Short Paper: Obelia: Scaling DAG-Based Blockchains to Hundreds of Validators

George Danezis[1,2], Lefteris Kokoris-Kogias[1], Alberto Sonnino[1,2], and Mingwei Tian[1]

[1] Mysten Labs
[2] University College London (UCL)

Abstract. Obelia improves upon structured DAG-based consensus protocols used in proof-of-stake systems, allowing them to effectively scale to accommodate hundreds of validators. Obelia implements a two-tier validator system. A core group of high-stake validators that propose blocks as in current protocols and a larger group of lower-stake auxiliary validators that occasionally author blocks. Obelia incentivizes auxiliary validators to assist recovering core validators and integrates seamlessly with existing protocols. We show that Obelia does not introduce visible overhead compared to the original protocol, even when scaling to hundreds of validators, or when a large number of auxiliary validators are unreliable.

Keywords: Blockchain · BFT Consensus · Dynamic Participation

1 Introduction

Blockchains using BFT quorum systems [37, 34, 35] divide time into 24-hour epochs, during which a committee of about 100 *validators*, elected through a Sybil-resistant mechanism [12], often a variant of proof-of-stake [22], operates the system using a BFT consensus protocol [6, 16, 3]. Their voting power correlates with their stake, allowing agreement on blocks of client transactions. Recent advancements in BFT protocols utilize directed acyclic graphs (DAG) [11, 33, 2, 3, 31, 24, 10, 20, 13, 25], achieving high throughput ($>$ 100k tx/s) and robustness against faults and network asynchrony [17, 11].

However, these consensus protocols limit operation to approximately 100 validators, sidelining many potential participants—often in the hundreds [36]. This exclusion is a sharp contrast to more traditional blockchains like Bitcoin [27] and Ethereum [4], which engage all participants, and is responsible of key weaknesses of quorum-based blockchains. First, only the subset of the total stake hold by these validators can be used to decentralize the system and benefit the blockchain ecosystem. Lower-stake players cannot participate in block proposals and typically resort to running full nodes without incentives [21]. Second the high throughput of DAG-based systems complicates state catch-up for new or crash-recovering validators, who either strain the committee's resources or depend on external unincentivized entities for recovery.

This paper introduces Obelia, an enhancement to DAG-based consensus that increases participation by enabling all stakeholders to sporadically author blocks. It incentivizes these participants to assist recovering validators and integrates seamlessly with existing protocols. However, developing Obelia involves overcoming significant challenges. (1) Obelia must avoid all-to-all communications between stakeholders as their large number makes it impractical. (2) It cannot rely on a classic BFT assumption for entities that have less stake and thus less incentive to be reliable. This challenge results from the inherent poor reliability of these slow-stake entities that can be offline for long periods of time. Obelia must ensure that all data they contribute to the chain remains available. Where traditional systems rely on monetary penalties to disincentivise unreliability [18] by assuming network synchrony, Obelia cannot follow this guidance as it aims to operate in the weaker asynchronous or partially synchronous network model of existing quorum-based protocols. (3) The final challenge consists in allowing these low-stake entities to participate in the consensus without slowing it down, as this would compromise the major benefit of quorum-based systems.

Obelia addresses these challenges by introducing a two-tier validator system. A core group of high-stake validators proposes blocks as in existing protocols, while a larger group of lower-stake auxiliary validators occasionally authors blocks. Auxiliary validators operate outside the critical path, proposing blocks at a slower pace and only after obtaining a strong proof of availability for their predisseminated block. Our implementation and evaluation of Obelia demonstrate that it does not introduce noticeable overhead compared to the original protocol, even when scaled to hundreds of potentially unreliable auxiliary validators.

Contributions. This paper makes the following contributions:

- We present Obelia, a novel mechanism enhancing DAG-based protocols enabling all stakeholders to engage in consensus and incentivizing support for recovering validators.
- We demonstrate Obelia's safety and liveness within the same network model as its underlying quorum-based protocol.
- We implement and evaluate Obelia on a realistic geo-distributed testbed, showing it adds negligible overhead despite a large number of potentially unreliable low-stake validators.

