DATA AVAILABILITY: CHALLENGES, SOLUTIONS, AND OPEN PROBLEMS

Ifesi Onubogu
Computer Science Department
Columbia University
io2249@columbia.edu

Hideaki Takahashi
Computer Science Department
Columbia University
ht2673@columbia.edu

Begum Cicekdag
Computer Science Department
Columbia University
bc2975@columbia.edu
*

ABSTRACT

As blockchain networks scale and diversify, ensuring the availability of transaction data becomes a critical challenge. Data availability, the guarantee that all participants can access and verify the complete data of each block, is fundamental to maintaining the security, transparency, and decentralization of blockchain systems. Without robust data availability mechanisms, blockchain systems are vulnerable to issues such as data withholding attacks, increased centralization risks, and compromised trust.

In this paper, we present a systematic analysis of the data availability problem in blockchain systems. Our study encompasses a rigorous problem formulation, simulation experiments, an examination of real-world codebases, and interviews with developers of existing data availability protocols. We evaluate current solutions, including data availability sampling, erasure coding, and data availability committees, highlighting their respective strengths and limitations. Furthermore, we identify existing gaps and propose open problems to guide future research in this domain.

1 Introduction

Since its inception with Bitcoin in 2008 by Satoshi Nakamoto [1], blockchain technology has revolutionized industries far beyond cryptocurrency, becoming an ecosystem valued at more than \$1 trillion by 2025 that reshapes how we conceptualize trust in digital systems [2]. By enabling immutable, transparent ledgers without centralized authorities, blockchain has transformed financial services through DeFi protocols [3], healthcare record management [4], supply chain verification [5], and digital identity systems [6]. Yet as these networks evolve, processing millions of transactions daily across increasingly complex architectures, they face a fundamental challenge that threatens their foundational promise: **data availability**.

Data availability is a fundamental concept, the guarantee that transaction data is accessible and verifiable by all participants in a network [7, 8, 9]. Unlike traditional databases, where trust relies on administrators, blockchain's revolutionary potential stems from its ability to eliminate intermediaries through cryptographic verification. However, this verification becomes impossible when participants cannot access the underlying data. As layer-1 solutions implement sharding and layer-2 networks deploy *rollups* to scale transaction throughput, ensuring universal data access has emerged as the critical technical hurdle preventing blockchain technology from achieving mainstream adoption while preserving its core value proposition of decentralized, trustless verification [10, 11].

Without data availability, blockchain networks are vulnerable to issues such as data withholding attacks, which prevent users from verifying the correctness of the rollup states, thus exposing the rollup to risks of fraud and censorship [12, 13]. For instance, light clients, which do not store the entire blockchain, rely on the assumption that data is readily available. If a malicious actor withholds data, these clients cannot verify the correctness of transactions, undermining the integrity of the blockchain.

*

To address these challenges, several solutions have been proposed. Data Availability Sampling (DAS) allows nodes to probabilistically verify data availability without downloading entire blocks, enhancing scalability and efficiency [7]. Erasure coding techniques enable the reconstruction of missing data from partial information, reducing storage requirements for individual nodes while maintaining data integrity. Another solution is Data Availability Committees (DACs), which introduce trusted entities responsible for ensuring data availability; however, this approach may raise concerns about centralization [14, 15, 16].

Despite these advances, significant limitations persist [16]. DAS introduces new communication patterns that increase network complexity and impose stringent latency requirements. Erasure coding and commitment schemes must strike a balance between storage efficiency and computational overhead. DACs, while effective, may conflict with the decentralized ethos of blockchain systems.

This paper presents a comprehensive examination of data availability protocols, bridging foundational theory with practical implementations. Our key **contributions** are as follows:

- We introduce a rigorous and unified formulation of the data availability problem, encompassing both data availability sampling (DAS) and data availability commitments (DAC). This framework facilitates a systematic analysis of existing and emerging solutions.
- We conduct an in-depth analysis of three prominent data availability protocols: *Celestia* [17], *Avail* [18], and *EigenDA* [19]. Our evaluation combines standardized simulations with qualitative insights obtained through interviews with protocol developers, offering a holistic understanding of each protocol's design choices and performance characteristics.
- We investigate the integration of the protocols above within *ZKsync* [20], a leading Ethereum Layer 2 scaling solution, through real implementations. This case study highlights the practical considerations involved in the real-world deployment of data availability solutions.
- We delineate several critical open problems and propose actionable research directions to guide future work in the domain of data availability.

In § 2, we begin by formalizing the data availability problem and then present an overview of popular cryptographic primitives employed in current solutions. Building on this foundation, § 3 provides a detailed analysis, comparison, and simulation of three real-world data availability protocols, along with their specific implementations within ZKsync, as described in § 4. Finally, § 5 discusses the shortcomings of existing protocols and proposes promising avenues for future research.

2 Formulation and Preliminaries

In this section, we formally define the data availability protocol and its required properties, and we review several cryptographic primitives commonly used in real-world schemes, including Merkle trees, erasure codes, and KZG commitments. Our formulation is general enough to encompass both data availability sampling protocols, such as those used in Celestia and Avail, and data availability commitment schemes like EigenDA.

2.1 Formulation of Data Availability

Based on [7], we define a data availability scheme as follows. A unified data availability protocol with the security parameter λ, data alphabet Γ, encoding alphabet Σ, data length $K \in \mathbb{N}$, and encoding length $N \in \mathbb{N}$ is a tuple $\mathcal{P} = (\text{Setup}, \text{Encode}, \text{Commit}, \text{Verify}, \text{Reconstruct})$ of algorithms with the following syntax:

Definition 1: Data Availability Protocol

- $\text{Setup}(1^\lambda) \to \text{pp}$: Generates public parameters pp.
- $\text{Encode}(\text{pp}, D) \to \eta$: Encodes data $D \in \Gamma^K$ into encoded codewords $\eta \in \Sigma^N$.
- $\text{Commit}(\text{pp}, \eta) \to (\text{com}, \text{aux})$: Produces a commitment com for the encoding η and an auxiliary information if necessary.
- $\text{Verify}(\text{pp}, \text{com}, \text{aux}, I, \{\eta_i\}_{i \in I}) \to \{0, 1\}$: Verifies a subset of the encoding at indices $I \subseteq \{1, \dots, N\}$.
- $\text{Reconstruct}(\text{pp}, \{\eta_i\}_{i \in S}) \to D'$: Reconstructs the data from a subset $S \subseteq \{1, \dots, N\}$ of the encoding.

This formulation can model both data availability sampling and data availability commitment. For example, in the data availability sampling, Verify consists of two steps: Sample and VerifyProof.

- $\mathrm{Sample}(\mathrm{pp}, i, \eta) \to (\eta_i, \pi_i)$: Queries an encoding and its proof at the position $i \in \{1, \ldots, N\}$.
- $\mathrm{VerifyProof}(\mathrm{pp}, \eta_i, \pi_i, \mathrm{com}) \to \{0, 1\}$: Verifies the correctness.

2.2 Required Properties of Data Availability

The data availability protocol must ensure three key properties: *completeness*, *soundness*, and *consistency* [7].

Completeness Intuitively, completeness guarantees that "honest systems always operate correctly." Formally, suppose data D is encoded and committed according to the protocol. Then (i) any sufficiently large, randomly chosen subset of codewords $\{\eta_i\}_{i \in I}$ will pass the verification procedure with overwhelming probability (Verification Reliability), and (ii) once at least T valid codewords are collected ($|S| \geq T$), the original data D can be reconstructed exactly (Reconstruction Guarantee).

Definition 2: Completeness

$\mathcal{P}$ satisfies *completeness* if for all $\ell \geq T$ where $\ell = \mathsf{poly}(\lambda)$ and $|I| = \ell$:

$$\Pr\left[\begin{array}{l}\left[\begin{array}{l}\forall i \in I. \\ \mathrm{Verify}(\mathrm{pp}, \mathrm{com}, \mathrm{aux}, \{i\}, \eta_i) = 1\end{array}\right] \wedge \\ \mathrm{Reconstruct}(\mathrm{pp}, \{\eta_i\}_{i \in S}) = D\end{array} \middle| \begin{array}{l}\mathrm{pp} \leftarrow \mathrm{Setup}(1^\lambda), \\ \eta \leftarrow \mathrm{Encode}(\mathrm{pp}, D), \\ (\mathrm{com}, \mathrm{aux}) \leftarrow \mathrm{Commit}(\mathrm{pp}, \eta), \\ S \subseteq \{1, \ldots, N\},\ |S| \geq T\end{array}\right] \geq 1 - \mathrm{negl}(\lambda) \quad (1)$$

Soundness Intuitively, soundness ensures that "dishonest parties cannot deceive the system about data availability." Formally, no probabilistic polynomial-time adversary can output a commitment com that simultaneously (i) passes the verification check on sufficiently many randomly chosen subsets of codewords and (ii) yet makes it impossible to reconstruct any valid data D from any collection of at least T codewords.

