PANDAS: Peer-to-peer, Adaptive Networking for Data Availability Sampling within Ethereum Consensus Timebounds

Matthieu Pigaglio
UCLouvain
Belgium

Onur Ascigil
Lancaster University
United Kingdom

Michał Król
City, University of London
United Kingdom

Sergi Rene
Datahop Labs
United Kingdom

Felix Lange
Ethereum Foundation
Germany

Kaleem Peeroo
City, University of London
United Kingdom

Ramin Sadre
UCLouvain
Belgium

Vladimir Stankovic
City, University of London
United Kingdom

Etienne Rivière
UCLouvain
Belgium
etienne.riviere@uclouvain.be

ABSTRACT

Layer-2 protocols can assist Ethereum's limited throughput, but globally broadcasting layer-2 data limits their scalability. The *Danksharding* evolution of Ethereum aims to support the selective distribution of layer-2 data, whose availability in the network is verified using randomized data availability sampling (DAS). Integrating DAS into Ethereum's consensus process is challenging, as pieces of layer-2 data must be disseminated and sampled within four seconds of the beginning of each consensus slot. No existing solution can support dissemination and sampling under such strict time bounds.

We propose PANDAS, a practical approach to integrate DAS with Ethereum under Danksharding's requirements without modifying its protocols for consensus and node discovery. PANDAS disseminates layer-2 data and samples its availability using lightweight, direct exchanges. Its design accounts for message loss, node failures, and unresponsive participants while anticipating the need to scale out the Ethereum network. Our evaluation of PANDAS's prototype in a 1,000-node cluster and simulations for up to 20,000 peers shows that it allows layer-2 data dissemination and sampling under planetary-scale latencies within the 4-second deadline.

KEYWORDS

Ethereum, Data Availability Sampling, Peer-to-peer, Performance

1 INTRODUCTION

Ethereum, the largest blockchain supporting smart contracts, currently supports adding fewer than a few tens of transactions per second to its main (layer-1) chain. Complementarily to layer-1 scalability improvements [33, 57], expanding support for layer-2 protocols [30] is now a priority for the Ethereum community [23].

Layer-2 protocols such as *side chains* and *rollups* have the potential to process large amounts of transactions [30, 38]. These protocols periodically produce compressed or batched layer-2 transaction data, which they make available via the layer-1 blockchain. For instance, participants in an optimistic rollup can download this data, verify its correctness, and submit fraud proofs [26, 40, 53].

The throughput of layer-2 protocols depends on how much data they can attach to layer-1 blocks. Previously, the only solution was adding layer-2 data as costly *calldata* transactions. These transactions competed for permanent block space with other layer-1 transactions, such as ETH transfers and DeFi interactions. In March 2024, the EIP-4844 (*Proto-Danksharding*[1]) proposal [10] introduced the notion of *blobspace*. Layer-2 data can now be shared as opaque *binary objects* (blobs). Blobs are broadcast separately and referenced by *blob-carrying* transactions in the block, which include cryptographic commitments to their content. Nodes participating in consensus verify these commitments and make blob data available to layer-2 participants for a limited time (4,096 epochs, ~18 days). While improving over *calldata* transactions regarding costs and supported volume, EIP-4844 still requires blob data to be broadcast and received by all nodes.

A significant upcoming step towards scaling support for layer-2 data in Ethereum is implementing *data availability sampling* (DAS). The Ethereum *Danksharding*[1] [24] roadmap plans to support up to 32 MB blob data referenced by each layer-1 block.

To avoid broadcasting this volume of data globally, blob data is erasure-coded, split, and distributed as collections of *cells*, so that each node holds only a fraction of the data. This shift introduces a new challenge: no single node can independently verify the availability of the full blob. To address this, Ethereum plans to adopt *data availability sampling* (DAS), wherein nodes collect random *sample* cells from the network until they reach overwhelming confidence that the complete data can be reconstructed. The Danksharding parameters imply sending 140 MB of erasure-coded cells to the network, with each node randomly sampling 73 cells (40 KB).

Integrating DAS and Ethereum consensus is a challenge. The need for each node to collect randomly selected samples results in a multitude of exchanges over high-latency links. At the same time, Ethereum consensus imposes tight time constraints. A new block is generated every 12 seconds. A committee must validate each new block within the first four seconds after its creation. To avoid changes to the consensus protocol, DAS must also be completed within four seconds, allowing committee members to attest block validity and blob data availability simultaneously.

[1] *Danksharding* is named after Dankrad Feist, an Ethereum researcher. *Proto-danksharding* is named after him and Diederik Loerakker (*protolambda*).

Several approaches are under discussion for integrating DAS with Ethereum consensus, e.g., EIP-7594 (PeerDAS) [5], SubnetDAS [14], or FullDAS [42]. These leverage peer-to-peer networks already used by other Ethereum functions, particularly GossipSub [64], a broadcasting network that pre-establishes dissemination overlays and uses them as channels for gossiping data. There is a discrepancy between the high costs of establishing new channels and the randomized nature of DAS. Multi-hop gossiping also has inherently high latency. These two factors lead these proposals to suggest that random sampling happens *after* the validation of blocks by the committee, i.e., past the 4-second deadline. Completing sampling after committee validation may reverse validation decisions due to the delayed detection of blob data unavailability. This requires modifications to Ethereum's consensus to account for such decision-revert possibilities. This also impacts finality (i.e., how long before a block can be considered immutable) as pending data availability verifications delay decisions. Finally, the possibility of reverting past consensus opens the door to new attacks based on *ex-ante* reorganizations [15].

Contribution. We demonstrate that DAS can be integrated with the existing Ethereum consensus while meeting the Danksharding objectives. Dissemination and sampling of blob data can occur within the first four seconds of a consensus slot. This allows the committee to confirm blob data availability and block data correctness simultaneously and removes the need to adapt Ethereum's consensus to delayed availability decisions.

We present PANDAS, a peer-to-peer protocol that supports DAS in Ethereum. PANDAS builds upon the following key features:

- It aligns with recent Ethereum evolutions, including Proposer-Builder Separation (PBS) [25, 34], which introduces powerful builders responsible for preparing block and blob data and ordinary proposers elected by Proof-of-Stake consensus [22]. PANDAS leverages builders for efficient *seeding* of blob data.
- It uses Ethereum nodes to host and sample blob data using peer-to-peer interactions. In contrast to other proposals [5, 14, 42], PANDAS employs direct (one-hop) communication, using connectionless networking (UDP). Interactions adapt to nodes' unavailability and faults, meeting the 4-second deadline in adverse environments or under inconsistent network views by different participants.
- PANDAS supports Ethereum's objectives of openness, decentralization, and scalability. Nodes' and builders' bandwidth requirements align with the typical capacities of home servers and cloud instances, and do not increase with system size.

We implement PANDAS over `libp2p` [4], the network stack of the Ethereum Geth client [3], and deploy 1,000 nodes on an 80-server cluster using representative emulated WAN latencies. Additionally, we utilize a simulator whose results are cross-validated against prototype deployments. This enables us to confidently explore results for up to 20,000 nodes.

Our evaluation shows that PANDAS meets the 4-second sampling deadline at all nodes at moderate scales and for the vast majority of nodes at large scales, while maintaining low load on builders and nodes. In contrast, baseline solutions based on GossipSub [64] or the Kademlia DHT [50] do not scale as well, incurring higher overhead and failing to meet the 4-second deadline even at moderate network sizes. Experiments involving a significant fraction of unresponsive nodes and inconsistent views further demonstrate that PANDAS's operations are robust against faults, meeting the 4-second deadline for the majority of nodes even when up to 50% of the nodes are misbehaving, and systematically detect data unavailability.

Outline. This paper is organized as follows. We present preliminaries about Ethereum, layer-2 protocols, and PBS (Section 2). We detail the DAS principles and the *Danksharding* roadmap and analyze the associated networking and communication requirements (Section 3). We present our model and assumptions, and detail our design objectives (Section 4). PANDAS uses a deterministic assignment of blob data to nodes (Section 5). It operates in three phases, from the *seeding* of blob data by a builder to nodes *consolidation* of this data and its *sampling* (Section 6). PANDAS uses direct and efficient but unreliable UDP communications. An adaptive fetching protocol arbitrates between request redundancy and time constraints (Section 7). We evaluate PANDAS and compare it to baselines (Section 8). We discuss our results (Section 9) before covering related work (Section 10) and concluding (Section 11).

2 PRELIMINARIES

We provide an overview of Ethereum, its consensus, the Proposer-Builder Separation principle, and layer-2 protocols.

Ethereum. Ethereum is an open blockchain using Proof-of-Stake (PoS) consensus [22]. Holders of ETH, Ethereum's virtual currency, can lock 32 ETH (their *stake*) or more to operate a *validator*, i.e., a virtual entity participating in the validation of new blocks.

Time in Ethereum is divided into slots of 12 seconds and epochs of 32 slots. In every slot, a new block is added to the blockchain. A subset of validators is deterministically selected to participate in each consensus slot. One of them, the *proposer*, is responsible for forming and spreading a new block. Some validators produce *attestations* of this new block, while others collect these attestations and publish aggregate decisions. As a result, consensus is split into three phases: (1) the broadcast of a new block and its verification by the committee; (2) the propagation and collection of attestations; and (3) the generation and broadcast of aggregate decisions. Each phase accounts for a third of the slot duration, i.e., $12/3 = 4$ seconds.

Servers called full nodes, or simply "nodes" for the rest of this paper, participate in the Ethereum network. Nodes can, but do not have to, host validators. A node is identified by its IP address and a public key, which are shared through *Ethereum Node Records* (ENR) propagated through the network and stored in the underlying Kademlia DHT [43, 50]. While nodes can collect all ENRs by crawling the DHT [12, 20, 58, 63], the association between a node and a specific validator should not be public [35]. Deanonymizing the link between the two leads to security threats such as DDoS or targeted abuse of slashing mechanisms [54]. All nodes, whether they host a validator or not, use the consensus committee's aggregate decisions to determine whether a block is accepted.

The dissemination of new blocks, attestations, and aggregate decisions is supported by Gossipsub [64], a peer-to-peer overlay that enables multi-hop, controlled flooding of data.