2 System Overview

We present the settings in which Obelia operates.

2.1 Validators selection

Obelia introduces the distinction between *core validators* and *auxiliary validators*. Core validators are the validators that *continuously* operate the consensus protocol and process transactions. In contrast, auxiliary validators participate sporadically while maintaining a copy of the DAG generated by core validators.

Both core and auxiliary validators are selected using a sybil-resistant mechanism [12], typically based on proof-of-stake [22]. Core validators are chosen similarly to existing quorum-based blockchains, consisting of roughly the 100 entities with the highest stake or those meeting specific criteria, such as owning a minimum percentage of the total stake [37]. Auxiliary validators include all other stakeholding entities not in the core group, typically numbering in the several hundreds [36], significantly surpassing the number of core validators. In practice, we expect current full nodes to operate as auxiliary validators.

2.2 System and threat model

Obelia assumes a computationally bounded adversary, ensuring the security of cryptographic properties such as hash functions and digital signatures. It operates as a message-passing system where core and auxiliary validators collectively hold $n = n_c + n_a$ units of stake [30], with n_c held by core validators and n_a held by auxiliary validators. Each unit of stake represents one "identity" [12], while each unit held by a core validator signifies one "unit of voting power" in the consensus system [26, 37]. This model aligns with deployed quorum-based blockchains, where core validators possess the majority of total stake ($n_c \gg n_a$) [37, 34, 35]. Obelia makes the following assumptions for core and auxiliary validators:

Core validators. Obelia works with existing DAG-based consensus protocols, inheriting their assumptions. Specifically, it requires that $n_c \geq 3f + 1$, where f is the maximum number of *Byzantine* core validators that may deviate from the protocol. The remaining stake is held by *honest* core validators who adhere to the protocol. There are no additional assumptions about the network model, core validators operate in the same setting as the underlying DAG-based consensus protocol. Note that most deployed DAG-based consensus protocols are partially synchronous [14], while some blockchains consider asynchronous protocols [26]. Under these assumptions, Section 3.3 demonstrates that a DAG-based protocol enhanced with Obelia is *safe*, meaning no two correct validators can commit conflicting transactions.

Auxiliary validators. For auxiliary validators, Obelia adopts a relaxed model due to their lower stake and reduced incentives for resource dedication and reliability. It assumes that at least $t_a \leq n_a$ units of stake are consistently held by honest and active auxiliary validators, regardless of the total number of auxiliary validators. The parameter t_a can be adjusted to balance system liveness (see Section 3.3) against the minimum participation of auxiliary validators. Auxiliary validators do not communicate with one another and only occasionally communicate with core validators over an asynchronous network. Section 3.3 shows that, under these assumptions, a DAG-based protocol enhanced with Obelia is *live*, ensuring that honest validators eventually commit transactions. Importantly, if the assumptions concerning auxiliary validators fail, safety remains guaranteed.

2.3 Design goals and challenges

Beyond ensuring safety and liveness within the same network model as the underlying consensus protocol, Obelia achieves several design goals (discussed in Section 3): **Increased participation (G1):** It allows all entities holding stake to author blocks in the consensus protocol, rather than limiting participation to the top 100 validators. **Incentivized synchronizer helpers (G2):** Obelia leverages auxiliary validators to assist slow or recovering core validators in catching up to the latest state. This approach incentivizes auxiliary validators to function as full nodes, storing and providing the DAG state to core validators to facilitate synchronization. **Generic design (G3):** The design of Obelia is directly applicable to a wide range of structured DAG-based consensus protocols.

Obelia also has performance goals that we demonstrate empirically in Section 4: **Negligible overhead (G4):** Obelia introduces minimal overhead, allowing the system to progress at the same speed as the underlying consensus protocol. **Scalability (G5):** Obelia scales effectively with the number of auxiliary validators. **Fault tolerance (G6):** Obelia maintains robust performance, remaining visibly unaffected by the presence of crashed auxiliary validators.