Definition 3: Soundness

$\mathcal{P}$ satisfies *soundness* if for all PPT adversaries $\mathcal{A}$:

$$\Pr\left[\begin{array}{l}\mathrm{Verify}(\mathrm{pp}, \mathrm{com}, \mathrm{aux}, I, \{\eta_i\}) = 1 \wedge \\ \mathrm{Reconstruct}(\mathrm{pp}, \{\eta_i\}_{i \in S}) \neq D \\ \\ \end{array} \middle| \begin{array}{l}\mathrm{pp} \leftarrow \mathrm{Setup}(1^\lambda), \\ (\mathrm{com}, \mathrm{aux}, I, \{\eta_i\}) \leftarrow \mathcal{A}(\mathrm{pp}), \\ S \subseteq \{1, \ldots, N\},\ |S| \geq T, \\ D \leftarrow \mathrm{Reconstruct}(\mathrm{pp}, \{\eta_i\}_{i \in S})\end{array}\right] \leq \mathrm{negl}(\lambda) \quad (2)$$

Consistency Intuitively, consistency guarantees that all honest partial views of the encoding agree on the same underlying data. Formally, for any data D, every valid commitment com produced by the protocol is unique—no two distinct commitments can correspond to the same D.

Definition 4: Consistency

$\mathcal{P}$ satisfies *consistency* if for all PPT adversaries $\mathcal{A}$:

$$\Pr\left[\begin{array}{l}\left[\begin{array}{l}\quad \mathrm{Reconstruct}(\mathrm{pp}, \{\eta_i^{(1)}\}) \\ \neq \mathrm{Reconstruct}(\mathrm{pp}, \{\eta_j^{(2)}\})\end{array}\right] \wedge \\ \left[\begin{array}{l}\forall k \in \{1, 2\}. \\ \mathrm{Verify}(\mathrm{pp}, \mathrm{com}, \mathrm{aux}, I_k, \{\eta_i^{(k)}\}) = 1\end{array}\right]\end{array} \middle| \begin{array}{l}\mathrm{pp} \leftarrow \mathrm{Setup}(1^\lambda), \\ (\mathrm{com}, \mathrm{aux}, I_1, \{\eta_i^{(1)}\}, I_2, \{\eta_j^{(2)}\}) \leftarrow \mathcal{A}(\mathrm{pp})\end{array}\right] \leq \mathrm{negl}(\lambda) \quad (3)$$

2.3 Cryptographic Primitives

Merkle Tree A Merkle tree is a binary tree data structure used to efficiently and securely verify the integrity of large datasets [21]. Each leaf node contains the cryptographic hash of a data block, while each internal node stores the hash of the concatenation of its two child nodes. By recursively hashing pairs of child nodes up the tree, a single hash—called the Merkle root—is produced at the top. This root serves as a compact cryptographic commitment to the entire dataset: any modifications to a data block alters its hash, which propagates upward and results in a different Merkle root. Consequently, a Merkle root uniquely identifies the exact set of data and cannot be forged, assuming a secure, collision-resistant hash function.

One of the key benefits of Merkle trees is their support for efficient inclusion proofs. To verify that a particular data block is part of the tree, accessing only a small subset of hashes along the path from the corresponding leaf to the root is sufficient. This sequence of hashes is known as a Merkle proof. For a tree with n leaves, the size of a Merkle proof is approximately $\log(n)$ hashes, and both the construction and verification of the proof require only $\mathcal{O}(\log n)$ time. This logarithmic efficiency makes Merkle trees ideal for scalable and verifiable data structures in distributed systems like blockchains.

Reed-Solomon Erasure Coding Reed-Solomon erasure coding allows a system to tolerate data loss by encoding a piece of data into n total fragments (data + parity) such that any k of those n fragments suffice to reconstruct the original data (commonly denoted as an (n,k) code) [22]. It operates over a finite field $\mathbb{F}_q$, where q is typically a prime power large enough to represent all possible data symbols. The encoder views the original message as a sequence of k symbols $m_0, m_1, \ldots, m_{k-1} \in \mathbb{F}_q$, which are interpreted as the coefficients of a polynomial:

$$P(x) = m_0 + m_1 x + m_2 x^2 + \cdots + m_{k-1} x^{k-1}$$

The polynomial $P(x)$ has degree at most $k-1$. To encode the message, the encoder evaluates $P(x)$ at n distinct, publicly known points $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{F}_q$, where $n > k$. The resulting codeword is $(P(\alpha_1), P(\alpha_2), \ldots, P(\alpha_n))$ where each $P(\alpha_i)$ is a data share.

KZG Commitment A KZG commitment is a cryptographic commitment scheme used to commit to a polynomial and prove evaluations of that polynomial at specific points [23]. KZG commitments operate in elliptic curve groups G_1, G_2 with a bilinear pairing:

$$e : G_1 \times G_2 \to G_T, \quad \text{such that } e(g^a, h^b) = e(g,h)^{ab}$$

for all $a, b \in \mathbb{Z}_p$, where $g \in G_1$ and $h \in G_2$. They require a one-time trusted setup to create a Structured Reference String (SRS) that is available to all parties using a secret $\tau \in \mathbb{Z}_p$:

$$\{g^{\tau^0}, g^{\tau^1}, \ldots, g^{\tau^d}\} \subset G_1, \quad \{h, h^\tau\} \subset G_2.$$

The secret τ is later destroyed to prevent forgery. Let $f(x) = \sum_{i=0}^{d} a_i x^i$ be a polynomial of degree d. In order to commit to f, compute:

$$C_f = \prod_{i=0}^{d} (g^{\tau^i})^{a_i} = g^{f(\tau)}.$$

To prove that $f(u) = v$, compute the quotient polynomial:

$$q(x) = \frac{f(x) - v}{x - u}.$$

Then compute the proof:

$$\pi = g^{q(\tau)}.$$

Given C_f, u, v, π, the verifier checks:

$$e(C_f - g^v, h) \stackrel{?}{=} e(\pi, h^{\tau - u}).$$

3 Case Studies of Real-World Data Availability Protocols

This section presents case studies of three representative data availability protocols deployed in real-world systems: *Celestia* [24], *Avail* [25], and *EigenDA* [26]. Drawing on their white papers, official documentation, and interviews with core developers, we outline their architectural designs and analyze how they align with our formal definition of the data availability problem in § 2. We also provide a comprehensive comparison across key properties, including consensus mechanisms, validator selection, economic security, fraud detection, bridging support, and light client capabilities. To ensure a fair performance evaluation, we re-implement the core architecture of each protocol from scratch using a shared implementation of cryptographic primitives, isolating the impact of architectural design on performance.

3.1 Celestia

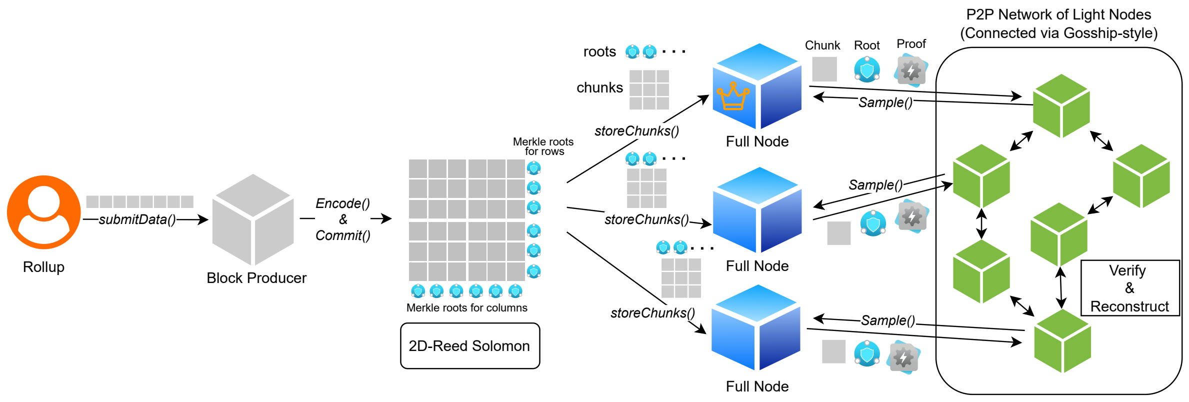


Figure 1: Architecture overview of Celestia. Celestia constructs data availability sampling with 2D Reed-Solomon encoding and Merkle Tree. Light clients randomly sample small portions of the encoded data; the client can be statistically confident that the entire data block is available if enough random samples are successfully retrieved.

Overview Celestia [24] is a modular data availability solution that decouples transaction execution from consensus. By employing data availability sampling (DAS) and 2D Reed-Solomon encoding, Celestia ensures data availability without requiring nodes to download the entire block [17]. Celestia's design ensures scalability by allowing light nodes to only sample random chunks of data, ensuring both trust-minimized and sensor-resistant infrastructure for decentralized applications.

In summary, Celestia functions as follows:

1. *Block Producer* [27]: The block producer collects transactions from rollup sequencers and encodes them using 2D Reed-Solomon encoding. For each chunk, Merkle roots are generated for rows and columns of the encoded data matrix.
2. *Light Nodes (Data Availability Sampling)* [28]: Light nodes sample random chunks of data from Celestia. If enough random chunks are available, the block header is accepted as valid. If the data is unavailable, the node rejects the block.
3. *Full Nodes* [29]: Full nodes download the entire block and validate all data. They verify the encoding and the Merkle roots of rows and columns, ensuring that data is available and not withheld.

Celestia guarantees censorship resistance and sensor-resistant infrastructure, which is essential for Web3 ecosystems where decentralization and trust-minimized data availability are crucial for rollups.

Formulation Let $\{\mathcal{O}_j\}_{j=1}^{q}$ define q quorums of light nodes sampling data from Celestia. The architecture of Celestia fits into the following formulation of a Data Availability Protocol:

$\mathrm{Setup}(1^\lambda) \to \mathrm{pp}$: The public parameters pp include parameters for 2D Reed-Solomon encoding, the Merkle root commitment scheme, and the light node sampling configuration $\{\mathcal{O}_j\}_{j=1}^{q}$.

