Enhancing Ethereum PoA Clique Network with DAG-based BFT Consensus

***Abstract*—Proof-of-Authority (PoA) is one of the popular blockchain consensus protocols, particularly for permissioned blockchains. However, PoA (as in Clique) still faces performance scalability issues. Its leader-based design restricts throughput and results in resource inefficiency, as it permits only one block to be accepted in each round, discarding all other concurrently proposed blocks in the same round. In this paper, we introduce DaPoA, an effort to enhance the Ethereum PoA Clique network to improve its performance scalability. The main idea of DaPoA is that it follows a parallel leader architecture that allows multiple sealers to propose their blocks in each round, which are then accepted in the same round. To achieve this, DaPoA employs a state-of-the-art directed acyclic graph (DAG)-based Byzantine Fault Tolerant (BFT) consensus algorithm, which enables parallel block proposals for high scalability with leader-less design. We have implemented and evaluated DaPoA on top of the Ethereum PoA Clique to demonstrate its performance scalability. In our setting of eight nodes, DaPoA achieves a 2.47x higher throughput and 5.76x lower latency compared to Clique.**

***Index Terms*—Proof-of-Authority, Clique, DAG-based BFT Consensus**

I. Introduction

In blockchain technology, consensus protocols enable trustworthy transactions among untrusted participants with properties such as immutability, transparency, and verifiability. Proof-of-Authority (PoA), which was first proposed in the Ethereum community [1], [2], is one of the popular consensus algorithms for permissioned blockchains due to its better performance and energy efficiency than Proof-of-Work. In PoA, a predetermined set of authority nodes (or sealers) follows a Byzantine-fault tolerant (BFT) style protocol, exchanging only a few messages to reach an agreement on blocks. Representatively, Clique [1] is the most widely used PoA consensus implementation in the Ethereum-based blockchains, including go-ethereum (Geth) [3], Binance Chain [4], Polygon (MATIC) [5], Consensys [6] and Hyperledger Besu [7].

However, PoA (as in Clique) still has several issues. First, it has limited block throughput because it allows only one block by a sealer to be accepted in a round. Second, it is inefficient in terms of resource usage, because the proposed blocks by other sealers in the same round are simply discarded, leading to the waste of computing resources. Third, it essentially centralizes the block generation authority to a certain sealer in a round, inducing a potential performance bottleneck [8], [9] and unfairness [10], [11].

To address these issues, we propose DaPoA in this paper, an effort to enhance PoA Clique network by leveraging a DAG-based BFT consensus. Specifically, DaPoA takes a leader-less design that allows multiple sealers to propose their blocks concurrently in a round in a decentralized manner. Also, these proposed blocks in the round, if valid, are integrated into the reliable DAG structure. For this, DaPoA employs the DAG-based BFT consensus [12]–[14], utilizing a reliable block exchange between sealers based on a DAG mempool (i.e., Narwhal) and a communication-free consensus algorithm (i.e., Tusk/Bullshark), to achieve high throughput and resource efficiency within a parallel leader architecture. We depict our approach in Fig. 1.

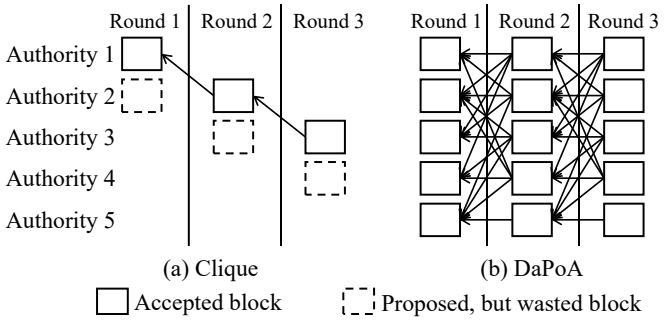


Fig. 1: Comparisons of Clique and Our Approach

We have implemented a prototype of DaPoA on top of the Ethereum Geth [3], using an available DAG BFT implementations [15] with minimal extensions. We evaluate the performance scalability of DaPoA using Hyperledger Caliper [16] with a simple payment smart contract written in Solidity, and show that it outperforms the existing Clique algorithm in terms of throughput and latency when the number of sealers becomes large. In our setting of eight nodes, DaPoA achieves a 2.47x higher throughput and 5.76x lower latency compared to Clique.

