marketPrism, a hybrid sharding-sidechain layer-2 solution. marketPrism divides the market into operational modules, each of which is assigned to a sidechain, and distributes the miners and the service-related workload among these sidechains. Thus, it simplifies scaling coordination and eliminates expensive cross-sidechain transactions.

System design. We introduce a novel *hybrid sharding-sidechain architecture* that enables parallel processing of the resource market's workload. It introduces an *adaptive functionality-oriented* approach; the resource market is divided into modules, each of which represents a functional unit and is assigned its own sidechain. Thus, the market workload is distributed among these sidechains based on the functionality they handle. For example, all transactions related to service agreements are assigned to the market matching module's sidechain, and all transactions related to service delivery and payments are assigned to the service-payment exchange module's sidechain, and so on. Moreover, we introduce a *hierarchical intra-module workload sharing* that subdivides overloaded modules, by having sub-sidechains, allowing for parallel processing even within the same module.

With this functionality-oriented workload splitting, transactions are contained within their modules' sidechains, thus *eliminating cross-sidechain transactions*, which significantly boosts performance. In turn, the miners are distributed among these sidechains/sub-sidechains, allowing for scaling further as the number of miners increases in the system.

marketPrism retains the underlying features of the sidechain architecture of [55]. These include managing the sidechain using a practical Byzantine fault-tolerant (PBFT)-based consensus to speed up agreement and reduce confirmation delays; maintaining the mainchain as the source for collective system state via sidechain syncing; and sidechain pruning that significantly reduces the size of the sidechain and subsequently the mainchain. Furthermore, a transaction is considered final once it appears on a sidechain, and workload processing correctness (based on the original logic of the market) is guaranteed by the security of consensus.

To promote resilience to security threats, we introduce a *weighted miner assignment* that assigns miners with high positive participation to sidechains managing critical modules. This reduces the chances of interruptions, caused by miner misbehavior, in these modules. Additionally, marketPrism is equipped with an *autorecovery protocol* that accounts for impacted modules during interruptions to allow safe recovery.

Security analysis. We analyze the security of marketPrism, showing that it preserves the correct and secure operation (i.e., liveness and safety) of the underlying resource market.

Implementation and evaluation. To assess performance, we build a proof-of-concept implementation, and conduct various experiments for a file storage market as a use case. Our results show that marketPrism achieves up to 4x increase in throughput and 5x reduction in confirmation latency compared to chainBoost. It also achieves a 2.5x improvement in throughput, 3.5x reduction in confirmation latency, and a 3x reduction in storage size compared to sharding.

To the best of our knowledge, marketPrism is the first to combine the parallelism of sharding and the features of dependent sidechains while achieving the best of both worlds. Coupled with its robustness and interruption resilience, our work is a leap forward towards making decentralized resource markets, and potentially other types of Web 3.0 systems, practically viable without compromising security.

2 Background

Decentralized resource markets. These are blockchain systems that offer digital services on top of the currency exchange medium. Users who can provide the service join as servers, while those who need the service join as clients, so resources are being traded among them for cryptocurrency payments. There are numerous deployments in practice, such as Filecoin [7] for file storage, Livepeer for video transcoding [12], and Session [17] for instant messaging.

A decentralized resource market consists of several modules: market matching, service-payment exchange, and dispute resolution. Servers submit offers detailing the service parameters they can provide, e.g., amount, duration, and prices, while clients submit asks detailing their service needs. Based on these offerings/asks, clients and servers are matched (this may include negotiating service terms). After that, the service-payment exchange phase starts, during which servers serve clients and clients pay for the delivered service. To avoid collecting payment without doing the promised work, servers submit service delivery proofs in order to be paid, e.g., proof of storage [33]. Also, to protect against clients who do not pay after being served, clients deposit payments in advance in an escrow that the miners use to dispense payments to the servers based on the submitted valid proofs.

Several security threats may arise due to having monetary incentives, where many of these misbehaviors cannot be mitigated in a cryptographic way. Instead, financial security countermeasures are employed via a dispute resolution module. Upon detecting a misbehavior, a proof-of-cheating is submitted. If this proof is found to be valid, financial penalties will imposed against the misbehaving parties, e.g., forfeiting (all or part of) their collateral or stake.

The chainBoost framework. chainBoost [55] introduces a new sidechain architecture targeting decentralized resource markets to boost their throughput, and reduce confirmation delays and blockchain size. It enables workload sharing and arbitrary data exchange between the mainchain and the sidechain, thus creating a mutual-dependency relationship between them.

That is, these chains live in the same domain—they have the same network protocol, transaction format, and miner population. Also, their security and correct operation depend on each other.

All service-related transactions that can be summarized are processed on the sidechain, while the rest stay on the mainchain. The system operates in epochs and rounds; an epoch is ω contiguous rounds, and a round is the time during which a block is produced. For each epoch, a committee from the mainchain miners is elected to manage the sidechain. This sidechain is composed of temporary meta-blocks that contain processed transactions, and permanent summary-blocks that contain summaries of the state changes induced by meta-blocks. In every round in an epoch, the committee produces a meta-block, while in the last round of this epoch, it produces a summary-block. Based on the summary-block, the committee issues a sync-transaction to sync the mainchain according to these summaries. Once this transaction is confirmed on the mainchain, all corresponding meta-blocks in that epoch are discarded, allowing for a significant reduction of blockchain size. The permanent summary-blocks support public verifiability as they attest to the validity of state changes.

In chainBoost, the sidechain employs a practical Byzantine fault tolerance (PBFT)-based consensus to speed up agreement, and thus allow processing service-related traffic in a timely manner. Any secure PBFT-based consensus with a secure committee election mechanism can be used. Hence, a transaction is final once it appears in a meta-block. As shown in [55], chainBoost achieves significant performance gains in terms of throughput, latency, and blockchain size. Nonetheless, it operates only one sidechain, and so all service-related transactions is directed to this sidechain, which would lead to a performance bottleneck as explained earlier.

3 Preliminaries

In this section, we present our notation, and the various models underlying our system design. We adapt the system, security, and adversary models from [55] to capture marketPrism setting, as shown below.

Notation. We use λ for the security parameter, pp for the system public parameters, tx for a transaction, and $\mathcal{L}$ for a ledger (or blockchain), with $\mathcal{L}_{\mathsf{mc}}$ denotes the mainchain ledger and $\mathcal{L}_{\mathsf{sc},\mathsf{module}}$ denotes the ledger of the sidechain assigned to the market module module. The set of market modules is denoted as $\mathsf{modules}$, and so we write $\mathsf{module} \in \mathsf{modules}$ to refer to a module within this set. Each participant maintains a secret and public key, sk and pk, respectively.

System model. A resource market is an open-access system that allows anyone to join and leave at any time, and is powered by a permissionless public blockchain. Parties are identified by their public keys, and are divided into clients $\mathcal{C}$, servers $\mathcal{S}$, and miners $\mathcal{M}$ who manage the blockchain and establish Sybil-resistant identities based on their mining power. We present a system model for a minimum viable, extensible, resource market composed of three modules: market matching, service-payment exchange, and dispute resolution. This market offers functionalities related to system and party setup, transaction issuance and verification, and chain extension.

marketPrism introduces several sidechains, each representing a resource market module, and the service-related transactions are split among these sidechains based on the modules they manage. Each sidechain is managed by an epoch committee elected from the mainchain miners that runs a PBFT-based consensus protocol. To simplify the presentation, we adopt a leader-based PBFT in which a leader proposes a block for the committee to agree on (as in [44])—nonetheless, voting-based PBFT (as in [38]) can be used instead.

The resource market offers the following functionalities:

$\mathsf{SystemSetup}(1^\lambda) \to (\mathsf{pp}, \mathcal{L}^0_{\mathsf{mc}})$**:** Takes the security parameter λ as input, and outputs the system public parameters pp and an initial ledger state $\mathcal{L}^0_{\mathsf{mc}}$ (which is the genesis block).[2]

$\mathsf{PartySetup}(\mathsf{pp}) \to (\mathsf{state})$**:** Takes the system public parameters pp as input, and outputs the initial state of the party state (containing a keypair $(\mathsf{sk}, \mathsf{pk})$, and for miners, the current ledger state $\mathcal{L}_{\mathsf{mc}}$).

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

- $\mathsf{tx}_{\mathsf{ask}}$: Used by a client to state its service needs.
- $\mathsf{tx}_{\mathsf{offer}}$: Used by a server to state its service offering.
- $\mathsf{tx}_{\mathsf{agreement}}$: Service agreement between a client and a server (or a set of servers).
- $\mathsf{tx}_{\mathsf{serviceProof}}$: Service delivery proof submitted by a server.
- $\mathsf{tx}_{\mathsf{servicePayment}}$: Server payment for the provided service.
- $\mathsf{tx}_{\mathsf{dispute}}$: Initiates a dispute for a misbehavior.
- $\mathsf{tx}_{\mathsf{transfer}}$: Currency transfer between participants.

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

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

$\mathsf{UpdateState}(\mathcal{L}_{\mathsf{mc}}, \{\mathsf{tx}_i\}) \to (\mathcal{L}'_{\mathsf{mc}})$**:** Takes as input the current mainchain ledger state $\mathcal{L}_{\mathsf{mc}}$, and a set of pending

[2] For the rest of the algorithms, if not mentioned, the input pp is implicit.

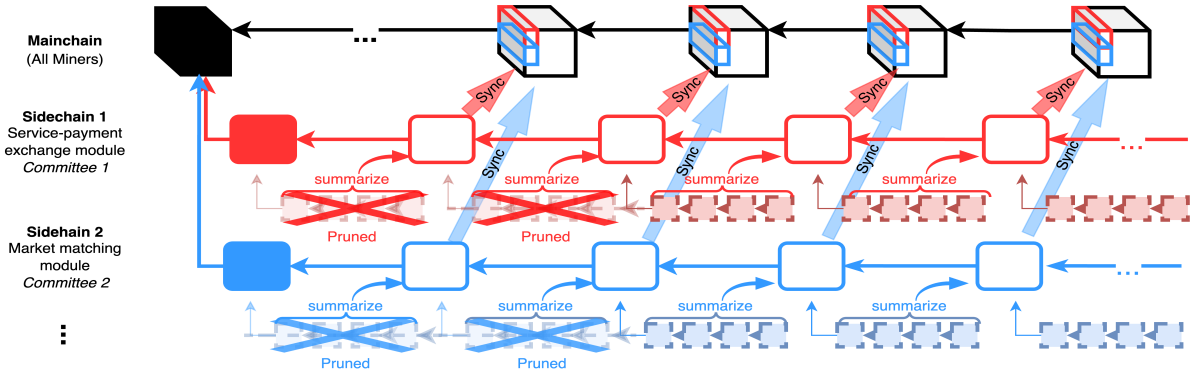


Figure 1: marketPrism architecture.

transactions $\{\mathsf{tx}_i\}$, updates the ledger state based on the changes induced by these transactions and outputs updated ledger $\mathcal{L}'_{\mathsf{mc}}$.

UpdateState extends the ledger with a new block containing new transactions based on the used consensus protocol.

It should be noted that the notion above captures a base resource market. Resource markets in practice may have additional modules, with an additional set of transactions, and the market may offer additional non-service-related transactions beyond just $\mathsf{tx}_{\mathsf{transfer}}$. The model above can be extended to cover these additional semantics.

marketPrism's functionalities can be abstracted as follows:

$\mathsf{Setup}(\mathcal{L}^0_{\mathsf{mc}}, \mathsf{modules}) \rightarrow (\{\mathcal{L}^0_{\mathsf{sc,module}}\}_{\mathsf{module}\in\mathsf{modules}}, \mathcal{L}'_{\mathsf{mc}})$**:** Takes as input the mainchain genesis block $\mathcal{L}^0_{\mathsf{mc}}$ and the set of the functional modules in the market modules. It creates a sidechain for each module and outputs the initial state of these sidechains $\{\mathcal{L}^0_{\mathsf{sc,module}}\}_{\mathsf{module}\in\mathsf{modules}}$. $\mathcal{L}^0_{\mathsf{sc,module}}$ is the genesis block containing the module/sidechain specific public parameters $\mathsf{pp}_{\mathsf{module}}$ including traffic classification, summary rules, and sidechain consensus configuration. Setup also creates the mainchain state variables that will be synced by the sidechains, thus producing an updated mainchain ledger state $\mathcal{L}'_{\mathsf{mc}}$.

