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 blockchain databases in which block proposers have full control over transaction ordering. To address this, a more fair approach to transaction ordering is essential.

Existing fairness protocols, such as Pompe and Themis, operate on leader-based consensus protocols, which not only suffer from low throughput, but also allow adversaries to manipulate transaction ordering. To address these limitations, we propose FairDAG-AB and FairDAG-RL that run fairness protocols on top of DAG-based consensus protocols, which improve protocol performance in both throughput and fairness quality, leveraging the multi-proposer design and validity of DAG-based consensus protocols.

We conducted a comprehensive analytical and experimental evaluation of our protocols. The results show that FairDAG-AB and FairDAG-RL outperform the prior fairness protocols in both throughput and fairness quality.

1 INTRODUCTION

The emergence of cryptocurrencies, including Bitcoin [37] and Ethereum [11], has sparked broad interest in blockchain database technology. Blockchain databases enable new class of applications, namely decentralized finance (DeFi) [7, 10, 43, 55], whose market capitalization exceeds 70 billion. Transaction ordering in DeFi has two requirements: consistency and fairness. The former ensures that all the blockchain nodes must agree on the same order, which is necessary for correctness. Consistent transaction ordering has been addressed under crash-failure settings of traditional distributed databases, for example [2, 23, 33, 42, 51], and under Byzantine-failure settings in blockchains and verifiable databases [11, 38, 56] where faulty replicas can have arbitrary malicious behavior. In the Byzantine-failure setting, although the transaction orderings are consistent across nodes, the fairness of ordering is vulnerable to manipulations by attackers. These manipulations, including front-running, back-running, sandwich attacks, liquidation manipulation, and time-bandit attacks, lead to significant financial losses for the victims [17, 39–41, 52]. These attacks can lower the incentives of the blockchain nodes for behaving correctly, thereby compromising the system's security.

This "order manipulation crisis" affects both permissionless blockchains (e.g., Ethereum) and permissioned blockchains (e.g., PBFT [15], HotStuff [54]), as block proposers have full control over transaction selection and ordering. Malicious proposers, such as leaders in Ethereum or PBFT, may reorder transactions to maximize personal profits [17]. This control over transaction order is inherently unfair to other participants.

The key to achieving fair ordering is to prevent the proposers from dominating the transactions within blocks. Existing works propose fairness protocols with different fairness guarantees [24, 30, 32, 36, 49, 50, 57]. Two representative protocols are Pompe [57] and Themis [30]. Pompe orders transactions based on final *assigned ordering indicators*. It achieves *Ordering Linearizability*, which guarantees that transaction T_1 is ordered before T_2 if every correct participant receives T_1 before any correct participant receives T_2. Themis constructs dependency graphs between transactions then orders the transactions along the edges. It supports a stronger fairness notion, *γ-batch-order-fairness*, which ensures 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.

In Pompe and Themis, each block contains not a list of transactions but a set of digitally signed local orderings, where each ordering reflects the sequence preferred by the signer. Once blocks are committed, a final transaction ordering is derived from the local orderings within them. The fairness properties can be ensured if the committed local orderings include ordering preferences from the majority of participants.

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

G1 **Limiting adversarial Manipulation.** A well-designed fairness protocol should restrict the influence of malicious participants on the final transaction ordering, thereby preserving fairness guarantees.

G2 **Minimal correct participants.** To mitigate the influence of adversarial participants, fairness protocols aggregate local orderings from correct participants. Given the same number of malicious actors, an effective fairness protocol should ensure fairness properties while relying on the minimal number of correct participants.

G3 **High performance.** As a specialized class of BFT protocols, fairness protocols should strive for high performance, including high throughput and low latency, similar to traditional BFT protocols.

Unfortunately, existing fairness protocols [30, 31, 57] rely on leader-based consensus protocols, such as PBFT [15] and HotStuff [54], to commit blocks, which hinders their ability to fully achieve the design objectives above. Leader-based consensus protocols rely on a single leader to collect local orderings from a quorum of participants. A malicious leader, however, can manipulate transaction ordering by selectively collecting local orderings. Additionally, even in the absence of adversary, a single leader may become a performance bottleneck if the workload exceeds its capacity.