Proposer-Builder Separation. Forming new blocks is increasingly computationally expensive, particularly with the rising importance of *Maximal Extractable Value* or MEV [31]. Any node hosting a

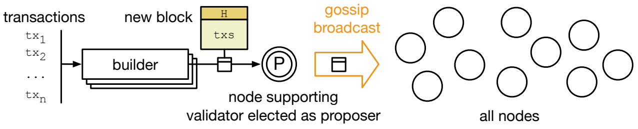


Figure 1: Proposer-Builder Separation (PBS). The proposer, elected based on stake, selects a block among those prepared by builders. The block is broadcast by gossip to all nodes.

validator that can be elected as a proposer would need to provision a powerful server. Preventing low-stake participants unable to do so from participating in consensus leads to the concentration of power among a limited number of actors. Proposer-Builder Separation (PBS), illustrated in Figure 1, addresses this risk by separating the role of *building* a block and the role of *proposing* it for consensus. This enables a few dedicated builders to form new blocks while maintaining decentralized consensus among many lightweight nodes that host validators. With PBS, the node supporting the proposer selects one of the blocks prepared by builders. The block is then broadcast to all nodes using Gossipsub. Today, PBS is responsible for around 90% of Ethereum block creation [34, 46], principally through the MEV-Boost network [28]. Builders receive block construction fees for blocks selected by proposers and accepted by consensus; therefore, they are incentivized to produce correct blocks.

Layer-2 protocols. The throughput of Ethereum's chain (i.e., layer-1) is limited to the number of transactions that can fit in a block. Layer-2 protocols move some of the transaction handling and validation processes to a separate layer while benefiting from the security and decentralization of the layer-1 chain. There exist many variants of layer-2 protocols [30, 38].

Rollups are exemplary layer-2 solutions that process transactions off-chain. They publish transaction state in a compressed form, together with a commitment to this state, via a call to a smart contract in a regular layer-1 transaction. Rollup variants include optimistic ones [26] posting compact hashes of transactions' states, e.g., Arbitrum [40] and Optimism/Bedrock [53], and ZK rollups [19] posting zero-knowledge proofs of validity, e.g., ZkSync [49] or Polygon [55]. The volume of layer-2 transactions that can be *anchored* to the layer-1 chain is directly linked to the supported volume of blob data. This data needs to be available for a sufficient time for a protocol's participants to verify it (e.g., verifying the ZK proof [19, 49, 55] or generating a fraud proof [26, 40, 53]). Unlike regular layer-1 transactions, layer-2 data does not need to persist indefinitely nor be verified for correctness by layer-1 nodes.

3 DATA AVAILABILITY SAMPLING

Ethereum's current mechanism for attaching layer-2 data to layer-1 blocks is EIP-4844 (*Proto-Danksharding*) [10]. It attaches a limited number of data *blobs* (binary objects) to each block. This blob data is broadcast to all nodes. Nodes hosting committee members must validate the corresponding commitments contained in blob-carrying transactions. To keep the costs of operating a node reasonable and preserve decentralization, EIP-4844 limits the number of 128-KB blobs to 3 (on average) to 6 (maximum) or 0.375 to 0.75 MB of data.

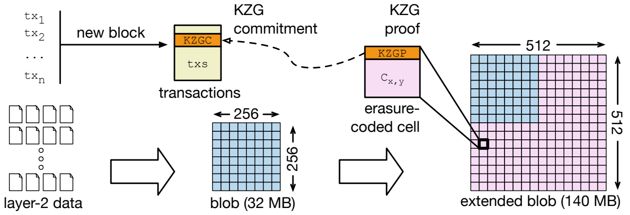


Figure 2: Builder preparatory operations for DAS. 32 MB of data is aggregated in a blob of 256×256 cells, extended to 512×512 cells using erasure coding. Each resulting cell includes a proof (KZGP) linking it with the Kate-Zaverucha-Goldberg commitment (KZGC) in the corresponding blob-carrying transaction.

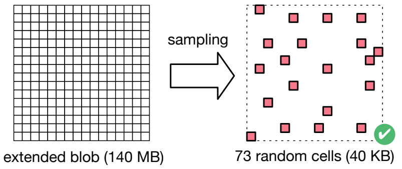


Figure 3: Each node samples 73 randomly chosen cells.

Danksharding [24] is a roadmap towards much more (32 MB) blob data attached to each block. It is intimately linked to PBS and relies on builders' computational and networking power to collect, aggregate, and share layer-2 data. With such volumes, fully disseminating blob data to all nodes is no longer realistic. Instead, each node receives and stores a subset (shard) of it. For an individual node, receiving a subset does not guarantee the availability of the *complete* blob data. Data Availability Sampling (DAS) enables this verification. It consists of three phases. First, the blob is extended using erasure coding. Second, each shard of extended blob data is distributed to a subset of nodes for hosting. Third, nodes collect *samples*, allowing them to consider the data they do not host available (or reconstructable) with an overwhelming probability.

Figure 2 details the construction of blob data. The blob aggregates 32 MB of data, split into cells of 512 B, organized as a 256×256 matrix. Releasing only a subset of blob data is a *data withholding attack*. The base blob is highly amenable to such an attack, as sharing all but one cell makes some data unavailable, threatening the security of layer-2 protocols. To prevent data withholding and allow data reconstruction after losses, the blob is *extended* using a two-dimensional Reed-Solomon erasure code [65]. Each row and column doubles in size but can now be reconstructed from any 50% of its cells. The resulting *extended* blob is now a 512×512-cell matrix. In addition to the 512 B of data, each cell includes a 48 B Kate-Zaverucha-Goldberg proof (KZGP) [41]. This proof links the cell's content to a commitment (KZGC) registered in a layer-1 blob-carrying transaction. In total, the extended blob is $(512 \times 512) \times (512 + 48) = 140$ MB in size including 12 MB of KZGPs.

Following the dissemination of extended blob data, nodes verify its availability by attempting to download randomly chosen cells, as illustrated by Figure 3. Collecting more random samples means higher confidence in the availability or reconstructability of blob data. The number of cells to sample depends on the maximum acceptable

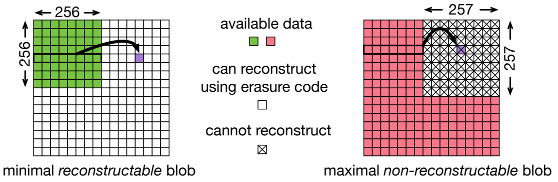


Figure 4: The minimal data enabling reconstruction (left), and the maximal data preventing it (right).

rate of false positives, i.e., of incorrectly determining availability, as we detail next.

The minimal amount of data necessary to enable reconstruction is half of the cells for either 256 distinct rows or 256 distinct columns, as illustrated by Figure 4-left (note that collecting *any* $256 \times 256 = 65,536$ cells *may not* provide this guarantee). The maximal amount of data that can be shared while preventing reconstruction is the 512×512 matrix minus a 257×257 square sub-matrix, as illustrated by Figure 4-right. If a fraction p of cells was not shared in the network, the probability of not hitting an unavailable cell with s samples is $(1-p)^s$. The false positive probability for availability sampling is, therefore, upper-bounded by $\prod_{i=0}^{s-1} 1 - \frac{257\times257}{512\times512-i}$. Discussions in the Ethereum community [21] suggest using $s = 73$ samples, which gives an upper bound false positive probability lower than 10^{-9}. We use this value of $s = 73$ in the remainder of the paper, corresponding to 73×560B $= 40$ KB worth of samples collected per node.

4 PANDAS: OBJECTIVES AND OVERVIEW

PANDAS is a peer-to-peer protocol integrating DAS without requiring modifications to Ethereum. It ensures that extended blob data is propagated to the network and sampled within 4 seconds of the block's creation. This section presents our assumptions, details our objectives, and gives an overview of PANDAS.

4.1 Model and Assumptions

This work is based on the following models and assumptions.

System model. The network comprises N nodes. Aligned with Ethereum, the system is open, but each node $n_i \in N$ is identified by an ID i, i.e., a cryptographic hash of its public key. Nodes periodically advertise to store (and refresh) their ENR records in the underlying Kademlia DHT. The ENR of a node contains its ID, public key, and contact information (IP and port). Nodes can be reached directly using this contact information.

The assignment of validators to nodes must remain unknown [35]. Therefore, it must be impossible to distinguish nodes that host validators from those that do not. To maintain decentralization, nodes must have commodity hardware and network requirements (i.e., a small home server with a 25 Mbps network connection [1]).

Dedicated builders propose blocks. For every slot, the elected proposer selects one block from a builder b. Builders have significantly better capacity and connectivity than nodes (e.g., a medium-range cloud instance with a recent multicore CPU and 10 Gbps network upload capacity). The selected builder b is responsible for sending extended blob data to the network of nodes. After that, nodes interact peer-to-peer to exchange this data for retrieval and DAS.

Network views. Each node, including builders, maintains a list of all nodes in the system as its *view* V, i.e., V_b is builder b's knowledge of existing nodes, and V_{n_1} is that of a node n_1. Views are filled by periodically crawling the DHT [20], which typically takes about a minute [12, 58, 63]. Views can be inconsistent (for any two nodes n_1 and n_2, we do not assume that $V_{n_1} = V_{n_2}$, and similarly for builders). They may also be incomplete ($V \cap N \subseteq N$) and contain departed nodes ($V - N \neq \emptyset$). However, thanks to the periodic crawls, views constantly converge towards the actual set of nodes.

Fault/attack model. Nodes may crash (fail-silent [56]) or refuse to answer incoming requests. Builders are rational and follow their economic interests. They aim to obtain block construction rewards while spending as few resources as possible. A selected builder can attempt a *data withholding* attack, i.e., avoid sharing some or all of the blob data to save on operational cost or because it did not produce it. However, it does not attempt to send *incorrect* data to the network, as doing so would be against its economic interests (i.e, this will be detected when checking KZGP and lead to no rewards, while still incurring bandwidth costs).

4.2 Objectives

The primary objective of PANDAS is to ensure that dissemination and sampling happen within four seconds of creating a block, and layer-2 clients easily retrieve blob data. We set the following goals:

- **[Robustness]** Sampling must meet the 4-second deadline even with a large fraction of unresponsive nodes and/or when nodes and builders have inconsistent views.
- **[Scalability]** Timing guarantees must hold with increasing system size, and the load imposed on nodes and builders must remain compatible with hardware profiles recommended for decentralization [1].
- **[Flexibility]** Participating entities may be free to implement local strategies for interacting with other system members, aligning with their financial incentives.

We target the *tight fork-choice* rule [16], i.e., DAS sampling is required before attesting to a block by committee members: a block with valid transactions but unavailable data is attested as invalid. As a result, we do not modify the consensus protocol beyond adding sampling as a verification step for nodes hosting committee member validators. This contrasts with the *trailing fork-choice* rule that postpones sampling to later, and requires non-trivial changes to consensus to be able to revert blocks with unavailable blob data. Similarly, we do not wish to modify Ethereum's discovery protocols (i.e., the DHT holding ENRs) and assume nodes use unmodified crawl mechanisms to collect their views [12, 58, 63].