The rest of this paper is organized as follows. Section II describes the background and related work of DaPoA. Section III presents the design and implementation of DaPoA. Section IV outlines a security analysis of DaPoA. Section V demonstrates our evaluation of DaPoA. Finally, Section VI concludes this paper.

II. Background and Related Work

A. Proof-of-Authority

A PoA Clique network consists of N fixed authorities, also called *sealers*. The Clique proceeds in sequential rounds, in which each has a rotating leader who is determined as $l = h \bmod N$, where h is the block number to propose. Besides the leader, Clique also allows multiple non-leaders (or edge-turn

sealers) to simultaneously propose blocks within a round to handle a faulty leader, albeit with a random delay. Note that out of N sealers, the number of edge-turn sealers and other remaining sealers (or out-of-turn sealers) are both $\lceil \frac{N}{2} \rceil - 2$ (i.e., smallest majority) and $\lfloor \frac{N}{2} \rfloor + 1$ (i.e., largest minority), respectively [11]. To break a tie among the multiple block proposals, Clique uses *difficulty* field in the block header, where the value is 2 and 1, for the leader and edge-turn sealers, respectively. Then, Clique resolves the fork problem based on the *difficulty* value, using the scoring mechanism of the GHOST protocol [17], which favors leader proposals. In the Clique algorithm, we observe that despite multiple blocks being proposed in each round, only one block is accepted per round, while almost half of the proposed blocks are regularly discarded. This leads to limited throughput and resource inefficiency. Moreover, out-of-turn sealers remain idle in terms of block proposals during each round, presenting an opportunity to exploit these idle resources for performance improvement.

CoPoA [18] represents a recent effort to enhance PoA performance by supporting concurrent block generation in the PoA network. In CoPoA, multiple authorities are allowed to propose blocks concurrently in a round, which are then incorporated into a super block made by a round leader. While the goal of exploiting idle resources in non-proposer nodes to improve block concurrency is similar, the key difference between our proposal and CoPoA lies in achieving decentralization. In CoPoA, although the blocks themselves are proposed concurrently, it is the single leader's responsibility to combine them into a super block. This leader-based approach potentially leads to a potential performance bottleneck especially under faulty leaders. In contrast, DaPoA does not depend on a single leader to achieve consensus, so it mitigates the leader bottleneck problems. Furthermore, CoPoA leverages parallel chains to achieve higher throughput, which is orthogonal to our proposal.

Due to its simple design, PoA is known to face several security challenges [10], [11], [19], [20]. One notable issue is unfairness, particularly related to order manipulation attacks, which has been extensively explored in PoA networks [10], [11]. In the context of fairness, two types have been identified in their work: block-level and transaction-level Block-level fairness implies that sealers should propose their blocks in a prearranged order, while transaction-level fairness concerns whether sealers selectively include transactions to maximize their profits. However, as DaPoA employs a parallel leader architecture, it can mitigate both types of unfairness by design.

B. DAG BFT

DAG BFT is a recent breakthrough for achieving high performance in the blockchain community [12]–[14]. DAG BFT addresses problems in the Byzantine Atomic Broadcast (BAB) protocol. In this approach, servers proceed in sequential rounds and broadcast their proposals in each round to reach consensus among correct servers in a partially asynchronous network, which tolerates f Byzantine failures out of $3f + 1$ servers. The BAB provides guarantees, including *agreement* for a consistent message delivery in a round, *integrity* for an exactly-once delivery of a message in a round, *validity* for eventual delivery of a message in a round, and *total order* for the same delivery order for messages [12].

The DAG BFT supports parallel block proposals by all servers in each round, which are then to be accepted in subsequent rounds with DAG structure. It consists of two steps. First, the servers disseminate transaction data through reliable broadcast to build a local view of DAG structure. Second, the servers interpret their local view of DAG structure to achieve consensus using deterministic ordering logic and shared randomness.