$\mathrm{Encode}(\mathrm{pp}, D) \to \eta$: Given data $D \in \Gamma^K$, the block producer applies 2D Reed-Solomon encoding to produce the encoded codeword $\eta = \mathrm{RS}_{2\mathrm{D}}(D) \in \Sigma^N$, with an expansion factor $\frac{N}{K}$.

$\mathrm{Commit}(\mathrm{pp}, \eta) \to (\mathrm{com}, \mathrm{aux})$:

- The encoded codeword η is split into chunks $\{\eta_i\}_{i=1}^m$, where m depends on the chunk size.
- For each chunk η_i, the block producer computes a Merkle commitment $\mathrm{com}_i = \mathrm{Merkle.Commit}(\eta_i)$ and a proof $\pi_i = \mathrm{Merkle.Prove}(\eta_i)$.
- The commitment com is the set of all chunk commitments: $\mathrm{com} = \{\mathrm{com}_i\}_{i=1}^m$.
- The auxiliary information aux includes all Merkle proofs $\{\pi_i\}_{i=1}^m$.

$\mathrm{Verify}(\mathrm{pp}, \mathrm{com}, \mathrm{aux}, I, \{\eta_i\}_{i \in I}) \to \{0, 1\}$: Celestia's verification process involves:

- *Data Availability Sampling (DAS)*: A light node samples chunks η_i randomly from the encoded data and verifies their integrity using Merkle proofs.
- *Full Node Verification*: Full nodes verify all chunks to ensure the data is available and correct.

$\mathrm{Reconstruct}(\mathrm{pp}, \{\eta_i\}_{i \in S}) \to D'$: Celestia's reconstruction process involves retrieving enough valid encoded chunks and applying the inverse Reed-Solomon decoding algorithm to recover the original data:

1. A full node or retriever queries the network for enough chunks $\{\eta_i\}_{i \in S}$.
2. The retrieved chunks are decoded to recover the original data D'.

Security Guarantee Celestia guarantees data availability with probabilistic assurance provided by DAS . Light nodes can verify data availability by sampling a small number of chunks, and the likelihood of failing to detect unavailable data decreases as more nodes sample.

3.2 AvailDA

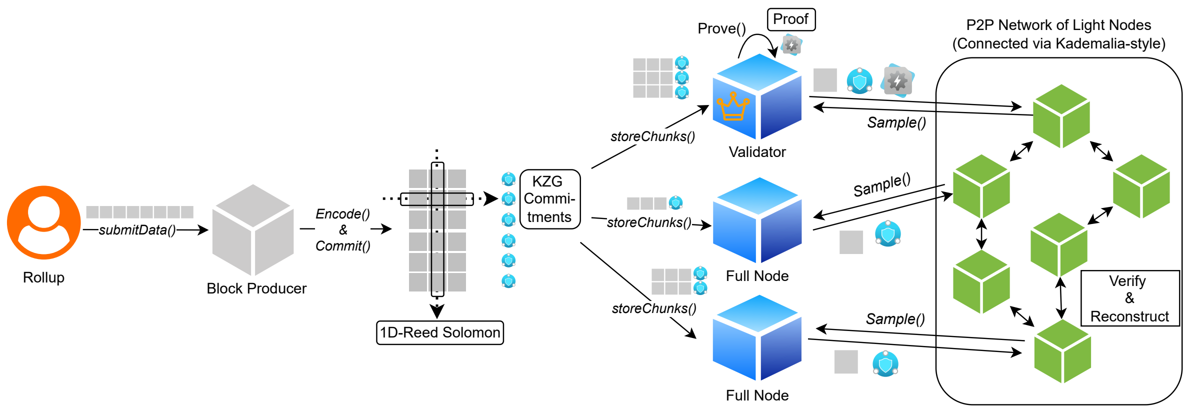


Figure 2: Architecture Overview of AvailDA. It reshapes data into a two-dimensional matrix, applies Reed-Solomon encoding to each column, and commits each row using KZG commitments. Light clients, connected through a Kademlia-style peer-to-peer network, perform data availability sampling (DAS) to verify that the entire data block is available probabilistically.

Overview AvailDA [25] is another popular data availability sampling protocols leveraging Reed-Solomon encoding and KZG commitments with four key players: *block producer*, *validator*, *fullnode*, and *light client*.

- *Block Producer*: The block producers are specialized full nodes that are responsible for creating new blocks on the AvailDA's network. They collect data from rollups or apps into a matrix. Each transaction has to include a unique application ID. Block producers apply 1D Reed-Solomon encoding for columns and generate KZG polynomial commitments for rows of the encoded data.
- *Validator*: Validators use these commitments to verify the data before it is attested and transmitted to the main chain through the AvailDA data attestation bridge (VectorX).
- *Full Node*: Full nodes serve data to Light Clients and other nodes when requested, ensuring that other network participants can retrieve specific data blocks efficiently.

- *Light Client*: AvailDA's light clients run on devices such as a user's phone to independently verify validity proofs. AvailDA's light clients are connected, forming a P2P overlay network, which creates a replica of Avail's blockchain in the P2P network. AvailDA's light clients can sample data availability directly from the P2P overlay, not just from full nodes.

Formulation Let $\{\mathcal{O}_j\}_{j=1}^{q}$ define q quorums of light nodes sampling data from AvailDA. The architecture of AvailDA fits into the following formulation of a Data Availability Protocol:

$\mathrm{Setup}(1^\lambda) \rightarrow \mathrm{pp}$: The public parameters pp include parameters for Reed-Solomon encoding, the parameters for KZG commitment and the light node sampling configuration $\{\mathcal{O}_j\}_{j=1}^{q}$. A 2D matrix layout for data blocks is also defined.

$\mathrm{Encode}(\mathrm{pp}, D) \rightarrow \eta$: Given an original data set D (e.g. a batch of blobs or transactions) for a new block, the block producer arranges it into a 2D matrix and performs column-wise Reed-Solomon erasure coding to produce an extended codeword η.

$\mathrm{Commit}(\mathrm{pp}, \eta) \rightarrow (\mathrm{com}, \mathrm{aux})$: Each row in η is viewed as a polynomial $P_i(x)$, where the row's entries are evaluations of $P_i(x)$. The producer uses the public SRS to compute a KZG commitment to each row polynomial P_i.

- **com:** The set of binding commitments to the encoded data: an array of KZG commitments, one per row, published in the block header.
- **aux:** Structured proof data to assist in verification of parts of η. This includes polynomial opening proofs for sampled shares, which will be provided during the Verify phase.

$\mathrm{Verify}(\mathrm{pp}, \mathrm{com}, \mathrm{aux}, I, \{\eta_i\}_{i\in I}) \rightarrow \{0,1\}$: The light client requests the data and verifies it against the KZG commitment in the block header, ensuring that the sampled data matches the commitment and is thus authentic. Sampling is done over a peer-to-peer (P2P) network, with light clients fetching data from a distributed hash table (DHT) populated by other light clients. If data is unavailable in the DHT, they can fall back to querying Avail nodes.

$\mathrm{Reconstruct}(\mathrm{pp}, \{\eta_i\}_{i\in S}) \rightarrow D'$: Any K available chunks from the erasure-coded set stored across the network is retrieved and the erasure decoding algorithm for Reed-Solomon is applied to reconstruct the original data, D', from these chunks.

3.3 EigenDA

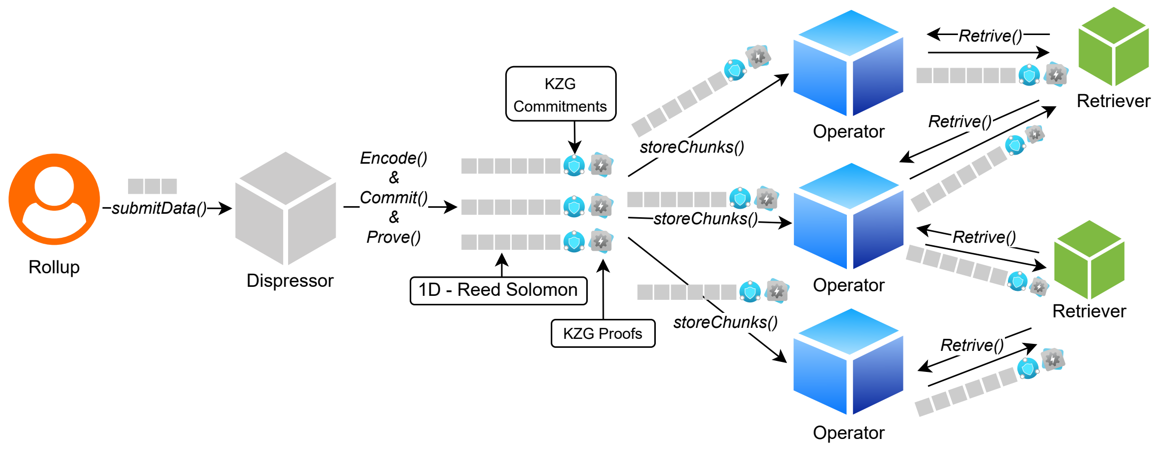


Figure 3: Architecture Overview of EigenDA. It employs erasure coding with KZG polynomial commitments to ensure data availability. Data is split into chunks, each stored by an operator node. A disperser encodes the data and distributes it to operators, who attest to its availability. Retrieval nodes can reconstruct the original data from these chunks.