4.3 PANDAS in a nutshell

The high-level principles of PANDAS are illustrated by Figure 5. At the beginning of a slot, the node hosting the elected proposer selects a block from one of the builders (❶). This block is disseminated via a dedicated, system-wide GossipSub channel (❷). At the same time, the same node requests the builder to publish blob data in the network. The builder *seeds* the network with extended blob cells, using direct communication to nodes in its view (❸). Every node is assigned a subset of cells that it must keep in *custody* for the rest of

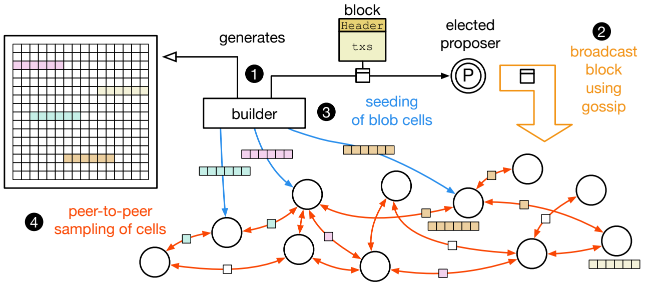


Figure 5: Distributed interactions following the selection of a new block by the proposer (❶). In parallel to the gossip block dissemination (❷), the builder distributes extended blob data to nodes in the network (❸). All nodes interact peer-to-peer to *consolidate* their assignment and collect random samples (❹).

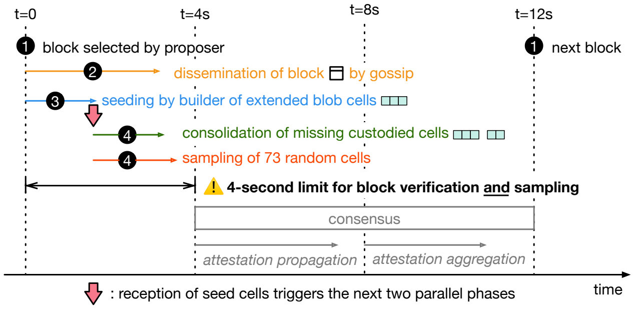

t=0
t=4s
t=8s
t=12s
❶ block selected by proposer
❶ next block
❷ dissemination of block by gossip
❸ seeding by builder of extended blob cells
❹ consolidation of missing custodied cells
❹ sampling of 73 random cells
⚠ 4-second limit for block verification and sampling
consensus
attestation propagation
attestation aggregation
time
: reception of seed cells triggers the next two parallel phases

Figure 6: Timeline of events within a slot. Starting from the selection of a new block by the proposer (time 0, ❶), two concurrent processes start for nodes that must both terminate within 4 seconds: block dissemination (❷) and extended blob data dissemination (❸), consolidation, and sampling (❹).

the network. The builder may send only a subset of this assigned data to each node directly. To serve all assigned data, nodes fetch missing cells from other nodes through *consolidation* (❹). In parallel, nodes select 73 cells randomly and send requests to nodes whose responsibility includes them, implementing the *sampling* phase.

Figure 6 represents the timeline of operations. The dissemination and verification of block and blob data are concurrent. Nodes initiate consolidation and sampling when they receive their seed cells from the builder. A node supporting an active validator can vote for a block if the block verification *and* the data availability sampling are successful before the 4-second deadline.

As ENRs do not allow distinguishing between nodes supporting validators and nodes that do not, all correct nodes are expected to custody data as long as they are registered in the DHT. We also assume all correct nodes perform DAS. In particular, we avoid having only committee members performing DAS, as it would reveal the association between validators in the committee and nodes [35].

Communication between all actors in PANDAS is based on one-way UDP networking with no signalling overhead (i.e., there is no establishment of connections or keep-alive messages). We stress that all Ethereum nodes already use UDP in the discovery protocol required to join the network [29]. Blob data is public and, therefore, sent unencrypted, avoiding a time-consuming encrypted channel establishment. Messages are authenticated with a digital signature using the recipient's public key. KZGPs further allow the authenticity of the received blob data to be verified. Peer-to-peer requests may fail silently due to packet loss or nodes that are unresponsive or have failed. To alleviate this and meet the deadline, PANDAS relies on builders adopting efficient seeding strategies, reconstructing cells using the erasure code, and nodes employing an adaptive fetching strategy that adapts request redundancy and aggressiveness to the available time budget.

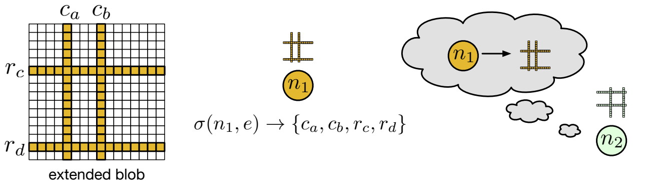


Figure 7: Assignment σ of cells to nodes. Node n_1 is assigned columns c_a and c_b and rows r_c and r_d in epoch e. Any node n_2 that knows n_1 deterministically determines its assigned cells, regardless of the rest of its view $V_{n_2} \setminus n_1$.

Similarly, the impact of incorrect nodes that do not participate in custody and consolidation is mitigated by redundancy. Incorrect nodes that forfeit the sampling phase only reduce the system load.

In the following sections, we detail the components of PANDAS. We start with the deterministic association between blob data and nodes (Section 5). Then, we present the three phases of *seeding*, *consolidation*, and *sampling* (Section 6). We finally detail the adaptive fetching strategy (Section 7).

5 CELL TO NODES ASSIGNMENT

The first component of PANDAS is an assignment between blob data and nodes. A function $\sigma(n_i)$ returns a list of cells from the 512×512. Node n_i is tasked with their custody, i.e., hosting and serving these cells for sampling queries and access by layer-2 participants.

All nodes and builders know σ, as illustrated by Figure 7. We set two requirements for σ: it must be *deterministic* and *short-lived*. Determinism means $\sigma(n_i)$ must be computed identically by two nodes n_a and n_b even if $V_{n_a} \neq V_{n_b}$.[2] Short-liveness means that the assignment must change periodically and be unpredictable. This prevents the emergence of attacks based on eclipsing nodes in charge of specific cells [48, 66] or censorship of specific data [61].

Even though adjacent cells likely contain data for distinct layer-2 protocols, storing them together on the same node favors efficient reconstruction, as it requires fetching multiple cells from the same row or column. Thus, PANDAS assigns complete rows and columns to each node. The number of rows and columns assigned to each node is a globally known parameter. By default, we use eight distinct

[2] Using consistent hashing, as in DHTs, does not meet this requirement: if n_a knows a node n_c that n_b does not know, n_a may associate cells to n_c that n_b associates to n_a.

rows and columns per node. Each node hosts $8 \times 512 + 8 \times (512 - 2)$ cells, i.e., $8,176 \times (512 + 48) = \simeq 4.4$ MB of data.[3]

To enable determinism and short-liveness, the assignment σ is a pseudo-random sortition. This is the exact mechanism used to select committees in the Ethereum consensus. For every epoch, a globally verifiable, pseudo-random sortition decides which nodes will be members of committees or proposers in each slot. This decision uses a pseudo-random number generator (PRNG) and an *epoch seed* known one epoch in advance (32 slots, ≃6.4 minutes) from a combination of random values proposed by validators (i.e., the "RANDAO" state [18]). PANDAS builds upon this existing mechanism by seeding the assignment function σ for an epoch e with its corresponding epoch seed s_e. We extend the definition of σ to include the epoch number, i.e., function $\sigma(n_i, e)$ generates eight distinct rows and eight distinct columns for n_i using a PRNG seeded by s_e.

6 PANDAS PROTOCOL PHASES

We detail the PANDAS protocol phases: seeding, consolidation, and sampling, illustrated in Figures 5 & 6. The latter two are concurrent.

6.1 Seeding phase

The interactions start with an initial *seeding* phase. This phase starts when a proposer selects a block from a builder b. In parallel to sending the block to the network via gossip, the proposer asks b to seed the corresponding blob data to the network.

All nodes know the proposer's identity and public key before the slot starts. However, they do not know who b is. Due to the strict time constraints, nodes cannot wait to receive the block via gossip to learn this information and start accepting blob data. To allow nodes to distinguish legitimate blob data, the proposer provides the builder with a digital signature binding b's identity (including its IP address) to the proposer's private key. This signature is attached to every seeding message.[4]

An objective of PANDAS is flexibility, i.e., the possibility for different actors to implement various strategies. This principle applies to blob seeding strategies. Builders are rational; their interest is in operational costs, particularly outgoing bandwidth. They also wish to maximize profits through block production rewards, which depend on the success of DAS.

A naive approach could be to have the selected builder b send all cells in $\sigma(n, e)$ to every node $n \in V_b$. The necessary outgoing bandwidth now depends on the size of the builders' view, close to or equal to that of the entire network. With ≃4.4 MB per node (eight rows and eight columns) and, say, 10,000 known nodes, the necessary bandwidth budget is 42.9 GB (343.7 Gb). With a 10 Gbps connection, as available with modern, medium-end cloud instances, the process takes more than 30 seconds, largely missing the 4-second deadline.

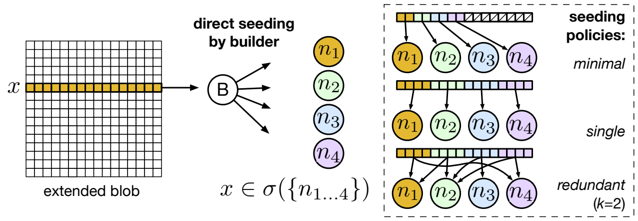


Figure 8: Three seeding policies. The *"minimal"* policy splits the first half of each row or column amongst known peers having it in their assignment. The *"single"* policy splits the entire row or column. The *"redundant"* policy shares each split to k nodes.

A better approach is to send a fixed amount of data to the network and determine a level of redundancy for the cells within each row and column. For a row (or column) x, b decides which cells of $x_1, \ldots, x_{512}$ to send to the network and with what degree of redundancy. It dispatches these cells to the nodes assigned to x in the current epoch e that it knows, i.e., $V_b(x) = \{n \in V_b \mid x \in \sigma(n, e)\}$. Each node in $V_b(x)$ receives only a *subset* of its assigned cells. Therefore, each node must still fetch the missing cells from its peers, a process we call *consolidation* that we detail in the following subsection.

Seeding policies. Figure 8 illustrates three example policies.

A *"minimal"* policy sends a single copy of *half* of the cells of x, i.e., $x_1, \ldots, x_{256}$ (i.e., the minimal amount of data necessary to reconstruct the row or column). The builder splits $x_1, \ldots, x_{256}$ into $|V_b(x)|$ parcels of adjacent cells and distributes them randomly to up to 256 nodes in $V_b(x)$. It repeats the process for all rows and columns. The total amount of data sent out is $256 \times 256 \times (512 + 48) = 35$ MB of data. This strategy is exceptionally fragile to message loss. We primarily use it as a baseline for the builders' costs.

A second, *"single"* policy leverages the redundancy allowed by the erasure code. It operates similarly to the minimal policy but sends a single copy of *all* of the cells of each row or column x, split to up to 512 nodes in $V_b(x)$. In total, it sends out the size of the extended blob, i.e., 140 MB. This strategy's rationale is that even if half of the cells are lost, nodes can still reconstruct the row or column using the erasure code.

