Brief Announcement: DAGs for the Masses

Michael Anoprenko
Institut Polytechnique de Paris, France

Andrei Tonkikh
Aptos Labs, New York, NY, USA

Alexander Spiegelman
Aptos Labs, New York, NY, USA

Petr Kuznetsov
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Abstract

A recent approach to building consensus protocols on top of Directed Acyclic Graphs (DAGs) shows much promise due to its simplicity and stable throughput. However, as each node in the DAG typically includes a *linear* number of references to the nodes in the previous round, prior DAG protocols only scale up to a certain point when the overhead of maintaining the graph becomes the bottleneck.

To enable large-scale deployments of DAG-based protocols, we propose a *sparse DAG* architecture, where each node includes only a *constant* number of references to random nodes in the previous round. We present a *sparse* version of Bullshark – one of the most prominent DAG-based consensus protocols – and demonstrate its improved scalability.

Remarkably, unlike other protocols that use random sampling to reduce communication complexity, we manage to avoid sacrificing resilience: the protocol can tolerate up to $f < n/3$ Byzantine faults (where n is the number of participants), same as its less scalable deterministic counterpart. The proposed "sparse" methodology can be applied to any protocol that maintains disseminated system updates and causal relations between them in a graph-like structure. Our simulations show that the considerable reduction of transmitted metadata in sparse DAGs results in more efficient network utilization and better scalability.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Consensus, Atomic Broadcast, Byzantine Fault Tolerance, DAGs, Scalability, Sampling

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.45

Related Version *Full Version*: `https://arxiv.org/abs/2506.13998` [2]

Funding *Petr Kuznetsov*: Supported by CHIST-ERA grant CHIST-ERA-22-SPiDDS-05 (REDONDA project) and Agence Nationale de la Recherche (ANR-23-CHR4-0009).

1 Introduction

Byzantine consensus is one of the central tasks in distributed computing and blockchains. It combines disseminating data blocks among the users and making sure that the blocks are placed by all users in the same order. A popular trend of DAG-based consensus ([14, 16, 15, 3, 4], to name a few) is to separate data dissemination from the ordering logic. In these protocols, all users disseminate blocks, and each block references a number of *parents* – previously disseminated blocks – thus forming a directed acyclic graph (DAG). The key observation is that by locally interpreting snapshots of this ever-growing graph, the users can infer a consistent ordering of the blocks. Compared to more traditional, leader-based solutions such as PBFT [8], this approach is simpler, exhibits a more stable throughput and scales to a larger number of participants.

Table 1 Communication complexity of Bullshark and Sparse Bullshark for different cryptographic setups and the estimated amount of egress traffic per-user per-round in the case when $n = 2\,000$ and $\lambda = 128$.

	Bullshark		Sparse Bullshark	
	complexity	ex. egress traffic	complexity	ex. egress traffic
Threshold signatures	$\Theta(n^2\lambda)$	171 MB	$\Theta(n^2 + n\lambda^2)$	17 MB
Multi-signatures	$\Theta(n^3)$	837 MB	$\Theta(n^2\lambda)$	81 MB
Regular signatures	$\Theta(n^3\lambda)$	171 GB	$\Theta(n^2\lambda^2)$	16 GB

The original DAG-based BAB protocols are, however, very heavy on the communication channels, which inherently limits their scalability. In this paper, we take a *probabilistic* approach to build a slimmer DAG structure, where each block carries *quasi-constant* amount of data about its predecessors, which results in a much less bandwidth-demanding solution. We show that the reduced complexity comes with the cost of a very small probability of consistency violation and a slight latency increase.

DAG consensus

DAG-based consensus protocols proceed in (asynchronous) rounds. In each round, each *user* (validator) involved in maintaining the DAG creates a node (typically containing a block of transactions submitted by the blockchain users) and disseminates it using reliable broadcast [6, 7]. The node contains references to $n - f$ nodes from the previous round, where n is the total number of validators and f is the maximum number of faulty validators (typically, $f = \lfloor \frac{n-1}{3} \rfloor$). In every round, a dedicated node is chosen as an *anchor*, where the choice can be either made deterministically based on the round number or at random using a common coin. The goal of consensus is now to place the anchors in an ordered sequence. In creating the sequence, the DAG nodes referencing the anchor as their parent are interpreted as its *votes*. Once the sequence of anchors is built, all DAG nodes in an anchor's causal past are placed before it in some deterministic way.

The burden of metadata

The central claim for DAG-based protocols is that they are able to amortize the space taken by the graph edges (*metadata*) provided enough user transactions (*payload*): If each block includes $\Omega(n)$ user transactions, the metadata will take at most a constant fraction of the block space. However, while alluring in theory, this claim does not stand the test of reality where we cannot simply keep increasing the block size indefinitely due to limited resources (network, storage, and CPU) and a limited demand.