Overview EigenDA [26, 19], a leading data availability (DA) solution built on EigenLayer, employs a sophisticated off-chain commitment scheme based on three core components: *operators*, *disperser*, and *retriever*.

In summary, EigenDA functions as follows:

1. *Disperser*: The disperser collects data blobs from the rollup sequencer and encodes them using two-dimensional Reed–Solomon erasure coding. For each resulting chunk, it generates KZG commitments and proofs. Next, the disperser splits the encoded data into redundant shards and distributes these shards—along with their KZG commitments and proofs—to the operators.
2. *Operator*: Operators receive the encoded shards and associated KZG commitments and proofs from the disperser, store the data, and earn rewards for their service. They then return a signature certifying that they have successfully stored the data.
3. *Retriver*: The retriever queries operators to fetch the stored blob chunks, verifies their integrity using the KZG commitments and proofs, and reconstructs the original data blobs. While EigenDA provides a default retriever, rollups can also deploy custom retrievers for enhanced control.

To safeguard against malicious behavior, operators are required to stake a minimum of 32 ETH (or 1 EIGEN), which functions as a security quorum. Once operators receive the data along with its commitments and proofs, they return signatures confirming storage. The disperser aggregates these signatures and submits them to Ethereum via the *EigenDAServiceManager* contract. This submission includes a call to the *BLSRegistry* contract, which verifies the signatures and confirms that the required quorum of staked funds is met. If an operator is found acting maliciously, a penalty mechanism known as slashing is triggered to burn the operator's stake.

Formulation Let $\{\mathcal{O}_j\}_{j=1}^{q}$ define q quorums, each containing a set of operators. The architecture of EigenDA fits into our formulation of a Data Availability Protocol as follows:

$\mathrm{Setup}(1^\lambda) \to \mathrm{pp}$: The public parameters pp implicitly include the parameters for the 1D Reed-Solomon coding (e.g., N and K), the KZG commitment scheme, and the quorum configuration $\{\mathcal{O}_j\}_{j=1}^{q}$ with their respective stake distributions.

$\mathrm{Encode}(\mathrm{pp}, D) \to \eta$: Given data $D \in \Gamma^K$, the depressor applies a 1D Reed-Solomon encoding to produce the codeword $\eta = \mathrm{RS}_{1\mathrm{D}}(D) \in \Sigma^N$, with an expansion factor of $\frac{N}{K}$.

$\mathrm{Commit}(\mathrm{pp}, \eta) \to (\mathrm{com}, \mathrm{aux})$:

- The encoded codeword η is divided into m chunks $\{\eta_i\}_{i=1}^{m}$, where each η_i corresponds to a segment of the encoded data. The number of chunks m depends on the chosen chunk size.
- For each chunk η_i, the depressor computes a KZG commitment $\mathrm{com}_i = \mathrm{KZG.Commit}(\eta_i)$ and a corresponding KZG proof $\pi_i = \mathrm{KZG.Prove}(\eta_i)$.
- The commitment com is the set of all individual chunk commitments: $\mathrm{com} = \{\mathrm{com}_i\}_{i=1}^{m}$.
- The auxiliary information aux includes the set of all KZG proofs: $\mathrm{aux} = \{\pi_i\}_{i=1}^{m}$.
- The encoded data chunks $\{\eta_i\}$, along with their corresponding KZG commitments $\{\mathrm{com}_i\}$ and proofs $\{\pi_i\}$, are distributed among the operators in the quorums.

$\mathrm{Verify}(\mathrm{pp}, \mathrm{com}, \mathrm{aux}, I, \{\eta_i\}_{i\in I}) \to \{0,1\}$: EigenDA's verification process involves two key aspects: depressor attestation and individual chunk verification.

- *Depressor Attestation:* While not a direct input to this function in the standard DAP definition, EigenDA relies on the depressor gathering signatures from operators on the commitment com. The verification of this attestation typically involves checking if a sufficient stake within each quorum has signed:

$$\forall j \in \{1, \cdots, q\} \quad \sum_{o\in\mathcal{S}_j} s_o \geq \alpha \sum_{o\in\mathcal{O}_j} s_o$$

  where $\mathcal{S}_j \subseteq \mathcal{O}_j$ is the set of operators in quorum j that have provided a valid signature on com, s_o is the stake of operator o, and α is a predefined threshold (e.g., $2/3$). This attestation provides probabilistic assurance of data availability.
- *Retriever Verification (Explicit):* A retriever who wants to verify a specific chunk at index $i \in I$ (where $I \subseteq \{1, \ldots, m\}$) obtains the following from the operator: 1) the commitment com_i 2) the chunk of encoded data η_i, and 3) the corresponding KZG proof π_i. Then, the retriever verifies the correctness of the proof against the commitment and the data: $\mathrm{KZG.Verify}(\mathrm{pp}, \mathrm{com}_i, i, \eta_i) = 1$. Here, we assume that the index i implicitly represents the position of the chunk within the original encoded data.

$\mathrm{Reconstruct}(\mathrm{pp}, \{\eta_i\}_{i\in S}) \to D'$: EigenDA's reconstruction process utilizes a retrieval subroutine.

1. $\mathrm{Retrieve}(\mathrm{pp}, \mathrm{com}, \{\mathcal{O}_j\}_{j=1}^{q}) \to \{\eta_i\}_{i\in S}$: A node (the reconstructor) queries operators within the quorums to obtain a sufficient set of encoded data chunks $\{\eta_i\}_{i\in S}$. The set S needs to be large enough to recover the original data, typically requiring $|S| \geq K$ (considering the properties of the Reed-Solomon code). The retrieval strategy might involve querying different operators across different quorums.
2. $\mathrm{RS}_{1\mathrm{D}}^{-1}(\{\eta_{\mathrm{i}}\}_{\mathrm{i}\in\mathrm{S}}) \to \mathrm{D}'$: Once a sufficient number of correct encoded data chunks $\{\eta_i\}_{i\in S}$ are retrieved (and potentially verified using the Verify function), the reconstructor applies the inverse 1D Reed-Solomon decoding algorithm $\mathrm{RS}_{1\mathrm{D}}^{-1}$ to recover the original data D'. If enough correct chunks are received, $D' = D$ with high probability.

Security Guarantee Currently, the quorum thresholds for signature confirmation are set to 55% of the registered stake for both the ETH and EIGEN token quorums [30]. In addition, EigenDA aims to tolerate up to 33% of the total stake in each quorum being adversarial. Moreover, the encoded blob size can be approximated by the following equation:

$$N = \frac{K}{\rho - \alpha} \tag{4}$$

, where N is the encoding length K is the original data length, ρ is the quorum threshold (55%), and α is the adversarial threshold (33%). Thus, the redundancy factor of the Reed–Solomon coding is currently configured to be 4.5 times the original blob size.

3.4 Feature Comparisons

	Celestia [24]	**AvailDA** [25]	**EigenDA** [26]
Consensus Protocol	Tendermint	BABE & GRANDPA	Dual Quorum
Validator Selection	Proof of Stake (PoS)	Nominated Proof of Stake (NPoS)	None
Economic Security	Slashing	Slashing	Slashing
Fraud Detection	Bad Encoding Fraud Proofs (BEFPs) + DAS	Validity Proof	Validity Proof
Bridge to Ethereum	Blobstream	VectorX	Service Manager
Light Clients	Supported	Supported	Planned

Table 1: Comparison of DA Solutions. They diverge significantly in trust assumptions and integration strategies.

This section presents a unified evaluation of Celestia, AvailDA, and EigenDA across critical dimensions: consensus protocols, validator selection, economic security, fraud detection, bridging mechanisms, and light-client support. We highlight each system's design trade-offs in terms of decentralization, performance, and security guarantees.

Consensus Protocol Celestia uses the *Tendermint* Byzantine-Fault-Tolerant (BFT) consensus protocol [31], which ensures fast finality at the block level without relying on proof-of-work (PoW) or other consensus protocols. This design delivers fast finality and low resource consumption, but its reliance on a fixed validator set introduces potential centralization pressures. In contrast, AvailDA employs a dual-layer approach: *BABE* (Blind Assignment for Blockchain Extension) orchestrates block production, while *GRANDPA* (GHOST-based Recursive Ancestor Deriving Prefix Agreement) provides probabilistic finality and strong safety guarantees [32]. This combination enhances throughput and scalability, though it does impose extra protocol complexity and coordination overhead. EigenDA employs a *Data Availability Committee* to ensure data availability through a small group of trusted participants. Consensus is achieved via a Dual Quorum mechanism built on a PoS network, allowing staking in both ETH and its native token, EIGEN [33].

Validator Selection In Celestia's *Proof-of-Stake (PoS)* model, validators participate in consensus by staking assets to earn the right to propose and validate blocks. Although this mechanism furnishes robust security through staking and incentivizes validators to act honestly, it tends to concentrate power among large stakeholders [34]. AvailDA mitigates this by adopting *Nominated PoS (NPoS)* [35], in which nominators select validators; this extra layer broadens participation and make it more decentralized thatn PoS, while it can be vulnerable to potential manipulation or collusion among nominators. EigenDA, by deferring to its DAC, does not employ a traditional staking mechanism;

committee membership is assigned off-chain according to governance rules, simplifying the protocol at the expense of economic-security guarantees, while suffering from centralized decision-making.

Economic Security Economic security in all of Celestia, AvailDA, and EigenDA is enforced via slashing [36, 37, 38]: validators are penalized for misbehaving or failing to fulfill their duties, providing stronger incentives for validators to behave honestly and enhancing validator accountability, while suffering from unfair penalties due to network issues and discouraged participation, leading to lower network decentralization.

