IEEE ICBC'24 1570981762

Scalability limitations of Kademlia DHTs when enabling Data Availability Sampling in Ethereum

***Abstract*—Scalability in blockchain remains a great challenge, especially when prioritizing decentralization and security. The Ethereum community has proposed comprehensive data-sharding techniques to overcome storage, computational, and network processing limitations. In this context, the propagation and availability of large blocks become the subject of research to achieve scalable data-sharding. This paper provides insights after exploring the usage of a Kademlia-based DHT to enable Data Availability Sampling (DAS) in Ethereum. It presents a DAS-DHT simulator to study this problem and validates the results of the simulator with experiments in real DHT networks. Our results help us understand what parts of DAS can be achieved based on DHT solutions and which ones cannot. We discuss the limitations of DHT solutions and discuss other alternatives.**

***Index Terms*—Ethereum, Blockchain Scalability, Data Availability Sampling, Kademlia DHT, DankSharding.**

I. Introduction

Scalability has proven to be one of the major challenges in blockchain technology, at least when trying to keep it decentralized and secure [1]. The protocol proposed by Ethereum [2] is not different from the rest in that respect [3]. To tackle those scalability limitations, Ethereum plans to rely on the so-called Layer 2s (L2) or roll-ups [4] [5]. These protocols, which include mature projects such as Arbitrum [6], Optimism [7], or zkSync [8], benefit from the speed of processing transactions off-chain. However, they still rely on the consensus and security of layer 1s to ensure that the interactions are traceable and immutable. L2s optimize their operation costs by aggregating multiple transactions into smaller ones, providing a set of proofs or commitments that can be used to verify the correctness of the operations.

However, L2s need space to store their data. Ethereum researchers have proposed data-sharding [9] to facilitate the space needed by roll-ups. The idea is to offer larger and cheaper block space (blobs) to fill up with L2 data. The nature of L2s generally relies on fault or verifiable proofs as a mechanism to validate transactions. But those proofs are not needed in the long term. Therefore, blob data can be considered ephemeral and discarded after a certain time. This protocol proposal aims to scale the block size from the current average of $100KB$ to $32MB$, allowing L2 protocols to submit orders of magnitude more data in those new blob spaces than today.

The proposal relies on Erasure Coding [10] and Data Availability Sampling (DAS) [11] techniques to split the block into segments/chunks/samples to reduce the bandwidth, storage and computational overhead that processing the entire block implies for nodes and validators. Splitting and sharing the block in smaller pieces makes the propagation of a block easier through the network, as the validators will only have to subscribe, receive, and process a smaller portion of the entire block. Ordinary (non-validator) nodes instead rely entirely on probabilistic sampling, receiving only a few segments of the block in order to verify that the block is reconstructable using the erasure code with very high probability.

This paper presents the results of simulating a Kademlia-based DHT to support DAS on Ethereum's data-sharding, exploring the time restrictions of the common DHT operations by simulating the expected DankSharding workload. Furthermore, the paper presents a configurable DHT simulator that can reproduce connection errors and network latencies and recreate known network conditions. We raise concerns about the scalability limitations of DHTs like IPFS's when they suffer a high throughput of store and retrieval operations. Especially when the protocol wants to keep the hardware resources close to commodity home hardware as Ethereum does [12] to promote decentralisation by allowing home node operators. Considering the outcome of this research, we propose a set of alternatives that could potentially overcome the listed limitations, ensuring low overhead for the nodes while still providing the expected requirements for blockchain scalability.

The remainder of this paper is organized as follows. Section II discusses related work done on high throughput DHTs and previous DAS attempts in the ecosystem. Section III introduces the methodology followed to simulate the DHT behaviour under Ethreum's DAS proposal. In Section IV, we introduce and analyze the results obtained by our study, comparing our simulated results with live DHT networks. Finally, Section V concludes the findings of this work and presents some future directions.

II. Related work

The overhead of scaling up the processing capabilities of modern blockchain can only be handled by splitting the workload across the participating actors. In that context of task division, preventing data withholding attacks and providing means to verify that the data had been released probabilistically is what Data Availability Sampling aims to solve. However, DAS is not a new concept in the blockchain ecosystem. Some prior works have attempted to bring DAS [13] proofs on-chain through the usage of Oracles [14]. However, the idea relies on a third trusted blockchain to submit the data availability proofs. Thus, the idea cannot be directly applied to a network

such as Ethereum, as it is one of the networks that generates trust for others.

