FairDAG: Consensus Fairness over Multi-Proposer Causal Design

Dakai Kang, Junchao Chen, Tien Tuan Anh Dinh†, Mohammad Sadoghi
Exploratory Systems Lab, University of California, Davis
†Deakin University

ABSTRACT

The rise of cryptocurrencies like Bitcoin and Ethereum has driven interest in blockchain database technology, with smart contracts enabling the growth of decentralized finance (DeFi). However, research has shown that adversaries exploit transaction ordering to extract profits through attacks like front-running, sandwich attacks, and liquidation manipulation. This issue affects blockchains where block proposers have full control over transaction ordering. To address this, a more fair transaction ordering mechanism is essential.

Existing fairness protocols, such as Pompe and Themis, operate on leader-based consensus protocols, which not only suffer from low throughput caused by single-leader bottleneck, but also give adversarial block proposers to manipulate transaction ordering. To address these limitations, we propose a new framework FairDAG that runs fairness protocols on top of DAG-based consensus protocols, which improves protocol performance in both throughput and fairness quality, leveraging the multi-proposer design and validity property of DAG-based consensus protocols.

We conducted a comprehensive analytical and experimental evaluation of two FairDAG variants—FairDAG-AB and FairDAG-RL. Our results demonstrate that FairDAG outperforms prior fairness protocols in both throughput and fairness quality.

PVLDB Reference Format:
Dakai Kang, Junchao Chen, Tien Tuan Anh Dinh, Mohammad Sadoghi. FairDAG: Consensus Fairness over Multi-Proposer Causal Design. PVLDB, 19(2): 265-278, 2025.
doi:10.14778/3773749.3773763

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/apache/incubator-resilientdb/tree/fairdag.

1 INTRODUCTION

The emergence of cryptocurrencies, including Bitcoin [54] and Ethereum [16], has sparked broad interest in blockchain database technology [1, 10, 13, 72]. Blockchain enables a new class of applications, namely decentralized finance (DeFi) [9, 14, 35, 61, 76], whose market capitalization exceeds 70 billion. DeFi requires consistency and fairness in transaction ordering. The former ensures that all participants agree on the same transaction order, which has been addressed under crash-failure settings of traditional distributed databases, for example [3, 36, 48, 60, 68], and under Byzantine-failure settings in blockchains and verifiable databases [16, 55, 78] where Byzantine participants can have arbitrary malicious behavior. In the presence of Byzantine participants, even though transaction ordering is consistent, its fairness remains vulnerable to ordering manipulation attacks. Such an "order manipulation crisis" is possible because block proposers have full control over transaction selection and ordering. Byzantine proposers can censor or reorder transactions to extract *Maximal Extractable Value (MEV)* from blocks [23, 37, 50, 66, 67, 70, 73, 77], which is unfair to other participants. Attacks include front-running, back-running, sandwich attacks, liquidation manipulation, and time-bandit attacks [23, 57–59, 69].

The key to achieving fair transaction ordering lies in preventing proposers from dominating the ordering. Existing studies [18, 22, 44, 46, 52, 53, 79] have proposed various fairness protocols. Unlike traditional protocols where each block contains a list of transactions, in fairness protocols, each block contains local orderings from a set of participants. Once blocks are committed, a final transaction ordering is derived from the local orderings. Different fairness protocols guarantee different fairness properties, reflecting the preferences of correct participants who honestly report the order in which they receive transactions. For example, Pompe [79] calculates the *assigned ordering indicator* for each transaction and orders transactions based on it. It guarantees that transaction T_1 is ordered before T_2 if every correct participant receives T_1 before any correct participant receives T_2, a property named *Ordering Linearizability*. Themis [44] constructs a dependency graph among transactions and determines the ordering according to the edges of the graph. It guarantees that if a γ proportion of correct participants receive T_1 before T_2, then T_1 will be ordered *no later* than T_2 -a property named *γ-Batch-Order-Fairness*.

We observe that a well-designed fairness protocol should achieve the following goals:

- G1 **Resilience to Ordering Manipulation.** The protocol should limit Byzantine participants' influence on the final transaction order to preserve fairness properties.
- G2 **Minimal Correct Participants Requirement.** The protocol should preserve fairness properties while relying on as few correct participants as possible.
- G3 **High Performance.** Fairness protocols should minimize the overhead introduced by fair ordering, achieving high throughput and low latency.

Unfortunately, previous fairness protocols [18, 44–46, 53, 79] leverage leader-based consensus protocols [20, 30, 74], relying on a single leader to collect local orderings from other participants. A Byzantine leader can manipulate transaction ordering by selectively collecting local orderings to maximize its *MEV*. Moreover, even without Byzantine behavior, a single leader may become a

Proceedings of the VLDB Endowment, Vol. 19, No. 2 ISSN 2150-8097.
doi:10.14778/3773749.3773763

performance bottleneck when the workload exceeds its capacity, as the message complexity of broadcasting collected local orderings grows quadratically with the number of participants.

To address the challenges posed by the underlying leader-based consensus protocols, we propose FairDAG, a novel framework that runs fairness protocols on top of DAG-based consensus protocols [6, 8, 24, 43, 62, 63], which provide the following features that can enhance ordering fairness:

- **Multi-Proposer High-throughput Design**: DAG-based protocols allow all participants to propose blocks in parallel. This approach improves system performance by alleviating the bottleneck introduced by a single leader.
- **Validity through Causal Design**: Blocks in DAG-based protocols reference blocks from other participants, forming a *Directed Acyclic Graph (DAG)*. The causal relationship in DAG guarantees that vertices from correct participants are eventually committed by all correct participants.

The *Multi-Proposer High-throughput Design* removes the bottleneck caused by a single leader (**G3**). The *validity* constrains the Byzantine participants' ability to selectively collect local orderings (**G1**), and thus lowers the requirement on the number of correct participants (**G2**).

In FairDAG, each participant proposes its block containing the local ordering as a DAG vertex and reliably broadcasts it, and a final transaction ordering is deterministically generated based on the vertices committed by the underlying DAG-based consensus protocols. We designed two variants of FairDAG, namely *absolute-ordering* FairDAG-AB and *relative-ordering* FairDAG-RL.

There are two main challenges in employing DAG-based protocols as the underlying consensus layer for fairness. First, to mitigate the high latency of DAG-based consensus protocols, participants should leverage uncommitted DAG vertices to reduce the latency for fairness decisions, but participants may hold inconsistent views of uncommitted vertices. Second, Byzantine participants may attempt to manipulate the ordering by selectively ignoring vertices proposed by specific participants. FairDAG addresses these challenges through a novel ordering indicator manager, adaptive fairness thresholds, and new DAG construction rules that collectively ensure fairness.

We make the following contributions:

(1) We propose FairDAG-AB, a *absolute* fairness protocol that guarantees *Ordering Linearizability*. We propose a *Ordering Indicator Manager* and an adaptive fairness threshold called *LPAOI* that are compatible with the multi-proposer design and commit rules of the DAG-based consensus protocols.
(2) We propose FairDAG-RL, a *relative* fairness protocol that guarantees γ*-batch-order-fairness*. Leveraging the validity property of DAG-based consensus protocols, we adopt new thresholds for dependency graph construction to improve system performance and reduce the requirement of minimal correct participant number.
(3) In FairDAG, we apply new rules for constructing a DAG to guarantee fairness against adversarial participants.

$R_1 : \{T_1, T_2, T_3, T_4\}$
$R_2 : \{T_2, T_3, T_4, T_1\}$
$R_3 : \{T_3, T_4, T_1, T_2\}$
$R_4 : \{T_4, T_1, T_2, T_3\}$

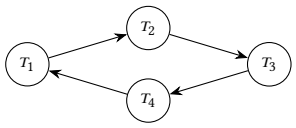


Figure 1: Condorcet Cycle

(4) We conducted comprehensive analytical and experimental evaluation of our protocols. The results show that: compared to Pompe and Themis, FairDAG-AB and FairDAG-RL outperform in both throughput and fairness quality, which reflects the fairness protocols' resilience against adversarial ordering manipulations.

This paper is structured as follows. Section 2 presents the background. Section 3 introduces the system model. Section 4 presents an overview of FairDAG protocols. Section 5 and Section 6 describe the details of FairDAG-AB and FairDAG-RL[1]. Section 7 analytically compares FairDAG with prior fairness protocols Pompe and Themis. Section 8 presents the experimental evaluation of FairDAG and baseline protocols. Section 9 discusses other related works, and Section 10 concludes.

2 BACKGROUND

FairDAG-AB and FairDAG-RL execute fairness protocols atop DAG-based consensus protocols. Beginning with this section, our discussion is framed within the context of *Byzantine Fault Tolerant (BFT)* protocols, where participants are referred to as replicas. *Byzantine replicas*, corrupted by an adversary, may exhibit arbitrary malicious behavior, whereas the remaining *correct replicas* behave benignly. In this section, we present the definitions of three fairness properties and introduce the DAG-based consensus protocols.

2.1 Receive-Order-Fairness

Definition 2.1. **Receive-Order-Fairness.** For any two transactions T_1 and T_2, if all correct replicas receive T_1 before T_2, then T_1 must be ordered *before* T_2 in the final ordering.

We show that it is impossible to always guarantee *Receive-Order-Fairness* in the presence of Byzantine replicas for *Condorcet Cycles*. Figure 1 illustrates this impossibility with a concrete example. Consider four replicas R_1, R_2, R_3, R_4, among which at most one may be Byzantine. Let there be four transactions T_1, T_2, T_3, T_4. As shown on the left side of the figure, for any two commands T_i and T_{i+1} (modulo 4) , three replicas receive T_i before T_{i+1} but the fourth differs. Since the identity of the Byzantine replica is unknown, we respect all majority-endorsed orderings supported by at least three replicas when determining the final ordering. The right side of Figure 1 depicts a directed graph where an edge $T_i \rightarrow T_j$ indicates that at least three replicas received T_i before T_j. The resulting graph contains a Condorcet Cycle [12, 21]—a cycle of pairwise preferences that cannot be linearly extended without violating at least one of them. Hence, the *Receive-Order-Fairness* is impossible in this case.

[1] see Section 8 in our extended report [39] for correctness proofs

R_1 : $\{(T_2, 1), (T_1, 2), (T_4, 3), (T_3, 4)\}$
R_2 : $\{(T_1, 1), (T_3, 2), (T_2, 3), (T_4, 4)\}$
R_3 : $\{(T_1, 1), (T_2, 1), (T_3, 3), (T_4, 4)\}$
R_4 : $\{(T_1, 1), (T_2, 2), (T_3, 3), (T_4, 4)\}$

Figure 2: T_1 will be ordered before T_4 if *Ordering Linearizability* holds regardless of the local ordering from R_4.