We summarized its operation as in [13]. At the beginning of each round, servers propose a block that contains transactions received from clients and each block corresponds to a vertex in the DAG. By using reliable broadcast, they exchange their block proposals with other servers to achieve the integrity and availability of their proposed blocks. Specifically, servers broadcast their block proposals in each round, then they reply with an acknowledgment if valid. Once $2f + 1$ distinct acknowledgments are collected for the round, each server combines them to create a certificate of block availability and also broadcasts it to other servers. This is then included in the block of the next round, resulting in the block having references, i.e., edges, to previous blocks in the previous rounds. Finally, by periodically and deterministically interpreting their local view of DAG structure, they achieve consensus by calculating the total order of the blocks with a randomly selected leader vertex.

III. Design and Implementation

A. Overview

We assume a consortium blockchain where the identities of all participants are known in advance and clients submit their transactions to a consortium member operating a DaPoA node in the PoA network that they trust. We assume that DaPoA runs in a partially asynchronous network with an unknown time bound Δ and tolerates f Byzantine failures out of $n = 3f + 1$ nodes. Also, we assume that standard cryptographic primitives are not subverted.

At a high level, DaPoA allows parallel block proposals in each round in PoA network, and these proposals, if valid, are incorporated into a reliable DAG block in a decentralized manner using the DAG BFT protocol. Specifically, DaPoA leverages reusable modules of DAG BFT, including DAG-based reliable broadcast (RBC) to disseminate block proposals and consensus to finalize the total order of block proposals in a round. With this approach, DaPoA can significantly improve the overall performance and resource efficiency within the PoA blockchain.

B. Algorithm

We present the operations of DaPoA in Algorithm 1. Each node initializes with locally maintained data structures, including *chain* for storing finalized blocks, round r representing a

Algorithm 1: Algorithm of DaPoA

```
1  Init():
2      chain ← {}; r_b ← 0; rdb ← {};
3  MainLoop() at p_i:
4      Extract Batch B from TxPool
5      b ← GenerateBlockProposal(B, r_b)
6      RBC.Broadcast(b)
7      dag_block ← Wait delivery of b at r_b
8      validate(dag_block)
9      chain[dag_block.round] ← dag_block
10     r_b ← r_b + 1
11 procedure GenerateBlockProposal(B, r_b) at p_i
12     b ← {}; b.batch ← B; b.source ← i
13     b.round ← r_b
14     return b
15 on receiving b from RBC.Deliver at p_i:
16     d ← digest(b)
17     rdb.put(d, b)
18 on receiving ordered_dag from Consensus at p_i:
19     dag_block ← {}
20     dag_block.round ← ordered_dag.r_d
21     for d ∈ ordered_dag.digests do
22         b ← rdb.get(d)
23         append(dag_block.blocks, b)
24     end
25     Notify dag_block at block round r_b to MainLoop()
```

sequence of proposed blocks, and rdb for keeping unordered blocks delivered by RBC (L1-2).

Each DaPoA node enters the MainLoop while continuously receiving transactions from nearby clients, which are then stored in the local TxPool. At the beginning of a round, each node p_i extracts a transaction batch B from the TxPool and generates a block proposal b from the batch B, the source node identifier i, and the current block round number r_b (L4-5, L11-14). An empty block is constructed if transactions do not exist in the TxPool at the moment. Subsequently, each p_i broadcasts b to other nodes using RBC for reliable dissemination of b (L6). Upon delivery of each b by RBC from other nodes at p_i, it is stored in rdb, keyed by the digest of b (L15-17).

After that, p_i waits until b is finalized in dag_block by the Consensus (L7). Once Consensus finalizes the total order of block proposals for round r_b, it delivers an $ordered\_dag$ to p_i, containing *digests*, a totally ordered list of digests of block proposals and r_d, a DAG round number of this committed sub DAG in terms of the whole DAG, determined by Cosensus (L18-20). The algorithm then iterates over this list of digests, retrieving the corresponding blocks from rdb to assemble them into a dag_block, after which it notifies the dag_block to the MainLoop (L21-25). Finally, the dag_block is validated and committed to the chain while increasing the current block round number by 1 (L8-10).