Other projects explored different solutions for DAS. Celestia [15] proposed a network with the single purpose of ordering and guaranteeing data availability [16], splitting the network into three main actors: consensus nodes, storage nodes and light clients. Despite achieving certain scalability results, the proposal only ensures that light clients don't need all the storage to validate chain transactions. This heavily relies on a low and limited number of "super nodes" that propose and download large blocks while serving all the segments to light clients. Due to their heavy hardware requirements, the approach exposes the network to a critical failure if storage nodes are overwhelmed by the workload.

DHTs have been part of the peer-to-peer and web3 ecosystem for a while due to their censorship resistance capabilities, workload balancing properties, and logarithmically scaling performance. Many projects like IPFS [17] [18], Libp2p [19] or even Ethereum [20] have relied on the Kademlia DHT implementation [21] for content routing or peer discovery. Some works like [22] have shown that despite needing some replication factor to overcome node churn in the network, the combination of erasure coding techniques and DHTs can reduce the bandwidth for the retrievability of the items. Since both techniques are robust and well-tested in the literature, one might expect that DHTs are a suitable solution to enable DAS on Ethereum. However, whether DHTs can guarantee the high throughput that DAS requires is uncertain.

There have been previous attempts to improve the latency and increase the throughput of DHTs. Previous work like [23] presented viable solutions that help increase the DHTs' throughput while providing congestion control and minimizing the latency between nodes. However, the authors do not contemplate the extra overhead that connecting and publishing into a larger network like the Ethereum one could have, extracting the available throughput from simulations with under 500 nodes.

This paper explores whether a modern Kademlia-based DHT can enable high-throughput native DAS techniques in a blockchain network. It focuses on a DHT schema that would help preserve the decentralized nature of Ethereum while preserving its resilience.

III. Methodology

The challenge of increasing the block size from an average of $100KB$ to $32MB$ is not simple. The upgrade to allocate more blobs in a cheaper ephemeral block space has many configurations and parameters that must be adjusted to make it "feasible". Figure 1 represents the proposed Execution Layer block's division for DAS, where the block will be first split into 256 rows and columns and then equally extended in both directions (rows and columns) applying Reed-Solomon Encoding [10] to ensure each row and column (and therefore the entire block) can be reconstructed as long as half of extended row or column samples are retrievable.

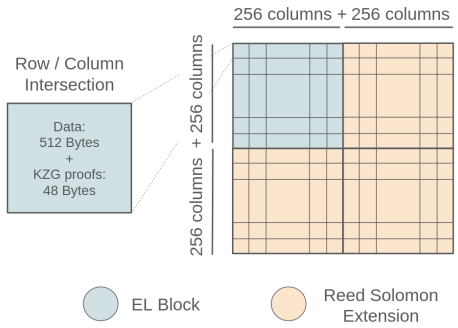

Fig. 1. Executions Layer's block split proposal for DAS.

With the current count of over 880.000 active validators distributed around 13.000 active Beacon Nodes, DAS aims to disseminate the block by entire rows and columns through the usage of distinct GossipSub topics [24]. It has been determined that each active validator will have to attest to a set of two random rows and columns in a single slot in an epoch [25]. This will ensure optimal bandwidth-efficient block propagation. However, since part of the network's privacy relies on not knowing which node in the network hosts which validator (preserving the anonymity of the validators), it complicates checking which data is available without losing that privacy.

The straight usage of a Kademlia DHT for content addressing, proven already on many live networks such as IPFS, makes it a suitable candidate to provide available samples to any interested node. Because the size of each sample, as introduced in Figure 1, is relatively small (512 bytes of data plus 48 bytes of KZG proofs [26]), we propose using a Kademlia-based DHT to evenly distribute the samples based on the Beacon Node's ID and the segment's distance.

A. Kademlia DHT

Since the parameters of the possible DHT are not defined (at the moment of writing this paper), the paper explores a possible DHT implementation for DAS through a simulator. The DAS simulator relies on a Python module that implements all the components of a Kademlia DHT. The module allows the creation of a network of the given size, initializing the routing table of the nodes while using the XOR binary distance between nodeIDs to fill up the K-buckets. Inside the module, the two main components, the DHT network and the DHT nodes (also known as DHT clients), have different purposes.