To achieve these goals, Obelia overcomes several challenges: **(Challenge 1):** Obelia cannot implement an all-to-all communication design due to the impractical number of auxiliary validators. **(Challenge 2):** Obelia cannot expect the classic BFT assumption to hold for these entities as they have less stake and thus less incentive to be reliable and prone to remain offline for long periods of time. **(Challenge 3):** Auxiliary validators must participate in the consensus without causing delays, as this would undermine the key advantage of quorum-based systems. They thus cannot take actions that impact the critical path.

3 The Obelia Design

We present the design of Obelia and argue its properties defined in Section 2.3.

3.1 DAG-based consensus protocols

DAG-based consensus protocols operate in logical *rounds*. In each round, every honest (core) validator creates a unique signed vertex. Byzantine validators may attempt to equivocate by producing conflicting vertices [19] or may abstain altogether. During each round, validators collect user transactions and vertices from other validators to construct their next vertex. Each vertex must reference a minimum number of vertices from the previous round (typically $2f + 1$ [11, 33, 3]) and adds fresh transactions that do not appear in preceding vertices.

Algorithm 1 outlines the operations of these validators, aligning with nearly all existing structured DAG-based consensus protocols [11, 33, 32, 2, 3, 19, 15, 39, 31, 24, 25, 10, 20, 9, 8, 7] and all DAG-based systems that have been deployed in production environments [11, 33, 3, 38].

When a core validator receives a new vertex v, it invokes ProcessCoreVertex(v) (Line 4). The validator first downloads v's causal history (Line 5) and verifies

Algorithm 1 Core Validator

```
1: T ← { }                                   ▷ Buffer client transactions
2: D_c ← { }                                 ▷ DAG of core vertices
3: D_a ← { }                                 ▷ DAG of auxiliary vertices

4: procedure ProcessCoreVertex(v)
5:     SyncCoreAncestors(v, D_c)
6:     SyncAuxAncestors(v, D_a)
7:     require ValidCoreVertex(v, D_c)
8:     AddToDag(v, D_c)
9:     L ← OrderNewLeaders(D_c)
10:    if L ≠⊥ then
11:        C ← Linearize(L, D_c, D_a)
12:        OutputToApplication(C)
13:    TryAdvance( )

14: procedure TryAdvance( )
15:    v' ← TryNewCoreVertex(T, D_c, D_a)
16:    if v' =⊥ then return
17:    AddToDag(v', D_c)
18:    SendToCoreValidators(v')

19: procedure ProcessAuxProposal(p)
20:    SyncCoreAncestors(v, D_c)
21:    require ValidAuxProposal(v, D_c)
22:    σ_p ← Sign(p)
23:    ReplyBack(σ_p)

24: procedure ProcessAuxVertex(v)
25:    DownloadCoreAncestors(v, D_c)
26:    require ValidAuxVertex(v, D_c)
27:    AddToDag(v, D_a)
28:    TryAdvance( )
```

Algorithm 2 Auxiliary Validator

```
1: T ← { }                                   ▷ Buffer client transactions
2: D_c ← { }                                 ▷ DAG of core vertices

3: procedure TryAdvance( )
4:     p ← TryNewProposal(T, D_c)
5:     if p =⊥ then return
6:     {σ_(p,i)} ← SendToCoreValidators(p)
7:     v' ← AssembleCertificate({σ_(p,i)})
8:     SendToCoreValidators(v')
```

v for validity (Line 7), which typically involves checking signatures, validating parent vertex references, and ensuring syntactical correctness. A valid v is then added to the local DAG view of the validator (Line 8).

Next, the validator checks if the new vertex triggers any commits. It derives a set of *leader* vertices either deterministically (in partially synchronous protocols [33, 32, 3]) or by reconstructing a global perfect coin [1] (in asynchronous protocols [11, 20]). Using a protocol-specific decision rule, it analyzes DAG patterns to establish a total order among the leaders (Line 9). If this yields a non-empty sequence, the validator linearizes the DAG into a sequence of vertices C that it outputs to the application layer (Line 11). This linearization step uses a deterministic function like depth-first search over the sub-DAG defined by each leader in the sequence [19, 11, 33].