$\mathsf{Elect}(\mathcal{L}_{\mathsf{mc}}) \rightarrow (\{\{\mathcal{C}_{i,\mathsf{module}}\}, \{\mathsf{leader}_{i,\mathsf{module}}\}\}_{\mathsf{module}\in\mathsf{modules}})$**:** Takes the current state of the mainchain $\mathcal{L}_{\mathsf{mc}}$ as input, and outputs a set of committees $\{\mathcal{C}_{i,\mathsf{module}}\}$ and their leaders $\{\mathsf{leader}_{i,\mathsf{module}}\}$ for all module sidechains., where $i \in \{1,\ldots,n\}$ and n is the size of the committee set. The first committee $\mathcal{C}_1$ in every $\{\mathcal{C}_i\}$ is the primary committee while the rest are backup committees.

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

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

$\mathsf{VerifyBlock}(\mathcal{L}_{\mathsf{sc,module}}, \mathsf{B}_{\mathsf{btype}}) \rightarrow (0/1)$**:** Takes as input a module's sidechain ledger state $\mathcal{L}_{\mathsf{sc,module}}$, a new block B of 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 the module, and 0 otherwise.

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

$\mathsf{Prune}(\mathcal{L}_{\mathsf{sc,module}}) \rightarrow (\mathcal{L}'_{\mathsf{sc,module}})$**:** Takes as input a module's sidechain ledger state $\mathcal{L}_{\mathsf{sc,sc}}$, and produces an updated state $\mathcal{L}'_{\mathsf{sc,sc}}$ in which all stale meta-blocks are discarded.

Security model. We aim to build a secure solution that preserves correctness and security of the underlying resource market. Starting with a secure market operating a secure blockchain (that satisfies liveness and safety [36]), deploying marketPrism preserves the invariant that the market remains secure and processes its traffic based on its original logic. Besides ledger security, a secure market addresses all service-related threats [24], including service theft (a client does not pay for a received service), service slacking (a server collects payments without offering services), service corruption (a server delivers a corrupt service), and DoS attacks (avoid publishing transactions issued by particular parties).

marketPrism aims to optimize the performance of a given resource market by boosting throughput and reducing confirmation delays as well as blockchain size. It does that by splitting the processing of market service-related workload among the module sidechains. These sidechains process their workload based on the original logic of the market. Thus,

in our security analysis, we show that marketPrism and all the techniques it introduces do not violate the security of the underlying resource market.

Adversary model. We assume the underlying resource market and its blockchain to be secure. For the sidechains, we account for three miner behaviors: honest, malicious who deviate arbitrarily, and honest-but-lazy who passively collaborate with the adversary, e.g., accepting blocks without validation. The adversary can corrupt miners but without going above the corruption threshold of the consensus protocol. As marketPrism adopts a PBFT-based consensus, the threshold of malicious miners in a committee cannot exceed f, where the committee size is $3f+1$ and $2f+1$ votes are needed to reach agreement. Similar to prior work [36, 44, 45, 58], we assume bounded message delivery, so any message will be delivered within Δ time delay. As we deal with public blockchains, the adversary can see all messages, reorder and delay them within Δ delay. The adversary is slowly adaptive [26]; it can corrupt miners only at the beginning of an epoch. Lastly, we deal with probabilistic polynomial time (PPT) adversaries who cannot break the security of the used cryptographic primitives with non-negligible probability.

4 marketPrism Design

marketPrism is a hybrid sharding-sidechain scaling framework that achieves the best of both worlds; supporting the high parallelism promised by sharding, while taking advantage of concise state change syncing, arbitrary data exchange and pruning offered by dependent-sidechains. marketPrism does that by introducing a functionality-oriented way for parallelism. As shown in Figure 1, it splits the resource market into logical functional modules, and assigns each module to a sidechain that handles this module's workload. By doing so, marketPrism eliminates the costly cross-sidechain transactions, a problem that hampered sharding solutions that randomly distribute the workload among shards.

In this section, we present the design of marketPrism including system setup, architecture and operation, handling interruptions, and its security. To simplify the discussion, we do that assuming a base market functionality covering the basic modules discussed in Section 2. Nonetheless, the same logic applies if a market has more modules, or if it supports multiple services each with its own modules.

4.1 Setup Phase

marketPrism is a service-type agnostic scalability framework; it can be used for any resource market regardless of the service type it offers. At the same time, marketPrism's generic architecture can be customized based on the market's modules and traffic patterns. In the setup phase (Figure 2), system designers determine the market's modules and traffic split among these modules. Each module will have a sidechain that will process the transactions related to this module. System designers also specify how these transactions should be summarized and define the mainchain state variables that will be synced based on these summaries. All these factors, in addition to the epoch length, round duration, and block sizes, are set in each sidechain public parameters $\mathsf{pp}_{\mathsf{sc}}$.

$\mathsf{Setup}(\mathcal{L}^0_{\mathsf{mc}}, \mathsf{modules})$: On input the mainchain genesis block and the market modules, generate configuration parameters:

- Epoch length ω.
- Public parameters $\mathsf{pp}_{\mathsf{module}}$ of each module's sidechain.
- Traffic classification.
- Summary rules for each module's sidechain.
- Mainchain summary state variables for all module sidechains $\mathsf{summary}_{\mathsf{variables,modules}}$.

Updates $\mathcal{L}_{\mathsf{mc}}$ to produce a new state $\mathcal{L}'_{\mathsf{mc}}$ containing summary state variables for all modules.

Initializes:

- $\mathsf{pendingTx}_{\mathsf{mc}}$, pending mainchain transaction pool.
- $\{\mathsf{pendingTx}_{\mathsf{module}}\}_{\mathsf{module}\in\mathsf{modules}}$, a set of pending transaction pools such that each module's sidechain has its own pool.

Outputs: epoch length ω, each sidechain's genesis block $\mathcal{L}^0_{\mathsf{sc,module}}$ (this block references $\mathcal{L}^0_{\mathsf{mc}}$) containing $\mathsf{pp}_{\mathsf{module}}$, traffic assignment and summary rules, and $\mathcal{L}'_{\mathsf{mc}}$.

Figure 2: System setup.

Module and traffic division. Parallelism by workload sharing is effective so long as each module can commit its assigned workload on its own — i.e., no transaction writes state owned by another module. Thus, in our functionality-oriented approach, a module represents a well-defined part of the market that achieves a particular purpose. For our abstract model, these include market matching, service-payment exchange, and dispute resolution. All service-related transactions under each module are directed to the sidechain assigned to that module. For example, $\mathsf{tx}_{\mathsf{ask}}$ and $\mathsf{tx}_{\mathsf{offer}}$ would pertain to the market matching module, $\mathsf{tx}_{\mathsf{serviceProof}}$ and $\mathsf{tx}_{\mathsf{servicePayment}}$ belong to the service-payment exchange module, and $\mathsf{tx}_{\mathsf{dispute}}$ goes into the dispute module.

The level of module granularity highly impacts system performance. For example, splitting the market matching module into two modules, one that handles $\mathsf{tx}_{\mathsf{ask}}$ and another that handles $\mathsf{tx}_{\mathsf{offer}}$, would lead to high volumes of (costly) cross-sidechain transactions. While merging, for example, the market matching and the dispute resolution modules into one would lead to an overloaded sidechain.

Traffic classification. We use a simple static approach, in which a transaction header contains field indicating its destination chain. This field includes a chain indicator, stating whether a transaction must be processed by the mainchain or

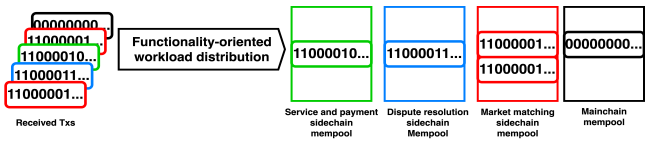


Figure 3: Annotated functionality-oriented load splitting.

one of the sidechains, and if it is the latter, there will be a module indicator, stating the destination module's sidechain. For example, with one byte prefix, first 2 bits of 00 means a mainchain transaction, while 11 means a sidechain transaction with the next 6 bits indicate which sidechain, e.g., 000001 is for the market matching sidechain, and so on. Figure 3 shows an annotation example based on our abstract market model.

Transaction summary rules. These are similar to those of chainBoost. For example, $\mathsf{tx}_{\mathsf{servicePayment}}$ and $\mathsf{tx}_{\mathsf{serviceProof}}$ are summarized by counting the number of proofs a server submitted in an epoch and its total payment, $\mathsf{tx}_{\mathsf{dispute}}$ are summarized by listing the outcome of a dispute and its proof, while $\mathsf{tx}_{\mathsf{ask}}$ and $\mathsf{tx}_{\mathsf{offer}}$ are summarized by the finalized agreements they produce encapsulated in $\mathsf{tx}_{\mathsf{agreement}}$. Different from chainBoost, in marketPrism summaries are distributed among the modules' sidechains based on the traffic they handle. Figure 4 presents the summary rules for the abstract resource market, which can be extended and adapted to match any additional modules defined during system setup.

4.2 Sidechain Management

The mainchain and sidechains in marketPrism work in parallel. During each epoch, a committee from the mainchain miners is elected for each sidechain. Each committee runs a PBFT-based consensus protocol. Every round within an epoch, the committee agrees on their shared state, then produces a new meta-block recording all processed transactions in that round. Then, in the last round of an epoch, the committee produces a summary-block summarizing the state changes induced by the meta-blocks produced within the epoch.

To reflect the state changes induced by sidechain transaction processing, each sidechain committee issues a sync-transaction (immediately after producing a summary-block) to sync the mainchain based on the summaries. Thanks to module splitting, sync-transactions of the sidechains in marketPrism can be processed independently. Also, since the mainchain stores the system state, any module sidechain can retrieve the needed state changes from the mainchain. Once a sync-transaction is confirmed on the mainchain, the corresponding epoch meta-blocks on the corresponding sidechain are discarded.

Like a regular blockchain, all miners have a copy of the mainchain. In marketPrism, they also have copies of all sidechains. This is done to speed up committee rotation. That is, when a committee for a sidechain is elected, each

Input: Epoch meta-blocks $\mathsf{B}^{\mathsf{meta},1}_{\mathsf{module}},\ldots,\mathsf{B}^{\mathsf{meta},n}_{\mathsf{module}}$ for $\mathsf{module} \in \{\mathsf{servicePaymentExchange}, \mathsf{dispute}, \mathsf{marketMatch}\}$.
Initialize a summary structure $\mathsf{sum}_{\mathsf{module}}$ for each module.
for $i \in \{1,\ldots,n\}$ and every $\mathsf{tx} \in \mathsf{B}^{\mathsf{meta},i}_{\mathsf{module}}$ **do**
  if $\mathsf{module} = \mathsf{servicePaymentExchange}$ **then**
    if $\mathsf{tx.txtype} = \mathsf{tx}_{\mathsf{serviceProof}}$ **then**
      // cid is the service contract ID
      ++ $\mathsf{sum}_{\mathsf{module}}[\mathsf{tx.cid}]$.count
    elseif $\mathsf{tx.txtype} = \mathsf{tx}_{\mathsf{servicePayment}}$ **then**
      $\mathsf{sum}_{\mathsf{module}}[\mathsf{tx.cid}]$.payment += tx.amount
  elseif $\mathsf{module} = \mathsf{dispute}$ **then**
    $\mathsf{sum}_{\mathsf{module}}[\mathsf{tx.cid}] = (\mathsf{tx.proof}, \mathsf{tx.outcome})$
  elseif $\mathsf{module} = \mathsf{marketMatch}$ **then**
    // tx under this module refers to $\mathsf{tx}_{\mathsf{agreement}}$ which
    // 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)
    $\mathsf{sum}_{\mathsf{module}}[\mathsf{tx.cid}] = (\mathsf{tx.s}, \mathsf{tx.cl}, \mathsf{tx.terms})$
Output $\mathsf{sum}_{\mathsf{module}}$ for all modules.