To address the challenges posed by underlying leader-based consensus protocols, we propose running fairness protocols on top of DAG-based consensus protocols [18, 29, 46], which offer the following features that we can adopt to help achieve the objectives above:

- **Multi-proposer high-throughput design**: DAG-based protocols employ a multi-proposer design, allowing all participants to propose blocks (called vertices) in parallel. This approach enhances system throughput.
- **Validity**: Vertices in DAG-based protocols reference vertices from other replicas, forming a *Directed Acyclic Graph (DAG)*. Vertices from correct participants will eventually be committed by all correct participants.

The *multi-proposer high-throughput design* eliminates the bottleneck caused by a single leader (**G3**). The *validity* limits the adversary's ability to selectively collect local orderings (**G1**), and thus lowers the requirement on the number of correct participants (**G2**).

Then, we propose FairDAG that runs fairness protocols on top of DAG-based protocols. In FairDAG, each participant proposes its local ordering as a vertex and reliably broadcasts the vertices, and a final transaction ordering is deterministically generated based on the vertices committed by the DAG-based protocols. We design two variants of FairDAG, namely FairDAG-AB with *absolute ordering*, and FairDAG-RL with *relative ordering* guarantees.

There are two challenges in using DAG-based protocols as underlying consensus protocols for fairness. First, participants may have inconsistent views on uncommitted vertices in the DAG. Second, malicious participants may attempt to manipulate orderings by ignoring vertices from specific participants. FairDAG addresses the challenges by designing a novel ordering indicator manager, adapting fairness threshold values, and applying new DAG formation rules for fairness guarantees.

In summary, we make the following contributions:

(1) We propose FairDAG-AB, a DAG-based *absolute* fairness protocol that guarantees fairness property *Ordering Linearizability*. We propose a *Ordering Indicator Manager* and a transaction execution threshold called *LPAOI* that are specific to the multi-proposer design and commit rules of the DAG-based consensus protocols.

(2) We propose FairDAG-RL, a DAG-based *relative* fairness protocol that guarantees fairness property *γ-batch-order-fairness*. We adopted new thresholds for dependency graph construction to improve performance, by leveraging the validity property of DAG-based consensus protocols.

(3) In FairDAG, we extend existing DAG-based protocols by applying new rules of forming DAG vertices within the DAG layer to guarantee fairness under adversarial conditions.

(4) We conducted comprehensive analytical and experimental evaluation of our protocols. The results show that: FairDAG-AB outperforms Pompe in throughput and fairness quality; FairDAG-RL not only outperforms Themis in throughput and fairness quality, but also has a lower requirement on the number of correct participants.

The remaining of the 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 Section 7 compares FairDAG with prior fairness protocols Pompe and Themis. Section **??** presents the correctness proof. Section 9 presents the experimental evaluation of FairDAG and baseline protocols. Section 10 discusses other related works, and Section 11 concludes.

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

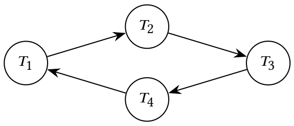


Figure 1: Condorcet Cycle

2 BACKGROUND

FairDAG-AB and FairDAG-RL run fairness protocols on top of DAG-based consensus protocols. Starting from this section, our discussion is framed within the context of *Byzantine Fault Tolerance (BFT)* protocols, where participants are replicas. *Byzantine replicas,* corrupted by the adversary, can have arbitrary malicious behavior; the remaining are *correct replicas*, having benign behaviors. In this section, we introduce the definitions of three fairness properties and 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 *Receive-Order-Fairness* is impossible to guarantee in the presence of malicious replicas due to the formation of a *Condorcet cycle.* 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 must consider all majority-endorsed orderings supported by at least three replicas. The right side of Figure 1 depicts a directed graph where an edge $T_i \to T_j$ indicates that at least three replicas received T_i before T_j. The resulting graph contains a Condorcet cycle [9, 16]—a cycle of pairwise preferences that cannot be linearly extended without violating at least one of them. Hence, the *Receive-Order-Fairness* is impossible if the identity of the Byzantine replicas is unknown.