To advance the round, the validator attempts to create a new vertex v' through TryAdvance( ) (Line 14). This succeeds if the validator possesses enough vertices from the previous round and, in partially synchronous protocols, enough leader vertices or if a timeout has occurred. If successful, the validator adds v' to its local DAG view and broadcasts it to the other validators (Line 17). Creating a vertex may involve reliable or consistent broadcasting [19, 11, 33, 32, 2], a best-effort broadcast [3, 20], or a hybrid of both [24, 10].

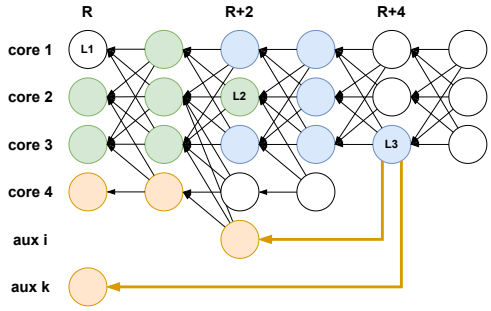

Fig. 1: Example of Obelia execution with 4 core validators and $t_a = 2$.

3.2 Vertex creation rule and commit rule

We present the protocol for auxiliary validators and the modifications made to the core validator protocol, using Figure 1 as an example. We denote vertices using the notation $v(author, round)$, where $author$ represents the validator that authored the vertex and $round$ indicates the round number.

Auxiliary validators. Auxiliary validators operate as full nodes, downloading the DAG generated by core validators while also collecting client transactions. Algorithm 2 illustrates their protocol. Each auxiliary validator composes a signed *proposal* containing client transactions and hash references to at least $2f + 1$ vertices created by core validators for a past round[3]. They then send it to the core validators who check their validity and reply with a counter signature. The auxiliary validator then assembles a vertex, made up of the $2f + 1$ counter-signatures, and rebroadcasts it to all core validators. For example, $v_a(aux_i, R{+}2)$ references the vertices from round $R+1$ created by core validators $core_2$, $core_3$, and $core_4$. This design choice solves **challenge 1** by forgoing communication between auxiliary validators and effectively leveraging the core validators as communication layer and to obtain a proof of availability for their data.

Core validators. Algorithm 1 shows the modifications to the core validator's protocol in orange. Core validators execute ProcessAuxProposal(p) to validate, download the causal history, and counter-sign an auxiliary validator's proposal p. Since p references $2f + 1$ core validator vertices, this verification allows core validators to synchronize any potentially missing vertices from the author of p. This protocol incentivizes auxiliary validators to collaborate, as inclusion of their vertices in the final commit sequence grants them a share of vertex rewards [4]. The function ProcessAuxVertices(v_a) shows how core validators process a vertex v_a (that is, a proposal p counter-signed by $2f + 1$ core validators). Core validators add this vertex to a new map, D_a, which will later be merged into the DAG D_c operated by core validators. This design choice solves **challenge 2** by ensuring that malicious or unreliable auxiliary validators cannot

[3] Auxiliary vertices referencing core vertices too far in the past are not included in the final commit sequence, as they will be pruned by garbage collection [11].

affect the protocol once they delivered their vertex to a core validator. Since a correctly signed auxiliary vertex indicates that at least $2f + 1$ core validators possess its data and causal history, the proposed data from auxiliary validators remains highly available despite their potential unreliability.

Next, we modify the vertex creation rules: every fixed number of rounds, the core validator leader(s) (e.g., L_3 in Figure 1) must reference vertices from auxiliary validators with a joint stake totaling at least t_a (see Section 2.2). In the figure, L_3 references $v_a(aux_i, R + 2)$ and $v_a(aux_k, R)$. This design solves **challenge 3** by allowing auxiliary validators to participate in the consensus without slowing it down. It ensures that auxiliary validators can asynchronously create vertices at a slower pace than core validators, without impacting the critical path, while still maintaining minimal required participation.

Auxiliary vertices are essentially treated as *weak links* [19], with core validators required to download them before processing a vertex. Auxiliary vertices are not used to establish the order of committed leaders but are included during the linearization step (Line 11). Designing Obelia to operate only at the linearization layer makes it compatible with nearly all DAG-based protocols: while leader ordering algorithms vary across protocols, linearization is a common procedure. The validator linearizes the vertices within the sub-DAG defined by each leader vertex through any deterministic procedure, such as a depth-first search [19]. If a vertex has already been linearized by a previous leader, the validator omits. Each leader is processed sequentially, ensuring all vertices appear in the final commit sequence in a deterministic order based on their causal dependencies.