2.2 Ordering Linearizability

Ordering Linearizability is a fairness property introduced by Pompe [79] and adopted by our protocol, FairDAG-AB. To achieve this property, each replica assigns a monotonically increasing *ordering indicator* (denoted oi) to each transaction reflecting the order it receives the transactions. The final ordering is then derived from ordering indicators collected from a majority of replicas.

Denoting by ois_i^C the set of ordering indicators for the transaction T_i from the correct replicas, we define *Ordering Linearizability*:

Definition 2.2. **Ordering Linearizability.** For any two transactions T_1 and T_2, if all ordering indicators in ois_1^C are smaller than all those in ois_2^C, i.e.,

$$\forall oi_1 \in ois_1^C, \forall oi_2 \in ois_2^C : oi_1 < oi_2.$$

then T_1 must be ordered before T_2 in the final ordering.

Figure 2 illustrates this definition with an example involving four replicas, where replicas R_1, R_2, R_3 are correct, and R_4 is Byzantine. Each replica assigns local ordering indicators to four transactions. For transactions T_1 and T_4, the correct replicas assign $ois_1^C = \{2, 1, 1\}$ and $ois_4^C = \{3, 4, 4\}$, respectively. Since all indicators in ois_1^C are smaller than those in ois_4^C, any protocol satisfying Ordering Linearizability—such as Pompe and FairDAG-AB—must place T_1 before T_4 in the final ordering, regardless of the Byzantine replica's input.

2.3 γ-Batch-Order-Fairness

In Section 2.1, we showed that it is impossible to guarantee *Receive-Order-Fairness* in the presence of unknown Byzantine replicas. However, a weaker yet practical variant, *γ-Batch-Order-Fairness*, can be achieved by both Themis [44] and our protocol FairDAG-RL.

We say that a transaction T_2 is **dependent** on transaction T_1, denoted as $T_1 \rightarrow T_2$, if T_1 must be ordered before T_2 in the final ordering. Due to the presence of *Condorcet cycles*, such dependencies can form cycles among transactions.

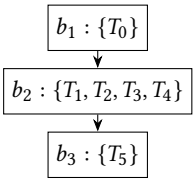
Definition 2.3. A batch S of transactions is *cyclic dependent* if for any two transactions T_1, T_2 in S, there is a list of transactions that form a dependency path from T_1 to T_2.

The final ordering can be partitioned into a sequence of non-overlapping batches $S_1, S_2, \ldots$, where each S_i is a *maximal cyclically dependent batch*, i.e., for any i, $S_i \cup S_{i+1}$ is not cyclically dependent.

We say that T_1 is ordered **no later than** T_2 if T_1 is in the same or an earlier batch than T_2. And we define *γ-Batch-Order-Fairness* as follows:

Definition 2.4. **γ-Batch-Order-Fairness.** For any two transactions T_1 and T_2, if at least a fraction γ of correct replicas receive T_1 before T_2, then T_1 must be ordered *no later than* T_2 in the final ordering.

$R_1 : \{T_0, T_1, T_2, T_3, T_4, T_5\}$
$R_2 : \{T_0, T_2, T_3, T_4, T_1, T_5\}$
$R_3 : \{T_0, T_3, T_4, T_1, T_2, T_5\}$
$R_4 : \{T_0, T_4, T_1, T_2, T_3, T_5\}$
Final Ordering: $\{T_0, T_1, T_2, T_3, T_4, T_5\}$
(a)



(b)

Figure 3: A final ordering of six transactions that satisfies γ-Batch-Order-Fairness with $\gamma = \frac{2}{3}$, 3 correct replicas, and 1 Byzantine replica.

Figure 3 presents an example satisfying *γ-Batch-Order-Fairness*, where we have $\mathbf{n} = 4$ replicas (at most $\mathbf{f} = 1$ replica can be Byzantine) and 6 transactions. Assuming that the fairness protocol generates a final ordering that can be split into three *cyclic dependent batches*, for any two transactions, *γ-Batch-Order-Fairness* holds. For example, T_0 is received earlier than T_1 by correct replicas of $\gamma(\mathbf{n}-\mathbf{f}) = 3$ and T_0 is ordered in a batch earlier than T_1.

2.4 DAG-based Consensus Protocols

DAG-based BFT consensus protocols [24, 43, 63] operate in rounds. In each round r, every replica proposes a block, referred to as a DAG vertex. Each vertex references multiple vertices from the previous round $r-1$, represented as edges that encode the causal dependencies between DAG vertices, forming a *Directed Acyclic Graph (DAG)*. The *causal history* of a DAG vertex includes all vertices reachable via the reference paths.

Every k rounds (e.g. $k = 2$ in Tusk [24]), a random or predetermined leader vertex is elected. These leader vertices are committed in an ascending order of round, and their causal histories are committed in a deterministic order. To ensure reliable dissemination, most DAG-based protocols employ reliable broadcast (RBC) mechanisms. With RBC and carefully designed commit rules, DAG-based protocols guarantee the following properties even in an *asynchronous network* without message delay bound:

- **Agreement**: If a correct replica commits a vertex v, then all correct replicas eventually commit v.
- **Total Order**: If a correct replica commits v before v', then every correct replica commits v before v'.
- **Validity**: If a correct replica broadcasts a vertex v, then all correct replicas eventually commit v.

Compared to consensus protocols with a single leader, the multi-proposer design of DAG-based protocols enables higher throughput. But this comes at the cost of higher commit latency due to the overhead of RBC and multi-round commit rules.

3 SYSTEM MODEL

We consider a distributed system consisting of a set of replicas and a potentially unbounded number of clients. The system is subject to Byzantine faults and operates under either asynchronous

or partially synchronous network conditions. Our model covers client behavior, replica corruption, authentication assumptions, and fairness-specific threat considerations.

3.1 Clients

Clients issue transactions to replicas and wait for execution results. They may behave arbitrarily with no correctness assumptions.

3.2 Replicas

In the system, there are a total of $\mathbf{n}$ replicas and an *adaptive adversary* capable of corrupting up to $\mathbf{f}$ replicas during execution. The corrupted replicas, referred to as Byzantine or malicious replicas, may exhibit arbitrary malicious behavior.

Regarding transaction ordering, Byzantine replicas may reorder transactions in local orderings and ignore unfavorable local orderings from other replicas. Correct replicas are honest about local orderings in which the transactions are received.

Fairness protocols differ in resilience to Byzantine faults. Specifically, FairDAG-AB and Pompe require $\mathbf{n} > 3\mathbf{f}$; Themis requires $\mathbf{n} > \frac{(2\gamma+2)\mathbf{f}}{2\gamma-1}$; and FairDAG-RL requires $\mathbf{n} > \frac{(2\gamma+1)\mathbf{f}}{2\gamma-1}$, $\frac{1}{2} < \gamma \leq 1$.

3.3 Authentication

We assume *authenticated communication*, where Byzantine replicas cannot forge messages from correct replicas. Authentication is enforced through *Public Key Infrastructure (PKI)* [42].

For integrity verification, each transaction T_i is associated with a digest d_i, computed using a *secure collision-resistant cryptographic hash function* [42].

3.4 Network

External network (client–replica). We make no assumptions about synchrony but assume the external network is *non-adversarial*. This assumption is necessary for preserving fairness, as an adversarial external network could arbitrarily control the order in which replicas receive transactions and then the final ordering.

Internal network (replica–replica). The internal network may operate either *asynchronously* or *partially synchronously*:

- **Asynchronous network:** Messages are never lost and are eventually delivered, but there is no bound on the message delays.
- **Partially synchronous network:** There exists an unknown *Global Stabilization Time (GST)* after which the message delays are bounded by a known constant Δ, i.e., any message sent at time t will be delivered by $\max(t, \text{GST}) + \Delta$.

4 OVERVIEW

Previous fairness protocols, such as Pompe [79] and Themis [44], are built atop leader-based consensus. Although the final ordering is derived from local orderings collected from a majority of the replicas, existing fairness protocols have these problems: (1) Byzantine leaders manipulate orderings and compromise fairness by selectively picking up the local orderings. (2) Byzantine or slow leaders can become system performance bottlenecks. To address these problems, FairDAG runs fairness protocols on top of multi-proposer DAG-based consensus with two layers:

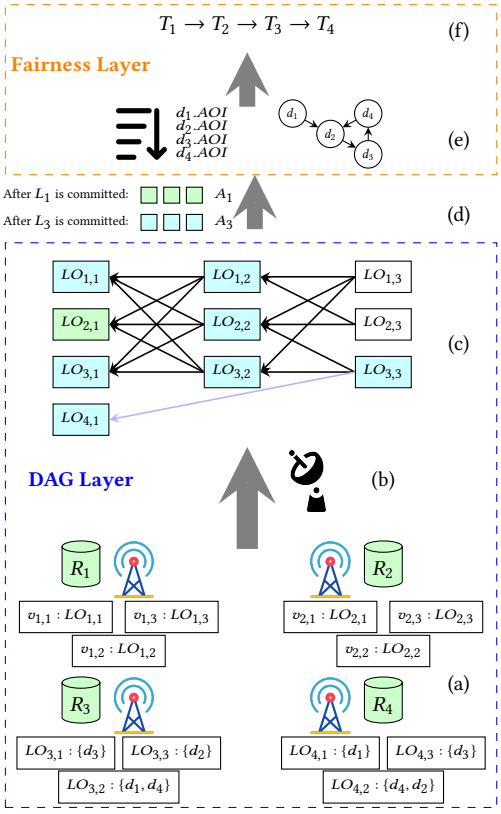


Figure 4: Architecture of FairDAG: (a) Replicas reliably broadcast blocks containing their local ordering fragments. (b) Each replica receives blocks delivered through Reliable Broadcast. (c) Each replica forms a local view of the DAG using received blocks and reference links, where different colors represent the subdags of different committed leader vertices. (d) Local orderings in subdag A_r are used as input of the fairness layer after L_r is committed. (e) Finalize transaction ordering using absolute ordering mechanism (left) or relative ordering mechanism (right), based on the committed local orderings. (f) A final transaction ordering is generated.

- *DAG Layer* handles the dissemination (Figure 4(a,b)) and commitment (Figure 4(c)) of local orderings.
- *Fairness Layer* takes these committed local orderings (Figure 4(d)) as input to a transaction ordering algorithm (Figure 4(e)) to produce a final ordering (Figure 4(f)).