The third, *"redundant"* strategy, adds further redundancy by sending k copies of each cell. It starts from the single policy, splitting the cells of x between nodes in $|V_b(r)|$. Then, each parcel is further assigned to $k - 1$ randomly selected distinct nodes in $|V_b(r)|$. We use $k = 8$ by default for this strategy. The outgoing bandwidth usage for the builder is, therefore, 1,120 MB = 1.09 GB of data. This is precisely the volume of data a builder would use to send every row and every column to GossipSub [64] channels for dissemination, as the typical fanout for the root of GossipSub dissemination trees is eight peers [62]. As using GossipSub channels is the approach proposed by concurrent designs to integrate DAS in Ethereum [5, 14, 42], this allows us to compare the performance of PANDAS under the same resource utilization.

6.2 Consolidation phase

The objective of the second phase, consolidation, is for a node n to rapidly get hold of all the cells of rows and columns assigned by

[3] While this is the expected amount of data stored by a node to enable DAS, nothing prevents collecting more. Typically, nodes participating in layer-2 protocols may obtain all relevant data and cache it for other participants in their network.

[4] While the proposer's signature allows verifying the legitimacy of b, the correctness of the received cells' KZGP from b cannot be checked against the KZGC before receiving the block and its blob-carrying transactions. It is, however, not in the builder's interest to send fake blob data that will eventually cause it to lose the rewards.

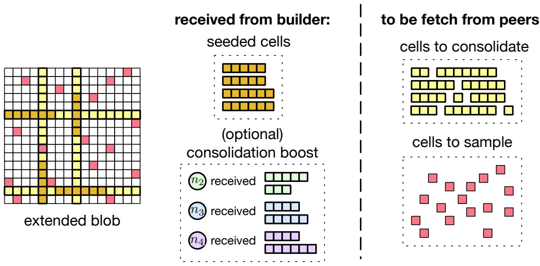


Figure 9: Consolidation and sampling phases must fetch cells from other nodes. Information from the builder consists of initial cells and an optional consolidation boost map informing n what other nodes received as seed data.

$\sigma(n, e)$. Thanks to the erasure code, collecting half of the cells of a given row or column is sufficient to consolidate it.

Consolidation starts at n upon reception of seed cells from the builder b. If n receives a request from another node linked to a slot for which it has not yet received its seed cells, it activates a timer (we use a default value of 400 ms). Consolidation starts without seed data if the timer expires before n receives cells from b because of a packet loss or because b does not know n yet (i.e., $n \notin V_b$).

The orchestration and timing of requests for fetching missing cells are delegated to PANDAS's fetching strategy, shared with the sampling phase, and detailed in the next section.

To fetch cells missing for consolidation, n contacts peers with overlapping rows and columns, $C_n(x) = \{n' \in V_n \mid x \in \sigma(n, e) \wedge x \in \sigma(n', e)\}$. In a view with 10,000 nodes assigned eight rows and eight columns each, each row or column is assigned to $\frac{10000\times(8+8)}{512\times2} \simeq 156$ nodes on average. Thus, $C_n(x)$ contains about 624 peers. Depending on the builder's seeding strategy, each may have received only a subset of the data. Asking many peers for cells may increase the chances of "hitting" the ones that received the needed cells via seeding, but it leads to many duplicates. In contrast, asking only a few random peers may require the selected nodes to finish their consolidation to respond, leading to a lengthy response delay.

A fast and effective consolidation aligns with the economic interests of the builder. If consolidation is fast and efficient, it improves the odds that sampling finishes on time, and the builder may afford to send less data to the network. We improve these two factors with *consolidation boosting*, as illustrated by Figure 9. The builder b attaches to the seeding message to n a map *CB*. For every row and column $x \in \sigma(n, e)$, $CB(x)$ lists the cells received by other nodes $n' \in V_b$ where $x \in \sigma(n', e)$. The consolidation boosting map *CB* allows n to know which nodes are likely to receive specific cells faster and prioritize them for the requests.

6.3 Sampling phase

The third phase of PANDAS is the *sampling* phase. It starts at the same time as consolidation and takes place concurrently.

Node n randomly selects 73 cells to sample. This selection must be unpredictable (i.e., unlike σ). For every sample, n can determine the nodes hosting an intersecting row or column in V_n. In a 10,000-node network, 156 nodes on average can have a copy of a given cell. The selection of targets for sampling and the orchestration and scheduling of requests are delegated to PANDAS's fetching algorithm that we describe next.

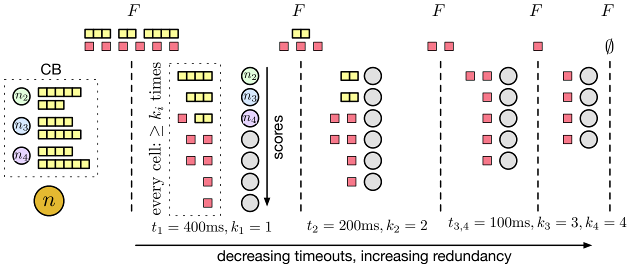

F
F
F
F
F
$\emptyset$
CB
n_2
n_3
n_4
n
every cell: $\geq k_i$ times
scores
$t_1 = 400\text{ms}, k_1 = 1$
$t_2 = 200\text{ms}, k_2 = 2$
$t_{3,4} = 100\text{ms}, k_3 = 3, k_4 = 4$
decreasing timeouts, increasing redundancy

Figure 10: Node n determines a set of nodes to query at each round. The adaptive strategy adjusts the redundancy of queries (i.e., the number of nodes queried for each missing cell) and the timeout before the next round.

7 ADAPTIVE FETCHING

Both consolidation and sampling require fetching cells from other nodes. This collection is handled as a single task by the adaptive fetching algorithm we detail in this section.

The fetching algorithm inputs a set of cell identifiers, as illustrated by Figure 9. In addition, it may receive a consolidation boost map *CB*. The algorithm aims to retrieve all cells before the 4 s deadline.

Target nodes are identified by a node n from its view V_n using the assignment σ. Some of these nodes may be offline or unresponsive. As PANDAS uses connectionless communications using UDP, the network may silently lose queries or response messages. Sending queries for cells sequentially bears the risk of missing the deadline. On the contrary, sending queries to multiple nodes that hold a copy of each desired cell generates a swarm of messages in the network. This leads to congestion risks, suggesting the need for a compromise between cautious fetching initially and a more aggressive approach with more redundant queries as the deadline approaches.

Consolidation processes at different nodes are executed concurrently. A queried node n_i may have a cell c in its assignment $\sigma(n_i, e)$ but have not yet received it from the builder or through consolidation. Nodes receiving a query for assigned cells they do not yet have buffer this query and respond when they can (if the cells are never received, they never respond, i.e., there is no negative acknowledgment). Thus, a querying node may allow sufficient slack time for queried nodes to respond, particularly early in the slot.

Fetching algorithm. Fetching operates in rounds, as illustrated in Figure 10. It adapts query redundancy and timeouts as time progresses and the deadline nears. For this purpose, each round i is associated with a timeout t_i and a redundancy factor k_i. Algorithm 1 details the process at a node n. The FETCH procedure receives a set of cells to fetch F and an optional consolidation boost map *CB* (Algorithm 1). Node n considers as *queryable* nodes all of its view V_n upon the initial call to FETCH, saved as a working copy Q (Algorithm 1). Any node in Q will be queried at most once. The fetching process is in three steps: *scoring*, *planning*, and *execution*.

In the scoring step (Algorithms 1 to 1), queryable nodes in Q are assigned a score, i.e., the number of their assigned cells still missing for n (Algorithm 1). If a consolidation boost was received, nodes are

Algorithm 1 Adaptive fetching at node n in epoch e.

```
procedure FETCH($F$, $CB$)
    $Q \leftarrow V_n$; $i \leftarrow 1$                                  # Queryable nodes and round number
    while $F \neq \emptyset \wedge i < i_{\max}$ do                      # Until all fetched or too many rounds
        for each $q \in Q$ do                                            # Assign scores to queryable nodes
            $q_{\text{cells}} = \{\sigma(q, e) \subseteq F\}$                    # Cells of interest ...
            $q_{\text{score}} = \|q_{\text{cells}}\|$                            # ... score is number of cells
            if $CB_q \neq \emptyset$ then                                  # If node in consolidation boost map ...
                $q_{\text{score}} \leftarrow q_{\text{score}} + (|F \cap CB_q|) \times cb\_boost$
                                                   # ... boost score for each cell received by seeding
        sort $Q$ by decreasing node score as $q_1, \ldots, q_{|Q|}$
        $P = \emptyset$; $U = F$; $j = 1$                                # Query Plan and cells Under redundancy
        while $U \neq \emptyset \wedge j \leq |Q|$ do                    # Should/can plan more queries
            if $(q_j.\text{cells} \cap U) \neq \emptyset$ then             # At least one cell of interest
                $P \leftarrow P \cup (q_j, q_j.\text{cells} \cap U)$         # Plan query
                $U \leftarrow \{c \in F \mid |\{p \in P \mid c \in p.\text{cells}\}| < k_i\}$
                                                   # Update set of cells with insufficient redundancy
            $j \leftarrow j + 1$                                           # Consider next node in sorted $Q$
        for each $p$ in $P$ do                                           # Send out queries from the query plan
            QUERYCELLS($p$.node,$p$.cells)                                # UDP async. query
            $Q \leftarrow Q \setminus p$.node                              # Nodes are queried only once
        SLEEP($t_i$); $i \leftarrow i + 1$                                 # Wait before next round
    return $(F \neq \emptyset)$                                          # Success if all cells fetched within round limit
procedure UPONRECEIVE($C$)                                               # Receiving a set of cells $C$
    $F \leftarrow F \setminus C$                                           # Receive new cells
    while $\exists$ row or column $x$ with $[256 : 512)$ cells do          # Can use code
        $x \leftarrow$ RECONSTRUCT($x$,$R$)                                 # Reconstruct full row/column
        $F \leftarrow F \setminus \{c \in x\}$                             # No need to fetch reconstructed cells
```

given a score boost of *cb_boost* for each cell declared as seeded by the builder and missing from F (Algorithm 1). The set of queryable nodes is then sorted by decreasing score values (Algorithm 1).

The planning step (Algorithms 1 to 1) prepares the set of queries as a set P. Each planned query $p \in P$ is associated with a node p.node and queried cells p.cells. Each missing cell from F must be queried from k_i nodes. Starting from the node with the highest score, q_1, the step greedily selects nodes with cells of interest as long as this criterion is not met. For this, it maintains a set U listing the cells for which insufficient redundancy currently exists in P. A node q_j is planned to be queried for cells with insufficient redundancy (Algorithm 1), before updating U (Algorithm 1).