Fraud Detection Celestia integrates Data Availability Sampling (DAS) with Bad-Encoding Fraud Proofs (BEFPs). Light clients probabilistically sample block shards to detect missing data, while full nodes can challenge incorrect encodings via succinct fraud proofs—together yielding a high-assurance system at the cost of added verification logic [39]. AvailDA uses DAS, permitting light clients to sample a small portion of data to verify its availability efficiently without downloading the entire block. Both AvailDA and EigenDA adopt the validity proof with KZG commitment and do not require additional challenge.

Bridge to Ethereum To bridge Celestia data into Ethereum, Celestia nodes stream "blobs" directly into Ethereum, achieving real-time availability assurances. This approach ensures timely updates but can incur congestion and elevated gas costs under heavy load. AvailDA's VectorX bridge introduces a structured method for efficient and optimized data transfers, while potentially introducing compatibility issues if VectorX is not widely supported or understood by different ecosystems [40]. EigenDA's Service Manager model centralizes the bridge logic in a governed off-chain service, simplifying integration but creating a single point of failure and limiting extensibility across diverse rollup ecosystems [41].

Light Clients Celestia provides native support for light clients [42]: by sampling a small, random subset of block fragments, they achieve probabilistic guarantees of data availability without downloading full blocks. This yields strong scalability and accessibility for nodes with limited resources, although light clients might have security risks if they cannot fully verify the correctness of the data they sample. AvailDA also supports light clients, offering similar sampling-based assurances [43]. Light clients for EigenDA are also planned to ensure that validator nodes cannot withhold data from retrieval nodes without such withholding being widely detectable [19].

3.5 Benchmarking

We re-implemented the core architectures of Avail, Celestia, and EigenDA from scratch using the same implementations of Reed-Solomon encoding, KZG commitments, and Merkle trees. This standardized implementation allows for a fair and consistent comparison of their architectural differences. Our evaluation focuses on the Encode, Commit, and Verify operations, excluding any simulation of network communication.

The core design principles of the three data availability solutions are summarized as follows:

- *Avail*: Reshapes the input data into a 2D matrix, applies 1D Reed-Solomon encoding to each column, and commits each row using KZG commitments.
- *Celestia*: Reshapes the input data into a 2D matrix, applies 2D Reed-Solomon encoding, and commits each row and column using Merkle trees (SHA-256 as the hash function).
- *EigenDA*: Splits the input data into fixed-length chunks and commits each chunk using KZG commitments.

We evaluate four data sizes: 16, 64, 256, and 1024 bytes. Following the actual protocol specifications, we use a Reed-Solomon expansion factor of 2 for AvailDA and Celestia, and 4 for EigenDA. In Avail, the number of rows is fixed at 8. In EigenDA, we fix the number of operators at 8, with each operator responsible for one chunk. The number of columns in AvailDA and the chunk size in EigenDA are determined based on these fixed parameters and the total data size. For data sampling, we follow protocol recommendations: AvailDA samples 8 random locations, which is the minimum required, and Celestia samples 16 locations. For EigenDA, we simulate data retrieval from 5 operators, which is sufficient to fully recover the original data. The implementation used for this experiment is available at `https://github.com/Koukyosyumei/MyZKP/blob/main/myzkp/examples/da.rs`.

Tab. 3.5 presents the results, reporting the total size of commitments and proofs, as well as the overall verification time. AvailDA and EigenDA exhibit significantly longer verification times than Celestia due to the use of KZG commitments, which require costly cryptographic operations such as elliptic curve pairings. Celestia, by contrast, uses Merkle trees, which allow for more efficient verification. However, the size of commitments and proofs in AvailDA and EigenDA remains constant regardless of the input data size. In Celestia, these sizes grow proportionally with the data size. This difference arises because KZG commitments produce fixed-size outputs per chunk, and the total number of chunks is fixed—equal to the number of operators in EigenDA and the number of rows multiplied by the expansion factor in AvailDA. In Celestia, however, a Merkle root is required for each row and column in the 2D Reed-Solomon-encoded matrix, and Merkle proof sizes increase with tree depth as the data size grows.

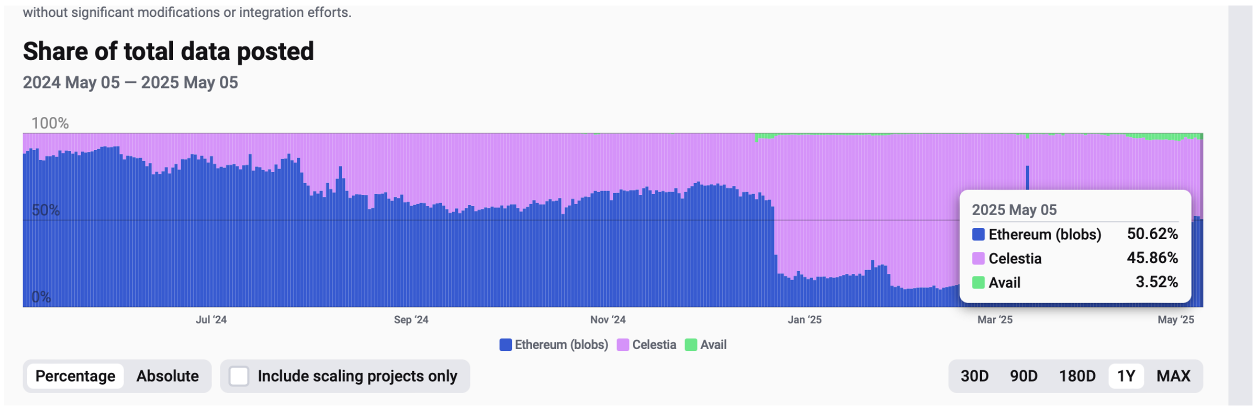
These results suggest that although architectures using KZG commitments incur high verification costs, they benefit from minimal network communication overhead, regardless of the data size. In terms of commitment and proof size, EigenDA performs the best. This aligns with publicly available benchmarks based on their official implementation. However, it is important to note that EigenDA adopts a more centralized architecture compared to the others.

	AvailDA			Celestia			EigenDA		
Data Size	Commitment Size	Proof Size	Verification Time	Commitment Size	Proof Size	Verification Time	Commitment Size	Proof Size	Verification Time
16 B	1024 B	768 B	23.7 s	544 B	1040 B	15.2 μs	512 B	768 B	16.4 s
64 B	1024 B	768 B	23.7 s	1056 B	1552 B	19.2 μs	512 B	768 B	15.2 s
256 B	1024 B	768 B	23.4 s	2080 B	2064 B	24.0 μs	512 B	768 B	15.3 s
1024 B	1024 B	768 B	23.6 s	4128 B	2576 B	32.0 μs	512 B	768 B	15.1 s

Table 2: Benchmark results comparing Avail, Celestia, and EigenDA using a unified implementation of encoding and commitment methods. Architectures using KZG commitment suffer from higher verification cost, while requiring less communication cost due to smaller commitment and proof size.

3.6 Proposal for Share of Total Data Posted Comparisons

The L2Beat dashboard currently presents the distribution of total data posted by various Data Availability (DA) systems. As of May 2025, Ethereum blobs account for 50.62%, Celestia for 45.86%, and AvailDA for 3.52% of the total data posted [44], as shown in Figure 4.


without significant modifications or integration efforts.

Share of total data posted

2024 May 05 — 2025 May 05

2025 May 05	
Ethereum (blobs)	50.62%
Celestia	45.86%
Avail	3.52%

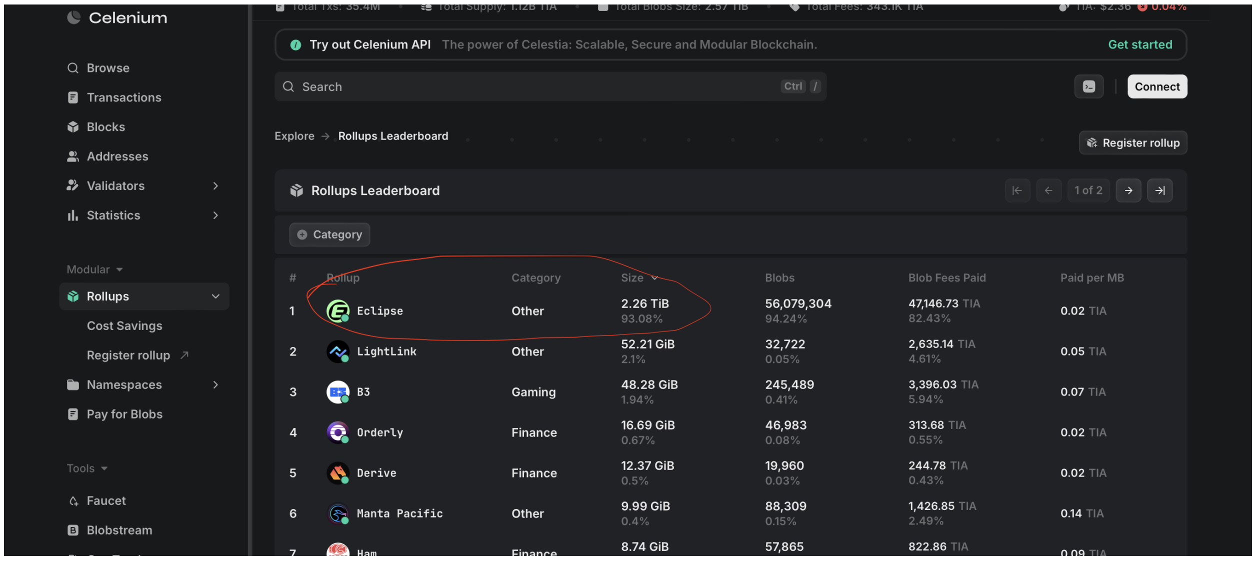
Figure 4: Share of total data posted dashboard on L2Beat, May 2025

However, this percentage breakdown may not accurately reflect the true utility of the data posted. For example, Celestia's data share appears heavily skewed due to a single contributor: Eclipse. According to the Celestia Rollup Dashboard [45], 93.08% of all data posted to Celestia originates from Eclipse (Figure 5).