2.2 Ordering Linearizability

Ordering Linearizability is a fairness property introduced by Pompe [57] and adopted by our protocol, FairDAG-AB. To achieve this property, each replica assigns a monotonically increasing *ordering indicator* (denoted *oi*) to each transaction based on its local order of receiving the transactions. The final ordering is then derived from these indicators.

Denote by ois_i^C the set of ordering indicators for transaction T_i from correct replicas. We define *Ordering Linearizability* as follows:

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

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

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 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. Replicas R_1, R_2, and R_3 are correct, while R_4 is Byzantine. Each replica assigns 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 [30] and our protocol FairDAG-RL.

We say that a transaction T_2 is **dependent** on transaction T_1, denoted as $T_1 \longrightarrow T_2$, 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 $T_1, T_{m_1}, T_{m_2}, ..., T_{m_k}, T_2$ such that:

- $\forall j \leq k$, T_{m_j} is dependent on $T_{m_{j-1}}$, i.e., $T_{m_{j-1}} \longrightarrow T_{m_j}$;
- $T_1 \longrightarrow T_{m_1}$; $T_{m_k} \longrightarrow T_2$.

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*—meaning that $S_i \cup S_{i+1}$ does not form a cyclically dependent set.

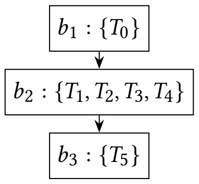
We say that T_1 is ordered **no later than** T_2 if T_1 appears in the same or an earlier batch than T_2. Based on this notion, we define *γ-Batch-Order-Fairness* as follows:

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

For example, as shown in Figure 3 (a), we have four replicas receiving six transactions in different orders, and a final transaction ordering is generated. By splitting the final ordering into three *cyclic dependency* batches (the details related to adding dependencies between transactions will be explained in Section 6), for each transaction pair, the fairness property *γ-Batch-Order-Fairness* holds. For instance, T_0 is received earlier than T_1 by $\gamma(\mathbf{n}-\mathbf{f}) = 3$ correct replicas, the fairness property holds as T_0 is ordered in an earlier batch than T_1.

$$
\begin{aligned}
&R_1 : \{T_0, T_1, T_2, T_3, T_4, T_5\}\\
&R_2 : \{T_0, T_2, T_3, T_4, T_1, T_5\}\\
&R_3 : \{T_0, T_3, T_4, T_1, T_2, T_5\}\\
&R_4 : \{T_0, T_4, T_1, T_2, T_3, T_5\}\\
&\text{Final Ordering: } \{T_0, T_1, T_2, T_3, T_4, T_5\}
\end{aligned}
$$

(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.

For example, as shown in Figure 3(a), four replicas (three of which are correct) receive six transactions in different orders, and a final ordering is generated. This ordering is partitioned into three *cyclically dependent* batches (the construction of dependencies is detailed in Section 6). For each transaction pair, the *γ-Batch-Order-Fairness* property holds, $\gamma = \frac{2}{3}$. For example, T_1 is received before T_2 by two correct replicas—satisfying the γ-fraction threshold—and T_1 appears in the same batch b_1 as T_2, thus preserving fairness.

2.4 DAG-based Consensus Protocols

DAG-based BFT consensus protocols [18, 29, 46] operate in rounds. In each round r, every participant proposes a block, represented as a DAG vertex. Each vertex references multiple vertices from the previous round $r-1$, represented as edges that encode causal dependencies, forming an *Directed Acyclic Graph (DAG)*. The *causal history* of a vertex includes all vertices reachable via directed paths.

Every k rounds (e.g., $k = 2$ in Tusk), a leader vertex is either randomly selected or predetermined. These leader vertices are committed in round order, 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*:

- **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 leader-based protocols, the multi-proposer design of DAG-based protocols has 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. We also specify the conditions under which different fairness protocols can guarantee their respective properties.