As each user needs to receive, process, and store each other user's block in each round, and each block includes $n - f$ references, per-user per-round communication, computational, and storage complexity is at least $\Omega(n^2)$ multiplied by the size of a single reference. In practice, in most protocols [14, 10, 16, 15, 3], each reference includes either $n - f$ signatures, a multi-signature with a bit-vector of size n, or a threshold signature [11, 5], which brings the per-user per-round complexity to $\Omega(n^3\lambda)$, $\Omega(n^3)$, or $\Omega(n^2\lambda)$ respectively, depending on the cryptographic primitives available, where λ is the security parameter. [1]

[1] Intuitively, one can think of λ as the number of "bits of security", i.e., as the logarithm of the number

This issue is a significant barrier not only for effective resource utilization, but also for the scalability of the system: modern production blockchain deployments operate with around 100 validators and poorly scale beyond that.

Our contribution

We introduce a simple but surprisingly efficient technique for reducing the number of edges in the DAG by randomly sampling a constant ($D = O(\lambda)$) number of parents for each DAG node from a quorum of nodes in the previous round. The communication complexity is thus brought down to $O(n^2\lambda^2)$ with plain signatures, $O(n^2\lambda)$ with multi-signatures, or $O(n^2 + n\lambda^2)$ with threshold signatures. We provide a detailed break-down of the communication complexity in Section 3.2. This significantly lowers the overhead of DAG-based protocols, enabling larger-scale deployments. The communication complexity comparison is presented in Table 1.

Notably, unlike prior solutions to scaling distributed systems with random sampling ([13, 9, 1], to name a few), our approach maintains the optimal resilience and tolerates up to $f < n/3$ Byzantine faults.

We present *Sparse Bullshark*, a variant of Bullshark [16] – a state-of-the-art DAG-based consensus protocol – and demonstrate with simulations that it achieves better scalability.

2 Protocol

2.1 System Model

We use the standard BFT system model. For simplicity, we assume $n = 3f + 1$ participants $\Pi = \{p_1, \ldots, p_n\}$ (or *validators*), up to f of which might be *Byzantine* faulty. We assume partial synchrony with a known upper bound Δ on the network delays after GST and a (polynomially) computationally bound adaptive adversary.

We implement the *Byzantine Atomic Broadcast* [14] (BAB) abstraction satisfying the standard properties: **Validity**, **Agreement**, **Integrity**, **Total order**.

2.2 Sparse Bullshark

Our protocol is based on the partially synchronous version of Bullshark [16, 17].

Pseudocodes of both Bullshark and Sparse Bullshark with highlighted modifications can be found in the full version of this paper[2].

2.2.1 Verifiable sampling

The key distinguishing feature of Sparse Bullshark is that a DAG node stores just a constant-size *sample* of nodes from the previous round instead of a full quorum. We denote the sample size by D. Intuitively, when a correct validator collects $n - f$ round-r nodes, it selects, uniformly at random, a subset of size D of them and creates a new node in the round $r + 1$ referencing the selected subset. The specific choice of D affects communication complexity, latency, and the security of the protocol. More specifically, the protocol is secure against computationally bounded adversaries as long as D is $\Omega(\lambda)$. We discuss the impact of the sample size on latency in detail in Section 3.1.

of operations required to break the cryptosystem. Most hash functions and digital signature schemes have output of size $O(\lambda)$. However, multi-signatures require additional $O(n)$ bits to store the set of nodes who contributed to the signature. In this paper, we assume $n \gg \lambda$.

Instead of taking a random sample, Byzantine validators might attempt to undermine safety and liveness of the protocol by selecting a set of D nodes from the previous round maliciously. To anticipate this behavior, we employ *verifiable sampling*: Each validator must provide a *proof* that it has indeed collected a quorum of $n - f$ nodes in a round and has fairly sampled D of them in a pseudo-random manner.

To enable such proofs, whenever validator p_i broadcasts a round-r block, it attaches $Sign_i(r)$, a signature on the current round number. To prove that a validator indeed has seen a quorum of $n - f$ nodes of round r, it provides an aggregated *multi-signature* on the round number r and a bitset encoding the set of validators it received the round-r blocks from. The validator then uses the multi-signature on the round number as the seed for a pseudo-random number generator, which in turn is used to choose D parents from the collected quorum. To verify the sample, any other validator simply needs to check the multi-signature and replay the pseudo-random sampling procedure.

This approach, while restricting the behavior of Byzantine nodes to some degree, still leaves open the possibility of *grinding attacks*: a malicious validator that receives more than $n - f$ round-r blocks may try many different subsets of size $n - f$ to find a more "favorable" sample. However, as long as $D = \Omega(\lambda)$, this attack is, intuitively, as efficient as any other naive attack on the cryptographic primitives, such as simply trying to recover a correct node's private key by a brute force search. We address it formally in [2].