Figure 4: Summary rules (assuming one server per service contract, if the market assigns multiple servers per contract then indexing should indicate the server ID).

committee member must have the latest state of this sidechain ledger so they can work on extending it. By requiring miners to have copies of the sidechains, committees will avoid the excessive delays of retrieving the sidechain ledger from the prior epoch committee. Furthermore, this also allows miners in the system to read state owned by the different system modules as a local lookup.

Note that correctness of layer 2 (i.e., sidechain) transaction processing is established by the security of consensus; the sidechain committee processes its workload using the original logic of the resource market and only agrees on valid state changes. No need for additional costly mechanisms such as ZK proofs or contestation periods as in rollups.

Weighted miner assignment. chainBoost and sharding solutions employ a random committee election approach, which is also supported in marketPrism. However, marketPrism recognizes that some modules are more critical than others in their impact on market operation, so interruptions in these modules are more devastating than others. For example, the dispute solving module is critical as it decides sanctions for misbehaving parties and affects their participation in providing/receiving service. Given that miners would have different vested interest in the system, those with large involvement are incentivized to behave honestly. By assigning them to critical modules, we can reduce the chances of interruptions and even have fast recovery if interruptions happen.

To this end, we devise a *weighted miner committee election*, facilitating cryptographic sortition to run this election locally (thus reducing overhead). It classifies miners into

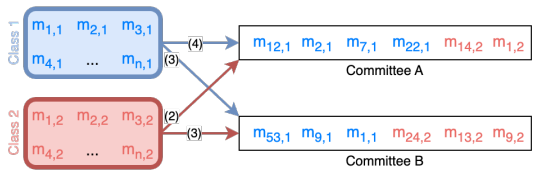


Figure 5: An example of weighted miner election, where a more critical module committee (A) is formed using 4 class 1 and 2 class 2 miners, and the other committee (B) has a more balanced selection, by selecting 3 miners from each class.

classes based on their participation (represented as a score) in the system. Then, it allocates a plurality of the critical sidechain committee memberships to miners positive participation, while membership of less critical committees will be more balanced, i.e., have same number of miners from all classes on expectation.

A simple example is shown in Figure 5 assuming two sidechains A and B (A is more critical than B) each requiring a committee of 6 miners. Here, we have $N = 2n$ miners split into two classes where class 1 contains n miners with the highest scores ($m_{i,1}$ and $m_{i,2}$ refer to miners in class 1 and class 2, respectively). The more critical chain A picks 4 miners from class 1 and the rest from class 2, while chain B uses a balanced selection, picking 3 miners from each class.

The weighted miner committee election algorithm. We capture positive participation via scores assigned to miners. These scores account for important parameters reflecting vested interest in the system. In detail, each miner calculates a score s based on its mining power P_m, previous participation in sidechain consensus S_c, and the disputes it was involved in D. This is computed as $s = \alpha P_m + \beta S_c - \gamma D$, where $\alpha+\beta+\gamma = 1$ with values calibrated by system designers. The mining power P_m can be computed by tracking how many blocks a miner mined (in proof-of-work), or its stake amount (in proof-of-stake), or amount of service it provides (in systems that rely on other resources), etc. For S_c, this is computed based on the number of agreements a miner has participated in for any sidechain (by observing signatures when running PBFT to agree on meta and summary blocks). While D represent the number of disputes for which the miner is found guilty.[3]

Then, the miners in the system are ranked into C classes based on this score, such that $C \geq |\text{modules}|$. Say we have a total of N miners, these miners are ranked based on their scores into C classes of equal sizes. This is done by having the highest scoring $\mu = \lfloor \frac{N}{C} \rfloor$ miners be class 1, the following μ miners in class 2, and so on. Then, each module's sidechain will be randomly assigned a number of miners $n_{c,\text{module}}$ from each class c, where $c \in \{1,\ldots,C\}$, based on the module priority. Note that miners' scores are computed based on public values from the mainchain, allowing anyone to compute and verify the class of any miner.

This construction also determines how the election behaves under Sybil attacks. Recall that miner identities are Sybil-resistant, so the mining power term P_m cannot be duplicated: an adversary that spreads its holdings over several identities spreads P_m among them as well. The participation term S_c is constrained by the class ranking itself. Fresh identities all enter at zero score and populate the lowest class. Thus, adding identities enlarges the population competing for that fixed share and lowers the probability that any one of them is elected. Since committee seats are the only channel through which S_c accrues, Sybil identities dilute their own access to it, and an adversary cannot accelerate its ascent into the high-participation classes. That ascent requires sustained participation across many epochs, which is precisely the behavior the score is meant to reward. The same effect raises the cost of abandoning an identity involved in disputes and re-entering with a clean one, as the replacement must re-earn both P_m and S_c from the lowest class.

The election mechanism is depicted in Algorithm 1. Each miner computes its score s, and the score set of other miners S, and uses these to determine its class c (S can be cached locally and updated each epoch). A miner determines if it is elected by flipping a coin biased by the probability of its class being elected for any committee. This is done by, first, computing a miner-specific random value using a verifiable random function (VRF) [38] ($seed_1$ and $seed_2$ are public per-epoch fresh randomness computed in the system using, e.g., a random beacon). Then, the normalized VRF output is compared to the probability of being selected in any committee that requires class c miners, which is $\frac{n_{c,all}}{\mu}$.[4] If a miner is elected, then it determines the exact committee based on a second normalized VRF output. That is, each sidechain is assigned a sub-range within [0,1) based on the number of class c miners it needs, and a miner finds the sub-range where its normalized VRF output belongs. At the end, the miner publishes its score and election result (VRF outputs, proofs, and assignment) so anyone can verify the election outcome.

In Appendix A, we show how to configure the number of miners from each class to achieve a target committee failure probability (chosen based on module importance).

Agreeing on a shared view. To ensure consistent block validation, members of each module's consensus committee must agree on a common view of the system. At the beginning of each consensus round for module m, every member of its consensus committee sends the leader a signed *view report*

[3] Recall that in many resource markets servers also play the role of miners, which is the assumption we have here.

[4] So, the miner-specific coin toss is $\frac{rnd_2}{2^{|rnd_2|}}$. The original expression for the bias, or target probability, is $\frac{\binom{\mu-1}{n_{c,all}-1}}{\binom{\mu}{n_{c,all}}}$, which when simplified produces $\frac{n_{c,all}}{\mu}$. It captures the following: if you choose $n_{c,all}$ miners uniformly at random from a population of μ c-class members (recall that μ is the total number of miners belonging to any class), what is the probability that a miner (of class c) is included among the $n_{c,all}$ miners selected.

Algorithm 1 Weighted miner election

procedure ELECT($seed_1, seed_2, s, S = [s_i \text{ for } i \in \{0, \cdots, N\}]$)

//Percentile of miner's score compared to all scores

$\upsilon = 1 - GetPercentile(s, S)$

//The class of the miner from the percentile, recall that C is //the total number of classes

$c = \lceil \upsilon \cdot C \rceil$

//Total number of class c miners needed for all sidechains. //For simplicity, we iterate over the modules //module $\in$ modules using index i.

$n_{c,all} = \sum_{i=1}^{|\text{modules}|} n_{c,\text{module}_i}$

//Compute a miner-specific VRF output

$rnd_1, \pi_1 \leftarrow VRF_{sk}(seed_1 || pk)$

//Normalize VRF output and flip a biased coin to check if the //miner is elected for its class for any committee. //Recall that $\mu = \lfloor \frac{N}{C} \rfloor$, which is the size of any class.

$elected = \frac{rnd_1}{2^{|rnd_1|}} < \frac{n_{c,all}}{\mu}$

if *elected* **then**

//Compute another miner-specific VRF output.

$rnd_2, \pi_2 \leftarrow VRF_{sk}(seed_2 || pk)$

//Normalize VRF output and determine the module (i.e., //the module sidechain assignment for the miner) based on //the position in the range [0,1).

if $0 \leq \frac{rnd_2}{2^{|rnd_2|}} < \frac{n_{c,\text{module}_1}}{n_{c,all}}$ **then**

$a \leftarrow \text{module}_1$

else if $\frac{rnd_2}{2^{|rnd_2|}} < \frac{n_{c,\text{module}_1} + n_{c,\text{module}_2}}{n_{c,all}}$ **then**

$a \leftarrow \text{module}_2$

...

else

$a \leftarrow \text{module}_{|modules|}$

end if

end if

return $s, rnd_1, \pi_1, rnd_2, \pi_2, a$

end procedure

containing, for each other chain ($\mathcal{L}_{\text{mc}}$ and $\mathcal{L}_{\text{sc},j}$ for all $j \neq m$), the height of its latest finalized block and the corresponding PBFT commit certificate.[5] The leader collects $2f+1$ valid reports and computes the component-wise minimum of the reported heights, identifying the longest prefix of each chain shared by all reporting members. These prefixes collectively form a *snapshot*, which the leader includes in the proposed block's header. An honest committee member votes for the proposal only if the snapshot is (i) at least the snapshot of the proposed block's parent, (ii) a valid prefix of its local view, and (iii) all proposed transactions can be validated against it.

[5] A commit certificate is the combination of committee member votes, i.e., their signatures, agreeing on the block.

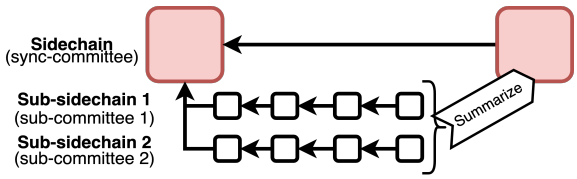


Figure 6: Scaling using sub-sidechains.

4.3 Hierarchical Intra-module Scaling

In resource markets, it is expected that some modules would be more overloaded than others, making the functionality-oriented workload distribution imbalanced. For example, the service-payment exchange module, whose traffic scales linearly with the number of servers/clients, would be overloaded. To address this issue, marketPrism introduces a *hierarchical intra-module scaling* that employs the sharding-sidechain hybrid paradigm within the same module. This is done by spinning out sub-sidechains that process the module's traffic in parallel. To reduce on-chain traffic, the results of all sub-sidechains are summarized into one summary-block, and hence, only one sync-transaction is issued.

All sidechains with full meta-blocks, and whose mempools cannot be emptied in one epoch, are marked by their committees as heavy. The committee estimates the number of sub-sidechains needed to process the full traffic, and reports that in the sync-transaction. This triggers an increase in the number of committees to be elected for the next epoch to account for these sub-sidechains. If the current miner population cannot accommodate additional large-enough committees, then fewer sub-sidechains will be created.[6] These sub-sidechains record only meta-blocks as shown in Figure 6. A sync-committee will be elected from the members of the sub-committees (following the same distribution of these sub-committees) to produce one summary-block and issue a sync-transaction.

As an example, suppose miners are divided into 4 classes (A, B, C, D), each of size 50 miners. The market has two heavy modules and a few non-heavy ones. Non-heavy modules need $(27A, 30B, 20C, 10D)$ miners for their committees, leaving $(23A, 35B, 30C, 40D)$ miners for the heavy modules. Assume the first module needs 5 sub-sidechains each with a committee of $(4A, 3B, 2C, 1D)$ miners, and the second module needs 3 sub-sidechains with a combined $(9A, 12B, 3C, 6D)$ miners. These cannot be satisfied based on the current miner population. Thus, we form committees as close as possible to modules' needs, having 4 sub-sidechains using $(16A, 12B, 8C, 4D)$ miners for the first module, and 2 sub-sidechains using $(6A, 8B, 2C, 4D)$ miners for the second.