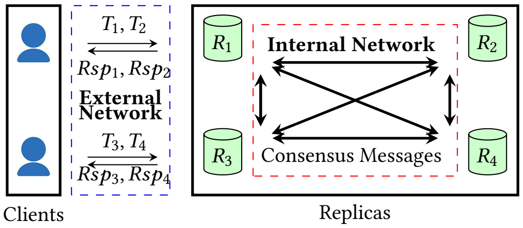


Figure 4: Network topology with Clients, Replicas, and Networks. ***Rsp_i*** **represents a client response for transaction** ***T_i*****, including the execution results of** ***T_i*****.**

3.1 Clients

Clients issue transactions to replicas and await execution results. They may behave arbitrarily; thus, we make no correctness assumptions about client behavior.

3.2 Replicas

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

In the context of fair ordering, Byzantine replicas may falsify local transaction orderings or attempt to filter out those unfavorable to their interests. In contrast, correct replicas strictly follow the protocol, reporting local orderings based on the order of receiving from clients.

Fairness protocols differ in the guarantees they provide and then in their 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 may impersonate each other but cannot forge messages from correct replicas. Authentication is enforced via *digital signatures* [28]. For integrity verification, each transaction T_i is associated with a digest d_i, computed using a *secure collision-resistant cryptographic hash function* [28].

3.4 Network

As illustrated in Figure 4, we distinguish between two types of networks: (1) *external communication* between clients and replicas, and (2) *internal communication* among replicas.

External network (client–replica). We make no assumptions about synchrony but assume the external network is *non-adversarial*—i.e., it cannot be fully controlled by a malicious party. This assumption is necessary for preserving fairness, as an adversarial external network could arbitrarily control the order replicas receive transactions and then the final ordering.

Internal network (replica–replica). The internal network may operate under either an *asynchronous* or a *partially synchronous* model:

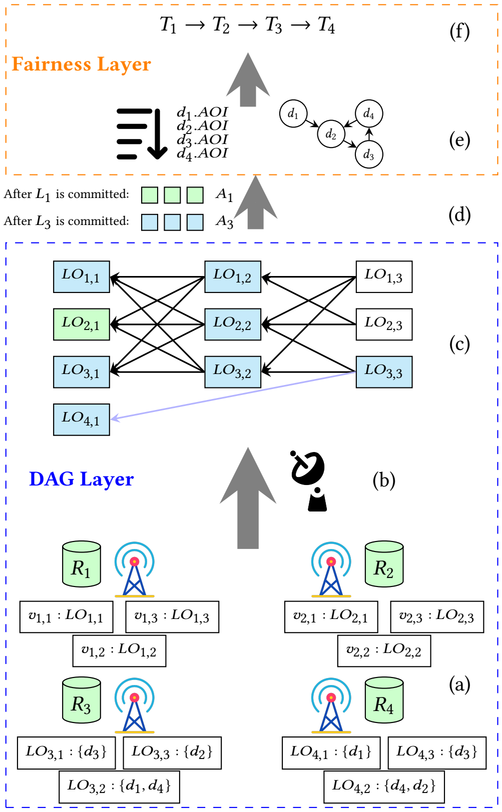


Figure 5: Overview Architecture of FairDAG: (a) Replicas reliably broadcast blocks containing their local ordering fragments. (b) Each replica receives blocks delivered through Reliable Broadcast. (c) Each replica forms a local view of the DAG using received blocks and reference links, where different colors represent the causal history of different committed leader vertices. (d) Local orderings in ***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.**

- **Asynchronous network:** Messages are never lost and are eventually delivered, but delivery delays may be unbounded.
- **Partially synchronous network:** There exists an unknown *Global Stabilization Time (GST)* after which 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 [57] and Themis [30], are built atop leader-based consensus. Although the final ordering is derived from local orderings collected from a quorum, existing fairness protocols have the following problems: (1) Leaders can selectively pick up the quorum, allowing manipulation and compromising fairness. (2) Faulty or slow leaders can become system bottlenecks. To address these problems, FAIRDAG runs fairness protocols over multi-proposer DAG-based consensus. FAIRDAG is structured into two layers:

- *DAG Layer* handles the dissemination (Figure 5(a,b)) and commitment (Figure 5(c)) of local orderings.
- *Fairness Layer* takes these committed local orderings (Figure 5(d)) and runs a fair-ordering mechanism (Figure 5(e)) to produce a final ordering (Figure 5(f)) that satisfies defined fairness properties.