The network module offers a common shared perspective for the simulation, as shown in Figure 2. It keeps track of all the available nodes and is the one that simulates the communication between two nodes by introducing latencies and eventual losses. Because peer-to-peer networks tend to be subject to node congestion, node churn, or even connection churn [27] [28], the network configuration allows to define the following set of parameters that helps reproduce any network behaviour:

- Number of nodes participating in the DHT network.

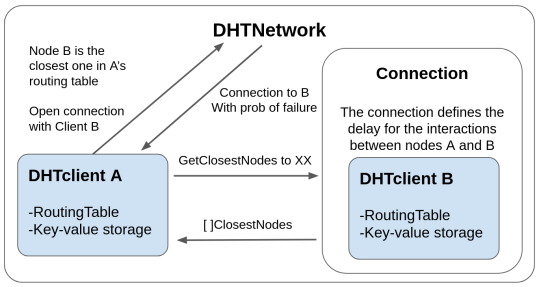

Fig. 2. Description of the DHT components in the *dht* simulator, and an example of the internal logic of their interaction.

- Fast error rate: ratio of connections that the remote peer will refuse.
- Slow error rate: ratio of the connections simulating a timeout when connecting a remote peer.
- Connection delay range: latencies that will be applied to the communication between two nodes.
- Fast delay range: latencies that will be applied when a *fast error* occurs.
- Slow delay range: latencies that will be applied when a *slow error* occurs.
- Gamma: the incremental delay applied to the communication between nodes when a peer is contacted by others, simulating the overhead of handling multiple connections.

On the other hand, the DHT clients are the ones that populate the simulated network. They include all the logic of the DHT, allowing themselves to swap information if the connection through the network interface succeeds. The simulator recreates the performance of a DHT, performing events such as initialization of the routing table, updating the routing table, looking up the closest peers to a key, providing a given value to the network, lookup for a specific key. The DHT clients have the configuration of the Kademlia DHT, including the following parameters:

- K: replication factor for the DHT and k-buckets' size.
- Alpha: maximum number of concurrent connections a DHT client can have while looking for a key/value or the closest clients to a key.
- Beta: number of close clients a remote peer provides when looking for a key.

We refer the reader to [21] for the exact definition of these parameters.

The *dht* [reference removed for double-blind review] module extracts all the timing and accuracy metrics from each individual operation, allowing the inspection of the network's performance under certain circumstances.

B. Verification of the DHT simulations

To validate the simulated results of the *dht* module, the paper includes a comparison with the IPFS live network's performance. The results were validated by comparing the performance of the most basic operations in a DHT, such as DHT provides and DHT lookups in IPFS, with simulations

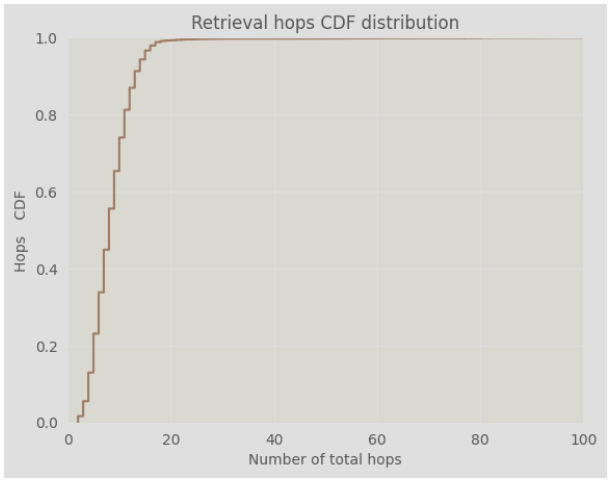

Fig. 3. CDF of the DHT hops while performing 100 concurrent retrievals in the IPFS network.

replicating the same DHT and network parameters. To obtain those performance measurements from the IPFS network, the team developed a dedicated tool [reference removed for double-blind review] that generates, provides, and requests a given number of Content Identifiers (CIDs) in parallel. This enables tracking the individual performance of each independent operation and the overhead of performing multiple low-demanding operations from a single node.

IV. Evaluation

A. DHT lookups

One of the key operations in a DHT is its ability to find which node in the network has the value for a specific key. This operation is generally known as *DHT lookup*, and it consists of a set of recursive operations that, using the XOR distance between the node IDs and the given key, discover closer and closer nodes until, eventually, a node that has the value is found (assuming some node has it). DAS needs fast data retrievability from the DHT to ensure that blobs can be verified "fast enough"[1]. For this reason, measuring DHT's performance is critical to check if it would be a viable option for DAS in Ethereum. A peculiarity of DAS verification is that a node is starting concurrent lookup queries for a number of randomly selected samples. The DAS verification is considered complete when every single selected sample is retrieved. This section describes the main findings observed when simulating such concurrent queries on a DHT. We use 80 queries from a node, which is assumed to provide high enough probabilistic guarantees on a 512-by-512 segment 2D encoded block.