Finally, the execution step sends out the queries asynchronously (Algorithms 1 to 1) before waiting for t_i ms before the next round. A queried node is removed from Q and is not used again. Upon correct reception by the target node, the handler either responds with the queried cells if all are available or buffers the query for a delayed reply. The response is received by the UPONRECEIVE function as a set of cells C (Algorithms 1 to 1).[5] When receiving new cells, the algorithm checks if an incomplete row or column now contains 256 or more cells (Algorithm 1) and, if so, reconstructs them (Algorithm 1).

Default parameters. The fetching algorithm is primarily parameterized by the round durations and query redundancy vectors t and k, as well as the score boost for consolidation *cb_boost*. We use the following universal parameters, but stress that nodes could select them differently, e.g., based on local connectivity.

In the first round, $i = 1$, the strategy aims to maximize the number of cells received (and reconstructed) using as few messages as possible (i.e., $k_1 = 1$). We use a duration of $t_1 = 400ms$ based on estimated time for the builder to send out initial cells and on inter-node latencies, as we will detail in Section 8. In subsequent rounds, we reduce this time by half but no lower than $100ms$, i.e., $t_2 = 200ms$ and $\forall j \geq 3, t_j = 100ms$ (up to t_{50}). Similarly, we increase the aggressiveness of queries by increasing the redundancy factor by two every round until a maximum of 10, i.e., $r_2 = 2, r_3 = 4, \ldots, \forall j \geq 6, r_j = 10$. Finally, we set $cb\_boost = 10,000$ to give an overwhelming advantage to nodes with seeded cells of interest.

8 EVALUATION

We structure our evaluation around the following claims:

- **C1:** PANDAS completes DAS within 4 s and supports the tight-fork choice rule under Danksharding requirements.
- **C2:** PANDAS bandwidth requirements for nodes are below Ethereum suggestions for decentralization (25 Mbps [1]) and, for builders, below typical cloud offerings (10 Gbps).
- **C3:** PANDAS satisfies **C1** even under a high percentage of unresponsive nodes and with highly inconsistent views.
- **C4:** PANDAS satisfies **C1–C3** scaling up to 20,000 nodes.
- **C5:** Relying on existing peer-to-peer overlays (Gossipsub [64] and Kademlia [50]) for DAS does not allow satisfying **C1**.

PANDAS is implemented in Go, extending `libp2p` [4], the network stack of the Ethereum Geth client [3]. Block dissemination relies on `libp2p`'s GossipSub implementation.

We aim to evaluate the PANDAS prototype in a real-world environment and verify its scalability in large networks, up to 20,000 nodes. Achieving both objectives with the PANDAS prototype would require a prohibitive amount of resources. We thus opt for a hybrid approach. We deploy 1,000 instances of PANDAS in a cluster, emulating representative WAN latencies. To evaluate PANDAS up to 20,000 nodes, we use a simulator, whose accuracy is validated against deployment results.

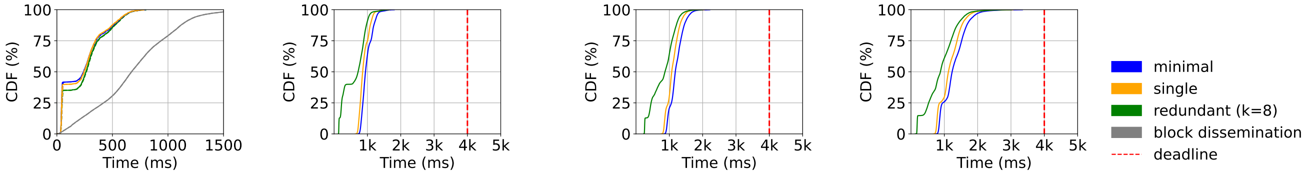
8.1 Prototype deployments

We run 1,000 PANDAS instances on a cluster of 80 servers, each equipped with an 18-core Intel Xeon Gold 5220 CPU and 96 GB of RAM. This level of consolidation (13 instances per server) was selected through careful load testing to avoid CPU contention and increased latencies compared to non-consolidated deployments.

Network emulation. We use network emulation using `tc` to reproduce WAN settings. There is no publicly available data on node-to-node latencies in the Ethereum network. However, a recent large-scale measurement campaign [45] has collected all-pair latencies in IPFS [8], a planetary-scale storage system that shares the scale and decentralization objectives of Ethereum. We use this trace for our network emulation. Round-trip latencies range from 8 ms to 438 ms with an average of 64 ms. The topology contains 10,000 vertices, to which we assign nodes randomly. We limit each node connection to 25 Mbps. We deploy a builder as a dedicated server, with a connection capped to 10 Gbps, assigning it to a vertex in the topology randomly selected among the 20% with the best average latency to all other nodes, i.e., nodes likely deployed in a cloud. UDP communication in the cluster is subject to a packet loss rate of 3%, according to our observations.

Evaluation metrics. Our primary metric of interest is the distribution of completion times for PANDAS's three phases, from the moment the builder is selected. The *time to seeding* is when a node

[5] For clarity, we omit in Algorithm 1 the verification checks performed when receiving C (e.g., verifying the cells KZMPs if/when the block header is available).



(a) Seeding (from start) **(b)** Consolidation (from seeding) **(c)** Consolidation (from start) **(d)** Sampling (from start)

Figure 11: Distribution of times for the three phases of PANDAS across all nodes, for the three seeding strategies. All times are from the start of the slot, except for Figure 11b where time is counted from the reception of the seed cells (as shown by Figure 11a).

has received its initial seed data. *Time to consolidation* and *time to sampling* refer to the periods when a node has received (or can reconstruct) its assigned eight rows and columns, and its 73 random cells, respectively. Additionally, we monitor the bandwidth costs and the number of messages for all nodes and the builder. We consider a fault-free scenario in this section, where all nodes participate in the protocol and have a complete view of the system (i.e., $\forall n \in N, V_n = N$). For all experiments, we present distributions over 10 slots (i.e., 10 cycles of seeding, consolidation, and sampling).

Phases timing. Figure 11 presents the distributions of times to seeding, consolidation, and sampling. We consider the three seeding strategies of Section 6.1: *minimal*, *single*, and *redundant* with $k = 8$. Only solid lines are of interest in this section; dashed ones represent simulator results that we will discuss in the next section. We illustrate the distribution of the reception time of the block via a global GossipSub channel (initiated by a randomly chosen node serving as the proposer), for comparison purposes, in Figure 11a.

We observe that the time to seeding is similar for the three strategies, as our builder's available bandwidth is not a bottleneck (the amount of data sent out is 36.6 MB, 149 MB, and 1,208 MB, respectively, for the three strategies). We observe an impact on the tail of the distributions: the maximum time to seeding is 700, 819, or 936 ms, respectively, for the three strategies (99th percentiles, or P99, are 698, 705, and 715 ms). The "step" around 64 ms corresponds to nodes assigned to well-connected vertices in the emulated topology, which are typically nodes deployed in the same cloud and/or region as the builder. Overall, all nodes receive their seed cells before the end of the first second of the slot.

We present the time to consolidation both from the reception of the seed data by a node (Figure 11b) and from the beginning of the slot (Figure 11c). The builder provides the consolidation boosting map to the nodes. We can observe the impact of the builder's seeding strategy. The minimal strategy results in a consolidation taking up to 2,2213 ms (P99=1,756 ms) from the reception of the seed data, and the single strategy has a maximum time of 2,046 ms (P99=1,595 ms). In contrast, the redundant strategy reduces this time to 1,985 ms (P99=1,558 ms). Median times to consolidation for the minimal, single, and redundant strategies (from the beginning of the slot) are 1,178 ms, 1,072 ms, and 869 ms, respectively.

The time to sampling distribution, our primary metric of interest, is given by Figure 11d. This distribution depends on the builder's seeding strategy, which impacts the time to consolidation. The minimal strategy results in a maximum of 3,341 ms (P99=2,303 ms); still, 100% of the nodes fetch their samples by the deadline. The single strategy also meets the deadline, with a maximum delay of 3,062 ms, (P99=2,068 ms). Finally, the redundant strategy matches the deadline safely for all nodes, with a maximum of 3,009 ms (P99=2,020 ms). The median times to sampling are, respectively, 1,235 ms, 1,122 ms, and 882 ms. The reduction in sampling times with increased availability of seed cells (via increased redundancy) is due to reduced contention on peer bandwidth, which in turn speeds up the fetching operation. We observe, however, that if the block dissemination latency (Figure 11a) were to be added to these times, meeting the 4 s deadline would be at risk for many nodes, even with the redundant strategy. This confirms our claim that DAS must start *concurrently* to block dissemination if we are to integrate it with consensus under the tight fork-choice rule.

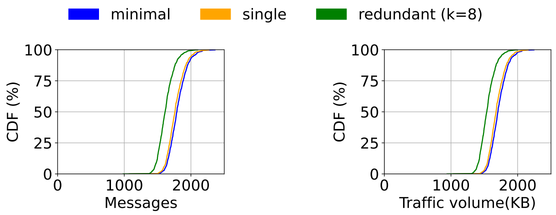


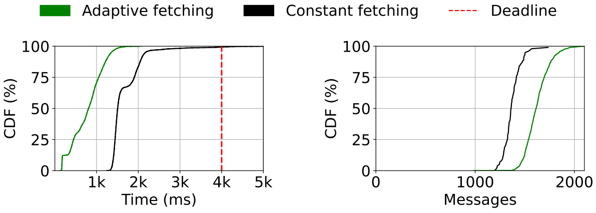
Figure 12: Distribution of messages and traffic volume for fetching across nodes, for different seeding strategies.

Bandwidth consumption. Figure 12 presents the distribution of the number of messages used by nodes in the fetching phase, and the corresponding traffic volume, summed for both directions. The redundant seeding strategy results in fewer messages exchanged between nodes and, as a result, lower bandwidth requirements. This is because more nodes already hold the requested cells, which reduces the need for retries (i.e., fewer rounds) during consolidation and sampling. Even with the single seeding strategy, which only marginally differs from the minimal one, the requirements are far below the Ethereum recommendations of 25 Mbps; the maximum traffic volumes are 2.26, 2, and 1.99 MB for the three strategies.

Fetching analysis. Table 1 presents an analysis of the progress of fetching for the first four rounds. All values discussed in this paragraph are averages over the 1,000 nodes, together with the standard deviation. We use the redundant seeding strategy, and nodes receive 2420 cells (± 180). Starting from 4,174, the number of requested cells decreases as the coverage of F (i.e., set of cells to

Round	1	2	3	4
Messages sent	341 ± 20	261 ± 58	185 ± 35	113 ± 22
Cells requested	4174 ± 100	2426 ± 96	923 ± 63	294 ± 40
Replies received in round	228 ± 22	143 ± 14	120 ± 20	69 ± 25
Replies received after round	107 ± 39	114 ± 25	56 ± 20	61 ± 3
Cells received in round	2420 ± 180	949 ± 170	535 ± 82	191 ± 22
Cells received after round	1128 ± 113	1478 ± 91	383 ± 52	23 ± 8
Received cells duplicates	0 ± 0	187 ± 42	142 ± 29	64 ± 12
Cells reconstructed	615 ± 126	566 ± 90	86 ± 29	32 ± 17
Cumulative coverage of F	56%	81%	96%	99%

Table 1: Fetching algorithm performance in successive rounds (values averaged over all nodes, ± is the standard deviation).