4.1 DAG Layer

The DAG layer runs existing protocols—such as Tusk [18], DAG-Rider [29], and Bullshark [46]—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 the final ordering reflects the preferences of the majority. (2) To preserve the integrity of a replica's local ordering, each vertex $v_{i,r}$ must include a strong edge to its immediate predecessor $v_{i,r-1}$ (for $r > 0$). Figure 5(c) shows an example DAG: squares represent vertices, 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 [8, 14, 18, 21] a vertex that contains a slice of LO_i, which is the local ordering of transactions of R_i.

In traditional DAG protocols, each vertex contains a list of transactions in the order chosen by the proposer. In contrast, as Figure 5(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.

Forming a DAG

As illustrated in Figure 5 (a,b), the DAG layer of FAIRDAG enables replicas to reliably broadcast and receive vertices from one another. This multi-proposer design alleviates the performance bottlenecks associated with a single leader, thereby addressing issue (2).

Using the received vertices and the references within them, each replica constructs its local view of the DAG, as shown in Figure 5(c). DAG protocols ensure that all correct replicas eventually have consistent DAG views.

Committing DAG vertices

The commit rules in DAG-based consensus protocols—leveraging random leader selection and awareness of causal dependencies between vertices—further mitigate leader manipulation in quorum selection, thereby 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 ($L_{r_1}, L_{r_2}, \ldots$) such that $C_{r_i} \subset C_{r_{i+1}}$ for all i.

A vertex v is said to be *anchored* to 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 set of vertices anchored to L_{r_i}. The committed DAG can thus be partitioned into non-overlapping segments ($A_{r_1}, A_{r_2}, \ldots$), each corresponding to a committed leader.

For example, in Figure 5(c), assume L_1 is the green vertex $v_{2,1}$ and L_3 is the blue vertex with $v_{3,3}$. Then A_1 consists of the green vertices, and A_3 consists of the blue ones.

As shown in Figure 5(d), each time a leader L_r is committed, the fairness layer processes A_r to compute the final ordering.

This approach ensures that the output aligns with the preferences of the majority of correct replicas, as each 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 ones. (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 causal history as input, the fairness layer runs a fair-ordering mechanism to compute the final transaction ordering.

In **absolute** fairness protocols (Figure 5(e), left), each transaction gets an *assigned ordering indicator (AOI) based on the input local orderings.* The final ordering (Figure 5(f)) is then derived by sorting transactions according to the *AOI* values. Protocols in this category, such as FAIRDAG-AB, satisfy the *Ordering Linearizability* property.

In **relative** fairness protocols (Figure 5(e), right) construct a dependency graph where nodes represent transactions, and edges encode pairwise ordering dependencies derived from the input local orderings. The final ordering is obtained by computing a *Hamiltonian path* through this graph (Figure 5(f)). Protocols in this category, such as FAIRDAG-RL, satisfy the *γ-Batch-Order-Fairness* property.

5 FAIRDAG-AB

FAIRDAG-AB is an **absolute** fairness protocol. Each transaction gets an *assigned ordering indicator* based on local ordering indicators from a quorum of replicas. Transactions are then ordered based on their assigned ordering indicators.

5.1 Transaction Dissemination

To prevent leader-driven manipulation, clients broadcast transactions to all replicas rather than submitting them to a single replica. This prevents adversarial replicas from selectively delaying transactions to influence their relative order (e.g., to back-run certain transactions).

Upon receiving a client transaction T, replica R gives T a monotonically increasing ordering indicator oi from the local timer, appends digest d of T and oi to the lists dgs and ois (Figure 7, Line 4), respectively, which are local variables storing transaction digests and corresponding ordering indicators. R also updates its $local\_highest\_oi$ (Lines 9-13).