For the correct operation of the algorithm, we consider a consistent relative order between r_b and r_d. In other words, for two block proposals $b1$ with r_{b1} and $b2$ with r_{b2} where $r_{b1} < r_{b2}$, $b1$ must be delivered before $b2$ in the DAG. As the DaPoA algorithm requires the proposal of the previous block round to be committed in a certain DAG round before submitting the next block, i.e., synchronous block proposal, their relative order is guaranteed to be consistent.

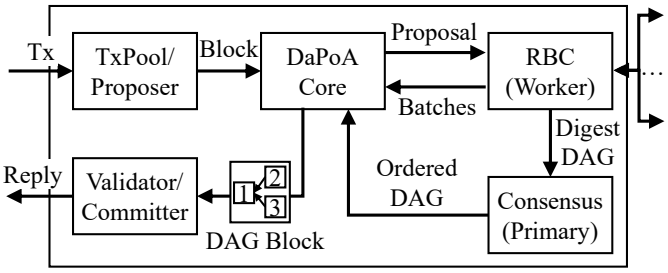
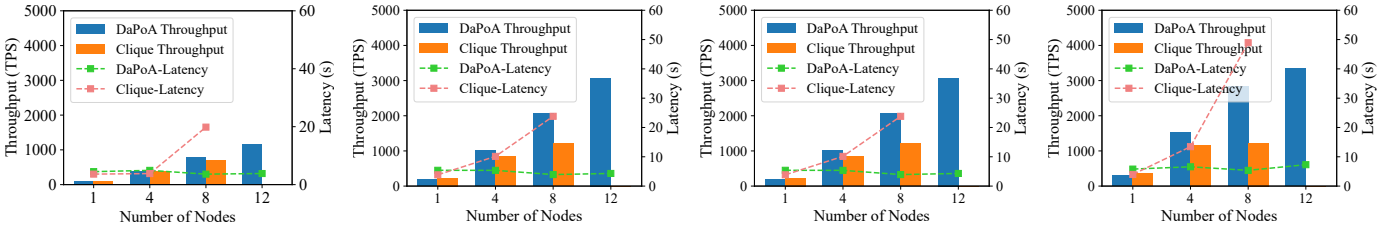


Fig. 2: Implementation of DaPoA.

C. Implementation

To integrate the existing design of DAG BFT into the PoA network, we developed DaPoA Core, a module that facilitates seamless communication between PoA Clique and DAG BFT components. We implemented DaPoA Core in GoLang and is installed on Geth [21] and used an initial version of Narwhal [15] writtin in Rust for the DAG BFT component. To communicate between the two, we use ProtocolBuffers over gRPC [22]. We depict our implementation of DaPoA node in Fig. 2. We reuse and extend the existing modules of DAG BFT, including Worker and Primary for RBC of transaction blocks and consensus, respectively to enable execution in the PoA Clique netowrk. Specifically, we extend Worker and Primary, both for notifying the exchanged block proposals through RBC to the DaPoA Core and their orders in block digest, respectively. For the Worker module, we added a notification function inside the Processor task [23], which is responsible for processing the block proposals exchanged between nodes through the RBC. For Primary, we extended analyze async function [24] that continuously receives a sequence of committed certificates from Consensus to notify the DaPoA Core of the ordered digests present in each certificate.

When the DaPoA Core receives a block from the TxPool/Proposer, it builds a proposal and submits it to RBC that is performed by Worker process in DAG BFT. The Workers exchange proposals with other nodes via RBC, and those exchanged proposals are delivered to the DaPoA Core in the form of batches. Then, DaPoA node stores them in a local database as key/value pairs of (digest, block). After the RBC module achieves availability of block proposals, it sends their digests to the Consensus module, which is performed by Primary process to finalize the total order of digests of block proposals locally. Then, the Primary notifies the OrderedDAG that contains a list of ordered digests and a dag number of this sub-dag to DaPoA Core. Following this, DaPoA Core iterates the ordered digests to retrieve the corresponding blocks from the local database and combine them into a DAG



(a) SendRate=100. (b) SendRate=300. (c) SendRate=500. (d) SendRate=700.