When scaling a module, transactions are divided among the sub-sidechains randomly, in a way that guarantees that related

[6] The miner population size can be determined by tracking miner activity on the mainchain, or by having an identity sidechain where miners register when joining (similar to the identity shard in [45]).

transactions end up in the same sub-sidechain. In our abstract resource market, the service-payment exchange, and the dispute modules can be assigned based on their service contract identifier. For the market matching module, a standing ask or offer is posted randomly using its own identifier, and all transactions referencing it are routed to that same sub-sidechain. Agreements are assigned based on the service contract ID they define.

4.4 Eliminating Cross-sidechain Transactions

As mentioned before, system designers identify the modules of the resource market and assign a sidechain to each. This division follows the general functionality of resource markets, centered mainly around market matching, payment–service exchange, and dispute resolution. Each module's sidechain processes that module's transactions, while information originating elsewhere—e.g., checking that a client created an escrow—is read from the miner's local copy of the mainchain or of another sidechain (recall that in marketPrism all miners maintain the mainchain and keep copies of the other sidechains to speed up committee switching). We now show that this functionality-oriented partition admits *no cross-sidechain transactions*.

Definition 4.1 (Cross-chain transaction)**.** *Under an assignment of transactions to chains, a transaction* tx *is* cross-chain *if committing its effects atomically requires a decision from more than one consensus instance.*

Definition 4.2 (Properties of a system's partition scheme)**.** *A system's partition scheme satisfies*

- Write locality *if the write set of every transaction* tx *with home module m is contained in $\mathcal{L}_{\mathsf{sc},m}$; and*
- Local read availability *if, for every* tx *with home module m, each value in the read set of* tx *is determined by state that every honest member of $\mathcal{C}_m$ holds locally and agrees upon at execution time.*

Lemma 4.1. *The functional partition of marketPrism satisfies write locality and local read availability.*

Proof. Write locality. Since in marketPrism modules were carved along functional boundaries, no transaction type has a write set spanning two ledgers.

Local read availability. During a round, miners in committee agree on their shared state of all $\mathcal{L}_{\mathsf{sc},\cdot}$ and of $\mathcal{L}_{\mathsf{mc}}$. Each sidechain runs PBFT and thus has deterministic one-block finality, so no honest miner ever holds a shared state that is later reverted, or that diverges from another honest miner's. Hence, for any value v read by tx, every honest member of $\mathcal{C}_m$ resolves v identically from local state. □

Theorem 4.2. *A system's partition scheme satisfying write locality and local read availability has no cross-chain transactions. In particular, by Lemma 4.1, marketPrism has none.*

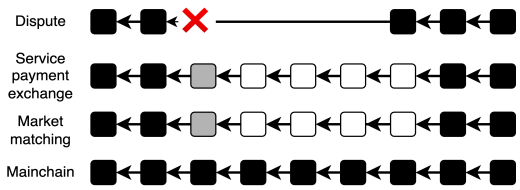


Figure 7: Autorecovery upon failure of the dispute sidechain (gray block signals interruption detection, and white blocks indicate empty blocks).

Proof. Fix tx with home module m and committee $\mathcal{C}_m$. By local read availability, executing tx is a deterministic function of tx and the shared state already held by honest member of $\mathcal{C}_m$; no external committee needs to certify any input, so $\mathcal{C}_m$ reaches agreement on the resulting write set within a single PBFT instance. By write locality, that write set lies entirely in $\mathcal{L}_{\mathsf{sc},m}$, so the decision of $\mathcal{C}_m$ suffices to commit it, and atomicity is that of a single consensus decision. No second instance participates, so tx is not cross-chain. □

4.5 Resilience to Interruptions

marketPrism's sidechains have a mutual-dependency relation with the mainchain as they share the market workload. marketPrism also creates a module criticality hierarchy, e.g., the dispute module issuing penalties against misbehaving miners can impact market matching and service-payment exchange modules. Moreover, there is the mainchain rollback issue, where abandoning recent blocks on its blockchain may lead to losing sync-transactions. Thus, to ensure system operation continuity and security, marketPrism needs an *autorecovery protocol* to handle three types of interruptions: rollbacks, intra-module sidechain interruptions, and inter-module issues.

Handling rollbacks and intra-module recovery. marketPrism uses the same mechanisms as chainBoost [55]. Mass-syncing allows a committee to issue a sync-transaction for its epoch and all prior epochs impacted by a rollback. While view-change addresses misbehaving committee leaders, and backup committees handle misbehaving primary committees, thus ensuring intra-module recovery.

Inter-module issues. We introduce a simple mechanism that allows a critical module to recover without impacting the liveness of other modules' sidechains. Say there is module_1 whose behavior depends on information processed in a critical module module_2. The miners in module_1's sidechain committee detect that module_2 is interrupted if they do not see new blocks on module_2's sidechain after a timeout η. At that moment, module_1's committee mines empty meta-blocks until module_2 recovers. Figure 7 depicts this protocol; the dispute module is a critical module, e.g., the service-payment module must not process payments for a server that violates the service agreement. If the dispute module experiences an interruption, the two other modules stop transaction pro-

cessing, and mine empty blocks until the dispute module recovers. The mainchain continues to operate normally but will have empty summaries from these sidechains during the interruption.

4.6 Security

In Appendix B, we analyze the security of marketPrism covering the failure probability of the autorecovery protocol, and showing that marketPrism design preserves liveness and safety of the underlying resource market.

5 Implementation and Setup

To evaluate the performance gains that marketPrism can achieve, we implement a proof-of-concept prototype emulating a distributed file storage market, inspired by Filecoin [7], as a use case. The prototype defines modules and executes the marketPrism protocol, using synthetic workloads corresponding to the patterns observed in Filecoin (discussed below). Execution progress is measured in logical blockchain rounds, each of which comprises the time to share the state and run consensus on the block.[7] Our code repository can be found at [13].

Sidechain implementation. We extend the source code from [55] to have sidechains for the following modules:

Market matching: Here, a client asks for file storage service for a specific duration, file size, and price, via a contract-proposal transaction, and the server responds with a contract-response transaction that either approves or declines a client's ask. If a server accepts a proposal, it also issues a contract-deal transaction, which is the client's contract-proposal (originally signed by the client) signed also by the server; it sets the service terms and the client escrow used to pay for service. Otherwise, the two parties keep negotiating. In our implementation, the service contracts are assigned negotiation durations from a normal distribution. Whenever a contract expires, it enters into a negotiation phase that ends with the issuance of a contract-deal and reactivation of the contract.

Service-payment exchange: Servers prove storing files by submitting compact proofs-of-retrievability (PoR) [59] transactions. The total payments for valid submitted PoRs is dispensed to the server once the storage contract ends.

Dispute resolution: When a miner observes a misbehavior (i.e., a server submits an invalid PoR transaction), it issues a dispute-transaction containing the proof of misbehavior. The miners in this module verify the proof and, if correct, publish the outcome on the module's sidechain. In our implementation, the misbehavior rate is a configurable parameter.

[7] Since the prototype does not emulate propagation delay, all committee members hold identical views, and so we do not implement view report exchange as it would not change our measurements. Its cost in practice is in-round latency that the round duration must accommodate.

Each module has a single sidechain, except the service-payment exchange module that can have up to three sub-sidechains. All sidechains use same block size, round duration, and epoch length. For sidechain consensus, we use BLS Collective Signing (CoSi)-based PBFT inspired by [44] as implemented in [2] using the default committee election option (random cryptographic sortition). We evaluate the impact of our weighted sortition in a separate experiment.

Traffic generation. We use the same traffic generation patterns and sizes from [55], and augment them with dispute transactions (515 bytes) and contract-deals (716 bytes). We apply the traffic distribution observed in Filecoin using [9]; 98% service-related transactions (destined to the sidechains) and 2% currency transfers (destined to the mainchain). Transactions are created at the beginning of each mainchain round.

Comparison to chainBoost. We extend the source code from [55] to support the new transactions introduced above. We use this extended implementation to run benchmark chainBoost under the same workload as marketPrism.

Comparison to sharding. We implement a proof-of-concept sharded resource market (that employs random workload distribution) inspired by Omniledger [45] and Rapidchain [64]. Miners are assigned to shards randomly at the beginning of each epoch. Cross-shard transactions (those that have an input from another shard) are handled using inter-committee transaction forwarding as in Rapidchain [64]. That is, when a transaction has an input in another shard, a cross-shard transaction is forwarded to the destination shard who puts this transaction in its mempool to be processed.

Evaluating autorecovery under different committee election mechanisms. We write a C++ program to evaluate marketPrism's autorecovery protocol under random and weighted committee elections. The program configures the percentages of lazy and malicious miners in a large miner population. Then, it creates one primary and several backup committees using both election approaches. Then, the program tracks the number of committee failures until the system recovers or fails (where a committee fails if the assigned number of adversarial nodes violates liveness). This allows quantifying the time needed to recover.

Experiment setup. We deploy our experiments on a computing cluster composed of 10 12-core, 130 GiB RAM, host-pinned VMs, connected with a 1 Gbps network link. Unless stated otherwise, our experiments run over a network of 8000 nodes, each serving 8 contracts. An experiment lasts for 61 mainchain rounds, with an epoch length $\omega = 10$ mainchain rounds corresponding to $\zeta = 30$ sidechain rounds, and a block size of 1 MB. A sidechain committee contains 500 miners. For the traffic pattern described before, 10% of the active contracts generate disputes. We measure the time in mainchain blocks and convert to Filecoin's block duration: $\tau_{mc} = 30$

seconds. We collect experiment data using SQLite [18]. In reporting our results, we measure the following metrics:

Throughput: The average number of transactions processed per mainchain round, including transactions appearing in a mainchain block B_{mc} (or round) and those appearing in all sidechain blocks S during the same mainchain round. We calculate throughput using the formula $T = \frac{\mathsf{Count}(B_{\mathsf{mc}}.txs)+\sum_{B_{\mathsf{sc}}\in S}\mathsf{Count}(B_{\mathsf{sc}}.txs)}{\tau_{\mathsf{mc}}}$

Confirmation time: The time (in seconds) a transaction takes to be confirmed from the moment it enters the queue. As we use PBFT, a transaction is confirmed once it is published in a meta-block. This metric is given by the following formulas: $L_{\mathsf{mc}} = (r_{c,\mathsf{mc}} - r_{q,\mathsf{mc}})\tau_{\mathsf{mc}}$, $L_{\mathsf{sc}} = (r_{c,\mathsf{sc}} - r_{q,\mathsf{sc}})\frac{\omega}{\zeta}\tau_{\mathsf{mc}}$, where $r_{q,i}, r_{c,i}$ for $i \in \{\mathsf{mc},\mathsf{sc}\}$ are the rounds where the transaction was queued and confirmed, respectively.

Storage footprint: The persistent blockchain size (mainchain blocks and sidechain summary-blocks) in megabytes. It is calculated as $S = \sum_{B_{\mathsf{mc}}\in\mathsf{mc}}|B_{\mathsf{mc}}| + \sum_{B_{sum}\in\{B_{\mathsf{sc}}\in\mathsf{sc}|B_{\mathsf{sc}}.type=summary\}}|B_{sum}|$

Time to autorecover: The (worst case) average time needed to autorecover from interruption in a sidechain. This is obtained from the monte-carlo simulation.