We also run similar concurrent queries on the live IPFS network. This serves as a validation for the correctness of the simulator, where the number of hops and the aggregated delay for the operation match a live network if the parameters are correctly set.

[1]There is an ongoing debate in the Ethereum community about how fast this verification should be. It is commonly argued that one slot time (12 seconds) is fast enough

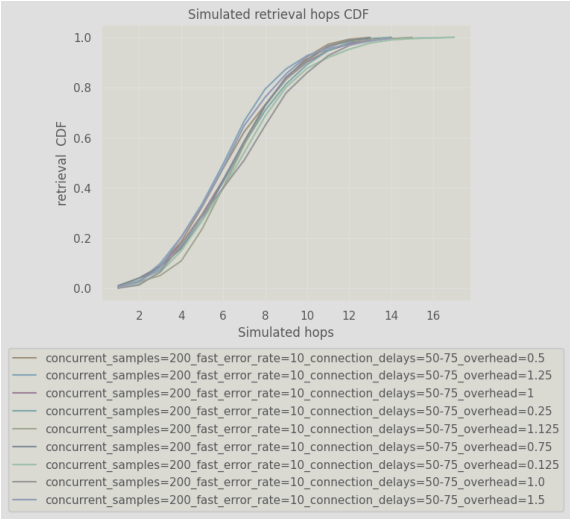

Fig. 4. CDF of the simulated number of hops while looking for 200 concurrent keys with different overheads.

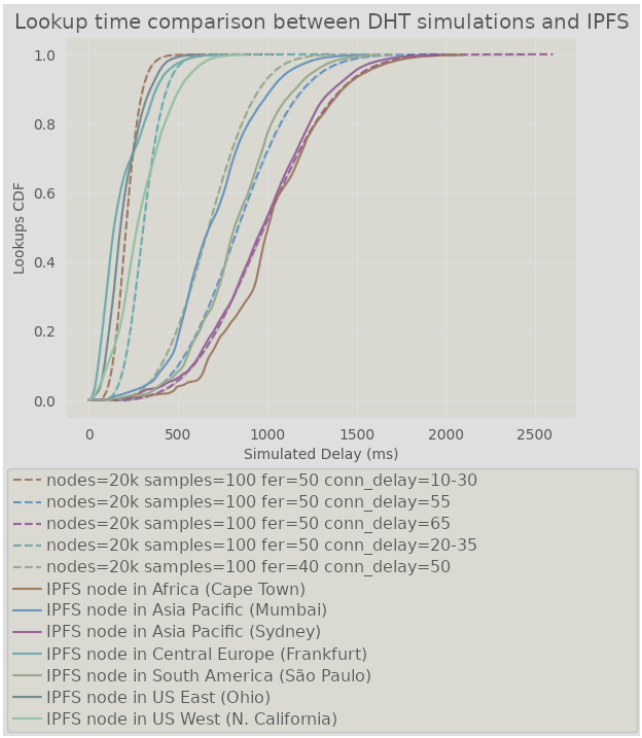

Fig. 5. Accuracy of the DHT simulator model matching lookup performances in the IPFS network from different locations.

a) Lookup hops: A hop is an individual step in the recursive lookup operation. It implies asking a remote node in the network for a number ($beta$) of the closest peers it has to the given key or the value of the key if it has it. This is how the DHT narrows down the lookup of a value. Figure 3 shows the CDF of the number of hops done by an IPFS DHT node when performing 100 sets of 80 concurrent DHT lookups. The figure shows how the 99% of lookups were done in under 18 hops, with a small tail in the last 1% that can reach the 100 hops on some rare occasions. The numbers measured with the DHT simulator report similar results, as Figure 4 shows. With a fast-failure rate of 10% of the connections, the simulation of 100 sets of 200 concurrent lookups report a 99th percentile of 12 to 14 hops. We also notice that the concurrency overhead parameter does not impact the number of hops, as we see a negligible difference between the distributions for different overhead values. This makes sense as the time it takes a node to answer does not change the content of its routing table, and, therefore, the number of hops is not impacted.