4.1 DAG Layer

The DAG layer adopts existing DAG-based consensus protocols—such as Tusk [24], DAG-Rider [43], and Bullshark [63]—with minimal modifications to support fairness guarantees.

Directed-Acylic-Graph

DAG-based protocols proceed in rounds where each replica concurrently proposes one vertex per round r. Each vertex references

previous vertices, forming a DAG, where references are **edges**. Specifically, a vertex in round r references at least $\mathbf{n}-\mathbf{f}$ vertices from round $r-1$ via **strong edges**, and optionally up to $\mathbf{f}$ vertices from earlier rounds via **weak edges**.

Let $v_{i,r}$ denote the vertex proposed by replica R_i in round r. A valid vertex satisfies:

- *strong_edges*: includes at least $\mathbf{n}-\mathbf{f}$ *strong edges*,
- *weak_edges*: includes up to $\mathbf{f}$ *weak edges*.

To guarantee the fairness properties in the presence of adversary, we extend existing DAG-based protocols by applying new rules of forming DAG vertices:

(1) In FairDAG, replicas include weak edges in vertices to incorporate local orderings from slower replicas, ensuring that all correct local orderings eventually get committed.
(2) To preserve the integrity of a replica's local ordering, each vertex $v_{i,r}$ must include a strong edge to its vertex in the previous round $v_{i,r-1}$ (for $r > 0$).

Figure 4(c) shows an DAG example where black arrows represent strong edges, and blue arrows represent weak edges.

Reliably broadcasting a vertex

To address issue (1), in FairDAG, instead of relying on a leader to collect local orderings, each replica R_i autonomously reliably broadcast [11, 19, 24, 29] its local ordering LO_i. As Figure 4(a) shows, FairDAG vertex $v_{i,r}$ contains r-th slice of the replica R_i's *local ordering* ($LO_{i,r}$)—a sequence reflecting the order in which transactions were received by replica R_i. Each local ordering is represented as a sequence of transaction digests paired with monotonically increasing ordering indicators.

Constructing a DAG

As illustrated in Figure 4 (a,b), the DAG layer of FairDAG enables replicas to reliably broadcast and deliver vertices from one another. This multi-proposer design alleviates the performance bottlenecks associated with a single leader, thereby addressing issue (2).

Using the delivered vertices and the references within them, each replica constructs its local view of the DAG, as shown in Figure 4(c). As mentioned in Section 2.4, the DAG protocols guarantee that all correct replicas eventually have consistent DAG views.

Committing DAG vertices

The commit rules in DAG-based consensus protocols—leveraging the causal dependencies between vertices—further mitigate leader manipulation in selecting local orderings, addressing issue (1).

DAG protocols group rounds into *waves*, each containing one or more leader vertices. A leader vertex L_{r_i} in round r_i is *committed* once specific conditions are met. Let C_{r_i} denote its *causal history*—the set of vertices reachable from L_{r_i}, including L_{r_i} itself. DAG protocols ensure that all correct replicas commit leader vertices in a consistent, round-increasing order $(L_{r_1}, L_{r_2}, \dots)$ such that $C_{r_i} \subset C_{r_{i+1}}$ for all i.

A vertex v is said to be in the subdag of leader L_{r_i} if $v \in C_{r_i}$ and $v \notin C_{r_j}$ for all $j < i$. Let A_{r_i} denote the vertices in the subdag of L_{r_i}. The committed DAG can thus be partitioned into non-overlapping subdags $(A_{r_1}, A_{r_2}, \dots)$, each corresponding to a committed leader.

For example, in Figure 4(c), L_1 is the green vertex $v_{2,1}$ and L_3 is the blue vertex $v_{3,3}$. Then A_1 consists of the green vertices, and A_3 consists of the blue ones.

As shown in Figure 4(d), each time a leader L_r is committed, the fairness layer processes A_r to compute the final ordering.

The DAG layer satisfies a necessary condition to guarantee that final ordering aligns with the preferences of the majority of correct replicas, as each subdag A_r aggregates local orderings from at least $\mathbf{n}-\mathbf{f}$ replicas—guaranteeing the inclusion of at least $\mathbf{n}-2\mathbf{f}$ correct replicas. (Note: $\mathbf{n}-2\mathbf{f} \geq \frac{\mathbf{n}-\mathbf{f}}{2}$ holds for all protocols.)

4.2 Fairness Layer

Taking local orderings within the committed subdags as input, the fairness layer runs a transaction ordering algorithm to calculate the final transaction ordering.

In **absolute** fairness protocols (Figure 4(e), left), each transaction gets an *assigned ordering indicator (AOI) based on the input local orderings*. The final ordering (Figure 4(f)) is then derived, sorting transactions by their *AOI*. FairDAG-AB is an *absolute* fairness protocols and satisfies the *Ordering Linearizability* property.

In **relative** fairness protocols (Figure 4(e), right) construct a dependency graph of transaction nodes, where edges encode pairwise ordering dependencies derived from the input local orderings. The final ordering is a *Hamiltonian path* within the graph (Figure 4(f)). FairDAG-RL is an *relative* fairness protocol and satisfies the *γ-Batch-Order-Fairness* property.

5 FAIRDAG-AB

FairDAG-AB is an **absolute** fairness protocol. In this section, we demonstrate how FairDAG-AB calculates the *assigned ordering indicator (AOI)* for each transaction and orders the transactions based on the AOI values.

5.1 Transaction Dissemination

To prevent leader-driven manipulation, clients broadcast transactions to all replicas. This prevents adversarial replicas from censoring transactions to influence their position in final ordering.

Upon receiving a client transaction T, replica R gives T a monotonically increasing ordering indicator oi from its local timer, appends digest d of T and oi to its lists dgs and ois (Figure 5, Line 4), respectively, which are in-memory variables storing transaction digests and ordering indicators of pending transactions (Lines 9-12).

5.2 FairDAG-AB Vertex

Built atop a DAG-based consensus protocol, each FairDAG-AB replica constructs and reliably broadcasts a DAG vertex when protocol conditions are met. In addition to DAG-specific metadata (as described in Section 4.1), a FairDAG-AB vertex includes (and clears) the replica's local ordering of pending transactions it received after it broadcast the last vertex, encoded in dgs, ois (Lines 13-17).

5.3 Managing and Assigning Ordering Indicators

FairDAG-AB leverages local orderings within committed DAG vertices to determine the final transaction ordering. However, a delay exists between receiving and committing the DAG vertices. To efficiently manage and utilize local orderings from both committed and uncommitted vertices, FairDAG-AB maintains an *Ordering*

```
1:  State Variables (per replica)
2:  txns_w_assigned_oi := {}
3:  highest_oi_list[1..n] := 0
4:  dgs, ois := []
5:  current_round, replica_id

6:  Ordering Indicator Manager (OIM)
7:  seen_ois[1..n] := ∞, committed_ois[1..n] := ∞
8:  LPAOI := ∞, AOI := ∞

    Client Thread (processing client transactions) :
9:  event On receive transaction T do
10:     if T is valid then
11:         oi := local_timer
12:         dgs.append(T.digest); ois.append(oi)

    DAG Layer Thread (constructing the DAG) :
13: event On propose DAG vertex do
14:     v := DAGVertex(replica_id, current_round)
15:     Add pending local ordering dgs and ois into v
16:     Clear dgs and ois
17:     Reliably broadcast v

18: event On deliver v_{i,r} do
19:     highest_oi_list[i] := the highest oi in v_{i,r}.ois
20:     for (d, oi) ∈ (v_{i,r}.dgs, v_{i,r}.ois) do
21:         OIM(d).seen_ois[i] := oi

22: event On commit L_r do
23:     Send A_r to Fairness Layer

    Fairness Layer Thread (Ordering transactions) :
24: ▷Update committed_ois
25: event On receive A_r do
26:     for v ∈ A_r, (d, oi) ∈ (v.dgs, v.ois) do
27:         OIM(d).committed_ois[v.replica_id] := oi
28:     LPAOI_min := ∞
29:     for d such that OIM(d) = ∞ do
30:         if OIM(d) has at least n − f committed_ois then
31:             ▷Calculate AOI
32:             OIM(d).AOI := (f+1)-th smallest in committed_ois
33:             Add d to txns_w_assigned_oi
34:         else
35:             ▷Calculate LPAOI
36:             for i = 1 to n do
37:                 lp_ois[i] := min(d.seen_ois[i], highest_oi_list[i])
38:             d.LPAOI := (f+1)-th smallest in lp_ois
39:             ▷Track the minimal LPAOI of all transactions without an AOI
40:             LPAOI_min := min(LPAOI_min, d.LPAOI)
41:     sort txns_w_assigned_oi sorted by AOI
42:     execute all transactions with an AOI lower than LPAOI_min
```

Figure 5: FairDAG-AB Algorithm.

Indicator Manager ($OIM(d)$) for each transaction digest d, which tracks the replica's local view of the DAG and ordering indicators within the vertices.

We say that a replica R has ***seen*** an ordering indicator oi if oi is in a vertex in R's local DAG view. R has ***committed*** oi if oi is in a committed vertex. $OIM(d)$ contains the following information (Lines 6-8):

- *seen_ois*: ordering indicators of d that R has seen.
- *committed_ois*: ordering indicators of d that R has committed.
- *LPAOI*: the lowest possible value of the assigned ordering indicator of d.
- *AOI*: the value of the assigned ordering indicator of d.

The final ordering is determined using *AOI* values derived from *committed_ois*. *LPAOI*, computed from *seen_ois*, helps determine when it is safe to decide the position of a transaction in the final ordering (see Section 5.4). The *seen_ois* and *committed_ois* are indexed from 1 to **n** and each item is initialized to ∞. For example, if $OIM(d).seen\_ois[2] = 3$, it means that R has seen an ordering indicator 3 for the transaction with digest d from replica R_2.

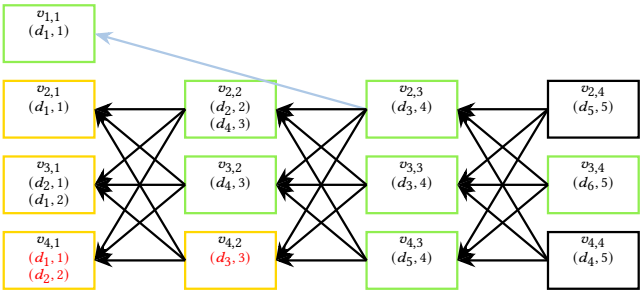


Figure 6: Example of calculating AOI and LPAOI.