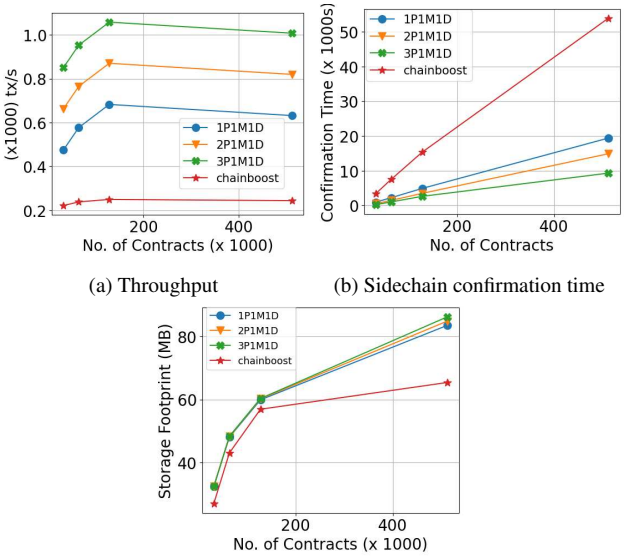
After an experiment run ends, we continue processing traffic to empty the queues so we can measure confirmation times accurately. All other metrics are reported when the run ends.

6 Performance Evaluation Results

We test the impact of several parameters, including sidechain block size, ratio of sidechain to mainchain rounds in an epoch, and the number of sub-sidechains in heavy modules. We also compare marketPrism to chainBoost and sharding, and we study the impact of our weighted committee selection in reducing autorecovery overhead. To differentiate between configurations of marketPrism, we notate them as aPbMcD-marketPrism, where a is the number of sub-sidechains in the service-payment exchange (P) module, b is the number of sub-sidechains in the market matching (M) module, and c is the number of sub-sidechains in the dispute (D) module.

Scalability. We vary the number of service contracts within $\{32K, 64K, 128K, 512K\}$. During each epoch, a contract would result in a PoR transaction per mainchain round when active, a service payment when it expires, and a number of deal negotiation transactions before its re-activation. We compare chainBoost to various configurations of marketPrism.

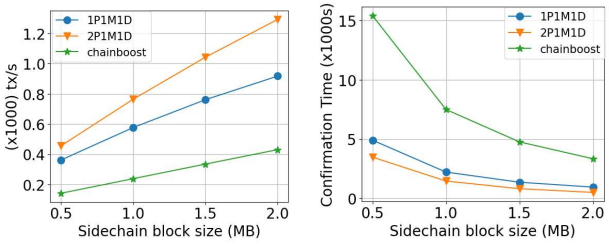
Throughput-wise, as shown in Figure 8a, marketPrism significantly outperforms chainBoost. At 32K contracts, 1P1M1D-marketPrism and 3P1M1D-marketPrism have 2x and 3.8x, respectively, the throughput of chainBoost. We also notice that throughput peaks at 128K contracts, realizing a 4x increase. At 512K contracts, all versions of marketPrism show a slight drop in throughput; this is due to the congestion in the system that exceeds the capacity of the sidechains.



(a) Throughput (b) Sidechain confirmation time

(c) Storage footprint

Figure 8: Scalability results under various workloads.



(a) Throughput (b) Sidechain confirmation time

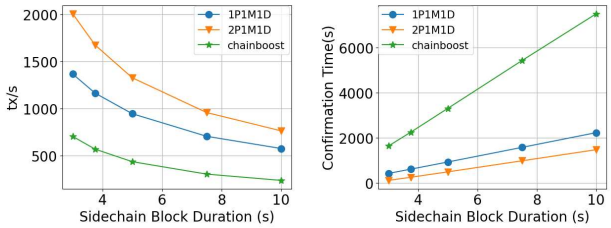
Figure 9: Impact of block size.

For confirmation time, marketPrism also outperforms chainBoost (Figure 8b). At 32K contracts, this time is 3,380 s for chainBoost while it is 944 s for 1P1M1D-marketPrism—a 3.5x reduction in latency. The gap amplifies as workload increases; at 512K contracts, chainBoost's confirmation time is 53,860 s, while it is 19,393 s for 1P1M1D-marketPrism.

In terms of storage footprint, marketPrism has a similar trend as chainBoost until up to 128K contracts. For 512K contracts, we notice that marketPrism needs around 20 MB of additional storage. This is because under congestion, chainBoost (which has one sidechain) will process less workload (service contracts and disputes), leading to smaller summary-blocks. marketPrism, with its multi-sidechain design, can deal with larger workloads, making its summary-blocks larger.

Impact of block size. We vary the sidechain block size within $\{0.5, 1, 1.5, 2\}$ MB, and compare chainBoost to various configurations of marketPrism in terms of throughput and confirmation time.

In Figure 9a we observe that marketPrism outperforms chainBoost in terms of throughput, even at smaller block sizes,



(a) Throughput (b) Sidechain confirmation time

Figure 10: Impact of sidechain round duration.

where the difference is 2.5x under 1P1M1D-marketPrism, and 3.22x under 2P1M1D-marketPrism. As expected, marketPrism's throughput improves at larger block sizes, achieving around 1293 tx/s under 2P1M1D an a block size of 2MB. For confirmation time, Figure 9b shows that, at the smallest block size, 1P1M1D-marketPrism and 2P1M1D-marketPrism achieve 3.13x and 4.42x improvement, respectively, over chainBoost (and these improvements reach 3.5x and 6.5 at the largest block size). This is again a result of the congestion that chainBoost experiences as it has one sidechain.

Overall, marketPrism with its multi-sidechain design and the hierarchical intra-module workload sharing for heavy modules, coupled with the right sidechain block size, achieves a high parallelism level. This makes marketPrism a more viable solution for resource markets in practice than chainBoost.

Number of sidechain rounds per epoch. We vary this number within $\{30, 40, 60, 80, 100\}$. An epoch length in our setup is 10 mainchain rounds, so the sidechain round durations are $\{3, 3.75, 5, 7.5, 10\}$ s. We report the maximum throughput that can be achieved and the confirmation time.

Figure 10a shows that marketPrism can achieve a throughput of 2000 tx/s when the sidechain round is 3 s (i.e., 10 sidechain rounds per mainchain round); a 2.66x improvement over chainBoost. Throughput drops as the sidechain block duration increases because blocks now would contain the same number of transactions (block size is fixed) but are published at a slow rate. A similar trend is observed for confirmation time in Figure 10b; for a 3 s sidechain round, chainBoost has 3.93x and 5.11x the confirmation time achieved by 1P1M1D and 2P1M1D, respectively. Confirmation time increases for longer round durations. As such, the use of short rounds will promote the performance gains achieved by marketPrism. However, system designers must account for the fact that a round duration must cover the time needed for consensus on the sidechain when configuring the system parameters.

Comparison to sharding. We compare marketPrism to a sharding-based resource market (denoted as ShardedMarket). The sharded version has the same number of shards as the number of chains (both main and side chains) in marketPrism's version of the market. We report throughput, confirmation time, storage footprint, the latency overhead added by cross-chain transactions, and their ratio (CTR) for a 1P1M1D-marketPrism and ShardedMarket with four shards, under a workload of 128K contracts.

Table 2: Comparing ShardedMarket vs. 1P1M1D-marketPrism.

Solution	Throughput (tx/s)	Confirmation Time (s)	CT Latency Overhead (s)	CTR %	Storage (MB)
ShardedMarket	269.48	13419.92	2966.09	21.08	244
1P1M1D-marketPrism	672.38	3868.75	0	0	62

As shown in Table 2, marketPrism outperforms sharding in terms of throughput and confirmation time. It achieves 2.5x the throughput of ShardedMarket. This is due to the sidechain design allowing for publishing blocks at a faster rate than the mainchain, thus processing traffic at a faster rate than sharding. marketPrism achieves 3.5x improvement in latency. This is due to the fast traffic processing and the elimination of cross-chain transactions, which constitute 21.08% of all transactions in ShardedMarket, and add 2966 seconds of latency overhead to the confirmation time. Storage-wise, since ShardedMarket does not prune blocks, the system's total storage grows to 244 MB, which is 3x the cost incurred by marketPrism.

Autorecovery overhead. We evaluate the positive impact of marketPrism's weighted miner election mechanism on system resilience. We configure a network of 10^6 nodes, of which $(p_l, p_A) \in \{0.25, 0.3, 0.33\}^2$ are lazy and malicious miners, respectively. We organize the network into committees of size $S_c = 10^4$. We set the time needed for a backup committee to detect misbehavior and step in to be 5 min (Following from Appendix D.3 in [55], this is the worst case when the system detects misbehavior at the end of an epoch, which spans 5 min in our experiments). We compare the default (random) and weighted committee elections under two miner classes C1 and C2. For C1, we vary its configuration to have 15%, 30%, and 45% of the total adversarial nodes (C2 would contain the rest of the adversarial nodes—85%, 70%, and 55%, respectively). For the weighted election, we vary the committee configuration to have 40%, 50%, and 60% of its members from C1 and the rest from C2. We use the notation WXX-AYY, where XX is the percentage chosen from C1 and YY is the adversarial node percentage in C1. We conduct 10000 runs and report the average time needed to recover.

Our results can be found in Figure 11. As expected, under both random and weighted election, time to recover (which includes the time needed for backup committees step in until a safe committee is found—if any) depends on the percentage of misbehaving nodes in the miner population. A larger percentage means a larger chance of having a misbehaving committee that violates liveness (i.e., the total number of non-voting nodes exceeds $f+1$ in the committee). However, weighted election (where the larger committee share is selected from a class with fewer adversarial nodes) performs better than random election, reducing the average autorecov-

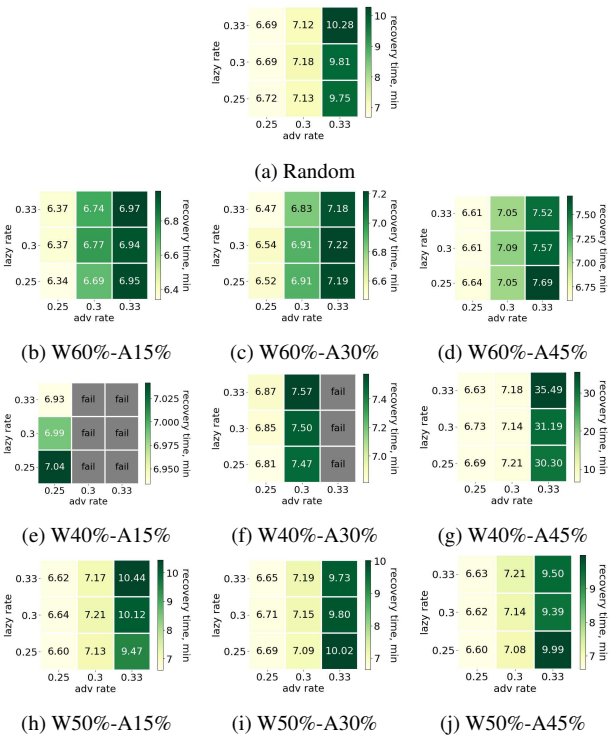


Figure 11: Average autorecovery time for random and 2-class weighted miner committee election under different adversarial distributions.

ery time by up to 3.31 min. This confirms the positive impact of our weighted approach in promoting security and reducing downtime of critical modules.

Our experiments also show that a low-priority weighted sortition for less critical modules harms performance and may compromise security. In other words, using a weighted election where the majority of the committee miners are picked from classes with higher adversarial miners can slow down the autorecovery by up to 35 min, and even leads to failures. Thus, a more balanced approach (that is closer to random sortition) for less critical modules is more favorable.

7 Conclusion

We introduced marketPrism, a secure layer-2 scaling solution for decentralized resource markets. marketPrism encompasses a new hybrid sidechain-sharding architecture allowing for high parallelism of workload processing. It splits the resource market into functional modules, each of which is assigned its own sidechain. This is done without introducing any cross-chain transactions, which hampered earlier sharding solutions. We analyzed the security of our system and conducted thorough performance evaluations. Our results show that marketPrism can achieve significant performance gains and improve threat resilience.