Fig. 3: Performance Scalability Evaluation.

Block. Finally, this DAG Block is validated and prepared for commitment in the Committer/Validator module.

IV. Security Analysis

We briefly outline three attack scenarios for the security analysis of DaPoA: fork attack, omission attack, and unfairness. First, in the case of a fork attack, a malicious node may attempt to create two different block proposals in the same round to violate consistency. However, it is guaranteed that correct nodes deliver the same proposals in the same round due to *agreement* property, i.e., if a correct node delivers a dag_block in round r_d, then every correct node delivers the same dag_block in round r_d. Second, in the case of an omission attack, a node may intentionally fail to make any proposal in a certain round. However, DaPoA can proceed with the protocol despite up to f failures out of $3f+1$ nodes, by allowing and accepting multiple proposals in a round due to its leaderless approach. Therefore, apart from performance degradation, there is no liveness violation due to omission attacks.

Third, in the case of an unfairness attack, two cases need to be considered. In terms of block-level unfairness, nodes in DaPoA participate equally in the protocol, meaning all nodes can submit their block proposals in a round and reach an agreement for these proposals to be accepted in that round in a decentralized manner. Therefore, unlike Clique, which accepts only one block per round, DaPoA can mitigate block-level unfairness due to its design. In terms of transaction-level unfairness, a profitable node may manipulate the order of transactions in their block proposal to maximize their profit. However, since DaPoA does not rely on a single node to propose and finalize a block, the impact of an unfairness attack is confined to each node's own block proposal. Furthermore, assuming a consortium blockchain setup where each member operates their own DaPoA node and clients submit their transactions to a DaPoA node they trust, the significance of transaction-level unfairness in each block proposal diminishes.

V. Evaluation

We evaluated DaPoA to demonstrate its performance scalability in the number of nodes up to twelve nodes, with comparisons to Clique. We used Hyperledger Caliper [16] to generate transaction workloads with simple payment application written in a Solidity smart contract [25] and measure throughput and latency of DaPoA. For Caliper configurations, we varied the rate of transaction submission (SendRate) with *fixed-rate* and *tps*. We set the gaslimit to 10^9, period to 5, transaction confirmation blocks to 2 in the Clique settings. For DAG BFT configurations, each DaPoA node corresponds to a single Narwhal node, consisting of one worker and one primary with Bullshark consensus. Note that for a single node experiment, we configured one DaPoA node with four Narwhal nodes to satisfy the $3f+1$ nodes condition required for the DAG BFT. We set the batch size to 2MB and used the default parameters for the remaining configurations as in benchmarks/fabfile.py [15]. We conduct our experiments on a local machine equipped with AMD Ryzen Threadripper 3990X CPU, 256 GB RAM, 1TB SSD with Ubuntu 22.04.

We demonstrate the experimental results of DaPoA's performance scalability in comparison to Clique, as shown in Fig. 3. Overall, when the number of nodes is small, we observe that DaPoA does not have any performance benefits over Clique due to the introduction of additional overhead by DaPoA, which rarely exploits parallelism. However, as the number of nodes increases, DaPoA's throughput linearly increases, up to 3606 in twelve nodes where SendRate is 700, outperforming Clique. In the case of the eight nodes experiment, DaPoA achieves a 2.47x higher throughput and 5.76x lower latency compared to Clique. This is attributed to the effective processing of parallel block proposals in DaPoA, while Clique accepts only one block per round regardless of the number of nodes. Note that when the number of nodes is twelve, we cannot run Clique in our environment due to its high overhead. From this experiment, we see that DaPoA demonstrates better scalability compared to Clique.

VI. Conclusion

In this paper, we present DaPoA, an effort to enhance Ethereum PoA Clique network via leveraging a DAG-based BFT consensus. DaPoA allows concurrent block proposals to be accepted in a round in a decentralized manner. We implemented and evalulated DaPoA on top of the Ethereum Geth to demonstrate its performance scalability.