Managing Ordering Indicators

Besides the *OIM* of each transaction digest d, each FairDAG-AB replica R also maintains *highest_ois*: a vector of the highest ordering indicators received from each replica, initialized to be 0 and indexed from 1 to **n** (Line 3). This vector is essential for checking whether it is safe to determine the position of a transaction in final ordering (See details in Section 5.4).

Upon a DAG vertex $v_{i,r}$ is delivered and added to replica R's local DAG view, R updates $highest\_ois[i]$ to the highest ordering indicator in $v_{i,r}$. For each digest d in $v_{i,r}$, R updates $OIM(d).seen\_ois[i]$ (Lines 18-21).

Upon a DAG leader vertex L_r is committed, its subdag A_r is input to the fairness layer (Lines 22-23). And R updates *committed_ois* for the digests in A_r accordingly (Lines 25-27).

Calculating Assigned Ordering Indicator

If $OIM(d)$ has at least **n**−**f** non-∞ *committed_oi* values R calculates $OIM(d).AOI$. The *AOI* is the (**f**+1)-th smallest value of the *committed_ois*, which definitely falls within the range of ordering indicators from correct replicas. Then, d is added to *txns_w_assigned_oi*, which is set of transactions digests with valid *AOI* values (Lines 29-33). *Note: The value of AOI is immutable once calculated, even if more ordering indicators of d are committed.*

Example 5.1. Figure 6 illustrates how to calculate *AOI* values. Suppose $v_{4,2}$ and $v_{3,4}$ are committed leader vertices L_2 and L_4, with yellow A_2 and green A_4, respectively. In A_2, digest d_1 has committed ordering indicators ∞, 1, 2, 1. Since this includes at least **n**−**f** valid

values, its *AOI* is the (**f**+1)-th lowest, which is 1 (Lines 29-33). Although an additional *seen_ois* of 1 appears in $v_{1,1} \in A_4$, it is ignored since *AOI* is immutable once calculated after L_2 is committed. In A_4, digest d_2 has committed ordering indicators $\infty, 1, 2, 3$, yielding an *AOI* of 2. Similarly, d_3 receives an *AOI* of 3. Other digests in A_4 lack sufficient *committed_ois* data for *AOI*.

5.4 Global Ordering

Due to the randomness in message arrival times, DAG vertices with smaller local ordering indicators may be committed later than vertices from other replicas with higher ordering indicators. As a result, a transaction d_1 may get a smaller *AOI* than d_2, even if d_1 obtains its *AOI* in a later round than d_2. To ensure strict ordering of transactions based on their *AOI* values—which is essential for achieving *Ordering Linearizability*—it is necessary to guarantee that no other transaction could get a lower *AOI* before determining the position of a transaction in the final ordering. To enforce it, for each transaction without an *AOI*, we calculate its *LPAOI*, the lowest AOI value it could possibly get. And we track $LPAOI_{min}$, the minimal *LPAOI* of transactions without an *AOI*.

Calculating LPAOI

For each transaction digest d without a valid *AOI*, replica R constructs a vector lp_ois, where each entry is computed as $lp\_ois[i] = \min(OIM(d).seen\_ois[i], highest\_ois[i])$. The *LPAOI* of d is then defined as the (**f**+1)-th smallest value in lp_ois. And $LPAOI_{\min}$ is the minimal value across all *LPAOI* values (Lines 34-40).

$LPAOI_{\min}$ serves as a threshold: only transactions with $AOI < LPAOI_{\min}$ can determine its position in final ordering. The fairness layer sorts all digests in *txns_w_assigned_oi* by *AOI*, executes all transactions with an *AOI* lower than the threshold. The ordered digests are then removed from *txns_w_assigned_oi* (Lines 41-42).

Example 5.2. As shown in Figure 6, transaction digests d_1, d_2, and d_3 have *AOI* values of 1, 2, and 4, respectively. Given $highest\_ois = (2, 6, 6, 6)$ and $OIM(d_4).seen\_ois = (\infty, 3, 3, 5)$, digest d_4 derives its $lp\_ois = (2, 3, 3, 5)$ and obtains $LPAOI = 3$. Similarly, d_5 and d_6 have *LPAOI* values of 4 and 5. $LPAOI_{\min}$ is then 3, (Lines 34-40) so d_1 and d_2 can be ordered and executed. In contrast, d_3 cannot be executed since its *AOI* exceeds $LPAOI_{\min}$, implying that d_4 could possibly get a lower *AOI* than d_3.

6 FAIRDAG-RL

FairDAG-RL is a **relative** fairness protocol. In this section, we demonstrate how to construct dependency graphs between transactions and derive a final transaction ordering from the dependency graphs.

6.1 Transaction Dissemination

As in FairDAG-AB, clients broadcast transactions to all replicas for censorship resistance. Upon meeting the conditions in DAG protocols, each replica broadcasts a vertex. A FairDAG-RL vertex is a restricted form of a FairDAG-AB vertex: it includes a sequence of incrementing counter values as ordering indicators corresponding to received transactions (Figure 7 Lines 7-8). For simplicity, we omit the ordering indicators in Figure 9.

```
1:  State Variables
2:  graphs := [ ]
3:  current_round, replica_id
4:  counter := 0

    Client Thread (processing transactions from clients) :
5:  event On receive a transaction T do
6:      if T is valid then
7:          counter := counter + 1
8:          ois.append(counter),   dgs.append(T.digest)

    DAG Layer Thread (forming the DAG) :
9:  event On propose vertex do
10:     v := DAGVertex(replica_id, current_round)
11:     Add pending local ordering dgs and ois into v
12:     Clear dgs and ois
13:     Reliably broadcast v

14: event On commit(L_r) do
15:     Send A_r to Ordering Layer
```

Figure 7: FairDAG-RL: Transaction dissemination and DAG vertex proposal.

The local orderings in subdag A_r will be input to the fairness layer after L_r is committed in FairDAG-RL (Lines 14-15).

6.2 Dependency Graph Construction

The fairness layer of FairDAG-RL utilizes local orderings in committed DAG vertices to construct dependency graphs that reflect the ordering preferences of replicas. Every time a leader vertex is committed, a new dependency graph is constructed:

First, transaction digests from A_r are added as graph nodes if it has not been added to any dependency graph is previously rounds. Each transaction digest d is associated with a node that stores:

- *type*: the type of the transaction node (See details later in **Adding nodes**)
- *committed_ois*: a vector of committed ordering indicators for d.
- *committed_rounds*: the rounds in which the corresponding *commited_oi* is committed.
- G: the graph to which the node is added.

Second, pairwise ordering preferences are aggregated into a weight function: $weight(d_1, d_2)$ denotes the number of replicas who have a lower *committed_oi* for d_1 than d_2. Each dependency graph contains the following information:

- *nodes*: the set of transaction digest nodes.
- *weight* : mapping of the pairs of nodes to their weights.
- *edges*: directed edges representing inferred ordering constraints between the transaction digest nodes.

Third, a directed edge from d_1 to d_2 is added if $weight(d_1, d_2)$ exceeds a quorum-based threshold $\frac{\mathbf{n}-\mathbf{f}}{2}$ and is not lower than $weight(d_2, d_1)$.

Thresholds

To construct a dependency graph that reflects the ordering preferences of the majority and mitigates manipulation by Byzantine

replicas, relative fairness protocols should leverage as many correct local orderings as possible. Prior protocols such as Themis [44] and Rashnu [53] rely on a single leader to collect local orderings, where a Byzantine leader may intentionally exclude local orderings from up to $\mathbf{f}$ correct replicas. As a result, a transaction may appear in at most $\mathbf{n}-2\mathbf{f}$ committed local orderings if Byzantine replicas ignore the transaction and the Byzantine leader excludes $\mathbf{f}$ correct local orderings, increasing both the susceptibility to ordering manipulation and the minimal requirement of number of correct replicas (see Section 7 for details).

In contrast, DAG-based consensus protocols ensure that all correct local orderings are eventually committed. This guarantees that each transaction can appear in at least $\mathbf{n}-\mathbf{f}$ committed local orderings. Consequently, FairDAG-AB raises the thresholds used in the construction of dependency graphs from $\mathbf{n}-2\mathbf{f}$ and $\frac{\mathbf{n}-2\mathbf{f}}{2}$ (used in Themis and Rashnu) to $\mathbf{n}-\mathbf{f}$ and $\frac{\mathbf{n}-\mathbf{f}}{2}$, respectively. We will elaborate on the details below.

Adding nodes

For each transaction digest d in A_r, let $node(d)$ represent its dependency graph node. Its *committed_ois* and *committed_rounds* are updated using information contained in A_r, and the node is added to *updated_nodes*, a set of transactions whose *committed_ois* are updated because of A_r (Figure 8 Lines 3-10) [2].

Then, we check if any transaction digests can be added as nodes into the new dependency graph of round r. We define $ap(d, r)$ as the number of ordering indicators for d that are committed by round r:

$$ap(d, r) := |\{i | node(d).committed\_rounds[i] \leq r\}|$$

Each node in *updated_nodes* whose type is *blank*, which means that it has not been added into any dependency graph previously, is classified as:

- *solid*, if $ap(d, r) \geq \mathbf{n}-\mathbf{f}$.
- *shaded*, if $\frac{\mathbf{n}-\mathbf{f}}{2} \leq ap(d, r) < \mathbf{n}-\mathbf{f}$.
- *blank*, if $ap(d, r) < \frac{\mathbf{n}-\mathbf{f}}{2}$.

The we add the *non-blank* nodes to G_r (Lines 11-18). Since classification is only applied to previously *blank* nodes, it is guaranteed that each digest is inserted into at most one dependency graph.

Updating weights between nodes

After classifying and adding nodes to G_r, we update edge weights based on local orderings in A_r. Vertices in A_r are processed in round-increasing order.

For each vertex v from replica R_i, we iterate through the transaction digests in $v.dgs$. For each digest d, we compare the value of $node(d).committed\_ois[i]$ with all other nodes in the graph G of $node(d)$. For each pair (d, d_2), we increment $G.weight[(d, d_2)]$ if $node(d).committed\_ois[i] < node(d_2).committed\_ois[i]$; otherwise, we increment $G.weight[(d_2, d)]$ (Lines 21-30).

During this process, we maintain a set *addable_edges* to track the edges that can be added to the dependency graph. If $G.weight[(d, d_2)]$ or $G.weight[(d_2, d)]$ reaches the threshold $\frac{\mathbf{n}-\mathbf{f}}{2}$, the corresponding pair is added to *addable_edges* (Lines 31-32).