Acknowledgment

This work is supported by the National Science Foundation (NSF) under grant No. CNS-2226932.

AI Usage

This paper was edited for grammar and editorial changes using Overleaf and Grammarly. The class composition and autorecovery analyses were verified using Claude.

References

[1] Can i run a verifier - optimism docs. https://help.optimism.io/hc/en-us/articles/4413155125403-Can-I-run-a-verifier.

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

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

[4] Data availability in zksync. https://docs.zksync.io/zksync-protocol/rollup/data-availability.

[5] Eip-4844. https://eips.ethereum.org/EIPS/eip-4844.

[6] Ethereum blockchain explorer. https://etherscan.io/chart/tx.

[7] Filecoin. https://filecoin.io/.

[8] Finality - zksync era docs. https://era.zksync.io/docs/reference/concepts/finality.html.

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

[10] Golem. https://golem.network/.

[11] Google Drive users are reporting the loss of months of data — tomshardware.com. https://www.tomshardware.com/software/cloud-storage/google-drive-users-are-reporting-the-loss-of-months-of-data.

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

[13] marketprism code. https://github.com/CSSL-UConn/marketprism-release.

[14] Optimism. https://optimism.io/.

[15] Optimistic rollup is not secure enough than you think. https://medium.com/onther-tech/optimistic-rollup-is-not-secure-enough-than-you-think-cb23e6e6f11c.

[16] Polkadot. https://polkadot.com/.

[17] Session. https://getsession.org/.

[18] Sqlite. https://www.sqlite.org/index.html.

[19] The Pantheon of Zero Knowledge Proof Development Frameworks. https://blog.celer.network/2023/08/04/the-pantheon-of-zero-knowledge-proof-development-frameworks/.

[20] The state of Arbitrum's progressive decentralization | Arbitrum DAO - Governance docs — docs.arbitrum.foundation. https://docs.arbitrum.foundation/state-of-progressive-decentralization.

[21] Withdrawal flow — docs.optimism.io. https://docs.optimism.io/stack/transactions/withdrawal-flow.

[22] [zip-4] reduce the execution delay from 21 hours to 3 hours. https://forum.zknation.io/t/zip-4-reduce-the-execution-delay-from-21-hours-to-3-hours/373/3.

[23] zksync. https://zksync.io/.

[24] Ghada Almashaqbeh, Allison Bishop, and Justin Cappos. Abc: A cryptocurrency-focused threat modeling framework. In *INFOCOM Workshops*, 2019. https://doi.org/10.1109/INFCOMW.2019.8845101.

[25] Nasreen Anjum, Dmytro Karamshuk, Mohammad Shikh-Bahaei, and Nishanth Sastry. Survey on peer-assisted content delivery networks. *Computer Networks*, 2017. https://doi.org/10.1016/j.comnet.2017.02.008.

[26] Zeta Avarikioti, Antoine Desjardins, Lefteris Kokoris-Kogias, and Roger Wattenhofer. Divide & scale: Formalization and roadmap to robust sharding. In *Structural Information and Communication Complexity*, 2023. https://doi.org/10.1007/978-3-031-32733-9_10.

[27] 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.

[28] Stefanos Chaliasos, Itamar Reif, Adrià Torralba-Agell, Jens Ernstberger, Assimakis Kattis, and Benjamin Livshits. Analyzing and benchmarking zk-rollups. In *AFT*, 2024. https://doi.org/10.4230/LIPIcs.AFT.2024.6.

[29] George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. In *NDSS*, 2016. https://doi.org/10.14722/ndss.2016.23187.

[30] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, 2015. https://doi.org/10.1007/978-3-319-21741-3_1.

[31] Devdatt P Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *BRICS Report Series*, 3(25), 1996.

[32] Michal Feldman, Christos Papadimitriou, John Chuang, and Ion Stoica. Free-riding and whitewashing in peer-to-peer systems. In *ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*, 2004. https://doi.org/10.1145/1016527.1016539.

[33] Ben Fisch. Tight proofs of space and replication. In *EUROCRYPT*, 2019. https://doi.org/10.1007/978-3-030-17656-3_12.

[34] Fangyu Gai, Jianyu Niu, Seyed Ali Tabatabaee, Chen Feng, and Mohammad Jalalzai. Cumulus: A secure bft-based sidechain for off-chain scaling. In *IWQoS*, 2021. https://doi.org/10.1109/IWQOS52092.2021.9521363.

[35] Jianbo Gao, Jiashuo Zhang, Yue Li, Jiakun Hao, Ke Wang, Zhi Guan, and Zhong Chen. Pshard: a practical sharding protocol for enterprise blockchain. In *International Conference on Blockchain Technology and Applications*, 2022. https://doi.org/10.1145/3581971.3581987.

[36] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, 2015. https://doi.org/10.1007/978-3-662-46803-6_10.

[37] Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *IEEE S&P*, 2019. https://doi.org/10.1109/SP.2019.00040.

[38] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*, 2017. https://doi.org/10.1145/3132747.3132757.

[39] Zicong Hong, Song Guo, Peng Li, and Wuhui Chen. Pyramid: A layered sharding blockchain system. In *IEEE INFOCOM*, 2021. https://doi.org/10.1109/INFOCOM42981.2021.9488747.

[40] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium*, 2018. https://dl.acm.org/doi/10.5555/3277203.3277305.

[41] Aggelos Kiayias and Dionysis Zindros. Proof-of-work sidechains. In *Financial Cryptography and Data Security*, 2019. https://doi.org/10.1007/978-3-030-43725-1_3.

[42] Achim Klenke and Lutz Mattner. Stochastic ordering of classical discrete distributions. *Advances in Applied probability*, 42(2):392–410, 2010.

[43] Adrian Koegl, Zeeshan Meghji, Donato Pellegrino, Jan Gorzny, and Martin Derka. Attacks on rollups. In *Workshop on Distributed Infrastructure for the Common Good*, 2023. https://doi.org/10.1145/3631310.3633493.

[44] 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 Symposium*, 2016. https://dl.acm.org/doi/10.5555/3241094.3241117.

[45] 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 S&P*, 2018. https://doi.org/10.1109/SP.2018.000-5.

[46] Chon Kit Lao, Sophie Zhou, Luyao Zhang, Fan Zhang, and Kanye Ye Wang. Unpacking long-latency transactions in ethereum. In *Workshop on Decentralized Finance and Security*, 2024. https://doi.org/10.1145/3689931.3694909.

[47] Nam-Yong Lee. Hierarchical multi-blockchain system for parallel computation in cryptocurrency transfers and smart contracts. *Applied Sciences*, 11(21), 2021. https://doi.org/10.3390/app112110173.

[48] Canlin Li, Huawei Huang, Yetong Zhao, Xiaowen Peng, Ruijie Yang, Zibin Zheng, and Song Guo. Achieving scalability and load balance across blockchain shards for state sharding. In *SRDS*, 2022. https://doi.org/10.1109/SRDS55811.2022.00034.

[49] Jiasun Li. On the security of optimistic blockchain mechanisms. *Available at SSRN 4499357*, 2023. https://dx.doi.org/10.2139/ssrn.4499357.

[50] Ao Liu, Jing Chen, Kun He, Ruiying Du, Jiahua Xu, Cong Wu, Yebo Feng, Teng Li, and Jianfeng Ma. Dynashard: Secure and adaptive blockchain sharding protocol with hybrid consensus and dynamic shard management. *IEEE Internet of Things Journal*, 2024. https://doi.org/10.1109/JIOT.2024.3490036.

[51] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. In *IEEE S&P*, 2024. https://doi.org/10.1109/SP54263.2024.00035.

[52] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *CCS*, 2016. https://doi.org/10.1145/2976749.2978389.

[53] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, 2 edition, 2017.

[54] Avi Mizrahi and Ori Rottenstreich. Blockchain state sharding with space-aware representations. *IEEE Transactions on Network and Service Management*, 18(2), 2020. https://doi.org/10.1109/TNSM.2020.3031355.

[55] Zahra Motaqy, Mohamed E Najd, and Ghada Almashaqbeh. chainboost: A secure performance booster for blockchain-based resource markets. In *IEEE EuroS&P*, 2024. https://doi.org/10.1109/EuroSP60621.2024.00022.

[56] Lan N Nguyen, Truc DT Nguyen, Thang N Dinh, and My T Thai. Optchain: Optimal transactions placement for scalable blockchain sharding. In *ICDCS*, 2019. https://doi.org/10.1109/ICDCS.2019.00059.

[57] Russell O'Connor and Marta Piekarska. Enhancing bitcoin transactions with covenants. In *Financial Cryptography and Data Security*, 2017. https://doi.org/10.1007/978-3-319-70278-0_12.

[58] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT*, 2017. https://doi.org/10.1007/978-3-319-56614-6_22.

[59] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *ASIACRYPT*, 2008. https://doi.org/10.1007/978-3-540-89255-7_7.

[60] Moshe Shaked and J George Shanthikumar. *Stochastic orders*. Springer, 2007.

[61] Liping Tao, Yang Lu, Yuqi Fan, Lei Shi, Chee Wei Tan, and Zhen Wei. Throughput-scalable shard reorganization tailored to node relations in sharding blockchain networks. *IEEE Transactions on Computational Social Systems*, 2024. https://doi.org/10.1109/TCSS.2024.3406769.

[62] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *USENIX NSDI*, 2019. https://dl.acm.org/doi/10.5555/3323234.3323243.

[63] Beverly Yang and Hector Garcia-Molina. Ppay: Micropayments for peer-to-peer systems. In *CCS*, 2003. https://doi.org/10.1145/948109.948150.

[64] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *CCS*, 2018. https://doi.org/10.1145/3243734.3243853.

[65] Peilin Zheng, Quanqing Xu, Zibin Zheng, Zhiyuan Zhou, Ying Yan, and Hui Zhang. Meepo: Sharded consortium blockchain. In *ICDE*, 2021. https://doi.org/10.1109/ICDE51399.2021.00165.

A Class Composition Analysis

In this section, we determine the class composition of a weighted-sortition committee such that its failure probability is lower than that of a committee elected using random sortition. We start by defining the distributions governing committee memberships, then define the conditions under which a committee fails, and derive an epoch-independent upper bound on the failure probability using stochastic dominance and a Chernoff bound. Finally, we use this bound to determine the number of miners $n_{i,module}$ to be sampled from each class i, for a module module, subject to a target failure probability $F_w < F$, where F denotes the committee failure probability when elected using random sortition (under the same failure definition).

Notation. For a module module, miner class i, we denote the number of miners from class i as $n_i = n_{i,\mathsf{module}}$, and the miners needed for all modules as $n_{i,all}$. During each epoch e, a class $i \in \{1,\cdots,C\}$ contains $\mathcal{M}_{i,e}$ misbehaving (lazy or malicious) and $\mathcal{W}_{i,e}$ well-behaved miners. We denote the total population in the system as N, and assume that each class has exactly $\mu = \lfloor N/C \rfloor$ miners, so that $\mathcal{M}_{i,e} + \mathcal{W}_{i,e} = \mu$.

Lemma A.1 (Committee memberships distributions)**.** *Under the weighted sortition algorithm, the numbers of misbehaving and well-behaved committee members from class i in epoch e, $B_{i,e}$ and $G_{i,e}$, respectively, and those across all classes, B_e and G_e, respectively, satisfy the following and all $B_{i,e}$ and $G_{i,e}$ mutually independent:*

$$B_e = \sum_{i=1}^{C} B_{i,e}, \quad B_{i,e} \sim \mathrm{Bin}(\mathcal{M}_{i,e}, q_i)$$

$$G_e = \sum_{i=1}^{C} G_{i,e}, \quad G_{i,e} \sim \mathrm{Bin}(\mathcal{W}_{i,e}, q_i)$$

where $q_i = \frac{n_i}{\mu}$.