b) Time performance: Nodes join and leave the network as they please in permission-less networks such as IPFS and Ethereum. However, the users in the Ethereum network have shown more stability in maintaining their nodes for long periods. Users are generally staking or extracting on-chain data from nodes, so they are somehow incentivised to keep the nodes up and running. As a result, this stability can create healthier routing tables and potentially increase the DHT lookup time performance. In an attempt to validate the impact of the network conditions on the DHT lookup performance, Figure 5 compiles the lookup time's CDF in the IPFS network done by the Probelab Team [29], with the respective estimated network conditions in each of the regions. It is not a surprise that different regions have different connectivity. The difference on the 90th percentile of the regions varies from 700 milliseconds to retrieve a value in the US and Europe to 1.7 seconds and 1.5 milliseconds in South America and Africa. Moreover, this connectivity difference can magnify in p2p networks where nodes cluster more in certain regions. The figure also showcases the possibility of configuring the *dht* module to fit any realistic network conditions. The figure shows that with the variation of the delay range at the simulated network, the results can accurately fit the measurements done in the IPFS DHT network from the lookup node's perspective.

c) Concurrency overhead: We've mentioned that there are differences between the IPFS and the Ethereum network. The nature of IPFS, where 80% of users altruistically spawn a node and disconnect it after 8 hours [30], makes it more subjected to node churn [28]. This generally impacts the performance, as these nodes might be present in the routing table of other nodes for several minutes. Despite the expected node churn in Ethereum is expected to be lower, having non-active nodes inside the nodes' routing table doesn't only affect the direct time performance of the DHT operations. The main solution to overcome a certain connection failure rate is to attempt multiple simultaneous ones, with the extra cost of handling those extra concurrent requests.

The presented $alpha$ parameter helps overcome the time impact of facing unreachable nodes, which in IPFS' is set to 3 concurrent connections. However, it multiplies the cost of making a single lookup. Figure 6 shows the time impact of different concurrency overhead parameters on 200 concurrent DHT lookups from a single retrieving node's perspective. The overhead is applied to nodes contacted during the concurrent

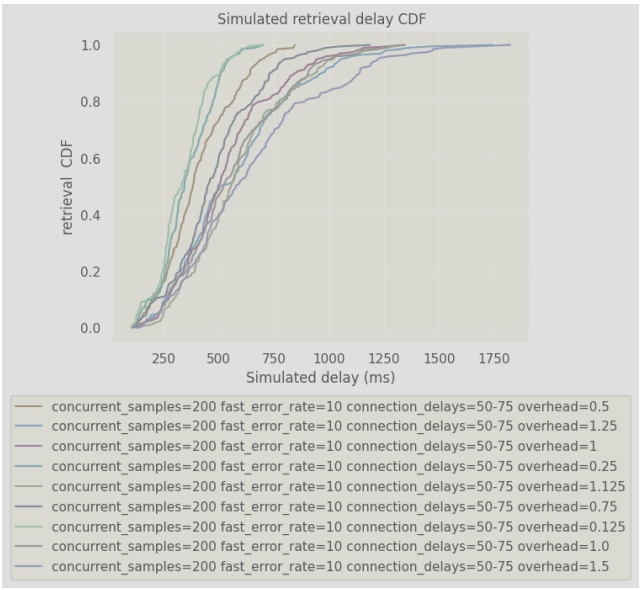

Fig. 6. CDF of simulated 100 retrieval sets of 200 concurrent keys with different overheads.

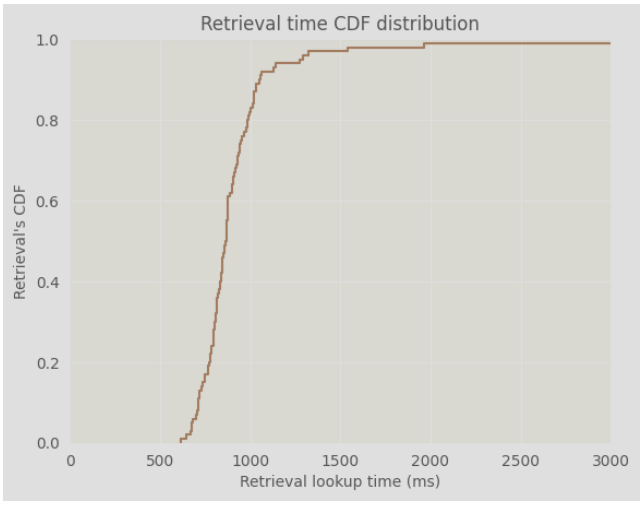

Fig. 7. CDF of the time to finish 100 sets of 80 concurrent retrievals from the IPFS network.

simulated operations. This means that each node connected during the lookup operation will apply an only increasing overhead delay to its connection delay as other concurrent operations contact the same peer. If a set 200 connection needs to contact peer A on three occasions, the first won't have any overhead delay, the second one will have a penalty of the overhead parameter, and the third one will have a penalty of two times the parameter.