In the example shown in Figure 1, the sequence of committed leaders (output from Algorithm 1) is ($[L_1, L_2, L_3]$). L_1 does not define any sub-DAG (the process begins at round R), so only L_1 is added to the commit sequence. L_2 defines a sub-DAG of the green vertices, which are linearly ordered, as e.g., $v_c(core_1, R + 1)$, $v_c(core_2, R)$, $v_c(core3, R)$, $v_c(core_2, R + 1)$, $v_c(core_3, R + 1)$, and L_2. While processing L_3, which defines the sub-DAG of both blue and orange vertices, the validator collects and linearizes all such vertices. As a result, although the original DAG might have excluded the core orange vertices ($v_c(core_4, R)$ and $v_c(core_4, R+1)$) and would have omitted the auxiliary vertices ($v_a(aux_i, R+2)$ and $v_a(aux_k, R)$), Obelia guarantees their inclusion in the final commit sequence. This inclusion helps core validator $core_3$ to synchronize parts of the DAG that were potential missing from its local view.

3.3 Security analysis

Algorithm 1 shows that Obelia modifies the original consensus protocols only in three places: (1) Line 6 (to sync auxiliary vertices' ancestors), (2) Line 11 (the linearization layer), and (3) Line 15 (proposing a new vertex).

Safety. Obelia does not alter the way the underlying protocol commits leaders, ensuring safety as long as DAG linearization (Line 11) is deterministic, as is typical in existing DAG-based protocols. Safety holds regardless of the number of honest auxiliary validators.

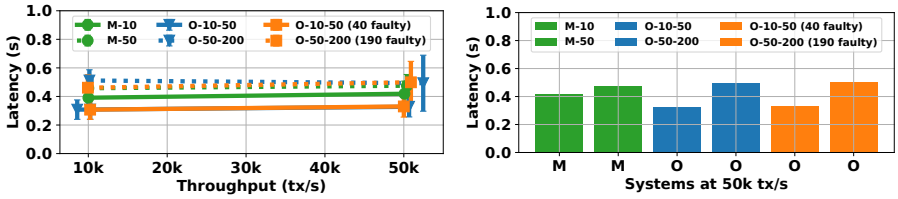

Fig. 2: Comparative evaluation of Mysticeti (10 and 50 validators) and Obelia (10 core + 50 auxiliary validators, 50 core + 200 auxiliary validators). Left: throughput-latency graph. Right: Latency zoom at 50k tx/s.

Liveness. Obelia maintains liveness under the same network model as the base protocol by ensuring (1) core validators can synchronize missing auxiliary vertices (Line 6), (2) the DAG linearization terminates (Line 11), and (3) core validators can eventually create new vertices (Line 15). Point (1) holds as core validators only reference certified auxiliary vertices, guaranteeing at least $f+1$ honest validators hold them [5]. Point (2) is satisfied as auxiliary vertices are linearized like core ones. Point (3) follows from the assumption that at least t_a auxiliary validators are honest (which we assume in Section 2.2), ensuring at least $2f+1$ core validators successfully call ProcessAuxProposal(·) (Line 19) and ProcessAuxVertex(·) (Line 24) which update their local DAGs with new auxiliary vertices, allowing the creation of new core vertices.

4 Implementation and Evaluation

We implement[4] Obelia as a fork of Mysticeti [3] and evaluate it on a geo-distributed AWS testbed. We use the same setup as the Mysticeti paper [3][5] and only test for loads up to 50k tx/s to limit costs. We set $t_a = 10\%$ (Section 2.2) and auxiliary validators propose blocks every few seconds. We use the notation M-X to indicate Mysticeti running with X validators, and O-X-Y to indicate Obelia running with X core validators and Y auxiliary validators.