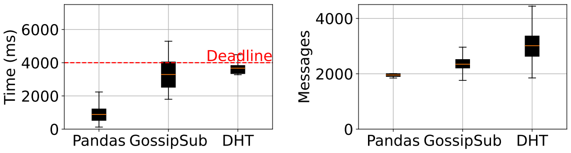
(a) Time to sampling **(b) Messages in and out**

Figure 13: Comparison of the performance of adaptive fetching, as used by PANDAS, and a non-adaptive approach.

fetch) increases, either through reception or reconstruction (e.g., 615 reconstructed cells in the first round). We distinguish between replies received *in* a round i, i.e., before the timeout t_i expires, and after. The latter case leads to redundant requests but illustrates the tradeoff between caution and eagerness Algorithm 1 implements. A majority of requests result in replies before the timeout, and a majority of cells are received on time in the round. Receptions after the round generally occur with a significant delay; adjusting timeouts to account for such tail latencies leads to lower success rates. While Table 1 only shows the first four rounds—after which 99% of the nodes have completed fetching—the process requires up to 6 rounds for the slowest nodes (P90=3, P99=4).

Impact of adaptive fetching. We evaluate in Figure 13 the impact of adaptive fetching. We consider the redundant seeding policy, i.e., the green distribution in Figure 13a is the same as in Figure 11d. For comparison, we employ a *constant* fetching strategy, which utilizes a fixed timeout for all rounds (t = 400 ms) and a fixed redundancy (k = 1), as represented in black. The constant strategy uses fewer messages, as it asks only a minimum of one node for each missing cell in each round, and leaves more time for nodes to respond. However, it drastically impacts the time to sampling, resulting in a maximum of 4,129 ms (P99=3,513 ms, median=1,546 ms), and some nodes miss the deadline. This illustrates the interest of dynamically adapting aggressiveness and redundancy to cope with the tight time constraints imposed by the tight fork-choice rule.

Comparison to alternative proposals. We finally compare our approach to two alternative methods based on the use of existing peer-to-peer protocols available in `libp2p`.



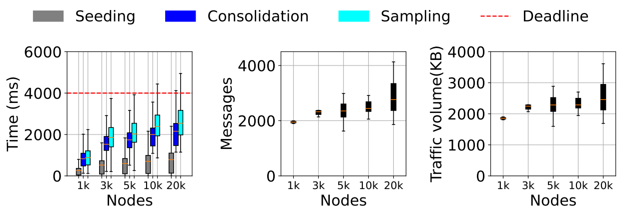
(a) Time to sampling **(b) Messages**

Figure 14: Distribution of time to sampling and messages compared to baselines based on GossipSub and the Kademlia DHT.

Some proposals [14, 42, 59], that we will further discuss in Section 10, suggest using GossipSub [64] for the dissemination of rows and columns (i.e., our seeding phase), but typically do not detail how random sampling should happen. We instantiate this idea by having all nodes subscribe to GossipSub channels corresponding to each unit of custody—that is, each group of eight rows and eight columns—they are assigned by σ. We disable explicit consolidation, but instead rely on GossipSub's gossiping within each channel to disseminate the assigned cells, and use the same sampling phase as in PANDAS. Therefore, the main difference is that the dissemination of seed cells occurs through peer-to-peer gossip within each channel, rather than through direct seeding by the builder as in PANDAS. In this 1,000-node network, each GossipSub channel involves approximately $\sim$ 16 nodes assigned to the corresponding unit of custody. The builder sends $k = 8$ copies of each unit of custody to the nodes in the corresponding GossipSub channel, which is configured with the default fanout of eight peers. As a result, the builder's outgoing traffic volume is the same as in the redundant seeding strategy of PANDAS, i.e., eight times the total blob size.

Another proposed approach [13] is to use the Kademlia DHT [50] for storing and retrieving cells using multi-hop routing. We implement it by mapping rows and columns to one dimension and splitting it into *parcels* of 64 adjacent cells. Parcels are then stored in the DHT by the builder using the put(key) operation. To ensure a fair comparison with PANDAS and the GossipSub baseline, the builder performs eight put(key) operations per parcel, storing it at each of the eight closest peers to the hash of the parcel's contents—therefore, the builder uses the same total bandwidth as in the other approaches. Nodes are responsible for the range of keys (and, therefore, parcels) assigned by the DHT, and we disallow consolidation. Sampling uses get(key) operations to fetch necessary parcels.

Figure 14 shows the distribution of time to sampling for PANDAS using the redundant seeding strategy (k = 8) and for the two baselines, as well as the distribution of the number of messages. With 1,000 nodes, 24% of GossipSub nodes and 17% of DHT nodes fail to complete sampling within the 4 s deadline. The average sampling delay for GossipSub nodes is 3,660 ms (P99=3342 ms), while PANDAS nodes complete sampling significantly faster, i.e., on average in 882 ms (P99=1935 ms), with all nodes completing well within the 4 s deadline. In terms of messaging, the DHT and GossipSub baselines incur significantly higher overhead in the number of messages compared to PANDAS. On average, PANDAS, GossipSub, and DHT nodes send 1,613, 2,370, and 3,021 messages. For the DHT baseline, the messaging overhead of storing and retrieving parcels is especially high due to multi-hop routing (i.e., DHT traversal).



(a) Time

(b) Fetching messages

(c) Fetching bandwidth

Figure 15: Simulation of seeding, consolidation, and sampling times for PANDAS with a various number of nodes.

8.2 Large-scale simulations

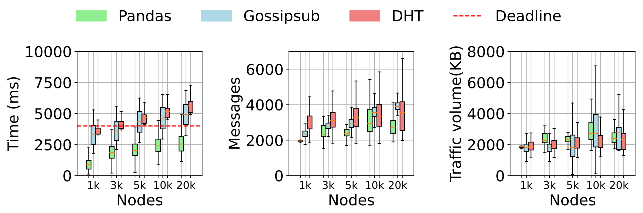
In addition to the prototype deployment detailed in the previous subsection, we implement PANDAS protocols in PeerSim [51], a Java simulator for large-scale evaluation of peer-to-peer systems. This implementation closely follows the one over `libp2p`. We also implement the two baselines detailed above.

We simulate the same latency trace as for the deployment.[6] We also enforce a fixed 3% loss rate for UDP packets as experienced in the testbed. When running on a server with 256 GB of memory, the simulator can scale up to 20,000 nodes.

Simulator validation. Before considering larger scales, we validate that the simulator results match those of the deployments. In all plots of Section 8.1, dashed lines report the results obtained with 1,000 simulated nodes. In all cases, the two lines are (almost) indistinguishable. Our evaluations at smaller scales (not shown) have the same property. The validation of simulation results at moderate scales gives us confidence in the simulator's ability to provide accurate results at higher scales.

Scaling. We first investigate PANDAS's scalability with 1,000 to 10,000 nodes and up to 20,000 nodes. Figure 15 presents the distribution of times to seeding, consolidation, and sampling using the redundant seeding strategy (Figure 15a) and the corresponding messages (Figure 15b) and bandwidth (Figure 15c). With 10,000 nodes, the current scale of the Ethereum network [2], all nodes successfully sample before the 4 s deadline. With 20,000 nodes, 10% fail to meet the deadline, mapping to nodes with poor simulated connectivity (connected from remote area of the geo-distributed network). We identify the scattering of seed data and the cost of consolidation as the primary reasons, as nodes have to contact more peers to collect their rows and samples, and take more time before being able to answer sampling requests. Nodes located in clouds do not suffer from significantly higher times, highlighting the need to host validator-hosting nodes in well-connected infrastructure.

The impact of the increasing scattering of seed cells with larger network sizes is also reflected in Figure 15b and Figure 15c, which show the number of messages and traffic volumes for fetching cells during consolidation and sampling. The average number of messages per node for networks of 1K, 3K, 5K, 10K, and 20K nodes is 1,956, 2,231, 2,247, 2,291, and 2,443, respectively. The corresponding peak traffic volumes are 1.9, 2.1, 2.2, 2.2, and 2.4 MB. We observe that even in the most demanding scenario with 20K nodes, the maximum traffic volume is transmitted in approximately 2.2 seconds, keeping the average bandwidth requirement well within the 25 Mbps target.

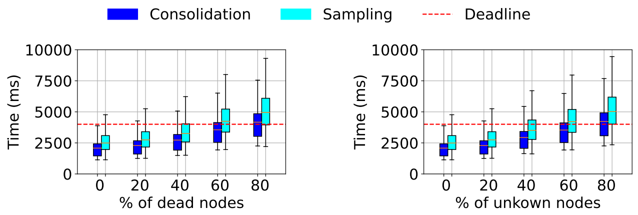


(a) Time to sampling

(b) Messages

(c) Bandwidth

Figure 16: Simulation of blob dissemination time for PANDAS and the two baselines, for various number of nodes.



(a) Dead nodes

(b) Out-of-view nodes

Figure 17: Simulation of time to consolidation and time to sampling for increasing number of dead nodes and out-of-view nodes, in a 10.000-node network.

Comparison to baselines. Figure 16 compares PANDAS to the two baselines in scales up to 20,000 nodes. Results for 1,000 nodes are consistent with the ones for testbed deployment reported in Figure 14. While the GossipSub-based baseline meets the deadline for a majority of nodes with 1,000 nodes, it fails to do so starting with 5,000 nodes. However, it plateaus for higher node counts, as GossipSub topics become more efficient with a higher number of participants. The DHT-based baseline is unable to meet deadlines for most nodes at all scales and shows linearly increasing times to sampling for increased system sizes. For both systems, the gap to PANDAS in terms of time-to-sampling latency widens as the system size grows. The number of messages is also significantly higher for the baselines than for PANDAS, with important variability for the GossipSub-based baseline as the system size increases.

Behavior under faults. We evaluate PANDAS's robustness under two types of faults: *dead nodes* and *out-of-view nodes*. In the *dead nodes* scenario, a fraction of nodes is assumed to have crashed and do not respond to any messages. The builder and the remaining correct nodes are unaware of these failures. Therefore, the builder seeds data to all nodes, including the dead ones, and includes them in consolidation boost maps. As a result, some seed cells are lost, and correct nodes may attempt to contact dead nodes during fetching which will lead to timeouts and retries. On the other hand, in the *out-of-view nodes* scenario, all nodes are correct and receive their assigned seed cells from the builder. However, each node only has an incomplete view of the network, and these views are not consistent.