2.2.2 Referencing the anchor

Since the references to the anchor in round r by the nodes in round $r + 1$ are used as votes for the direct commit rule, on top of selecting D random parents, if a correct node observers the anchor of the previous round, it always includes it into its edges. Notice that trying to enforce this rule on Byzantine users would be unhelpful as they can always simply pretend to not have received the anchor node.

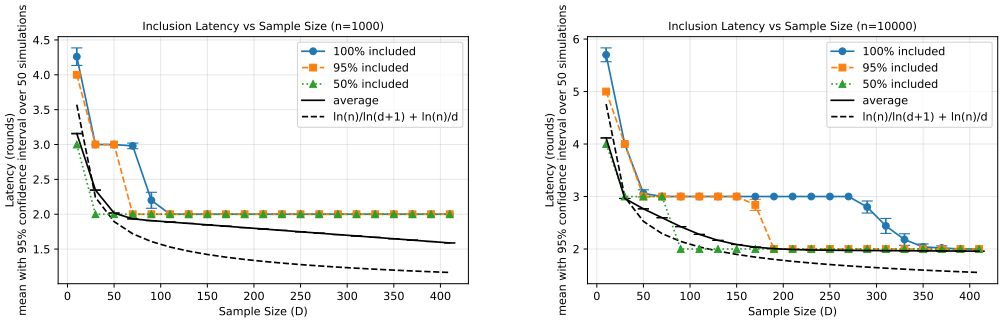
2.2.3 Direct commit rule

Finally, in Sparse Bullshark, we modify the direct commit rule, raising the threshold from $f + 1$ to $n - f = 2f + 1$. This does not affect the liveness or latency of the protocol in partial synchrony. Indeed, if the user producing the anchor node is correct and the network is in the stable period, all (at least $n - f$) correct users will reference the anchor node in round $r + 1$.

This modification is necessary in order to ensure that, with high probability, any committed anchor is reachable from any consequent anchor, and thus guarantee **Total order**.

3 Analysis

Correctness of Sparse Bullshark relies on the same key propery as Bullshark: if an anchor has been committed by an honest validator using the direct commit rule then there will be a path in DAG from any anchor of any future round to this anchor. This property is satisfied with high probability in Sparse Bullshark, implying the **Total order** property. We refer to the full version of the paper [2] for detailed correctness proofs.



(a) inclusion latency for $n = 1000$ nodes. **(b)** inclusion latency for $n = 10\,000$ nodes.

Figure 1 Inclusion latency (in DAG rounds) simulation, for varying sample size (D), assuming each node has D nodes from the previous round uniformly at random as its parents.

3.1 Latency

During periods of synchrony, all blocks created by correct validators in odd rounds will link to the anchor of the previous round because of the additional timer delay before creating a new vertex. Therefore, all anchor nodes by correct validators will be committed. The frequency of anchors being committed in Sparse Bullshark is the same as in the original Bullshark.

However, as the anchor node in Sparse Bullshark only references D nodes from the previous round instead of $n - f$, the protocol incurs additional *inclusion latency*: the delay between a vertex being broadcast and being included in some anchor's causal history. Assuming that each node has its parents selected at random from the previous round, the exact delay will follow the rumor-spreading pattern.

We provide a simulation of the inclusion delay (in terms of DAG rounds) in Figures 1a and 1b for 1000 and 10 000 users respectively. For example, selecting $D = 70$ for 1000 users or $D = 190$ for 10 000 users is enough to ensure that 95% of nodes will have inclusion latency of at most 2, i.e., will be referenced by some node referenced directly by the anchor. In contrast, Bullshark has inclusion latency of 1 for the $n - 1$ nodes referenced directly by the leader and 2 (assuming random message ordering, with high probability) for the rest of the nodes.

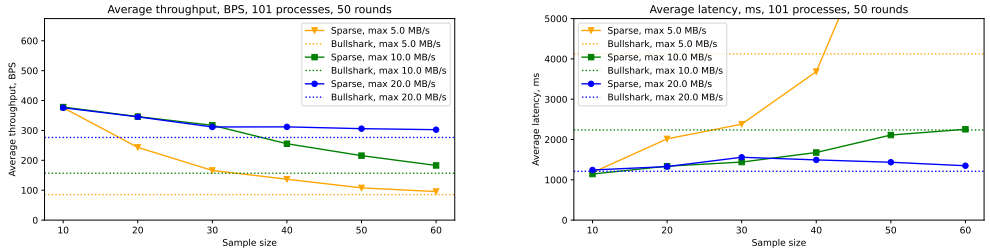
3.2 Communication complexity

We estimate the per-node metadata communication complexity. In Sparse Bullshark, each DAG node contains up to $D+2 = \Theta(\lambda)$ references to other nodes compared to $\Theta(n)$ references in Bullshark. Each reference is a certificate for the corresponding DAG node and its size is:

- $\Theta(\lambda)$ in case threshold signatures are available;
- $\Theta(n + \lambda) = \Theta(n)$ in case multi-signatures are available;
- $\Theta(n\lambda)$ otherwise.