Figure 2 shows that all system configurations maintain a latency of approximately 400ms when processing loads of either 10k tx/s or 50 tx/s. Notably, regardless of the committee size, there is no statistical difference between Mysticeti in its barebone configuration and when equipped with Obelia, confirming our claim **G4** (from Section 2.3) that Obelia introduces negligible overhead. Additionally, Figure 2 demonstrates that Obelia can scale to 200 auxiliary validators, thus validating our scalability claim **G5**. Lastly, Figure 2 (orange lines) illustrates that even a large number (up to 190) of crashed auxiliary validators do not noticeably impact protocol performance, supporting our claim **G6**.

[4] `https://github.com/asonnino/mysticeti/tree/obelia` (commit `fe74642`)

[5] 13 different AWS regions; `m5.8xlarge` instances (with 32 vCPUs, 128GB RAM, and 10Gbps network); 512 bytes transactions; each data point is the average latency and error bar represent one stdev; benchmarks run for multiple minutes under fixed load.

5 Discussion and Related Work

To the best of our knowledge, Obelia has no close related work. However, it draws inspiration from DagRider's weak links [19], which include older blocks not required for leader selection (although for different reasons than Obelia), and Narwhal's vertex-creation rule [11], which ensures vertex availability before inclusion in the DAG. Future work include the analysis of Obelia where auxiliary validators operated under the sleepy model [29] to explore potential improvements in censorship resistance; whether auxiliary validators can enhance the protocol's safety for clients who trade latency, as in OFlex [23, 28]; and whether they can aid in fork recovery when more than f Byzantine core validators are present. Lastly, we leave as future work the incentive analysis and its impact on the relationship between n_c and n_a ($n_c < n_a$, $n_c > n_a$, or $n_c \approx n_a$).

Acknowledgments. This work is supported by Mysten Labs. We thank Srivatsan Sridhar for his feedback on the paper and discussions on future work.

References

1. Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G.: Bingo: Adaptivity and Asynchrony in Verifiable Secret Sharing and Distributed Key Generation. In: Advances in Cryptology – CRYPTO 2023 (2023)
2. Arun, B., Li, Z., Suri-Payer, F., Das, S., Spiegelman, A.: Shoal++: High throughput dag bft can be fast! arXiv preprint arXiv:2405.20488 (2024)
3. Babel, K., Chursin, A., Danezis, G., Kokoris-Kogias, L., Sonnino, A.: Mysticeti: Low-Latency DAG Consensus with Fast Commit Path (2024)
4. Buterin, V., et al.: Ethereum white paper. GitHub repository **1**, 22–23 (2013)
5. Cachin, C., Guerraoui, R., Rodrigues, L.: Introduction to reliable and secure distributed programming. Springer Science & Business Media (2011)
6. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Seltzer, M.I., Leach, P.J. (eds.) Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999. pp. 173–186. USENIX Association (1999)
7. Dai, X., Li, W., Wang, G., Xiao, J., Chen, H., Li, S., Zomaya, A.Y., Jin, H.: Remora: A low-latency dag-based bft through optimistic paths. IEEE Transactions on Computers (2024)
8. Dai, X., Wang, G., Xiao, J., Guo, Z., Hao, R., Xie, X., Jin, H.: Lightdag: A low-latency dag-based bft consensus through lightweight broadcast. Cryptology ePrint Archive (2024)
9. Dai, X., Zhang, Z., Guo, Z., Ding, C., Xiao, J., Xie, X., Hao, R., Jin, H.: Wahoo: A dag-based bft consensus with low latency and low communication overhead. IEEE Transactions on Information Forensics and Security (2024)
10. Dai, X., Zhang, Z., Xiao, J., Yue, J., Xie, X., Jin, H.: Gradeddag: An asynchronous dag-based bft consensus with lower latency. In: 2023 42nd International Symposium on Reliable Distributed Systems (SRDS). pp. 107–117. IEEE (2023)
11. Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.: Bridging the Gap of Timing Assumptions in Byzantine Consensus. In: EuroSys '22: Proceedings of the Seventeenth European Conference on Computer Systems (2022)