[6] When using more than 10,000 nodes, we reuse vertices randomly for the assignment.

For example, if 20% of nodes are out-of-view, each node is only aware of a randomly chosen 80% of the full node set. This affects both consolidation and sampling, as requests may fail due to the sender lacking the knowledge of a suitable peer.

In Figure 17, we vary the proportion of dead or out-of-view nodes from 0% to 80% (in 20% increments) and measure the impact on both time to consolidation and sampling. The network size is 10,000 nodes. We observe that for 0%, 20%, 40%, 60%, and 80% of *dead nodes*, 92%, 83%, 74%, 45%, and 27% of nodes complete sampling within the 4-second deadline, respectively. In the case of *out-of-view nodes*, for 0%, 20%, 40%, 60%, and 80% of nodes being out of view, 92%, 83%, 67%, 47%, and 25% of nodes complete sampling within the deadline, respectively.

In both scenarios, we observe that beyond 50% dead or out-of-view nodes, over half of the correct nodes fail to meet the deadline which would prevent consensus from being reached and causing the blockchain to stall. These scenarios are, however, unlikely in practice and would impact other key mechanisms, such as block dissemination, preventing consensus to succeed anyway.

Summary. Our evaluation using a prototype deployment and large-scale simulations of PANDAS and two baselines confirmes our claims. PANDAS supports DAS within 4 s for all nodes (up to 10,000) and the vast majority of them (for 20,000) and enable the tight fork-choice rule (**C1** and **C4**). The bandwidth requirements for builders and nodes are below Ethereum recommendations and compatible with its decentralization objectives under the PBS principles (**C2** and **C4**). The evaluation of fault scenarios show that PANDAS supports these claims with a large fraction of failed or out-of-view nodes (**C3** and **C4**). In contrast, the two baselines fail to meet these criteria in particular as the system size increases (**C5**).

9 DISCUSSION

We discuss PANDAS and classical concerns in decentralized systems.

Impact of Sybils. A node in Ethereum, and thus PANDAS, does not have to support validators to participate in peer-to-peer interactions, e.g., block dissemination. This opens possibilities for *Sybil* attacks, where an attacker operates multiple nodes to bias the system operation.

Sybils can perform *general* attacks, where they join the DHT and GossipSub channels and stop answering queries or forwarding data, disrupting the system merely by their overwhelming presence. Our evaluation shows that PANDAS is robust against many nodes that ignore sampling and consolidation requests, provided the builder uses sufficient redundancy in its seeding strategy. Proposals for increasing IP diversity in Ethereum's discovery mechanisms [43] could strengthen this robustness.

A *targeted* use of Sybils consists of carefully placing them in the peer-to-peer network to prevent specific nodes from interacting with it (an *Eclipse* attack) or to censor specific information [36, 48, 61, 66]. PANDAS makes the network fully connected and randomized exchanges, making Eclipse attacks irrelevant. This contrasts with designs based on GossipSub trees, where an attacker could position its Sybils as the first neighbors of the builder and disrupt the early dissemination of blob data. Another targeted attack scenario targets specific *content*. In PANDAS, disrupting the sharing of specific blob data would require (1) knowing before blob data dissemination which cell will contain such data and (2) positioning Sybil nodes in the network to make the corresponding row and column difficult to reconstruct (i.e., disallow fetching half of its cells). Condition (1) does not hold as the cell location is known only upon reception of the block. Condition (2) would require the attacker to generate enough identities to control the corresponding row and column, which is highly improbable considering that σ changes unpredictably every 6.5 minutes, less time than what ENR crawling requires.

Limiting openness. As one of PANDAS goals is to avoid modification to Ethereum other mechanisms, it follows its open-network design. An alternative design could limit participation to validator-holding nodes and restrict other nodes to being only observers. This would drastically reduce any potential risk associated with Sybil attacks, as an attacker can only generate one identity for every 32 ETH they hold. It would, however, limit decentralization by switching to a semi-permissioned system, where only stakeholders can participate. *Proof-of-validator* [39] is an anonymous credential scheme based on zero-knowledge proofs (ZKP). It could enable this limitation if integrated with node discovery, i.e., the crawl of the DHT for ENRs. This approach would come at a significant complexity cost, even for an observer (i.e., gathering the list of validators and verifying a ZKP for every crawled ENR).

Handling free riders. Nodes using the system while contributing minimal resources, or free riders, are unavoidable in decentralized systems [37]. In Ethereum, block and blob building, proposal, and validation are incentivized through monetary rewards; however, there is no incentive for interactions within the peer-to-peer network (e.g., for correctly answering ENR discovery requests in the DHT). Similarly, PANDAS does not have incentives for nodes to participate in the consolidation, sampling, and hosting of blob data. Nodes hosting committee members could even vote for blob availability without sampling, if the expected rewards outweigh bandwidth costs.

Also departing from our objective of no modification to Ethereum, a possible direction to include incentives for DAS operations would be to integrate *proof-of-custody* mechanisms [27] and the associated slashing mechanisms with consensus. Proof of custody is a negative incentive in which the builder infrequently inserts in blob data a cryptographic "bomb" targeted at a randomly-selected validator. A node that attests to blob data with a bomb for itself is slashed for a large amount of stake, making the correct behavior of downloading and verifying data more profitable than free riding. As only nodes holding stake can be targeted, and not observers, such a mechanism would probably depend on proof-of-validator integration.

10 RELATED WORK

We start by detailing alternative proposals for implementing DAS in Ethereum. Then, we explore the broader history and foundational literature on data availability. Finally, we discuss earlier P2P techniques designed for purpose-specific data dissemination.

Alternative DAS Proposals. The Ethereum community has so far mainly explored gossip-based approaches to support DAS and the Danksharding roadmap, primarily using GossipSub [64], where each row and column is disseminated through a distinct channel with long-lived subscriptions [11, 42, 59]. The scalability of this approach remains uncertain due to the large number of channels required and potential security risks associated with long-lived node-to-channel assignments [44]. More importantly, the slow, multi-hop propagation of rows and columns resulted in proposals considering the removal of random sampling from the critical path of consensus. Instead, these approaches adopt a "lightweight" custody-based verification, where nodes supporting committee members evaluate the availability of a blob (and vote accordingly) based solely on the successful receipt of their assigned rows and columns [14]. This approach provides significantly weaker assurances of availability compared to random sampling. By foregoing independent verification of randomly chosen cells, validators risk being slashed or locked onto an unavailable chain. This, in turn, could prevent them from forming a minority chain or participating in a manual fork of the canonical chain under social consensus in the event of a malicious majority.

Alternative network-layer mechanisms for DAS have also been evaluated. A recent study [13] highlights the inefficiencies of using the Kademlia DHT [50] for DAS, particularly the overhead of seeding cells to nodes involving traversing the DHT.

Data Availability. The idea of verifying data availability by sampling a block extended with erasure coding was introduced by Al-Bassam *et al.* [7] and later adopted by LazyLedger [6], since evolving into Celestia. Celestia employs a *centralized* approach where validators, i.e., highly resourceful "super" nodes—retrieve and store the complete blob data, while light clients sample the blob from these super nodes. Unlike PANDAS's collaborative approach, where sampling and storage responsibilities are distributed among participants, Celestia's design results in overhead and costs that increase linearly with participation and blob size. Recent work by Nazirkhanova *et al.* [52] explores how erasure coding, combined with homomorphic vector commitments, can ensure verifiable data retrieval in rollups while maintaining storage and communication efficiency.

Recent work proposes alternative methods for node sampling in DAS. Honeybee [67] focuses on Sybil-resistant peer sampling via verifiable random walks. Honeybee assumes fetching random cells requires contacting random nodes, whereas PANDAS selects random cells and takes advantage of their deterministic assignment to nodes. Honeybee's interactive verification at each hop introduces potential latency, making it less suitable for strict timing constraints. Sheng *et al.* [60] propose an alternative approach where a group of oracle nodes collectively store erasure-coded data blobs and provide access to clients. Their work focuses on ensuring the integrity and correctness of the coded blob and relies on at least half of the oracle nodes being honest and reliable. In contrast, PANDAS can function even under a supermajority of nodes failing or experiencing omission faults.

Purpose-specific Data Dissemination Networks. One-hop communication in overlay networks was proposed in the early days of P2P research [32]. The Interplanetary File System (IPFS) [8] adopted a hybrid P2P networking approach by combining multi-hop search (i.e., through a Kademlia DHT) and one-hop communication. More specifically, each peer accesses a few directly connected peers to perform a one-hop search and retrieve popular content [17]. The DHT is used to discover peers hosting (less popular) content through a slow, multi-hop search process. While IPFS supports one-hop retrieval, it is not optimized for the DAS use case, where a large number of directly connected peers must be efficiently utilized to retrieve unpopular content in a timely manner, i.e., chunks of a data blob each hosted by roughly the same number of peers.

Beyond IPFS, several studies explore optimizations for blockchain and distributed systems that improve data dissemination efficiency. Mercury [68] optimizes blockchain transaction dissemination using network coordinates, while Perigee [47] improves Bitcoin's gossip efficiency. Similarly, Berendae et al. [9] propose Hyperledger Fabric optimizations to reduce dissemination delay and improve fairness. While these optimizations focus on transaction propagation rather than DAS, they highlight the importance of optimizing dissemination networks, which aligns with PANDAS's objectives.

11 CONCLUSION

We presented PANDAS, a practical approach to integrated data availability sampling (DAS) in the consensus workflow of Ethereum under the demanding Danksharding objectives. By favoring direct and lightweight communications between nodes, builder-led blob data dissemination, and adaptive fetching mechanisms, PANDAS allows large amounts of layer-2 data to propagate in the Ethereum network and be verified as available under the strict timing constraints imposed by Ethereum consensus.

This work opens interesting perspectives, among which is the design of *adaptive* policies. We presented and evaluated different fixed strategies for the builders and the nodes to follow. However, the design could support automatic adaptation mechanisms that select or update parameters based, for example, on observed networking and fault ratio conditions.

ACKNOWLEDGMENTS

This work was supported by a grant from the Ethereum foundation (grant #FY22-0753). Ramin Sadre and Etienne Rivière are supported by Belgian Wallonia region CyberExcellence program (grant #2110186).

REFERENCES

[1] Ethereum full node vs. archive node. https://www.quicknode.com/guides/infrastructure/node-setup/ethereum-full-node-vs-archive-node.

[2] Ethereum node tracker. https://etherscan.io/nodetracker.

[3] Go ethereum. official go implementation of the ethereum protocol. https://geth.ethereum.org/.

[4] libp2p: A modular network stack. https://libp2p.io.

[5] EIP-7594: PeerDAS. https://github.com/ethereum/consensus-specs/tree/dev/specs/_features/eip7594, May 2024.