Proof. Consider a class-i miner j, and define the following events:

- $E_{j,e}$: miner j is elected to participate in any committee in epoch e. This event happens with probability $n_{i,all}/\mu$, uniformly across epochs.
- $A_{j,e,\mathsf{module}}$: miner j is assigned to the module module for epoch e, conditional on being elected. This event happens with probability $n_i/n_{i,all}$, uniformly across epochs.

A miner joins the committee of module if both events occur. Recall that our weighted sortition relies on a VRF-based approach. Since the two selection stages use independent VRF outputs, we have:

$$\begin{aligned}\Pr[E_{j,e} \cap A_{j,e,\mathsf{module}}] &= \Pr[E_{j,e}] \Pr[A_{j,e,\mathsf{module}} \mid E_{j,e}] \\ &= \left(\frac{n_{i,\mathrm{all}}}{\mu}\right)\left(\frac{n_i}{n_{i,\mathrm{all}}}\right) \\ &= \frac{n_i}{\mu} = q_i.\end{aligned}$$

Furthermore, the VRF-based selection across miners is run independently, thus each class-i miner performs an independent Bernoulli trial with success probability q_i. Therefore,

$$G_{i,e} \sim \mathrm{Bin}(\mathcal{W}_{i,e}, q_i)\ ,\ B_{i,e} \sim \mathrm{Bin}(\mathcal{M}_{i,e}, q_i)$$

Since the class populations are disjoint and selection trials are independent, all $G_{i,e}$ and $B_{i,e}$ are mutually independent. Summing over these classes establishes B_e and G_e in the lemma. □

Lemma A.2 (Committee Failure)**.** *Suppose that for every class i and every admissible epoch e, we have $\mathcal{M}_{i,e} \le \overline{\mathcal{M}}_i$ and $\mathcal{W}_{i,e} \ge \underline{\mathcal{W}}_i$. Define the mutually independent random variables:*

$$\overline{B} = \sum_{i=1}^{C} \overline{B}_i, \qquad \overline{B}_i \sim \mathrm{Bin}(\overline{\mathcal{M}}_i, q_i),$$

$$\underline{G} = \sum_{i=1}^{C} \underline{G}_i, \qquad \underline{G}_i \sim \mathrm{Bin}(\underline{\mathcal{W}}_i, q_i).$$

And let

$$g_{\min} = \mathbb{E}[\underline{G}] = \sum_{i=1}^{C} \underline{\mathcal{W}}_i q_i.$$

Then, for a committee with expected elected size S_c and minimum required participating size $s_{\min}$, where $0 < s_{\min} \le g_{\min}$, its failure probability in any admissible epoch satisfies:

$$\Pr[sc\text{-}fail] \le \Pr[2\overline{B} \ge \underline{G}] + \exp\left(-\frac{g_{\min}}{2}\left(1-\frac{s_{\min}}{g_{\min}}\right)^2\right).$$

Proof. A weighted sortition committee, with size $S_e = B_e + G_e$ fails for two reasons: either it fails to hold its Byzantine fault threshold ($B_e \ge \lceil \frac{S_e}{3} \rceil$), or it produces a participating committee (members who publish their election proofs) of size S_e^{part} that is too small to provide a prescribed level of fault tolerance (e.g., tolerating at least $f_{\min}$ misbehaving miners). The committee-failure event is therefore contained in

$$\{sc\text{-}fail, e\} \subseteq \{B_e \ge \lceil \frac{S_e}{3} \rceil\} \cup \{S_e^{\text{part}} < s_{\min}\}.$$

where s_{min} is the minimum committee size that can guarantee a minimum level of fault tolerance.

Recall that a committee exceeds its Byzantine fault threshold if and only if:

$$B_e \ge \left\lceil \frac{S_e}{3} \right\rceil \iff 3B_e \ge S_e \iff 2B_e \ge G_e.$$

To bound the probability $\Pr[2B_e \ge G_e]$, we need to understand its ordering against $\Pr[2\overline{B} \ge \underline{G}]$, using stochastic dominance. A binomial with fixed success probability is stochastically increasing in its number of trials [42, Theorem 1.1(a)]. Thus, for $i \in \{1,\dots,C\}$ we have $B_{i,e} \preceq_{\text{st}} \overline{B}_i$ and $\underline{G}_i \preceq_{\text{st}} G_{i,e}$. The second relation implies $-G_{i,e} \preceq_{\text{st}} -\underline{G}_i$.

By Lemma A.1, the variables $B_{i,e}, G_{i,e}$ are independent. Applying the closure of stochastic order under sums of independent random variables [60, Theorem 1.A.3(b)], we obtain

$$\sum_{i=1}^{C}\left(2B_{i,e} - G_{i,e}\right) \preceq_{\text{st}} \sum_{i=1}^{C} 2\overline{B}_i - \underline{G}_i$$

Equivalently, we have $2B_e - G_e \preceq_{\text{st}} 2\overline{B} - \underline{G}$, and therefore:

$$\Pr[2B_e \ge G_e] \le \Pr[2\overline{B} \ge \underline{G}]$$

For bounding the participating committee size failure, we note that since all elected well-behaved miners publish their proofs, the actual participating committee size ($S_e^{\text{part}} \le S_e$) satisfies:

$$S_e^{\text{part}} = B_e^{\text{part}} + G_e \ge G_e.$$

where B_e^{part} is the number of misbehaving miners that publish their proofs. Consequently, the probability that fewer than $s_{\min}$ miners publish valid election proofs is bounded by

$$\Pr[S_e^{\text{part}} < s_{\min}] \le \Pr[G_e < s_{\min}].$$

By stochastic dominance, $\underline{G} \preceq_{\text{st}} G_e$, and hence

$$\begin{aligned}&\Pr[G_e \ge s_{\min}] \ge \Pr[\underline{G} \ge s_{\min}]\\ \iff& 1 - \Pr[G_e \ge s_{\min}] \le 1 - \Pr[\underline{G} \ge s_{\min}]\\ \iff& \Pr[G_e < s_{\min}] \le \Pr[\underline{G} < s_{\min}]\end{aligned}$$

By construction, $\underline{G}$ is a sum of independent Bernoulli random variables with expectation:

$$\mathbb{E}[\underline{G}] = \sum_{i=1}^{C} \underline{\mathcal{W}}_i q_i = g_{\min}.$$

The multiplicative Chernoff lower-tail bound, [53, Theorem 4.5] gives

$$\Pr[\underline{G} < (1-\delta)g_{\min}] \le \exp\left(-\frac{g_{\min}\delta^2}{2}\right), \qquad 0 < \delta < 1.$$

Setting $\delta = 1 - \frac{s_{\min}}{g_{min}}$, we obtain

$$\begin{aligned}\Pr[G_e < s_{\min}] &\le \Pr[\underline{G} < s_{\min}]\\ &\le \exp\left(-\frac{g_{\min}}{2}\left(1-\frac{s_{\min}}{g_{min}}\right)^2\right)\end{aligned}$$

To determine the committee failure, we apply a union bound to the Byzantine fault-threshold and minimum size failure events, and we have

$$\begin{aligned}\Pr[sc\text{-}fail, e] &\le \Pr[\{2B_e \ge G_e\} \cup \{S_e^{\text{part}} < s_{\min}\}]\\ &\le \Pr[2B_e \ge G_e] + \Pr[G_e < s_{\min}]\\ &\le \Pr[2\overline{B} \ge \underline{G}]\\ &\quad + \exp\left(-\frac{g_{\min}}{2}\left(1-\frac{s_{\min}}{g_{\min}}\right)^2\right).\end{aligned}$$

Since the class-population bounds, the probability of being elected to a committee, and the expected committee size do not depend on the epoch, the resulting bound holds uniformly over all admissible epochs. □

Numerical example. We consider a system with $C = 3$ classes whose misbehaving fractions are bounded by $\overline{p}_1 = 0.15$, $\overline{p}_2 = 0.25$ and $\overline{p}_3 = 0.35$, i.e., $\overline{\mathcal{M}}_i = \lfloor \overline{p}_i \mu \rfloor$ and $\underline{\mathcal{W}}_i = \mu - \overline{\mathcal{M}}_i$. A weighted-sortition committee draws $n_1 = 349$, $n_2 = 249$ and $n_3 = 149$ miners in expectation from the three classes, for an expected committee size of $S_c = 747$. We require a minimum committee size of $s_{\min} = 400$, which corresponds to tolerating at least $f_{\min} = 133$ Byzantine members. We compute the term $\Pr[2\overline{B} \ge \underline{G}]$ of Lemma A.2 exactly, and bound the minimum-size term with the Chernoff bound, using $g_{\min} = \sum_i \underline{\mathcal{W}}_i q_i = 580.25$.

For comparison, we evaluate random sortition under the same failure definition: a single population of $N = C\mu$ miners, 25% of whom are misbehaving, from which each miner is elected independently with probability S_c/N, giving the same

expected committee size of 747. Table 3 reports both bounds for different class sizes.

Across all considered class sizes, weighted sortition yields a failure probability bound approximately four to five orders of magnitude smaller than that of random sortition, despite both schemes having the same expected committee size.

Table 3: Failure probability bounds for an expected committee size of 747 and $s_{\min} = 400$.

		Failure probability bound	
μ	N	Weighted sortition	Random sortition
2,000	6,000	7.9×10^{-13}	2.9×10^{-8}
10,000	30,000	3.0×10^{-12}	1.4×10^{-7}
100,000	300,000	4.9×10^{-12}	2.0×10^{-7}

B Security Analysis of marketPrism

Recall that a decentralized resource market operates a blockchain that manages and processes the market workload. A secure market means that it operates a secure blockchain that satisfies liveness and safety. This means that all transactions in the market are processed based on its network protocol, only valid blocks (containing valid transactions) are accepted, and that the system is live in the sense that it processes its workload and its blockchain is growing over time. We recall the definition of a secure ledger below.

Ledger security. A ledger $\mathcal{L}$ is secure if it satisfies the following properties [36]:

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 $\mathcal{L}$ (which includes all blocks buried under at least k blocks, where k is the depth parameter) 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: A valid transaction tx broadcast at time round t will be recorded on $\mathcal{L}$ at time at most $t+u$ with overwhelming probability, where u is the liveness parameter.

Validity of market workload processing is governed by its network protocol. Thus, honest miners on the mainchain and the sidechain process the workload they receive using the original logic of the market protocol. To show that marketPrism is secure, we have to prove that deploying marketPrism does not violate the liveness and safety of the underlying market.

Corollary B.1 (Failure of any committee)**.** *Consider K modules, where module m has $q_{i,m} = n_{i,m}/\mu$ for each class i. Let $\overline{B}_m$, $\underline{G}_m$, and $g_{\min,m}$ be defined as in Lemma A.2 using $q_{i,m}$, with $0 < s_{\min} \leq g_{\min,m}$ for all m. The probability of the event that any marketPrism's sidechain committee fails (denoted as CF) can be expressed as:*

$$\Pr[CF] \leq \sum_{m=1}^{K} \left(\Pr[2\overline{B}_m \geq \underline{G}_m] + \exp\left(-\frac{g_{\min,m}}{2} \left(1 - \frac{s_{\min}}{g_{\min,m}} \right)^2 \right) \right).$$

Proof. Lemma A.2 applies to each module separately, with $q_i = q_{i,m}$. The result follows from the union bound. □

Lemma B.2 (Autorecovery Failure)**.** *Consider a module* module*, the probability of the event that the module's autorecovery fails (denoted as AF_{module}) can be expressed as:*

$$\Pr[AF_{\text{module}}] \leq \left(\Pr[2\overline{B} \geq \underline{G}] + \exp\left(-\frac{\mathbb{E}[\underline{G}]}{2} \left(1 - \frac{s_{\min}}{\mathbb{E}[\underline{G}]} \right)^2 \right) \right)^{\kappa+1}$$

where $\overline{B}$, $\underline{G}$, $s_{\min}$ are defined as in Appendix A, and κ is the number of backup committees.