[2] All operations related to dependency graph construction only apply to transactions that are not ordered yet. We omit this for simplicity in the protocol description and the pseudocode.

Fairness Layer Thread (Ordering Transactions) :

```
1:  event On receive A_r do
2:      G_r := NewGraph(),   graphs.push(G_r)
3:      ▷Find nodes updated with A_r
4:      updated_nodes := {}
5:      for v ∈ A_r do
6:          i := v.replica_id
7:          for (d, oi) ∈ (v.dgs, v.ois) do
8:              node(d).committed_ois[i] := oi
9:              node(d).committed_rounds[i] := r
10:             updated_nodes.insert(d)
11:     ▷Update node types and add new non-blank nodes to G_r
12:     for d ∈ updated_nodes do
13:         if node(d).type = blank then
14:             Let ap(d, r) be the number of ordering indicators for d that
                are committed by round r
15:             if ap(d, r) ≥ n − f then
16:                 node(d).type := solid;    G_r.nodes.add(node(d))
17:             else if ap(d, r) ≥ (n−f)/2 then
18:                 node(d).type := shaded;    G_r.nodes.add(node(d))
19:     ▷Find all candidate edges in all existing graphs G_r
20:     addable_edges := {}
21:     for v ∈ A_r do
22:         i := v.replica_id
23:         for (d, oi) ∈ (v.dgs, v.ois) do
24:             G' := node(d).G
25:             d_oi := node(d).committed_ois[i]
26:             for node(d_2) ∈ G'.nodes do
27:                 if d_oi < node(d_2).committed_ois[i] then
28:                     increment G'.weight[(d, d_2)]
29:                 else
30:                     increment G'.weight[(d_2, d)]
31:                 if either weight reaches threshold (n−f)/2 then
32:                     addable_edges.insert(d, d_2)
33:     ▷Add edge if weight reaches threshold
34:     for (d, d_2) ∈ addable_edges do
35:         G := node(d).G
36:         if G.weight[(d, d_2)] ≥ G.weight[(d_2, d)] then
37:             G.edges.add(e(d, d_2))
38:         else
39:             G.edges.add(e(d_2, d))
40:     ▷Finalize Transaction Ordering in G_r when G_r is a tournament
41:     OrderFinalization()
```

Figure 8: Dependency graph construction in FairDAG-RL.

Adding edges

For each pair $(d, d_2) \in addable\_edges$, if there is no edge between $node(d)$ and $node(d_2)$, an edge is added based on the majority preference (Lines 33-39):

- If $G.weight[(d, d_2)] \geq G.weight[(d_2, d)]$, add edge $e(d, d_2)$ from $node(d)$ to $node(d_2)$;
- Otherwise, add edge $e(d_2, d)$ from $node(d_2)$ to $node(d)$.

Example 6.1. Figure 9 illustrates how FairDAG-RL constructs dependency graphs. In Figure 9(a), after processing A_2, graph G_2 contains six nodes. Nodes d_0, d_1, d_2, and d_4 are classified as *solid* (solid circles), while d_3 and d_5 are *shaded* (dashed circles). All node pairs form edges except (d_3, d_5), as neither $G_2.weight[(d_3, d_5)]$

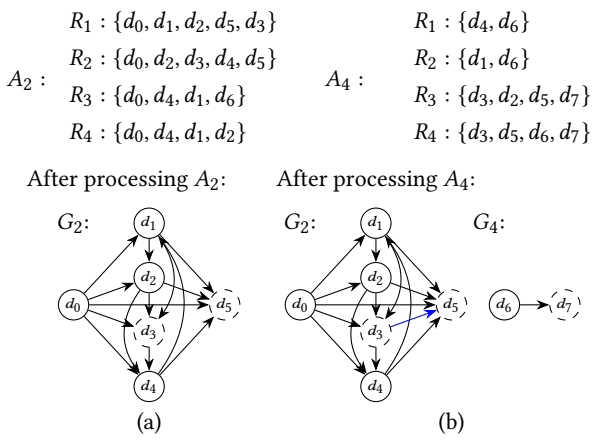


Figure 9: Constructing dependency graphs with $\mathbf{n} = 4, \gamma = 1, \mathbf{f} = 1$.

nor $G_2.weight[(d_5, d_3)]$ reaches the threshold $\frac{\mathbf{n}-\mathbf{f}}{2} = \frac{3}{2}$. In Figure 9(b), after processing A_4, G_4 is constructed with two additional nodes, and an edge between $node(d_3)$ and $node(d_5)$ is added once $G_2.weight[(d_3, d_5)]$ reaches the threshold.

6.3 Ordering Finalization

FairDAG-RL finalizes the ordering of transactions within a dependency graph after it becomes a *tournament*. A *tournament* is a dependency graph such that there is an edge between each pair of transaction nodes in the dependency graph. When finalizing the ordering, FairDAG-RL condenses the dependency graph into multiple *Strongly Connected Components (SCCs)* and generates a topological sorting of them. Transactions that are sorted behind the last *SCC* containing at least one *solid* transaction will be readded into later dependency graphs, as they might be dependent on some transactions in later dependency graphs (see more detailed explanation in Section 8.4 of our extended report [39].)

As we use the same algorithm to finalize transaction ordering within *tournaments* as Themis, we put the detailed algorithm in Section 6.3 our extended report [39].

7 COMPARING FAIRNESS PROTOCOLS

In this section, we demonstrate how FairDAG-AB and FairDAG-RL outperform Pompe [79] and Themis [44] in limiting the adversary's manipulation of transaction ordering. We achieve this by comparing how these protocols perform under adversarial conditions, such as those caused by Byzantine replicas or an asynchronous network.

7.1 Pompe and Themis

Pompe and Themis run fairness protocols atop leader-based consensus protocols such as PBFT [20] and HotStuff [74], where a single leader is responsible for collecting local orderings from a quorum of $\mathbf{n}-\mathbf{f}$ replicas.

Beyond the difference in underlying consensus protocols, Pompe assigns non-overlapping intervals to consensus rounds, allowing only transactions whose *AOI* fall within the corresponding interval to be executed, increasing the overhead of recovery when the leader fails. Additionally, in Pompe, each client transaction is sent to a single replica instead of being broadcast, making the protocol more susceptible to transaction censorship.

7.2 Adversarial Manipulation

We now analyze how an adversary can manipulate transaction ordering in Pompe and Themis by selectively collecting local orderings. Additionally, we demonstrate how FairDAG-AB and FairDAG-RL mitigate these vulnerabilities, ensuring more resilient and fair transaction orderings.

Pompe vs FairDAG-AB

In Pompe, a Byzantine replica who is responsible for collecting ordering indicators can selectively choose $\mathbf{n}-\mathbf{f}$ local ordering indicators to calculate an assigned ordering indicator. This selective collection allows the adversary to manipulate the ordering of transactions whose ranges of correct ordering indicators overlap.

Furthermore, if at most $\mathbf{f}$ correct replicas have received the assigned ordering indicator for a transaction T before a Byzantine leader replica collects assigned ordering indicators for a new block, the leader can exclude T from the new block. Since the leader is only required to collect from $\mathbf{n}-\mathbf{f}$ replicas, it can selectively collect from $\mathbf{n}-\mathbf{f}$ replicas that have not received T, thus delaying the position of T in the final ordering.

In FairDAG-AB, the client broadcasts its transaction to all replicas, and each replica independently broadcasts its ordering indicators. The *Validity* property and the round-robin or random leader rotation of the underlying DAG protocols guarantees that reduce the chance of selective collection of ordering indicators by Byzantine replicas. Thus, FairDAG-AB is more resilient against ordering manipulation than Pompe.

Additionally, Pompe effectively mitigates the transaction censorship issue mentioned in Section 7.1, because transactions are broadcast to all replicas, and each replica independently generates and broadcasts its local ordering indicators.

Themis vs FairDAG-RL

In Themis, if a Byzantine leader ignore $\mathbf{f}$ correct local orderings containing transaction T and the Byzantine replicas exclude T from their local orderings, there would be at most $\mathbf{n}-2\mathbf{f}$ local orderings containing T. Thus, the threshold of deciding edge direction is $\frac{\mathbf{n}-2\mathbf{f}}{2}$ in Themis. To guarantee the *γ-Batch-Order-Fairness*, it is required that the votes of the opposite direction cannot reach the threshold, containing $\mathbf{f}$ votes from Byzantine replicas and $(1-\gamma)(\mathbf{n}-\mathbf{f})$ votes from correct replicas. That is, $\mathbf{f} + (1-\gamma)(\mathbf{n}-\mathbf{f}) < \frac{\mathbf{n}-2\mathbf{f}}{2}$, i.e., $\mathbf{n} > \frac{\mathbf{f}(2\gamma+2)}{2\gamma-1}$. When $\gamma = 1$, Themis requires $\mathbf{n} > 4\mathbf{f}$.

In FairDAG-RL, due to the Validity of DAG-based consensus protocols, all local orderings from correct replicas will eventually be committed by all replicas. Then, for each transaction, there are at least $\mathbf{n}-\mathbf{f}$ local orderings containing it. Thus, the threshold for deciding edge direction is $\frac{\mathbf{n}-\mathbf{f}}{2}$ in Themis. Then, to guarantee the *γ-Batch-Order-Fairness*, it is required that $\mathbf{f} + (1-\gamma)(\mathbf{n}-\mathbf{f}) < \frac{\mathbf{n}-\mathbf{f}}{2}$, i.e., $\mathbf{n} > \frac{\mathbf{f}(2\gamma+1)}{2\gamma-1}$. When $\gamma = 1$, FairDAG-RL requires $\mathbf{n} > 3\mathbf{f}$.

7.3 Leader Crash and Asynchronous Network

In Pompe and Themis, if the leader crashes, a recovery subprocess must be initiated to replace the leader with a new one, introducing an additional delay of $O(\Delta)$ before the protocol can resume normal operation. Moreover, in Pompe, each round corresponds to a distinct, non-overlapping time slot. If the designated leader crashes, transactions with assigned ordering indicators that fall within that time slot cannot be ordered or executed, resulting in a potential transaction loss or indefinite delays. In Themis, if the leader crashes, replicas have to resend their local orderings to the new leader, resulting in additional communication overhead.

If the network operates under asynchronous conditions where messages can experience indefinite delays, additional overhead will be introduced, similar to the overhead incurred during leader crashes.

FairDAG-AB and FairDAG-RL address the aforementioned issues through the multi-proposer design and the Reliable Broadcast inherent to DAG-based consensus protocols, which ensures that each correct local ordering eventually deliviers and commits even in asynchronous settings.