References

[1] (2017) Eip-225: Clique proof-of-authority consensus protocol. [Online]. Available: https://eips.ethereum.org/EIPS/eip-225

[2] (2015) Poa private chains. [Online]. Available: https://github.com/ethereum/guide/blob/master/poa.md

[3] (2023) clique. [Online]. Available: https://github.com/ethereum/go-ethereum/tree/master/consensus/clique/
[4] (2023) Binance chain clique. [Online]. Available: https://github.com/bnb-chain/bsc/tree/master/consensus/clique
[5] (2023) Polygon matic clique. [Online]. Available: https://github.com/maticnetwork/bor/tree/master/consensus
[6] (2023) Consensys clique. [Online]. Available: https://github.com/ConsenSys/quorum/blob/master/consensus/clique/clique.go
[7] H. Besu. (2023, Aug.) Hyperledger besu clique. [Online]. Available: https://github.com/hyperledger/besu/tree/main/consensus/clique/src
[8] K. Toyoda, K. Machi, Y. Ohtake, and A. N. Zhang, "Function-level bottleneck analysis of private proof-of-authority ethereum blockchain," *IEEE Access*, vol. 8, pp. 141 611–141 621, 2020.
[9] C. N. Samuel, S. Glock, F. Verdier, and P. Guitton-Ouhamou, "Choice of ethereum clients for private blockchain: Assessment from proof of authority perspective," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2021, pp. 1–5.
[10] Q. Wang, R. Li, Q. Wang, S. Chen, and Y. Xiang, "Exploring unfairness on proof of authority: Order manipulation attacks and remedies," in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 123–137. [Online]. Available: https://doi.org/10.1145/3488932.3517394
[11] X. Zhang, Q. Wang, R. Li, and Q. Wang, "Frontrunning block attack in poa clique: A case study," 2022.
[12] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is dag," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, ser. PODC'21. New York, NY, USA: Association for Computing Machinery, 2021, p. 165–175. [Online]. Available: https://doi.org/10.1145/3465084.3467905
[13] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: A dag-based mempool and efficient bft consensus," in *Proceedings of the Seventeenth European Conference on Computer Systems*, ser. EuroSys '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 34–50. [Online]. Available: https://doi.org/10.1145/3492321.3519594
[14] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: Dag bft protocols made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2705–2718. [Online]. Available: https://doi.org/10.1145/3548606.3559361
[15] (2021) narwhal. [Online]. Available: https://github.com/facebookresearch/narwhal
[16] (2023) Hyperledger caliper. [Online]. Available: https://hyperledger.github.io/caliper/
[17] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *Financial Cryptography and Data Security*, R. Böhme and T. Okamoto, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 507–527.
[18] X. Wu, J. Chang, H. Ling, and X. Feng, "Scaling proof-of-authority protocol to improve performance and security," *Peer-to-Peer Networking and Applications*, vol. 15, no. 6, pp. 2633–2649, 2022.
[19] P. Ekparinya, V. Gramoli, and G. Jourjon, "The attack of the clones against proof-of-authority," *arXiv preprint arXiv:1902.10244*, 2019.
[20] S. De Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, V. Sassone *et al.*, "Pbft vs proof-of-authority: Applying the cap theorem to permissioned blockchain," in *CEUR workshop proceedings*, vol. 2058. CEUR-WS, 2018.
[21] (2023) go-ethereum. [Online]. Available: https://github.com/ethereum/go-ethereum/commit/81d328a73e00454912edf79e74f7d041467fa2aa
[22] (2023) A native grpc client & server implementation with async/await support. [Online]. Available: https://github.com/hyperium/tonic
[23] (2021) A processor task of narwhal. [Online]. Available: https://github.com/facebookresearch/narwhal/blob/main/worker/src/processor.rs
[24] (2021) An analyze function of narwhal. [Online]. Available: https://github.com/facebookresearch/narwhal/blob/main/node/src/main.rs#L137
[25] (2019) simple smart contract. [Online]. Available: https://github.com/hyperledger/caliper-benchmarks/blob/main/src/ethereum/simple/simple.sol/