Eclipse uses Ethereum as its settlement layer and the Solana Virtual Machine (SVM) for execution, posting data to Celestia for availability. While this architecture is modular and forward-looking, one might reasonably wonder whether Eclipse posts even empty or low-data blocks at regular intervals, thus consuming blobspace without proportional user-driven activity. This has not been confirmed, but if true, it would suggest that current raw data volume metrics may overstate Eclipse's relative utility.

To provide a more objective basis for comparing posted data shares—especially when public data lacks granularity—we propose a model that weights each client's posted data by its estimated utility rather than raw volume.

Let U_i represent the utility of data posted by client i. We define:



#	Rollup	Category	Size	Blobs	Blob Fees Paid	Paid per MB
1	Eclipse	Other	2.26 TiB 93.08%	56,079,304 94.24%	47,146.73 TIA 82.43%	0.02 TIA
2	LightLink	Other	52.21 GiB 2.1%	32,722 0.05%	2,635.14 TIA 4.61%	0.05 TIA
3	B3	Gaming	48.28 GiB 1.94%	245,489 0.41%	3,396.03 TIA 5.94%	0.07 TIA
4	Orderly	Finance	16.69 GiB 0.67%	46,983 0.08%	313.68 TIA 0.55%	0.02 TIA
5	Derive	Finance	12.37 GiB 0.5%	19,960 0.03%	244.78 TIA 0.43%	0.02 TIA
6	Manta Pacific	Other	9.99 GiB 0.4%	88,309 0.15%	1,426.85 TIA 2.49%	0.14 TIA
7	Ham	Finance	8.74 GiB	57,865	822.86 TIA	0.09 TIA

Figure 5: Celenium's Celestia Rollup Dashboard, May 2025

$$U_i = \alpha T_i + \beta C_i + \gamma S_i + \delta D_i$$

Where:

- T_i: Transaction frequency of client i
- C_i: Data complexity (e.g., zk-proofs, recursive proofs)
- S_i: Client scale (size or reach of client)
- D_i: Raw volume of data posted
- $\alpha, \beta, \gamma, \delta$: Tunable weights representing relative importance

From this, a normalized utility-weighted share is computed:

$$T_{\text{share},i} = \frac{U_i}{\sum_j U_j}$$

This model allows us to compute a more reliable estimate of each client's contribution to a DA system's throughput, correcting for cases where data volume alone may be misleading.

Parameter Selection and Weighting Rationale The chosen parameters represent distinct, interpretable axes of utility:

- T_i (Transaction frequency) captures sustained user demand.
- C_i (Data complexity) reflects computational or cryptographic value per byte posted.
- S_i (Client scale) measures the client's breadth—useful for weighting contributions from widely-used systems.
- D_i (Raw data volume) is retained for completeness but down-weighted to avoid biasing results toward unstructured or low-signal data.

We recommend the following baseline weights:

$$\alpha = 0.4, \quad \beta = 0.3, \quad \gamma = 0.2, \quad \delta = 0.1$$

These weights are based on the insights from an interview with an engineer at one of the DA projects we study in this paper. They reflect a belief that transaction frequency and data complexity are stronger indicators of utility than raw volume. The linear structure of the model ensures interpretability and facilitates comparison across systems. It is designed to leave little room for subjective interpretation, providing a more consistent framework for benchmarking client-level DA contributions.

4 Real-World Implementation: Data Availability in ZKsync

This section examines real codebases from ZKsync [20], one of the most widely adopted Layer 2 scaling solutions for Ethereum, with a focus on its integration with Celestia, Avail, and EigenDA. In particular, we analyze the implementation of clients for these three data availability protocols, located in `https://github.com/matter-labs/ZKsync-era/tree/main/core/node/da_clients`.

4.1 Celestia

ZKsync integrates Celestia's data availability (DA) layer to ensure that transaction data is available before finalizing state transitions. This integration bypasses the need for a traditional light client by using Celestia's APIs directly. The process can be broken down into the following steps:

Setup The setup step involves initializing the CelestiaClient with the necessary configuration and secrets. The client is created using ‘CelestiaConfig‘ (which contains the Celestia API endpoint, timeout, and namespace) and ‘CelestiaSecrets‘ (which holds the private key for secure communication).

```
let grpc_channel = Endpoint::from_str(config.api_node_url.clone().as_str())?
    .timeout(time::Duration::from_millis(config.timeout_ms))
    .connect()
    .await?;

let private_key = secrets.private_key.0.expose_secret().to_string();
let client = RawCelestiaClient::new(grpc_channel, private_key, config.chain_id.clone())
    .expect("could not create Celestia client");

Ok(Self {
    config,
    client: Arc::new(client),
})
```

This code sets up a gRPC channel to communicate with Celestia's API and initializes the CelestiaClient. This allows ZKsync to interact with Celestia's DA layer using the provided configuration and private key for secure communications.

Encoding The ‘dispatch_blob‘ function is responsible for encoding the transaction data into a blob and ensuring data integrity. The blob is created by attaching a namespace to the data and hashing it.

```
let namespace_bytes =
    hex::decode(&self.config.namespace).map_err(to_non_retriable_da_error)?;
let namespace =
    Namespace::new_v0(namespace_bytes.as_slice()).map_err(to_non_retriable_da_error)?;
let blob = Blob::new(namespace, data).map_err(to_non_retriable_da_error)?;
```

The namespace is used to uniquely identify the data, and the blob contains the transaction data. This ensures the data's integrity and that it can be identified within Celestia's decentralized network.

Commitment Once the blob is created, a commitment is generated. This commitment is a hash of the blob, ensuring that the data cannot be altered without changing the hash.

```
let commitment = blob.commitment;
```

This commitment serves as a fingerprint of the transaction data, allowing ZKsync and Celestia to verify that the data has not been tampered with. The commitment is crucial for ZKsync's validation process, where state transitions are validated based on this data.

Verification The ‘ensure_finality‘ and ‘get_inclusion_data‘ methods are used to verify that the blob is included in Celestia and available for ZKsync to use. These methods ensure that the transaction data is not only available but also included in Celestia's network.

```rust
async fn ensure_finality(
    &self,
    dispatch_request_id: String,
) -> Result<Option<FinalityResponse>, DAError> {
    Ok(Some(FinalityResponse {
        blob_id: dispatch_request_id,
    }))
}

async fn get_inclusion_data(&self, _: &str) -> Result<Option<InclusionData>, DAError> {
    Ok(Some(InclusionData { data: vec![] }))
}
```

These methods allow ZKsync to ensure that the blobs are available before proceeding with state transition validation. The inclusion data provides the necessary information to verify that the transaction data is available on Celestia.

Reconstruction While the code provided does not explicitly define the reconstruction process, the commitment generated in step 3 allows for data reconstruction when needed. The commitment ensures that the data can be reconstructed by fetching the blob from Celestia and verifying its correctness.

The ability to reconstruct data from the commitment ensures that ZKsync can verify state transitions and make withdrawals, even if nodes do not have access to all the transaction data directly.

Bypassing the Need for a Light Client In traditional systems, a light client is used to download and verify data before proceeding with state transitions. However, in this implementation, the CelestiaClient bypasses the need for a light client by using Celestia's APIs directly. The following steps facilitate this:

- Data Availability: By streaming the blobs directly to Celestia, ZKsync ensures that the data is available and can be retrieved as needed without relying on an external light client.
- Data Integrity and Commitment: The commitment to the blob ensures that ZKsync can rely on Celestia for data integrity without needing to re-verify the data itself.
- Verification and Finality: The ‘ensure_finality‘ and ‘get_inclusion_data‘ methods ensure that ZKsync can verify data availability before proceeding with state transitions, making the light client unnecessary.

Missing Steps: Ensuring Erasure Encoding and Proofs While the current implementation ensures data availability and integrity, there is a missing step: verifying that the transaction data is correctly erasure encoded and included at the time the proof is sent to Ethereum. This could be handled by implementing a check to ensure that the blob commitment matches the expected encoding and inclusion data before submitting the proof to L1.

4.2 AvailDA

ZKsync's integration with AvailDA enables rollups to interact with the network through two different clients: a full client and a gas relay client. In contrast to the implementations for Celestia and EigenDA, the codebase includes a minimal re-implementation of AvailDA's SDK client. This is considered to be a temporary solution until a mature SDK is available on crates.io.

Setup Roll-up operators can choose the type of AvailDA's client to initialize by specifying the desired mode in the *AvailConfig* configuration file, defined in `ZKsync-era/core/lib/config/src/configs/da_client/avail.rs`. Independent of the selected mode, the configuration includes a bridge API URL, which facilitates communication between the AvailDA's network and Ethereum, and a request timeout in milliseconds. On the other hand, client-specific variables are defined in *AvailClientConfig*, nested inside *AvailConfig*:

- **Full Client:** Requires the API node URL, application ID, and an optional finality state.
- **Gas Relay Client:** Requires the gas relay API URL and the maximum number of retry attempts.