8 EVALUATION

We evaluate FairDAG-AB and FairDAG-RL by comparing their performance with other baseline protocols. We implemented the protocols [38] in Apache ResilientDB (Incubating) [2, 33]. Apache ResilientDB is an open-source incubating blockchain project that supports various consensus protocols. It provides a fair comparison of each protocol by offering a high-performance framework. Researchers can focus solely on their protocols without considering system structures such as the network and thread models. We set up our experiments on CloudLAB m510 machines with 64 vCPUs and 64GB of DDR3 memory. Each replica and client run on a separate machine.

We compared FairDAG-AB and FairDAG-RL with the following baseline protocols:

- Pbft [20]: A single-leader consensus protocol without fairness guarantees, $\mathbf{n} > 3\mathbf{f}$.
- Pompe [79]: an absolute fairness protocol running on top of Pbft, $\mathbf{n} > 3\mathbf{f}$.
- Themis [44]: a relative fairness protocol running on top of Pbft, $\mathbf{n} > \frac{\mathbf{f}(2\gamma+2)}{2\gamma-1}$.
- RCC [31]: a multi-proposer protocol that runs concurrent Pbft instances without fairness guarantees, $\mathbf{n} > 3\mathbf{f}$.
- Tusk [24], a multi-proposer DAG-based consensus protocol without fairness guarantees, $\mathbf{n} > 3\mathbf{f}$.

For Themis and FairDAG-RL, we set $\gamma = 1$ in the experiments by default. And we implement the DAG layer of FairDAG-AB and FairDAG-RL on top of a variant of Tusk with weak edges.

8.1 Scalability

In the scalability experiments, we measure two metrics:

- *Throughput* – the maximum number of transactions per second that the system reaches consensus.

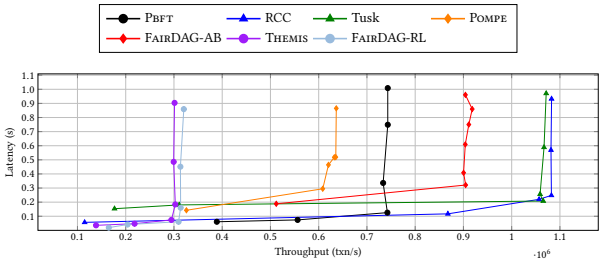


Figure 10: Throughput vs latency with $\mathbf{f} = 8$.

- *Client Latency* – the average duration between the time a client sends a transaction and the time the client receives $\mathbf{f}+1$ matching responses.

We compare the performance of the protocols with varying $\mathbf{f}$, the maximum number of faulty replicas allowed, from 5 to 8. With the same $\mathbf{f}$, different protocols have different minimum replica number requirements. For example, when $\mathbf{f} = 5$, Themis requires $\mathbf{n} = 21$ replicas, while other protocols require $\mathbf{n} = 16$ replicas.

Besides design, the performance of the protocols is highly related to the workloads. As shown in Figure 10, where we set $\mathbf{f} = 8$, as the workload increases, the throughput increases until the pipeline is fulfilled by the transactions. Then, after reaching the throughput limit, the latency increases as the workload increases. We define by *optimal point* the point with the lowest latency while maintaining the highest throughput. And we evaluate their scalability at the *optimal points* of the protocols with varying $\mathbf{f}$.

Throughput. Figure 11 shows that Tusk and RCC achieve higher throughput than other protocols because they have multiple proposers and no overhead for fairness guarantees. Due to the fairness overhead, when $\mathbf{f} = 5$ and $\mathbf{f} = 8$, FairDAG-AB reaches 83.5% and 84.9% throughput of Tusk, while FairDAG-RL reaches 11.9% and 12.6% throughput of Tusk.

However, compared to Pompe and Themis, the multi-proposer design of the DAG layer brings FairDAG-AB and FairDAG-RL advantages in throughput. When $\mathbf{f} = 5$ and $\mathbf{f} = 8$, FairDAG-AB obtains 30.2% and 52.6% higher throughput than Pompe, respectively. Similarly, FairDAG-RL reaches 7.5% and 5.1% higher throughput than Themis.

Latency. Without the fairness overhead, Tusk and Pbft, as the underlying consensus protocols, have lower latency than the fairness protocols running on top of them.

With $\mathbf{f} = 5$ and $\mathbf{f} = 8$, FairDAG-AB latency is 7.1% and 8.3% higher than Pompe, because Tusk, the underlying DAG consensus protocol of FairDAG-AB, has a higher commit latency than Pbft, the underlying consensus protocol of Pompe.

FairDAG-RL has a latency close to Themis when $\mathbf{f} = 5$. As $\mathbf{f}$ grows, FairDAG-RL has a lower latency than Themis, which is 20.9% lower when $\mathbf{f} = 8$. FairDAG-RL achieves a lower latency because Themis needs $\mathbf{f}$ more correct replicas to guarantee fairness, which causes higher overhead for both consensus and ordering. By comparing the latency of Themis with $\mathbf{f} = 6$ and FairDAG-RL with $\mathbf{f} = 8$, we can verify this claim: with the same replica number $\mathbf{n} = 25$, FairDAG-RL achieves a 4.6% higher latency than Themis.

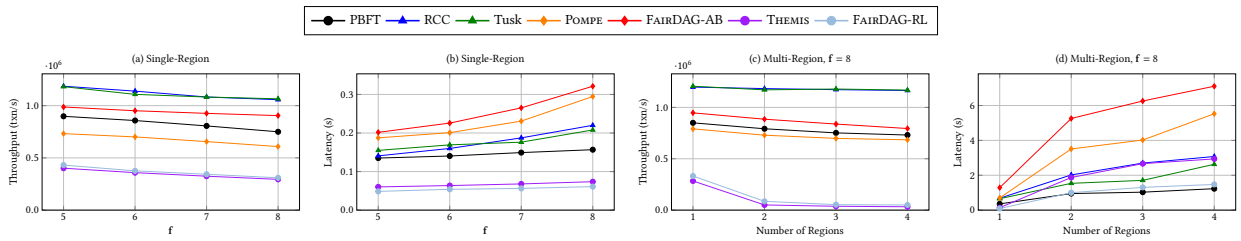


Figure 11: Performance of FairDAG and baseline protocols, with varying f (a,b), and varying number of regions (c,d).

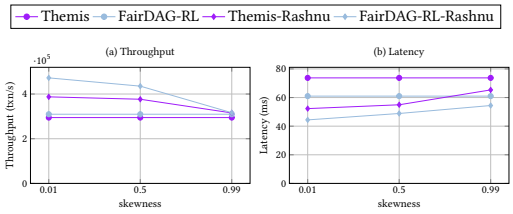


Figure 12: Performance of Rashnu-enhanced variants vs. relative fairness protocols.

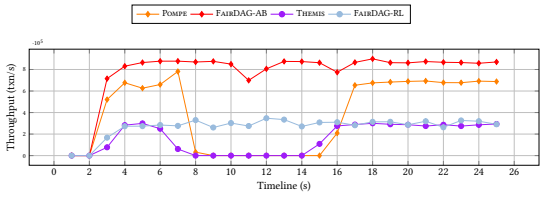


Figure 13: Real-Time throughput with a faulty leader.

Geo-distributed performance. We conducted experiments under geo-distributed settings by deploying the systems across multiple AWS regions. Specifically, we varied the number of regions from 1 to 4. The regions include North Virginia, Oregon, London, and Zurich. We fixed $\mathbf{f} = 8$ and deploys $\frac{\mathbf{n}}{k}$ replicas in each region, where k is the number of regions. Figure 11 (c, d) show that in the geo-distributed setting, the latencies of all the protocols are high and increase with the number of regions, caused by the high inter-regional message delays. Furthermore, FairDAG-AB has higher throughput than the other fairness protocols.

We found that for all protocols except the relative fairness protocols, increasing the batch size allowed us to achieve throughput values comparable to those in the single-region setting. However, in FairDAG-RL and Themis, a larger batch size leads to a higher overhead of the fairness layer, which increases quadratically with batch size. Moreover, while FairDAG-RL achieves only a 5.1% throughput improvement over Themis in a single-region setting, we observe that in the geo-distributed setting, FairDAG-RL outperforms Themis by at least 42.1%. This significant gain is attributed to the robust performance of the underlying multi-proposer DAG-based consensus protocol in geo-scale settings with limited bandwidth and higher message delays.

Data-dependent fairness. Rashnu [53] proposes a technique to reduce the overhead of the fairness layer in relative fairness protocols by computing edge directions between only data-dependent transactions. This method is orthogonal to both FairDAG-RL and Themis. We implemenetded two Rashnu-enhanced variants, called Themis-Rashnu and FairDAG-RL-Rashnu, and compared them to Themis and FairDAG-RL.

In this experiment, we implemented a transaction workload with keys following a Zipfian distribution. We fixed $\mathbf{f} = 8$ and varied the skewness parameter s from 0.01 to 0.99, where a higher skewness indicates greater data dependency between transactions. The results presented in Figure 12 show that the Rashnu variants outperform the non-Rashnu variants and perform better as the skewness decreases. This improvement stems from the reduced overhead in calculating edge directions when transactions are less interdependent.

8.2 Tolerance to Byzantine Behavior

Next, we will discuss the impact of Byzantine behaviors on the performance and transaction ordering fairness of the protocols.

Faulty leader. In this experiment, we make the consensus leader replica in Pompe and Themis faulty, which would trigger a view-change for leader replacement. While for FairDAG-AB and FairDAG-RL, we make a replica faulty due to the multi-proposer design. Figure 13 shows how the faulty leader affects the performance of Themis and Pompe. At time 7, the faulty leader stops sending any messages. After a period without progress, a view-change is triggered to replace the faulty leader. At time 15, the view-change is complete, and the throughput of Pompe and Themis recovers to the original level. In contrast, FairDAG-RL and FairDAG-AB are not affected because of the resilience provided by the multi-proposer design.

Adversarial Manipulation. We conduct two experiments in which Byzantine replicas attempt to manipulate transaction ordering. We evaluated the fairness quality of the fairness protocols under two Byzantine behaviors: (1) *Reversing order*: in each round, the Byzantine replicas reverse the local orderings of the transactions it has received. (2) *Targeted delay*: the Byzantine replicas intentionally delay targeted transactions by giving them higher local ordering indicators. In both cases, the Byzantine leader in Pompe and Themis excludes local orderings from $\mathbf{f}$ correct replicas.