We will denote the size of a certificate by C.

In Sparse Bullshark, the metadata of a node consists of $\Theta(\lambda)$ references and a bitmask of collected quorum (*sampleProof*), so metadata size for each node is $\Theta(n + \lambda C)$. In contrast, in Bullshark, the metadata consists of $\Theta(n)$ references, so the size of a node is $\Theta(nC)$.



(a) Throughput.

(b) Average commit latency.

Figure 2 Simulated statistics of Bullshark and Sparse Bullshark with various bandwidth limits and sample sizes.

Practical implementations rely on a combination of Signed Echo Consistent Broadcast [7] with randomized *pulling*[10, Section 4.1]. This approach allows us to surpass the theoretical bound on deterministic Reliable Broadcast communication complexity [12] and achieve linear expected communication complexity. With this approach, the expected communication complexity is $\Theta(n^2C)$ for Bullshark and $\Theta(n^2 + n\lambda C)$ for Sparse Bullshark. Refer to Table 1 for the exact complexity and example DAG node sizes depending on the choice of cryptographic tools.

3.3 Evaluation

We evaluated our Sparse Bullshark protocol using a simple simulator implemented in Python. We used emulated message delays with (50, 10)-normal distribution for 99% of messages and (500, 10)-normal distribution for a randomly selected 1% of the messages. We limited the network bandwidth of each simulated validator to emulate real life limitations of network bandwidth. Our goal was to evaluate the resources consumed by our protocol on the *metadata* exchange, compared to Bullshark [17].

Figure 2a shows the throughput of the protocol in blocks committed per second depending on the sample size for various per-node network bandwidth limits. We can observe that small sample sizes dramatically increase the throughput, allowing the protocol to be scalable when, under the same conditions, Bullshark saturates the network and does not scale.

Figure 2b shows the change in latency depending on the sample size for various per-node network bandwidth limits. Despite the theoretical result of increased latency in terms of message delays, we can see that in many cases under limited network bandwidth, commit latency actually decreases significantly due to sampling.

Evaluations confirm that while the sampling significantly decreases the amount of metadata transmitted through the network, which also leads to a decrease in latency when the bandwidth restriction is below a certain threshold.

Due to the limitations of the simulator implementation, we only tested a relatively small system size of 100 nodes with small sample sizes. However, the same trends will apply to larger systems, with proportionally larger sample sizes as well.

References

1 Ittai Abraham, Eli Chouatt, Ivan Damgård, Yossi Gilad, Gilad Stern, and Sophia Yakoubov. Asynchronous algorand: Reaching agreement with near linear communication and constant expected time. *Cryptology ePrint Archive*, 2025.

2 Michael Anoprenko, Andrei Tonkikh, Alexander Spiegelman, and Petr Kuznetsov. Dags for the masses, 2025. doi:10.48550/arXiv.2506.13998.

3 Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. Shoal++: High throughput DAG BFT can be fast! *CoRR*, abs/2405.20488, 2024. doi:10.48550/arXiv.2405.20488.

4 Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. Mysticeti: Reaching the limits of latency with uncertified dags, 2024. arXiv:2310.14821.

5 Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *International Workshop on Public Key Cryptography*, pages 31–46. Springer, 2002.

6 Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987. doi:10.1016/0890-5401(87)90054-X.

7 Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

8 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002. doi:10.1145/571637.571640.

9 Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a COINcidence: Sub-quadratic asynchronous byzantine agreement WHP. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 25:1–25:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.DISC.2020.25.

10 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis, editors, *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 34–50. ACM, 2022. doi:10.1145/3492321.3519594.

11 Yvo Desmedt. Threshold cryptosystems. In *International Workshop on the Theory and Application of Cryptographic Techniques*, pages 1–14. Springer, 1992.

12 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, January 1985. doi:10.1145/2455.214112.

13 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017. doi:10.1145/3132747.3132757.

14 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 165–175. ACM, 2021. doi:10.1145/3465084.3467905.

15 Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li. Shoal: Improving DAG-BFT latency and robustness. In Jeremy Clark and Elaine Shi, editors, *Financial Cryptography and Data Security - 28th International Conference, FC 2024, Willemstad, Curaçao, March 4-8, 2024, Revised Selected Papers, Part I*, volume 14744 of *Lecture Notes in Computer Science*, pages 92–109. Springer, 2024. doi:10.1007/978-3-031-78676-1_6.

16 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2705–2718. ACM, 2022. doi:10.1145/3548606.3559361.

17 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: The partially synchronous version, 2022. doi:10.48550/arXiv.2209.05633.