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

[7] Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin, and Ismail Khoffi. Fraud and data availability proofs: Detecting invalid blocks in light clients. In *International Conference on Financial Cryptography and Data Security*, FC. Springer, 2021.

[8] Juan Benet. IPFS: content addressed, versioned, P2P file system. *arXiv preprint arXiv:1407.3561*, 2014.

[9] Nicolae Berendea, Hugues Mercier, Emanuel Onica, and Etienne Riviere. Fair and efficient gossip in hyperledger fabric. In *40th International Conference on Distributed Computing Systems*, ICDCS, pages 190–200. IEEE, 2020.

[10] Vitalik Buterin, Dankrad Feist, Diederik Loerakker, George Kadianakis, Matt Garnett, Mofi Taiwo, and Ansgar Dietrichs. Eip-4844: Shard blob transactions. https://eips.ethereum.org/EIPS/eip-4844, 2024.

[11] 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.

[12] Mikel Cortes-Goicoechea and Leonardo Bautista-Gomez. Discovering the Ethereum2 P2P network. In *2021 Third International Conference on Blockchain Computing and Applications*, BCCA. IEEE, 2021.

[13] Mikel Cortes-Goicoechea, Csaba Kiraly, Dmitriy Ryajov, Jose Luis Muñoz-Tapia, and Leonardo Bautista-Gomez. Scalability limitations of Kademlia DHTs when enabling Data Availability Sampling in Ethereum, 2024.

[14] Francesco D'Amato and Ansgar Dietrichs. SubnetDAS - an intermediate DAS approach. https://ethresear.ch/t/subnetdas-an-intermediate-das-approach/17169, 2023.

[15] Francesco D'Amato and Luca Zanolini. Recent latest message driven ghost: Balancing dynamic availability with asynchrony resilience. In *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*, pages 127–142. IEEE, 2024.

[16] Francesco D'Amato, Luca Zanolini, and Roberto Saltini. Das fork-choice. https://ethresear.ch/t/das-fork-choice/19578, May 2024.

[17] Alfonso De la Rocha, David Dias, and Yiannis Psaras. Accelerating content routing with bitswap: A multi-path file transfer protocol in IPFS and Filecoin. Technical report, Protocol Labs, 2021.

[18] Ethereum. Randao: Ethereum random number generator. https://github.com/randao/randao, 2022.

[19] Ethereum. Zero-knowledge rollups. https://ethereum.org/en/developers/docs/scaling/zk-rollups/, 2024.

[20] Ethereum. Discv4 ENR periodic crawls. https://github.com/ethereum/discv4-dns-lists, 2025.

[21] Ethereum community. DAS query analysis notebook. https://colab.research.google.com/drive/1Di1-hBae8tZr1tZqcu1JqYycOFy8FdAy, 2024.

[22] Ethereum foundation. The merge: Ethereum switch to proof-of-stake. https://ethereum.org/en/upgrades/merge/, 2023.

[23] Ethereum foundation. Ethereum roadmap. https://ethereum.org/en/roadmap/, 2024.

[24] Ethereum foundation. Ethereum roadmap: Danksharding. https://ethereum.org/en/roadmap/danksharding/, 2024.

[25] Ethereum foundation. Ethereum roadmap: Proposer-builder separation. https://ethereum.org/en/roadmap/pbs/, 2024.

[26] Ethereum foundation. Optimistic rollups. https://ethereum.org/en/developers/docs/scaling/optimistic-rollups/, 2024.

[27] Dankrad Feist. Proofs of custody. https://dankradfeist.de/ethereum/2021/09/30/proofs-of-custody.html, 2021.

[28] Flashbots. MEV-Boost in a Nutshell. https://boost.flashbots.net, 2024.

[29] Ethereum Foundation. Node discovery protocol v5 - wire protocol. https://github.com/ethereum/devp2p/blob/master/discv5/discv5-wire.md#udp-communication.

[30] Ankit Gangwal, Haripriya Ravali Gangavalli, and Apoorva Thirupathi. A survey of Layer-two blockchain protocols. *Journal of Network and Computer Applications*, 209, 2023.

[31] Vincent Gramlich, Dennis Jelito, and Johannes Sedlmeir. Maximal extractable value: Current understanding, categorization, and open research questions. *Electronic Markets*, 34(1):49, 2024.

[32] Anjali Gupta and Barbara Liskov. One-hop lookups for peer-to-peer overlays. In *9th Workshop on Hot Topics in Operating Systems*, HotOS, 2003.

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

[34] Lioba Heimbach, Lucianna Kiffer, Christof Ferreira Torres, and Roger Wattenhofer. Ethereum's proposer-builder separation: Promises and realities. In *Internet Measurement Conference*, IMC, pages 406–420. ACM.

[35] Lioba Heimbach, Yann Vonlanthen, Juan Villacis, Lucianna Kiffer, and Roger Wattenhofer. Deanonymizing Ethereum validators: The P2P network has a privacy issue. In *USENIX Security Symposium*, 2025.

[36] Sebastian Henningsen, Daniel Teunis, Martin Florian, and Björn Scheuermann. Eclipsing Ethereum peers with false friends. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 300–309. IEEE, 2019.

[37] Cornelius Ihle, Dennis Trautwein, Moritz Schubotz, Norman Meuschke, and Bela Gipp. Incentive mechanisms in peer-to-peer networks—a systematic literature review. *ACM Computing Surveys*, 55(14s):1–69, 2023.

[38] Maxim Jourenko, Kanta Kurazumi, Mario Larangeira, and Keisuke Tanaka. SoK: A taxonomy for Layer-2 scalability related protocols for cryptocurrencies. Cryptology ePrint Archive, Paper 2019/352, 2019.

[39] George Kadianakis, Mary Maller, Andrija Novakovic, and Suphanat Chunhapanya. Proof of validator: A simple anonymous credential scheme for ethereum's dht. https://ethresear.ch/t/proof-of-validator-a-simple-anonymous-credential-scheme-for-ethereums-dht/16454, August 2023.

[40] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *27th USENIX Security Symposium*, 2018.

[41] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Intl. conf. on the theory and app. of cryptology and info. sec.*, ASIACRYPT, 2010.

[42] Csaba Kiraly, Leonardo Bautista-Gomez, and Dmitriy Ryajov. FullDAS - towards massive scalability with 32mb blocks and beyond https://ethresear.ch/t/fulldas-towards-massive-scalability-with-32mb-blocks-and-beyond/19529/1, May 2024.

[43] Michał Król, Onur Ascigil, Sergi Rene, Alberto Sonnino, Matthieu Pigaglio, Ramin Sadre, Felix Lange, and Etienne Riviere. Disc-NG: Robust service discovery in the Ethereum Global Network. In *9th European Symposium on Security and Privacy*, EuroS&P, pages 193–215. IEEE, 2024.

[44] Ankit Kumar, Max von Hippel, Panagiotis Manolios, and Cristina Nita-Rotaru. Formal model-driven analysis of resilience of gossipsub to attacks from misbehaving peers. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 2142–2160. IEEE, 2024.

[45] Probe Lab. Final report: NAT hole punching measurement campaign. https://github.com/plprobelab/network-measurements/blob/master/results/rfm15-nat-hole-punching.md, 2023.

[46] Labrys.io. MEV Watch. https://www.mevwatch.info, 2024.

[47] Yifan Mao, Soubhik Deb, Shaileshh Bojja Venkatakrishnan, Sreeram Kannan, and Kannan Srinivasan. Perigee: Efficient peer-to-peer network design for blockchains. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 428–437, 2020.

[48] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. Low-resource eclipse attacks on Ethereum's peer-to-peer network. *IACR Cryptology ePrint Archive*, 2018(236), 2018.

[49] Matter Labs. zkSync. https://zksync.io, 2024.

[50] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *International Workshop on Peer-to-Peer Systems*, IPTPS. Springer, 2002.

[51] Alberto Montresor and Márk Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer*, P2P, 2009.

[52] Kamilla Nazirkhanova, Joachim Neu, and David Tse. Information dispersal with provable retrievability for rollups. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, pages 180–197, 2022.

[53] Optimism. https://www.optimism.io, 2024.

[54] Ulysse Pavloff, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni. Byzantine attacks exploiting penalties in ethereum pos. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 53–65. IEEE, 2024.

[55] Polygon. https://polygon.technology, 2024.

[56] David Powell. Failure mode assumptions and assumption coverage. In *Predictably dependable computing systems*, pages 123–140. Springer, 1995.

[57] Gabriel Antonio F Rebello, Gustavo F Camilo, Lucas Airam C de Souza, Maria Potop-Butucaru, Marcelo Dias de Amorim, Miguel Elias M Campista, and Luís Henrique MK Costa. A survey on blockchain scalability: From hardware to layer-two protocols. *IEEE Communications Surveys & Tutorials*, 2024.

[58] Codex.storage Research. Crawling the ethereum discv5 network, fast. https://ethresear.ch/t/crawling-the-ethereum-discv5-network-fast/20962, 2024.

[59] Danny Ryan. PeerDAS – a simpler DAS approach using battle-tested P2P components. https://ethresear.ch/t/peerdas-a-simpler-das-approach-using-battle-tested-p2p-components/16541, 2023.

[60] Peiyao Sheng, Bowen Xue, Sreeram Kannan, and Pramod Viswanath. ACeD: Scalable data availability oracle. In *Financial Cryptography and Data Security*, FC. Springer, 2021.

[61] Srivatsan Sridhar, Onur Ascigil, Navin Keizer, François Genon, Sébastien Pierre, Yiannis Psaras, Etienne Rivière, and Michał Król. Content censorship in the interplanetary file system. In *Network and Distributed System Security Symposium*, NDSS, 2024.

[62] LibP2P Team. gossipsub v1.0: An extensible baseline pubsub protocol. https://github.com/libp2p/specs/blob/master/pubsub/gossipsub/gossipsub-v1.0.md.

[63] Dennis Trautwein. Nebula: A network agnostic dht crawler and monitor. https://github.com/dennis-tra/nebula, 2025.

[64] Dimitris Vyzovitis, Yusef Napora, Dirk McCormick, David Dias, and Yiannis Psaras. Gossipsub: Attack-resilient message propagation in the Filecoin and ETH2.0 networks. *arXiv preprint arXiv:2007.02754*, 2020.

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

[66] Karl Wüst and Arthur Gervais. Ethereum eclipse attacks. Technical report, ETH Zurich, 2016.

[67] Yunqi Zhang and Shaileshh Bojja Venkatakrishnan. Honeybee: Decentralized peer sampling with verifiable random walks for blockchain data sharding. *arXiv e-prints*, pages arXiv–2402, 2024.

[68] Mingxun Zhou, Liyi Zeng, Yilin Han, Peilun Li, Fan Long, Dong Zhou, Ivan Beschastnikh, and Ming Wu. Mercury: Fast transaction broadcast in high performance blockchain systems. In *IEEE Conference on Computer Communications*, INFOCOM, 2023.