A new client is initialized using these parameters, along with credentials from *AvailSecrets*, which includes either a seed phrase—used for signing transactions—or an API key for authenticating with the gas relay service.

Data Submission The `dispatch_blob` method, part of the DataAvailabilityClient trait implementation for AvailClient, manages the submission of blobs to the AvailDA's network.

- **Full clients** submit data directly to an AvailDA's node over a WebSocket connection, using the node URL specified in the `api_node_url` field of *AvailClientConfig*.

```
    let ws_client = WsClientBuilder::default()
            .build(default_config.api_node_url.clone().as_str())
            .await
            .map_err(to_non_retriable_da_error)?;
```

  The block hash and transaction ID from the block where the extrinsic was included is returned.

- **Gas relay clients** submit data to the gas relay API using the `post_data` function defined in ZKsync's AvailDA's SDK. A request id from the relay service, which serves as an identifier for tracking the submission, is returned.

```
    let submission_id = client
            .post_data(data)
            .await
            .map_err(to_retriable_da_error)?;
```

Finality Check The function `ensure_finality`, checks whether a data blob has been finalized on the AvailDA's network and, if so, returns the blob id, a colon-separated string consisting of the block hash and the transaction index. For full clients, finality is assumed since the block hash and transaction index are returned at the time of submission in the `dispatch_blob` method. In gas relay mode, `ensure_finality` queries the relay server to verify the block finality.

```
    AvailClientMode::Default(_) => Some(FinalityResponse {
            blob_id: dispatch_request_id,
        }),
    AvailClientMode::GasRelay(client) => {
            let Some((block_hash, extrinsic_index)) = client
                .check_finality(dispatch_request_id)
                .await
                .map_err(to_retriable_da_error)?
```

Inclusion Check The function `get_inclusion_data` fetches Merkle inclusion proof data for a blob given it's `blob_id`. The request URL is formed by parsing the bridge API URL defined in *AvailConfig* and appending `/eth/proof/{block_hash}?index={tx_idx}`. The information included in the response from Avail's data attestation bridge can be found in Table 3. If the response includes an error message or contains any empty fields, None will be returned. Otherwise, the attestation data will be converted into Ethereum ABI tokens and encoded.

```
            match bridge_response_to_merkle_proof_input(bridge_api_data) {
            Some(attestation_data) => Ok(Some(InclusionData {
                data: ethabi::encode(&attestation_data.into_tokens()),
            })),
            None => {
                tracing::info!(
                    "Bridge API response missing required fields. Data might not be available yet.");
                Ok(None) }}
```

4.3 EigenDA

Setup The following shows the code snippet of the Setup operation used in the client code of ZKsync for EigenDA. For example, it downloads `srs_points_source`, the generators and their powers on the elliptic curve used in the pairing operation for KZG commitment. The configuration, `eigen_config`, consists of necessary settings such as the RPC endpoint for connecting to EigenDA's disperser, the Ethereum RPC URL for blockchain interactions.

```
let srs_points_source = match config.points_source {
    PointsSource::Path(path) => SrsPointsSource::Path(path),
```

	Definition
`blob_root`	Merkle root of all data blobs published to AvailDA.
`bridge_root`	Merkle root relayed by the bridge to another chain (e.g., Ethereum) as proof of data attestation.
`data_root_index`	Position of a data root within a sequence or Merkle tree.
`data_root_proof`	Merkle proof showing a data root's inclusion in the attested Merkle root.
`leaf`	Individual data blob.
`leaf_index`	Position of a leaf in the Merkle tree.
`leaf_proof`	Merkle proof showing a leaf's inclusion in the Merkle tree.
`range_hash`	Hash representing a contiguous range of data blobs/leaves.

Table 3: AvailDA Attestation Bridge Contents

```
    PointsSource::Url(url) => SrsPointsSource::Url(url),
};

let eigen_config = rust_eigenda_client::config::EigenConfig::new(
    config.disperser_rpc,
    eth_rpc_url,
    config.settlement_layer_confirmation_depth,
    config.eigenda_svc_manager_address,
    config.wait_for_finalization,
    config.authenticated,
    srs_points_source,
    config.custom_quorum_numbers,
)?;
let private_key = PrivateKey::from_str(secrets.private_key.0.expose_secret())
    .map_err(|e| anyhow::anyhow!("Failed to parse private key: {}", e))?;
let eigen_secrets = rust_eigenda_client::config::EigenSecrets { private_key };
```

Encode The following `dispatch_blob` function dispatches a blob of data to the disperser, subsequently encoding the data with 1D Reed Solomon coding.

```
fn dispatch_blob(
    &self,
    _: u32, // batch number
    data: Vec<u8>,
) -> Result<DispatchResponse, DAError> {
    let blob_id = self
        .client
        .dispatch_blob(data)
        .await
        .map_err(to_retriable_da_error)?;

    Ok(DispatchResponse::from(blob_id))
}
```

Commit The process of receiving commitment data is implemented within `get_inclusion_data`, which internally calls `get_commitment` function, obtaining verification proof and verifying the corresponding commitment.

```
fn get_inclusion_data(&self, blob_id: &str) -> Result<Option<InclusionData>, DAError> {
    let inclusion_data = self
        .client
        .get_inclusion_data(blob_id)
        .await
        .map_err(to_retriable_da_error)?;
    if let Some(inclusion_data) = inclusion_data {
        Ok(Some(InclusionData {
            data: inclusion_data,
        }))
    } else {
        Ok(None)
    }
}
```

5 Open Problems and Future Directions

The limitations of current data availability protocols primarily arise from two fundamental challenges. First, the peer-to-peer (P2P) communication model introduces significant overhead, as data availability sampling requires frequent and bandwidth-intensive interactions between validators and local nodes [46]. This tight coupling places scalability constraints on the system, especially under high validator counts or adversarial conditions. Second, the design of encoding and commitment schemes presents both structural and computational limitations. For example, the combination of 2D Reed–Solomon erasure coding with Merkle tree commitments necessitates fraud proofs to detect incorrect data, which in turn requires that each node maintain connections to honest peers and some time delay for finality. Another example is that validity proofs based on KZG commitments impose heavy computational costs, rely on a trusted setup, and lack post-quantum security, raising concerns about long-term cryptographic robustness [16, 47].

P2P Network One of the promising future network architectures for data availability protocol is *Vacuum!* [48], which is a high-efficiency data propagation protocol implemented in the mamo-1 testnet for Celestia. Its three core innovations, 1) lazy gossiping, 2) Validator Availability Certificates (VACs), and 3) Pull-Based Broadcast Tree (PBBT), work in concert to optimize throughput. Lazy gossiping ensures that data is transmitted only to peers that explicitly request it, eliminating redundant transfers and maximizing bandwidth utilization. VACs enable each validator to attest to the specific blobs it holds and prioritize the propagation of the most critical data, streamlining the routing process. The pull-based broadcast tree leverages erasure coding and parity shards so that validators can reconstruct missing portions of a block from locally cached fragments rather than downloading the entire payload. Together, these enhancements allow the mamo-1 testnet to maintain 21.33 MB / s of throughput, over a 16× improvement compared to the current Celestia mainnet.

Despite these breakthroughs, mamo-1 still contends with two principal limitations. First, although erasure coding dramatically reduces the volume of data that must be transmitted, it cannot completely eliminate the risk of missing or corrupted fragments, and any failure in verifying correct encoding remains a potential vulnerability. Second, while the testnet's validator set spans three European cities (Amsterdam, Paris, and Warsaw), extending this model to a global network of thousands of validators will introduce new challenges in certificate distribution, inter-validator communication, and distributed storage management.

Future research directions for P2P networks in the context of data availability sampling include developing decentralized and scalable mechanisms for managing validator identities, enabling efficient data availability sampling across a larger and more geographically dispersed network, and designing incentive mechanisms that encourage reliable participation from diverse regions.

Encoding and Commitment One recent advancement in encoding and commitment schemes for data availability sampling is *Zero-Overhead Data Availability* (ZODA) [49, 47], an enhanced version of the 2D erasure coding and Merkle tree commitments used in Celestia. ZODA begins by encoding the columns of the two-dimensional data and committing to this intermediate encoding using a Merkle tree. It then derives entropy from this commitment to generate a random vector, constructs a diagonal matrix from this vector, and multiplies the data by the resulting random diagonal matrix. Finally, it performs row-wise encoding. This approach enables each row and column to prove their correctness without requiring any additional proofs, thereby eliminating the need for fraud proofs without incurring extra overhead. Unlike KZG, ZODA uses linear algebra and hash-based commitments, making it plausibly post-quantum secure without a trusted setup.

Another promising approach is *FRIDA* [50], which replaces KZG commitments with Fast Reed-Solomon Interactive Oracle Proofs of Proximity (FRI IOPP), thereby eliminating the need for a trusted setup. FRI is a cryptographic protocol that enables a prover to convince a verifier that a given codeword, representing a polynomial, is close to a low-degree polynomial, all without requiring a trusted setup. This makes it suitable for a variety of practical verification problems, including the data availability problem. [50] demonstrates that FRIDA achieves lower commitment sizes and per-query communication costs compared to other schemes, though it incurs a higher total encoding size.

Given the numerous candidates for both encoding and commitment, each with distinct properties and trade-offs in terms of security and efficiency, future research should tackle fine-grained theoretical analyses of each combination, complemented by realistic simulations to evaluate their behavior in practical settings. This would enable practitioners to select the most suitable combination based on their specific use cases."