5.2 FairDAG-AB Vertex

Using DAG consensus protocols, each FairDAG-AB replica constructs and reliably broadcasts a DAG vertex upon the fulfillment of the specific conditions outlined by the DAG protocols. As an absolute fairness protocol, as described in 4.1, in addition to the necessary information to form a DAG, each FairDAG-AB vertex contains the information of client transactions it has received since the last time it broadcast a DAG 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 transactions since its last broadcast, encoded in dgs, ois, and its current highest local ordering indicator $local\_highest\_oi$ (Lines 14-18).

5.3 Managing and Assigning Ordering Indicators

FairDAG-AB leverages local orderings within committed DAG vertices to determine the final transaction order. However, a delay often 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 Indicator Manager* ($OIM(d)$) for each transaction digest d, which tracks the replica's local view of the DAG and ordering indicators contained within the vertices.

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

- $seen\_ois$: vector of ordering indicators of d that R has seen.
- $committed\_ois$: vector of ordering indicators of d that R has committed.
- $LPAOI$: lowest possible assigned ordering indicator of d.
- AOI: 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 assign a final position (see Section 5.4). All vectors are indexed from 1 to $\mathbf{n}$ and initialized to ∞; values are considered valid once set to a finite number.

Managing Ordering Indicators

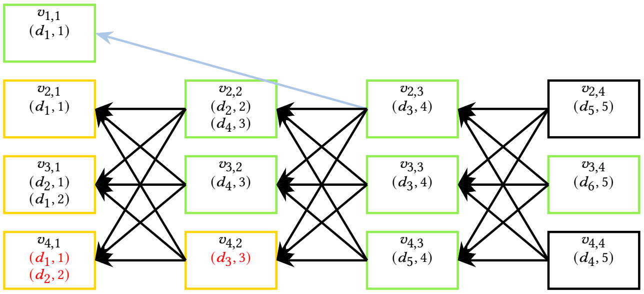


Figure 6: Example of calculating AOI and LPAOI.

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 $\mathbf{n}$ (Line 3). This vector is essential for the global ordering of transactions (See details in Section 5.4).

Upon a DAG vertex $v_{i,r}$ is reliably broadcast and added to replica R's local DAG view, R updates $highest\_ois[i]$ to $hoi_{i,r} + 1$, where $hoi_{i,r}$ is the highest ordering indicator in $v_{i,r}$. For each digest d in $v_{i,r}$, R updates $seen\_ois[i]$ of $OIM(d)$ (Lines 19-22).

Upon a DAG leader vertex L_r is committed, and its associated A_r is input to the fairness layer (Lines 23-24), R updates $committed\_ois$ for the digests in A_r accordingly (Lines 25-27).

Calculating Assigned Ordering Indicator

After committing at least $\mathbf{n}-\mathbf{f}$ non-∞ $committed\_oi$ values for a digest d, R calculates $OIM(d).AOI$. The AOI is the $(\mathbf{f}+1)$-th smallest value of $committed\_ois$, which definitely falls within the range of correct ordering indicators. Then, d is added to $txns\_w\_assigned\_oi$, which is set of transactions digests with valid AOI values (Lines 29-32). *Note: The value of AOI is immutable once calculated, even if more ordering indicators of d are committed.*