12. Douceur, J.R.: The sybil attack. In: International workshop on peer-to-peer systems. pp. 251–260. Springer (2002)
13. Duan, S., Wang, X., Zhang, H.: FIN: Practical Signature-Free Asynchronous Common Subset in Constant Time. In: CCS '23: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (2023)
14. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM (JACM) **35**(2), 288–323 (1988)
15. Gao, Y., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 1187–1201 (2022)
16. Gelashvili, R., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A., Xiang, Z.: Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback (2021)
17. Giuliari, G., Sonnino, A., Frei, M., Streun, F., Kokoris-Kogias, L., Perrig, A.: An empirical study of consensus protocols' dos resilience. In: Proceedings of the 19th ACM Asia Conference on Computer and Communications Security. pp. 1345–1360 (2024)
18. He, Z., Li, J., Wu, Z.: Don't trust, verify: The case of slashing from a popular ethereum explorer. In: Companion Proceedings of the ACM Web Conference 2023. pp. 1078–1084 (2023)
19. Keidar, I., Kokoris-Kogias, E., Naor, O., Spiegelman, A.: All You Need is DAG. In: PODC'21: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (2021)
20. Keidar, I., Naor, O., Poupko, O., Shapiro, E.: Cordial Miners: Fast and Efficient Consensus for Every Eventuality. In: 37th International Symposium on Distributed Computing (DISC 2023) (2022)
21. Król, M., Ascigil, O., Rene, S., Sonnino, A., Pigaglio, M., Sadre, R., Lange, F., Riviere, E.: Disc-ng: Robust service discovery in the ethereum global network. R (2024)
22. Kuçi, A., Cachin, C., Marson, G.A.: Proof-of-stake blockchain: Ouroboros (2021)
23. Malkhi, D., Nayak, K., Ren, L.: Flexible byzantine fault tolerance. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security. pp. 1041–1053 (2019)
24. Malkhi, D., Stathakopoulou, C., Yin, M.: Bbca-chain: One-message, low latency bft consensus on a dag. arXiv preprint arXiv:2310.06335 (2023)
25. Malkhi, D., Szalachowski, P.: Maximal extractable value (mev) protection on a dag. arXiv preprint arXiv:2208.00940 (2022)
26. Mysten Labs: The sui blockchain. `https://github.com/mystenLabs/sui` (2024)
27. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Satoshi Nakamoto (2008)
28. Neu, J., Sridhar, S., Yang, L., Tse, D.: Optimal flexible consensus and its application to ethereum. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 3885–3903. IEEE (2024)
29. Pass, R., Shi, E.: The sleepy model of consensus. In: ASIACRYPT (2017)
30. Saad, S.M.S., Radzi, R.Z.R.M.: Comparative review of the blockchain consensus algorithm between proof of stake (pos) and delegated proof of stake (dpos). International Journal of Innovative Computing **10**(2) (2020)
31. Shrestha, N., Shrothrium, R., Kate, A., Nayak, K.: Sailfish: Towards improving latency of dag-based bft. Cryptology ePrint Archive (2024)

32. Spiegelman, A., Arun, B., Gelashvili, R., Li, Z.: Shoal: Improving dag-bft latency and robustness. arXiv preprint arXiv:2306.03058 (2023)
33. Spiegelman, A., Giridharan, N., Sonnino, A., Kokoris-Kogias, L.: Bullshark: DAG BFT Protocols Made Practical. In: CCS '22: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (2022)
34. The Aptos team: Aptos. `https://aptoslabs.com` (2023)
35. The Solana team: Solana: Powerful for developers. fast for everyone. `https://solana.com` (2024)
36. The Sui scan team: suiscan. `https://suiscan.xyz/mainnet/home` (2024)
37. The Sui team: The sui blockchain. `http://sui.io` (2023)
38. Tsimos, G., Kichidis, A., Sonnino, A., Kokoris-Kogias, L.: Hammerhead: Leader reputation for dynamic scheduling. In: 2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS). pp. 1377–1387. IEEE (2024)
39. Yang, L., Park, S.J., Alizadeh, M., Kannan, S., Tse, D.: {DispersedLedger}:{High-Throughput} byzantine consensus on variable bandwidth networks. In: 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). pp. 493–512 (2022)