6 Conclusion

In this work, we present a unified framework for understanding and evaluating data availability protocols, bridging the gap between foundational theory and real-world deployment. We begin by formalizing the data availability problem. Based on this formulation, we provide a comprehensive comparison of three leading protocols: Celestia, Avail, and EigenDA through standardized simulations and developer interviews. We further demonstrate the practical challenges of real-world integration through a case study of ZKsync, a major Ethereum Layer 2 solution. This work not only surveys the current landscape of data availability protocols but also synthesizes key limitations and open questions, offering concrete research directions to address these gaps. It serves as a resource for researchers and protocol designers aiming to ensure robust, scalable, and trustless data availability in the next generation of blockchain systems. Looking ahead, we outline critical avenues for future work to advance the resilience and decentralization of data availability infrastructure.

7 Acknowledgments

The insights presented in this report were informed in part by conversations with engineers at two of the three data availability projects we studied, as well as a team working on a zero-knowledge rollup. Although these individuals have requested to remain anonymous, we are deeply grateful for their generosity in sharing their time and expertise.

Their willingness to explain core concepts and point us toward resources we might not have encountered independently—such as the Celenium dashboard and ZKsync's implementation of the Data Availability clients —greatly enriched our understanding and sharpened the focus of our analysis.

References

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. White paper, 2008.

[2] Bhaskar Tripathi and Rakesh Kumar Sharma. Cryptocurrency exchanges and traditional markets: A multi-algorithm liquidity comparison using multi-criteria decision analysis. *Computational Economics*, 65(5):2649–2677, 2025.

[3] Saulo Dos Santos, Japjeet Singh, Ruppa K Thulasiram, Shahin Kamali, Louis Sirico, and Lisa Loud. A new era of blockchain-powered decentralized finance (defi)-a review. In *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1286–1292. IEEE, 2022.

[4] Cornelius C Agbo, Qusay H Mahmoud, and J Mikael Eklund. Blockchain technology in healthcare: a systematic review. In *Healthcare*, volume 7, page 56. MDPI, 2019.

[5] Maciel M Queiroz, Renato Telles, and Silvia H Bonilla. Blockchain and supply chain management integration: a systematic review of the literature. *Supply chain management: An international journal*, 25(2):241–254, 2020.

[6] Seyed Mohammad Hosseini, Joaquim Ferreira, and Paulo C Bartolomeu. Blockchain-based decentralized identification in iot: An overview of existing frameworks and their limitations. *Electronics*, 12(6):1283, 2023.

[7] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Foundations of data availability sampling. *Cryptology ePrint Archive*, 2023.

[8] Arunima Chaudhuri, Sudipta Basak, Csaba Kiraly, Dmitriy Ryajov, and Leonardo Bautista-Gomez. On the design of ethereum's data availability sampling: A comprehensive simulation study. In *2024 6th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 1–4. IEEE, 2024.

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

[10] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*, 10:93039–93054, 2022.

[11] Jan Gorzny and Martin Derka. A rollup comparison framework. *arXiv preprint arXiv:2404.16150*, 2024.

[12] Adrian Koegl, Zeeshan Meghji, Donato Pellegrino, Jan Gorzny, and Martin Derka. Attacks on rollups. In *Proceedings of the 4th International Workshop on Distributed Infrastructure for the Common Good*, pages 25–30, 2023.

[13] Muhammad Bin Saif, Sara Migliorini, and Fausto Spoto. A survey on data availability in layer 2 blockchain rollups: Open challenges and future improvements. *Future Internet*, 16(9):315, 2024.

[14] Ertem Nusret Tas and Dan Boneh. Cryptoeconomic security for data availability committees. In *International Conference on Financial Cryptography and Data Security*, pages 310–326. Springer, 2023.

[15] Peiyao Sheng, Bowen Xue, Sreeram Kannan, and Pramod Viswanath. Aced: Scalable data availability oracle. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25*, pages 299–318. Springer, 2021.

[16] Kundu Chen and Jie Luo. zkrpc: Trustless bitcoin data availability network. *Computer Networks*, 258:110957, 2025.

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

[18] Avail Project. Avail: A Unifying Blockchain Network. `https://github.com/availproject/data-availability/tree/master/reference%20document`, 2024. Published 06/11/2024.

[19] EigenLabs. Eigenda core concepts: Overview. `https://docs.eigenda.xyz/core-concepts/overview/`. Accessed: 2025-05-01.

[20] Matter Labs. zksync: Trustless scaling and privacy engine for ethereum. `https://zksync.io/`, 2020. Accessed: 2025-05-03.

[21] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, page 369–378, Berlin, Heidelberg, 1987. Springer-Verlag.

[22] Stephen B Wicker and Vijay K Bhargava. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[23] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-Size Commitments to Polynomials and Their Applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 177–194, Berlin, Heidelberg, 2010. Springer.

[24] Celestia: A modular consensus and data availability layer. `https://celestia.org/`. Accessed: 2025-03-11.

[25] Avail: Scalable data availability for blockchain networks. `https://availproject.org/`. Accessed: 2025-03-11.

[26] Eigenda: Secure and decentralized data availability service. `https://eigenlayer.xyz/`. Accessed: 2025-03-11.

[27] Celestia Labs. Block producer. `https://celestia.org/glossary/block-producer/`. Accessed: 2025-05-06.

[28] Celestia Labs. Light node. `https://celestia.org/glossary/light-node/`. Accessed: 2025-05-06.

[29] Celestia Labs. Light node. `https://celestia.org/glossary/full-node/`. Accessed: 2025-05-06.

[30] EigenLabs. Security Model: BFT Security Model. `https://docs.eigenda.xyz/core-concepts/security/security-model#bft-security-model`, 2025. Accessed: 2025-05-06.

[31] Celestia Labs. Staking, governance, & supply. `https://docs.celestia.org/learn/staking-governance-supply`, 2025. Accessed: 2025-05-06.

[32] Avail Project. How to stake avail: An intro to nominated proof of stake (npos). `https://blog.availproject.org/stake-avail-earn-rewards/`, May 2024. Accessed: 2025-05-06.

[33] EigenLabs. Eigenlayer holesky testnet launch + dual quorum support for eigenda. `https://www.blog.eigenlayer.xyz/eigenlayer-holesky-testnet-launch-dual-quorum-support-for-eigenda/`, 2023. Accessed: 2025-05-06.

[34] Varul Srivastava, Sankarshan Damle, and Sujit Gujar. Centralization in proof-of-stake blockchains: A game-theoretic analysis of bootstrapping protocols. *arXiv preprint arXiv:2404.09627*, 2024.

[35] Avail Project. How to stake avail: An intro to nominated proof of stake (npos). `https://blog.availproject.org/stake-avail-earn-rewards/`, May 2024. Accessed: 2025-05-07.

[36] Celestia Labs. Jailing and slashing on celestia. `https://docs.celestia.org/how-to-guides/celestia-app-slashing`, November 2024. Accessed: 2025-05-07.

[37] Avail Project. Chill your validator. `https://docs.availproject.org/docs/operate-a-node/become-a-validator/chill-your-validator`, April 2025. Accessed: 2025-05-07.

[38] EigenLabs. Slashing. `https://docs.eigenlayer.xyz/eigenlayer/concepts/slashing/slashing-concept`, November 2024. Accessed: 2025-05-07.

[39] Celestia Labs. Data availability faq. https://docs.celestia.org/learn/how-celestia-works/data-availability-faq, 2023. Accessed: 2025-05-07.

[40] Avail Project. Vectorx. https://docs.availproject.org/docs/build-with-avail/vectorx, 2025. Accessed: 2025-05-07.

[41] Layr Labs. eigenda-proxy. https://github.com/Layr-Labs/eigenda-proxy, 2025. Accessed: 2025-05-07.

[42] Celestia Labs. Light client. https://celestia.org/glossary/light-client/. Accessed: 2025-05-08.

[43] Avail Project. Introduction to avail light clients. https://docs.availproject.org/docs/operate-a-node/run-a-light-client/Overview, 2025. Accessed: 2025-05-08.

[44] L2Beat. Share of total data posted. https://l2beat.com/data-availability/throughput, 2024. Accessed: 2025-05-06.

[45] Celenium. Rollups leaderboard. https://celenium.io/rollups?page=1. Accessed: 2025-05-06.

[46] Michał Król, Onur Ascigil, Sergi Rene, Etienne Rivière, Matthieu Pigaglio, Kaleem Peeroo, Vladimir Stankovic, Ramin Sadre, and Felix Lange. Data availability sampling in ethereum: Analysis of p2p networking requirements. *arXiv preprint arXiv:2306.11456*, 2023.

[47] Alex Evans, Nicolas Mohnblatt, and Guillermo Angeris. Zoda: Zero-overhead data availability. *Cryptology ePrint Archive*, 2025.

[48] Mustafa Al-Bassam. Announcing mamo-1: Celestia's 128mb block testnet. https://blog.celestia.org/mamo-1/, April 2025. Accessed: 2025-05-06.

[49] Alex Evans, Nicolas Mohnblatt, Guillermo Angeris, Sanaz Taheri, and Nashqueue. Zoda: An explainer. https://baincapitalcrypto.com/zoda-explainer/, December 2024. Accessed: 2025-05-03.

[50] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Frida: Data availability sampling from fri. In *Annual International Cryptology Conference*, pages 289–324. Springer, 2024.