For two transactions T_1 and T_2, we say that they are correctly ordered if T_1 is ordered before T_2 and:

(1) in relative fairness protocols, $weight[(d_1, d_2)] > \frac{n}{2}$;

(2) in absolute fairness protocols, the $\mathbf{f}$+1-th smallest local ordering indicator of d_1 is not larger than that of d_2.

For the *reversing order* attack, we consider all transaction pairs; for the *targeted delay* attack, we consider only the transaction pairs that involve the targeted transactions.

For FairDAG-RL and Themis, we measure the ratio of correctly ordered pairs with different $Dist$ values, where $Dist(d_1, d_2) = |weight[(d_1, d_2)] - weight[(d_2, d_1)]|$.

For FairDAG-AB and Pompe, we measure the ratio of correctly ordered pairs with different $Diff$ values, where $Diff(d_1, d_2)$ is the minimal number of Byzantine local ordering indicators needed to order d_1 before d_2. Formally, we denote by $asc\_ois_1^C[i]$ the i-th *smallest* value in the correct local ordering indicators of d_1, and by $desc\_ois_2^C[j]$ the j-th *largest* value in the correct local ordering indicators of d_2. To order d_1 before d_2, we need to find (1) a pair of i and j such that $asc\_ois_1^C[i] < desc\_ois_2^C[j]$, (2) $\mathbf{f}$+1$-i$ Byzantine local ordering indicators smaller than $asc\_ois_1^C[i]$, and (3) $\mathbf{f}$+1$-j$ Byzantine local ordering indicators larger than $desc\_ois_2^C[j]$. Thus, we have $Diff(d_1, d_2) = \min\{2(\mathbf{f}+1)-i-j \mid asc\_ois_1^C[i] < desc\_ois_2^C[j]\}$, which is the minimal number of Byzantine local ordering indicators needed. We only consider transaction pairs with $Diff(d_1, d_2) > 0$.

We set $\mathbf{f} = 10$ and vary $\mathbf{f}_a$, the actual number of Byzantine replicas, from 0 to 10. For example, Themis-7 denotes Themis with $\mathbf{f}_a = 7$. As shown in Figure 14, FairDAG-RL and FairDAG-AB consistently demonstrate better resilience against adversarial ordering manipulation in all experimental settings, compared to Themis and Pompe, respectively. The results substantiate our claim that FairDAG effectively mitigates adversarial ordering manipulation through the properties inherent in the DAG-based consensus layer.

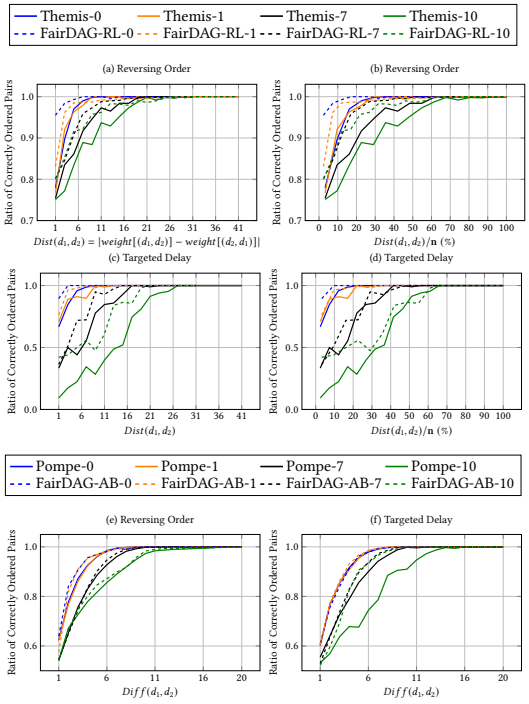


Figure 14: Fairness quality of the fairness protocols under adversarial ordering manipulation attacks.

9 RELATED WORK

In traditional Byzantine Fault Tolerance (BFT) research, protocols [5, 20, 30, 34] are designed to ensure both safety and liveness in the presence of malicious replicas. Although these protocols do not explicitly guarantee fair transaction ordering, they mitigate unfair ordering to some extent. Protocols such as Hot-Stuff [74], which employ leader rotation in a round-robin manner [15, 26, 28, 32, 40, 41, 49], provide each participant with the opportunity to propose a block. Multi-proposer approaches, including concurrent consensus protocols [27, 31, 41, 64] and DAG-based protocols [6, 8, 24, 43, 62, 63, 71], enable multiple participants to propose blocks concurrently, ordering them globally through predetermined or randomized mechanisms. Although these protocols reduce the reliance on a single leader and distribute transaction ordering authority, a Byzantine participant can still manipulate the ordering of transactions within the blocks it proposes.

Some protocols seek to eliminate the block proposers' oligarchy over the ordering of transactions within blocks via *censorship resistance* [4, 7, 17, 25, 47, 50, 51, 65, 71, 75]. In these protocols, a transaction is encrypted until the ordering of the transaction is determined. However, block proposers can still engage in censorship based on metadata, such as IP addresses, or prioritize their own transactions, knowing the content of encrypted transactions.

There are several prior fairness protocols that generate final ordering with collected local orderings. Wendy [46] guarantees *Timed-Relative-Fairness* similar to *Ordering Linearizability*, but it relies on synchronized local clocks, which are impractical in asynchronous networks. DCN [22] reaches δ-Median Fairness such that T_1 can be ordered before T_2 if T_1 is sent long enough earlier than T_2. Aequitas [45] guarantees batch-order-fairness but suffers from liveness issues due to the existence of infinite *Condorcet Cycles*, which Themis solves via a *batch unspooling* mechanism. Quick-Order-Fairness [18] reaches batch-order-fairness with $\mathbf{n} > 3\mathbf{f}$ replicas but incurs $O(\mathbf{n}^3)$ communication complexity for the consensus leader. Rashnu [53] improves Themis performance by guaranteeing *γ-Batch-Order-Fairness* between only data-dependent transactions, but suffers from the same problems as Themis. SpeedyFair [52] pipelines the consensus layer and fairness layer, but still relies on a single leader to collect local orderings. *Ambush* attacks are identified in [56], which FairDAG-RL inherently mitigates with the underlying DAG-based consensus.

10 CONCLUSION

In this paper, we introduced FairDAG, a fair-ordering framework designed to run fairness protocols atop DAG-based consensus protocols. Through theoretical demonstration and experimental evaluation, we show that unlike previous fairness protocols, FairDAG-AB and FairDAG-RL, the two variants of FairDAG, not only uphold fairness guarantees, but also achieve better performance under normal and adversarial conditions, effectively constraining adversarial manipulation of transaction ordering.

ACKNOWLEDGMENTS

This work is partially funded by NSF Award Number 2245373.

REFERENCES

[1] 2022. Aptos Whitepaper: Safe, Scalable, and Upgradeable Web3 Infrastructure. https://arxiv.org/abs/2201.01107

[2] 2024. Apache ResilientDB (Incubating). https://resilientdb.incubator.apache.org/

[3] Michael Abebe, Brad Glasbergen, and Khuzaima Daudjee. 2020. DynaMast: Adaptive dynamic mastering for replicated systems. In *2020 IEEE 36th international conference on data engineering (ICDE)*. IEEE, 1381–1392.

[4] Amit Agarwal, Kushal Babel, Sourav Das, and Babak Poorebrahim Gilkalaye. 2025. Time-Lock Encrypted Storage for Blockchains. *Cryptology ePrint Archive* (2025).

[5] Mohammad Javad Amiri, Chenyuan Wu, Divyakant Agrawal, Amr El Abbadi, Boon Thau Loo, and Mohammad Sadoghi. 2024. The Bedrock of Byzantine Fault Tolerance: A Unified Platform for BFT Protocols Analysis, Implementation, and Experimentation. In *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024*, Laurent Vanbever and Irene Zhang (Eds.). USENIX Association, 371–400.

[6] Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. 2024. Shoal++: High throughput dag bft can be fast! *arXiv preprint arXiv:2405.20488* (2024).

[7] Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. 2018. A fair consensus protocol for transaction ordering. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 55–65.

[8] Kushal Babel, Andrey Chursin, George Danezis, Lefteris Kokoris-Kogias, and Alberto Sonnino. 2023. Mysticeti: Low-Latency DAG Consensus with Fast Commit Path. *CoRR* abs/2310.14821 (2023).

[9] Paddy Baker and Omkar Godbole. 2020. Ethereum Fees Soaring to 2-Year High: Coin Metrics. *CoinDesk* (2020). https://www.coindesk.com/defi-hype-has-sent-ethereum-fees-soaring-to-2-year-high-coin-metrics

[10] Same Blackshear, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Xun Li, Mark Logan, Ashok Menon, Todd Nowacki, Alberto Sonnino, et al. 2023. Sui lutris: A blockchain combining broadcast and consensus. *arXiv preprint arXiv:2310.18042* (2023).

[11] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. *Information and Computation* 75, 2 (1987), 130–143.

[12] Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia. 2016. *Handbook of Computational Social Choice*. Cambridge University Press, Cambridge, UK.

[13] Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, et al. 2021. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. *Chainlink Labs* 1 (2021), 1–136.

[14] Christopher Brookins. 2020. DeFi Boom Has Saved Bitcoin From Plummeting. *Forbes* (2020). https://www.forbes.com/sites/christopherbrookins/2020/07/12/defi-boom-has-saved-bitcoin-from-plummeting/

[15] Ethan Buchman, Jae Kwon, and Zarko Milosevic. 2018. The latest gossip on BFT consensus. *CoRR* abs/1807.04938 (2018).

[16] Vitalik Buterin. 2013. Ethereum White Paper: A Next-Generation Smart Contract and Decentralized Application Platform. https://ethereum.org/en/whitepaper/.

[17] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*. Springer, 524–541.

[18] Christian Cachin, Jovana Mićić, Nathalie Steinhauer, and Luca Zanolini. 2022. Quick order fairness. In *International Conference on Financial Cryptography and Data Security*. Springer, 316–333.

[19] Christian Cachin and Stefano Tessaro. 2005. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*. IEEE, 191–201. https://doi.org/10.1109/SRDS.2005.36

[20] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.* 20, 4 (2002), 398–461. https://doi.org/10.1145/571637.571640

[21] Marquis de Condorcet. 1785. *Essay on the Application of Analysis to the Probability of Majority Decisions*. Imprimerie Royale, Paris.

[22] Andrei Constantinescu, Diana Ghinea, Lioba Heimbach, Zilin Wang, and Roger Wattenhofer. 2023. A fair and resilient decentralized clock network for transaction ordering. *arXiv preprint arXiv:2305.05206* (2023).

[23] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2019. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *arXiv preprint arXiv:1904.05234* (2019).