Users with access to larger hardware resources for the Ethereum node could be less restricted by the concurrency overhead of retrieving multiple block segments in sub-second lookups. In other occasions where the hardware resources are more limited, like in a solo-staker scenario, the DAS operations will be more penalized for handling so many interactions. This exposes a clear disadvantage for smaller users if the number of lookups is too high in the long-term projection of the protocol.

To compare the existing overhead in the IFPS DHT client, Figure 7 exposes the CDF time of concluding 80 concurrent lookups in the IPFS network in a set of 100 rounds. The figure shows that 90% of concurrent lookups performed between 600ms and 1.2 seconds. Compared to the numbers presented in Figure 5, we can clearly see that IPFS's client does see an impact in performance originating from a concurrency overhead, not at least at this number of concurrent requests. Therefore, it is also an expected effect in a possible DHT for Ethereum's DAS.

B. DHT limitations

The results of the experiments are clear: modern Kademlia-based DHTs can achieve sub-second sampling performance on its 90th percentile with the favourable conditions that the Ethereum network provides. Even if looking at less favourable conditions and at higher percentiles, we see latencies well below the slot time of 12 seconds.

However, despite the logarithmic scaling solution for content addressing, the idea of the DHT seems to be reaching its limits in highly demanding conditions. While the sampling itself seems to work, the first step of a DHT solution is seeding the block data into the DHT, where the samples are dispersed around the network according to their position in the DHT address space.

Ethereum's current network conditions report roughly 13.000 active nodes. Combined with the suggested standard Kademlia parameters [31] of $k = 20$ and $alpha = 3$, the seeding of 262.144 block segments (512 columns * 512 rows) would produce a ratio of 403.29 block segments that each node in the network would have to store, on average, every 12 seconds. Moreover, storing the block samples doesn't only affect the last contacted nodes. Providing the keys is a process that starts from a distinct lookup to find the closest k nodes to the segment's key, performing a median of 8 to 10 hops until the closest nodes are notified. If handled simply as a large number of individual storage requests, the overall network load can easily overload the network, considering that all this should happen fast before the actual sampling can begin.

When considering the network load of seeding the DHT, it is important to remember that block propagation is also done to validators using GossipSub. However, in that case, instead of sending individual samples to selected target nodes, the efficient mesh-based GossipSub protocol is used to send entire rows and columns. As mentioned before, serving sampling requests directly from validators would compromise the identity of individual validator nodes. However, it raises the question: who could seed the DHT, and how could it be done instead of many individual storage requests?

C. Seeding the DHT

The DHT could be seeded by the block proposer/builder, but also by validators (limited to the scope of their GossipSub subscription). However, we can state that both cases are problematic for several reasons explained in the following paragraphs:

a) Single seeder to the DHT: If the block proposer/builder is in charge of the DHT seeding, the nodes in its routing table will suffer the biggest workload of replying to the first hops of the 262.144 requests, creating a huge workload on the seeding and routing nodes. The tedious problem of having to perform many provide operations to the DHT network is already present in the IPFS DHT for large service providers like Cloudflare [32] or Pinata [33]. As a result, organizations and developers have moved the biggest part of the content in IPFS away from the DHT, relying on several solutions:

- BitSwap [34], a protocol initially designed to download content from a connected network participant optimally. Using BitSwap connections as a solution avoids seeding the "provider records" for each hosted item. However, it relies on the nature of the network to end up connected to the largest DHT nodes, which centralizes the network and doesn't provide any content-addressing solution. It relies on the probability of eventually being connected to one of these "big" nodes to retrieve the desired content, not guaranteeing minimal retrieval times for a non-popular item.
- Network Indexers [35]. It is a separate method that tackles the scalability problems of the DHT and the content addressing limitations of BitSwap. It offers a network of high throughput and low latency DHT-like databases present in the network as nodes that can be used by users supporting the protocols. As expected, this solution breaks the nature of distributed networks, as it relies on adding a particular actor whose only function is to provide the content addressing in the network. The solution, despite being scalable, magnifies the exposition of the network to a denial of service attack. Thus, it is not a viable option for Ethereum.
- The Accelerated Routing Table [36]. This experimental routing table enables the possibility of making batch provides or reprovide sweeps [37]. The method relies on the crawling capabilities of the DHT host to discover all the participants in the network. This allows it to, based on the whole pack of content it has to advertise to the network, dismiss every individual DHT lookup to discover the K closest peers because it already knows the entire list of participants. As a result, a node using an accelerated routing table can perform a bulk-provide operation per each node it needs to connect. The method saves many round-trip connections but exposes the network to the feared node churn. One could think about increasing the frequency of refreshing the routing table to overcome the node churn; however, it would also mean crawling the network more often, replacing the network's spam from DHT lookups and refreshing the accelerated routing table. Furthermore, the Ethereum network tends to have a low level of connections to limit the bandwidth usage [12], which makes it unviable to implement at the moment.
- Recursive (and forwarding) Kademlia. The cost of seeding a large number of small segments (key-value pairs) into a Kademlia DHT is expensive due to the large number of interactions needed from the iterative lookup process, the parallel lookups (alpha), and the number of target storage nodes (K). [38] introduced recursive Kademlia to reduce the iterative overhead, while [39], [40] extended this to support efficient broadcasting in the Kademlia address space. While DAS seeding is more complex than broadcasting, and recursive forwarding introduces compromises in the robustness of seeding the DHT, similar solutions can be considered to reduce the network load and time required.

b) Multiple seeders to the DHT: On the other hand, the seeding could be done more organically by peers subscribed to a small set of GossipSub topics. This would balance the workload of publishing many items to the DHT for individual routing tables of the seeding nodes. But even with more sophisticated solutions like Optimistic Provide [41], the network ends up over-flooded by the same message as more nodes perform the same provide operation. This solution could be improved by adding some logic to the consensus, where validators rely on deterministic randomness to define who must seed the DHT. This solution could follow a similar approach to the one that defines which validator aggregates and distributes the attestations at each slot committee. However, requires a heavy change in the current logic of the consensus and could expose to some degree the privacy of the validators.

c) Avoiding the DHT: Finally, we should mention that some early-stage proposals emerged while writing this paper that are trying to avoid using a DHT for sampling. PeerDAS [42] relies on a deterministic segment-to-nodeID mapping scheme to avoid expensive lookups. Sampling nodes are assumed to know enough node IDs to be able to perform their sampling based on the mapping. While a peer sampling service is assumed, the compromise of PeerDAS compared to the DHT-based solution is a reduced number of rows and columns. Another proposal, SubnetDAS [43] instead compromises on the randomness of the sampling, more specifically on the unlinkability of the queries. While in a DHT-based design every single node can query its selected random samples, in SubnetDAS nodes subscribe to GossipSub channels, and hold these subscriptions for a specified period, weakening the protection of nodes against double-spend.

V. Conclusion

This paper presents the results and limitations of implementing a direct Kademlia DHT as a DAS solution for Ethereum's scalability proposal. We present a DHT simulator that is validated using experiments in a production p2p network with a real Kademlia DHT implementation. Our results expose that, despite being the right fit for data sampling, it is problematic

to seed the DHT in the context of Ethereum DAS. We discuss several possible alternative ways to seed the DHT, as well as elements of other designs not relying on a DHT. As future work, we intend to explore the viability and performance of DHT solutions in the context of more flexible and less constrained DAS protocols.

REFERENCES

[1] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *Ieee Access*, vol. 8, pp. 16 440–16 455, 2020.

[2] "Ethereum whitepaper," https://ethereum.org/en/whitepaper/.

[3] A. Chauhan, O. P. Malviya, M. Verma, and T. S. Mor, "Blockchain and scalability," in *2018 IEEE international conference on software quality, reliability and security companion (QRS-C)*. IEEE, 2018, pp. 122–128.

[4] V. Buterin, "On-chain scaling to potentially 500 tx/sec through mass tx validation," https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477.

[5] L. T. Thibault, T. Sarry, and A. S. Hafid, "Blockchain scaling using rollups: A comprehensive survey," *IEEE Access*, 2022.

[6] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1353–1370.

[7] "Optimism," https://www.optimism.io/.

[8] "zksync," https://zksync.io/.

[9] "Danksharding," https://ethereum.org/en/roadmap/danksharding/.