Proof. Consider a class-i miner j, and a sidechain with $\kappa+1$ (primary and backup) committees. We assume that Algorithm 1 is extended to randomly assign an elected miner to one of $\kappa+1$ committees using an independent VRF-based sortition.

Define the following events:

- $E_{j,e}$: miner j is elected to participate in any committee, for any module in epoch e. This event happens with probability $(\kappa+1)(n_{i,all}/\mu)$, uniformly across epochs.
- $A_{j,e,\text{module}}$: miner j is assigned to the module module for epoch e, conditional on being elected. This event happens with probability $n_i/n_{i,all}$, uniformly across epochs.
- $O_{j,e,l,\text{module}}$: miner j is assigned to module's l^{th} committee. This event happens with probability $1/(\kappa+1)$, uniformly across epochs.

Under weighted sortition, we define the event of being elected to module's $l^{th} \in \{0, \cdots, \kappa\}$ committee, represented by the indicator random variable $I_{j,e,l,\text{module}}$. All three events need to happen. We obtain:

$$\begin{aligned}
\Pr[I_{j,e,l,\text{module}} = 1] &= \Pr[E_{j,e} \cap A_{j,e,\text{module}} \cap O_{j,e,l,\text{module}}] \\
&= \Pr[E_{j,e}] \\
&\quad \cdot \Pr[A_{j,e,\text{module}} \mid E_{j,e}] \\
&\quad \cdot \Pr[O_{j,e,l,\text{module}} | A_{j,e,\text{module}} \cap E_{j,e}] \\
&= \left((\kappa+1) \frac{n_{i,\text{all}}}{\mu} \right) \left(\frac{n_i}{n_{i,\text{all}}} \right) \frac{1}{(\kappa+1)} \\
&= \frac{n_i}{\mu} = q_i.
\end{aligned}$$

Thus, for any module module, epoch e, and committee l,

$$G_{i,e,l} \sim \text{Bin}(\mathcal{W}_{i,e}, q_i), \qquad B_{i,e,l} \sim \text{Bin}(\mathcal{M}_{i,e}, q_i).$$

We define

$$G_{e,l} = \sum_{i=1}^{C} G_{i,e,l}, \qquad B_{e,l} = \sum_{i=1}^{C} B_{i,e,l}.$$

and

$$\overline{B} = \sum_{i=1}^{C} \mathrm{Bin}(\overline{\mathcal{M}}_i, q_i), \qquad \underline{G} = \sum_{i=1}^{C} \mathrm{Bin}(\underline{\mathcal{W}}_i, q_i).$$

Then $\underline{G}$ is stochastically dominated by $G_{e,l}$, and $\overline{B}$ stochastically dominates $B_{e,l}$.

Committee l fails, denoted F_l, if $2B_{e,l} \geq G_{e,l}$ or $G_{e,l} < s_{\min}$. and thus

$$\Pr[F_l] \leq \Pr[2\overline{B} \geq \underline{G}] + \exp\left(-\frac{\mathbb{E}[\underline{G}]}{2}\left(1 - \frac{s_{\min}}{\mathbb{E}[\underline{G}]}\right)^2\right).$$

Autorecovery fails when all (primary and backup) committees fail, i.e., $\bigcap_{l=0}^{\kappa} F_l$. Thus, it remains to bound $\Pr[\bigcap_{l=0}^{\kappa} F_l]$, using negative association (NA).

Fix a miner j. Since j is assigned to at most one committee of module, the indicators $(I_{j,e,l,\mathsf{module}})_{l=0}^{\kappa}$ sum to at most one and are NA by the Zero-One Lemma [31]. The VRF outputs of distinct miners are independent, so by Proposition 7(1) of [31] the family $\{I_{j,e,l,\mathsf{module}}\}_{j,l}$ over all miners and all l is NA. Now $G_{e,l} = \sum_i \sum_{j \in \mathcal{W}_{i,e}} I_{j,e,l,\mathsf{module}}$ and $B_{e,l} = \sum_i \sum_{j \in \mathcal{M}_{i,e}} I_{j,e,l,\mathsf{module}}$ are non-decreasing functions of pairwise disjoint subsets of this family, so by Proposition 7(2) of [31] the vectors $\mathbf{G} = (G_{e,l})_{l=0}^{\kappa}$ and $\mathbf{B} = (B_{e,l})_{l=0}^{\kappa}$ are each NA. Applying Proposition 7(2) again with the non-increasing maps $h_l(x) = -x$ on the singletons $\{l\}$, $-\mathbf{G} = (-G_{e,l})_{l=0}^{\kappa}$ is NA. Since $\mathbf{B}$ and $\mathbf{G}$ are determined by the VRF outputs of the malicious and honest miners respectively, and these sets are disjoint, $\mathbf{B}$ and $-\mathbf{G}$ are independent; by Proposition 7(1) the joint vector $(B_{e,0}, \ldots, B_{e,\kappa}, -G_{e,0}, \ldots, -G_{e,\kappa})$ is NA.

Let $f(b,g) = \mathbf{1}\{2b + g \geq 0\} \vee \mathbf{1}\{g > -s_{\min}\}$, so that $\mathbf{1}\{F_l\} = f(B_{e,l}, -G_{e,l})$. The function f is non-decreasing: if $f(b_2, g_2) = 1$ and $b_1 \geq b_2$, $g_1 \geq g_2$, then either $2b_1 + g_1 \geq 2b_2 + g_2 \geq 0$ or $g_1 \geq g_2 > -s_{\min}$, so $f(b_1, g_1) = 1$. The pairs $(B_{e,l}, -G_{e,l})$ occupy disjoint coordinates of the NA vector above, so the product inequality gives

$$\begin{aligned}
\Pr\Big[\bigcap_{l=0}^{\kappa} F_l\Big] &= \mathbb{E}\Big[\prod_{l=0}^{\kappa} f(B_{e,l}, -G_{e,l})\Big] \\
&\leq \prod_{l=0}^{\kappa} \mathbb{E}[f(B_{e,l}, -G_{e,l})] \\
&= \prod_{l=0}^{\kappa} \Pr[F_l].
\end{aligned}$$

Consequently, we have:

$$\Pr[AF_{\mathsf{module}}] \leq \prod_{l=0}^{\kappa} \Pr[F_l] = (\Pr[F_l])^{\kappa+1}$$

which completes the proof. □

marketPrism has K sidechains and autorecovery in marketPrism fails if any of the autorecovery of any of its sidechains fails. Applying a union bound, we have the failure probability of marketPrism's autorecovery, denoted as $\Pr[AF_{marketPrism}]$, is $\Pr[AF_{marketPrism}] \leq \sum_{m=1}^{K} \Pr[AF_m]$.

Theorem B.3. *Deploying marketPrism on top of a secure decentralized resource market does not violate safety and liveness of this market.*

We prove this theorem by proving two lemmas as follows.

Lemma B.4 (Preserving safety)**.** *marketPrism preserves the safety of the underlying resource market*

Proof. Based on the mechanisms employed in marketPrism design, we identify the following threats to safety:

- Service agreement violation, slacking and theft: As meta-blocks are pruned, servers and clients may try to violate service agreement terms, e.g., a server offers less service than agreed upon or a client tries to pay less than the agreed-upon price. Also, a server (who did not do the work) may claim it offered a service but the service delivery proofs have been pruned, or a client (who received the service) claims that no service has been delivered, as the meta-blocks containing the proofs have been pruned, to avoid paying the servers.
- Dispute circumvention: A party keeps participating in the system under the same identity even if they were excluded or penalized because they misbehaved.
- Out-of-sync mainchain or invalid syncing: A sidechain committee does not issue a sync-transaction at the end of the epoch, causing the mainchain state to be out of sync. Or this committee issues a sync-transaction that contains invalid summaries, causing the mainchain to be synced using invalid records (thus violating mainchain quality).
- Producing invalid sidechain blocks: A sidechain committee produces invalid meta- or summary-blocks. This violates the sidechain quality, and in turn will lead to invalid mainchain syncing.

marketPrism is resilient to these attacks since it uses a secure PBFT-consensus run by committees elected in a non-biased way with a size that guarantees honest majority, and the use of a secure autorecovery protocol that ensures system recovery from interruption with overwhelming probability.

Service agreement violation, slacking and theft as well as *dispute circumvention* are addressed as follows. The modules process their workload using the original logic of the underlying resource market. Meta-blocks containing the processed workload are not dropped until their corresponding sync-transaction is confirmed on the mainchain. Thus, anyone

can verify the validity of the produced summaries based on the corresponding meta-blocks. Summary-blocks are permanent, where they contain summarized service agreements, service delivery proofs, and dispute verification/outcome. Also, payments are made to the servers on the mainchain based on the service records of these servers (i.e., clients do not control that). By the security of consensus, these summaries are guaranteed to be correct, and hence, any service claims that do not agree with these summaries will be rejected and corresponding malicious parties will be penalized. Our autorecovery protocol and the mass-syncing mechanism guarantee that the system will recover from any interruption, thus restoring its valid operation state.

Out-of-sync mainchain, invalid syncing, or producing invalid sidechain blocks are addressed by the security of our autorecovery protocol (along with mass-syncing), the security of consensus, and electing committees with a size that guarantees an honest majority. These ensure that committees follow the prescribed protocol, and thus produce valid summaries and sync-transactions to sync the mainchain based on correct records. The security of consensus guarantees that a committee with honest majority only agrees on valid blocks, with the autorecovery protocol taking care of cases when a committee fails by having backup committees step in.

As a result, marketPrism preserves the safety of the underlying resource market. □

Lemma B.5 (Preserving liveness)**.** *marketPrism preserves the liveness of the underlying resource market.*

Proof. As marketPrism employs committees to manage module sidechains, we identify the following threats to liveness:

- DoS attacks: A sidechain's committee excludes particular transactions from being published.
- Violating sidechain liveness: The committee managing important modules does not add blocks to its sidechain, or the leader does not issue a $\mathsf{tx}_{\mathsf{sync}}$. This will violate the liveness of this sidechain. Also, not adding blocks may cause other modules to mine empty blocks for an undetermined period.
- Violating public verifiability: This consists of all threats resulting from workload sharing and blockchain pruning on the public verifiability of the records on the side and main chains.

DoS attacks are unfeasible since we employ a secure PBFT-based consensus with rotating epoch committees and leaders. Thus, it is infeasible for an attacker to hold full control of the transaction inclusion process on the long term.

Violating sidechain liveness is addressed by the use of a secure PBFT-based consensus and the autorecovery protocol of marketPrism. Any disruption to liveness would be detected by the committee members (if the cause is the committee leader) and will be solved by a leader change process; or by having a backup committee step in (in case of an unresponsive committee). The loss of sync-transactions (due to rollbacks) or due to having a leader that does not issue them, is resolved by the mass-syncing process. By configuring the committee size and the autorecovery protocol parameters properly, system recovery is guaranteed with overwhelming probability.

Violating public verifiability is mitigated by the use of a secure PBFT-based consensus and our autorecovery protocol, which guarantee that only correct transactions are processed and published on the sidechains and that correct summaries are produced. Also, the mass-syncing mechanism will handle any interruptions that may happen to the syncing process, thus guaranteeing that no summaries will be lost. Hence, the mainchain will be synced based on the workload processed on the sidechain. Since meta-blocks are pruned after their corresponding $\mathsf{tx}_{\mathsf{sync}}$ is confirmed on the mainchain, anyone can verify that the (permanent) summaries are valid. By the security of consensus, it is guaranteed that only valid summaries are accepted and their correctness will not change once meta-blocks are pruned. All these factors preserve the public verifiability of market operation.

Thus, marketPrism preserves the liveness of the underlying resource market. □