[24] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*. ACM, 34–50. https://doi.org/10.1145/3492321.3519594

[25] Pranav Garimidi, Joachim Neu, and Max Resnick. 2025. Multiple Concurrent Proposers: Why and How. *arXiv preprint arXiv:2509.23984* (2025).

[26] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. 2022. Jolteon and Ditto: Network-adaptive efficient consensus with asynchronous fallback. In *International conference on financial cryptography and data security*. Springer, 296–315.

[27] Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. 2024. Autobahn: Seamless high speed BFT. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. 1–23.

[28] Neil Giridharan, Florian Suri-Payer, Matthew Ding, Heidi Howard, Ittai Abraham, and Natacha Crooks. 2023. BeeGees: Stayin' Alive in Chained BFT. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing* (Orlando, FL, USA) *(PODC '23)*. Association for Computing Machinery, New York, NY, USA, 233–243. https://doi.org/10.1145/3583668.3594572

[29] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. 2019. Scalable Byzantine Reliable Broadcast. In *33rd International Symposium on Distributed Computing (DISC 2019) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 146. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 21:1–21:17. https://doi.org/10.4230/LIPIcs.DISC.2019.21

[30] Suyash Gupta, Jelle Hellings, Sajjad Rahnama, and Mohammad Sadoghi. 2021. Proof-of-Execution: Reaching Consensus through Fault-Tolerant Speculation. In *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, Yannis Velegrakis, Demetris Zeinalipour-Yazti, Panos K. Chrysanthis, and Francesco Guerra (Eds.). OpenProceedings.org, 301–312. https://doi.org/10.5441/002/edbt.2021.27

[31] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. 2021. RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 1392–1403. https://doi.org/10.1109/ICDE51399.2021.00124

[32] Suyash Gupta, Dakai Kang, Dahlia Malkhi, and Mohammad Sadoghi. 2025. Brief Announcement: Carry the Tail in Consensus Protocols. In *39th International Symposium on Distributed Computing (DISC 2025)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 59–1.

[33] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2020. ResilientDB: Global Scale Resilient Blockchain Fabric. *Proc. VLDB Endow.* 13, 6 (2020), 868–883. https://doi.org/10.14778/3380750.3380757

[34] Suyash Gupta, Sajjad Rahnama, Shubham Pandey, Natacha Crooks, and Mohammad Sadoghi. 2023. Dissecting BFT Consensus: In Trusted Components we Trust!. In *Proceedings of the Eighteenth European Conference on Computer Systems*. ACM, 521–539. https://doi.org/10.1145/3552326.3587455

[35] Lioba Heimbach, Eric Schertenleib, and Roger Wattenhofer. 2022. Risks and returns of uniswap v3 liquidity providers. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*. 89–101.

[36] Joshua Hildred, Michael Abebe, and Khuzaima Daudjee. 2023. Caerus: Low-Latency Distributed Transactions for Geo-Replicated Systems. *Proceedings of the VLDB Endowment* 17, 3 (2023), 469–482.

[37] Yuming Huang, Jing Tang, Qianhao Cong, Andrew Lim, and Jianliang Xu. 2021. Do the Rich Get Richer? Fairness Analysis for Blockchain Incentives *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 790–803. https://doi.org/10.1145/3448016.3457285

[38] Dakai Kang, Junchao Chen, Anh Dinh, and Mohammad Sadoghi. 2025. FairDAG. https://github.com/apache/incubator-resilientdb/tree/fairdag Accessed: 2025-04-01.

[39] Dakai Kang, Junchao Chen, Tien Tuan Anh Dinh, and Mohammad Sadoghi. 2025. FairDAG: Consensus Fairness over Multi-Proposer Causal Design. *arXiv preprint arXiv:2504.02194* (2025).

[40] Dakai Kang, Suyash Gupta, Dahlia Malkhi, and Mohammad Sadoghi. 2024. HotStuff-1: Linear Consensus with One-Phase Speculation. *arXiv preprint arXiv:2408.04728* (2024).

[41] Dakai Kang, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2024. SpotLess: Concurrent Rotational Consensus Made Practical through Rapid View Synchronization. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, Netherlands, May 13-17, 2024*. IEEE.

[42] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to Modern Cryptography* (2nd ed.). Chapman and Hall/CRC.

[43] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. 2021. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. 165–175.

[44] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. 2023. Themis: Fast, strong order-fairness in byzantine consensus. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 475–489.

[45] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. 2020. Order-fairness for byzantine consensus. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III 40*. Springer, 451–480.

[46] Klaus Kursawe. 2020. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 25–36.

[47] Rujia Li, Xuanwei Hu, Qin Wang, Sisi Duan, and Qi Wang. 2025. Transaction fairness in blockchains, revisited. *IEEE Transactions on Dependable and Secure Computing* (2025).

[48] Hatem Mahmoud, Faisal Nawab, Alexander Pucher, Divyakant Agrawal, and Amr El Abbadi. 2013. Low-latency multi-datacenter databases using replicated commit. *Proceedings of the VLDB Endowment* 6, 9 (2013), 661–672.

[49] Dahlia Malkhi and Kartik Nayak. 2023. Hotstuff-2: Optimal two-phase responsive bft. *Cryptology ePrint Archive* (2023).

[50] Dahlia Malkhi and Pawel Szalachowski. 2022. Maximal extractable value (mev) protection on a dag. *arXiv preprint arXiv:2208.00940* (2022).

[51] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 31–42.

[52] Ke Mu, Bo Yin, Alia Asheralieva, and Xuetao Wei. 2024. Separation is good: A faster order-fairness Byzantine consensus. (2024).

[53] Heena Nagda, Shubhendra Pal Singhal, Mohammad Javad Amiri, and Boon Thau Loo. 2024. Rashnu: Data-Dependent Order-Fairness. *Proceedings of the VLDB Endowment* 17, 9 (2024), 2335–2348.

[54] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

[55] Senthil Nathan, Chander Govindarajan, Adarsh Saraf, Manish Sethi, and Praveen Jayachandran. 2019. Blockchain meets database: design and implementation of a blockchain relational database. *Proc. VLDB Endow.* 12, 11 (July 2019), 1539–1552. https://doi.org/10.14778/3342263.3342632

[56] Eunchan Park, Taeung Yoon, Hocheol Nam, Deepak Maram, and Min Suk Kang. 2025. On Frontrunning Risks in Batch-Order Fair Systems for Blockchains (Extended Version). Cryptology ePrint Archive, Paper 2025/1168. https://doi.org/10.1145/3719027.3744879

[57] Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. 2021. An empirical study of defi liquidations: Incentives, risks, and instabilities. In *Proceedings of the 21st ACM Internet Measurement Conference*. 336–350.

[58] Kaihua Qin, Liyi Zhou, and Arthur Gervais. 2022. Quantifying blockchain extractable value: How dark is the forest?. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 198–214.

[59] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. 2021. Attacking the defi ecosystem with flash loans for fun and profit. In *International conference on financial cryptography and data security*. Springer, 3–32.

[60] Kun Ren, Dennis Li, and Daniel J Abadi. 2019. Slog: Serializable, low-latency, geo-replicated transactions. *Proceedings of the VLDB Endowment* 12, 11 (2019).

[61] Fabian Schär. 2021. Decentralized finance: On blockchain-and smart contract-based financial markets. *FRB of St. Louis Review* (2021).

[62] Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li. 2023. Shoal: Improving DAG-BFT latency and robustness. *arXiv preprint arXiv:2306.03058* (2023).

[63] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. 2022. Bullshark: DAG BFT Protocols Made Practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM, 2705–2718.

[64] Chrysoula Stathakopoulou, Tudor David, and Marko Vukolic. 2019. Mir-BFT: High-Throughput BFT for Blockchains. http://arxiv.org/abs/1906.05552

[65] Chrysoula Stathakopoulou, Signe Rüsch, Marcus Brandenburger, and Marko Vukolić. 2021. Adding fairness to order: Preventing front-running attacks in bft protocols using tees. In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 34–45.

[66] Weijie Sun, Zihuan Xu, and Lei Chen. 2022. Fairness Matters: A Tit-for-Tat Strategy Against Selfish Mining. *Proc. VLDB Endow.* 15, 13 (Sept. 2022), 4048–4061. https://doi.org/10.14778/3565838.3565856

[67] Weijie Sun, Zihuan Xu, Wangze Ni, and Lei Chen. 2025. InTime: Towards Performance Predictability In Byzantine Fault Tolerant Proof-of-Stake Consensus. *Proc. ACM Manag. Data* 3, 1, Article 47 (Feb. 2025), 27 pages. https://doi.org/10.1145/3709740

[68] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J Abadi. 2012. Calvin: fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data*. 1–12.

[69] Ye Wang, Patrick Zuest, Yaxing Yao, Zhicong Lu, and Roger Wattenhofer. 2022. Impact and user perception of sandwich attacks in the defi ecosystem. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–15.

[70] Ben Weintraub, Christof Ferreira Torres, Cristina Nita-Rotaru, and Radu State. 2022. A flash (bot) in the pan: measuring maximal extractable value in private pools. In *Proceedings of the 22nd ACM Internet Measurement Conference*. 458–471.

[71] Shaokang Xie, Dakai Kang, Hanzheng Lyu, Jianyu Niu, and Mohammad Sadoghi. 2025. Fides: Scalable Censorship-Resistant DAG Consensus via Trusted Components. *arXiv preprint arXiv:2501.01062* (2025).

[72] Anatoly Yakovenko. 2018. Solana: A new architecture for a high performance blockchain v0. 8.13. *Whitepaper* (2018).

[73] Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. 2022. Sok: Mev countermeasures: Theory and practice. *arXiv preprint arXiv:2212.05111* (2022).

[74] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 347–356. https://doi.org/10.1145/3293611.3331591

[75] Pouriya Zarbafian and Vincent Gramoli. 2023. Lyra: Fast and scalable resilience to reordering attacks in blockchains. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 929–939.

[76] Dirk A Zetzsche, Douglas W Arner, and Ross P Buckley. 2020. Decentralized finance. *Journal of Financial Regulation* 6, 2 (2020), 172–203.

[77] Jianting Zhang and Aniket Kate. 2024. No fish is too big for flash boys! frontrunning on DAG-based blockchains. *Cryptology ePrint Archive* (2024).

[78] Meihui Zhang, Zhongle Xie, Cong Yue, and Ziyue Zhong. 2020. Spitz: a verifiable database system. 13, 12 (Aug. 2020), 3449–3460. https://doi.org/10.14778/3415478.3415567

[79] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. 2020. Byzantine ordered consensus without byzantine oligarchy. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 633–649.