[10] S. B. Wicker and V. K. Bhargava, *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[11] M. Hall-Andersen, M. Simkin, and B. Wagner, "Foundations of data availability sampling," *Cryptology ePrint Archive*, 2023.

[12] M. Cortes-Goicoechea, L. Franceschini, and L. Bautista-Gomez, "Resource analysis of ethereum 2.0 clients," in *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE, 2021, pp. 1–8.

[13] P. Sheng, B. Xue, S. Kannan, and P. Viswanath, "Aced: Scalable data availability oracle," in *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25*. Springer, 2021, pp. 299–318.

[14] H. Al-Breiki, M. H. U. Rehman, K. Salah, and D. Svetinovic, "Trustworthy blockchain oracles: review, comparison, and open research challenges," *IEEE access*, vol. 8, pp. 85 675–85 685, 2020.

[15] "Celestia," https://celestia.org/.

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

[17] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[18] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, and Y. Psaras, "Design and evaluation of ipfs: a storage layer for the decentralized web," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 739–752.

[19] "Libp2p," https://libp2p.io/.

[20] "Discv5," https://github.com/ethereum/devp2p/blob/master/discv5/discv5.md.

[21] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[22] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE transactions on information theory*, vol. 56, no. 9, pp. 4539–4551, 2010.

[23] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. T. Morris, "Designing a dht for low latency and high throughput." in *NSDI*, vol. 4, 2004, pp. 85–98.

[24] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, and Y. Psaras, "Gossipsub: Attack-resilient message propagation in the filecoin and eth2. 0 networks," *arXiv preprint arXiv:2007.02754*, 2020.

[25] M. Cortes-Goicoechea, T. Mohandas-Daryanani, J. L. Muñoz-Tapia, and L. Bautista-Gomez, "Autopsy of ethereum's post-merge reward system," *arXiv preprint arXiv:2303.09850*, 2023.

[26] "Kzg polynomial commitments," https://dankradfeist.de/ethereum/2020/06/16/kate-polynomial-commitments.html.

[27] S. Henningsen, S. Rust, M. Florian, and B. Scheuermann, "Crawling the ipfs network," in *2020 IFIP Networking Conference (Networking)*. IEEE, 2020, pp. 679–680.

[28] E. Daniel and F. Tschorsch, "Passively measuring ipfs churn and network size," in *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2022, pp. 60–65.

[29] "Probelab.io, ipfs lookups' performance," https://probelab.io/ipfsdht/#dht-lookup-performance-distribution-by-region.

[30] D. Trautwein, "Libp2p," https://github.com/plprobelab/network-measurements/blob/master/results/rfm2-uptime-and-churn.md.

[31] "Ipfs provider record's liveness," https://github.com/plprobelab/network-measurements/blob/master/results/rfm17-provider-record-liveness.md.

[32] "Cloudflare," https://www.cloudflare.com/.

[33] "Pinata," https://www.pinata.cloud/.

[34] A. De la Rocha, D. Dias, and Y. Psaras, "Accelerating content routing with bitswap: A multi-path file transfer protocol in ipfs and filecoin," *San Francisco, CA, USA*, p. 11, 2021.

[35] "Interplanetary network indexer," https://github.com/ipni.

[36] "Accelerated dht provide," https://github.com/ipfs/kubo/pull/8997/files.

[37] "Reprovide sweep," https://github.com/libp2p/go-libp2p-kad-dht/issues/824.

[38] B. Heep, "R/kademlia: Recursive and topology-aware overlay routing," in *2010 Australasian Telecommunication Networks and Applications Conference*. IEEE, 2010, pp. 102–107.

[39] Z. Czirkos and G. Hosszú, "Solution for the broadcasting in the kademlia peer-to-peer overlay," *Computer Networks*, vol. 57, no. 8, pp. 1853–1862, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128613000777

[40] E. Rohrer and F. Tschorsch, "Kadcast: A structured approach to broadcast in blockchain networks," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, ser. AFT '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 199–213. [Online]. Available: https://doi.org/10.1145/3318041.3355469

[41] "Optimistic provide docs," https://github.com/libp2p/go-libp2p-kad-dht/pull/783.

[42] "PeerDAS proposal," https://ethresear.ch/t/peerdas-a-simpler-das-approach-using-battle-tested-p2p-components/16541.

[43] "SubnetDAS proposal," https://ethresear.ch/t/subnetdas-an-intermediate-das-approach/17169.