Example 5.1. Figure 6 illustrates the assignment of AOI values. Suppose $v_{4,2}$ and $v_{3,4}$ are committed leader vertices L_2 and L_4, with yellow and green vertices corresponding to A_2 and A_4, respectively. In A_2, digest d_1 has committed ordering indicators $\infty, 1, 2, 1$. Since this includes at least $\mathbf{n}-\mathbf{f}$ valid values, its AOI is the $(\mathbf{f}+1)$-th lowest, which is 1 (Lines 29-32). 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 and leader selection in DAG-based protocols, DAG vertices with smaller local ordering indicators may be committed later. 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 transaction could get a lower AOI before executing a transaction. To enforce this constraint, we design a mechanism to compute $LPAOI_{min}$, an

```
1:  **State Variables (per replica)**
2:  $txns\_w\_assigned\_oi := \{\}$
3:  $highest\_oi\_list[1..\mathbf{n}] := 0$
4:  $dgs, ois := [\,]$
5:  $local\_highest\_oi, current\_round, replica\_id$

6:  **Ordering Indicator Manager** ($OIM$)
7:  $seen\_ois[1..\mathbf{n}] := \infty,\ committed\_ois[1..\mathbf{n}] := \infty$
8:  $LPAOI := \infty,\ AOI := \infty$

    Client Thread (processing client transactions) :
9:  event On receive transaction $T$ do
10:     if $T$ is valid then
11:         $oi := local\_timer.GetNextOI()$
12:         $dgs.append(T.digest)$; $ois.append(oi)$
13:         $local\_highest\_oi := oi$

    DAG Layer Thread (forming the DAG) :
14: event On propose DAG vertex do
15:     $v := DAGVertex(replica\_id, current\_round)$
16:     $v.dgs := dgs$, $v.ois := ois$, $v.hoi := local\_highest\_oi$
17:     $dgs.clear(), ois.clear()$
18:     Reliably broadcast $v$

19: event On receive $v_{i,r}$ do
20:     $highest\_oi\_list[i] := v_{i,r}.hoi + 1$
21:     for $(d, oi) \in (v_{i,r}.dgs, v_{i,r}.ois)$ do
22:         $OIM(d).seen\_ois[i] := oi$

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

    Fairness Layer Thread (Ordering transactions) :
25: event On receive $A_r$ do
26:     for $v \in A_r$, $(d, oi) \in (v.dgs, v.ois)$ do
27:         $OIM(d).committed\_ois[v.replica\_id] := oi$
28:     $LPAOI_{\min} := \infty$
29:     for $d$ such that $OIM(d) = \infty$ do
30:         if count of non-$\infty$ in $OIM(d).committed\_ois \geq \mathbf{n}-\mathbf{f}$ then
31:             $OIM(d).AOI := (\mathbf{f}+1)$-th smallest in $committed\_ois$
32:             add $d$ to $txns\_w\_assigned\_oi$
33:         else
34:             for $i = 1$ to $\mathbf{n}$ do
35:                 $lp\_ois[i] := \min(d.seen\_ois[i], highest\_oi\_list[i])$
36:             $d.LPAOI := (\mathbf{f}+1)$-th smallest in $lp\_ois$
37:             $LPAOI_{\min} := \min(LPAOI_{\min}, d.LPAOI)$
38:     for $d \in txns\_w\_assigned\_oi$ sorted by $AOI$ do
39:         if $d.AOI < LPAOI_{\min}$ then
40:             execute the transaction of $d$;
41:             remove $d$ from $txns\_w\_assigned\_oi$
42:         else
43:             break
```

Figure 7: FairDAG-AB Algorithm.

AOI threshold that must be satisfied before a transaction can be executed.

LPAOI: Lowest Possible Assigned Ordering Indicator

While assigning *AOI* values, the fairness layer computes a lower bound, $LPAOI_{\min}$, for the remaining digests without a valid non-∞ *AOI*. This ensures that transactions are only ordered when it is safe to do so. For each 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 $(\mathbf{f}+1)$-th smallest value in lp_ois. The minimum value across all *LPAOI* values, is denoted as $LPAOI_{\min}$ (Lines 33-37).

$LPAOI_{\min}$ serves as a threshold: only digests with $AOI < LPAOI_{\min}$ are eligible for execution. The fairness layer sorts all digests in *txns_w_assigned_oi* by *AOI*, executes them sequentially, and stops when *AOI* reaches or exceeds the threshold. The ordered digests are then removed from *txns_w_assigned_oi* (Lines 38-43).

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 are assigned *LPAOI* values of 4 and 5. $LPAOI_{\min}$, the minimum *LPAOI* across all digests is 3, (Lines 33-37) so d_1 and d_2 can be ordered and executed. In contrast, d_3 cannot be ordered since its *AOI* exceeds $LPAOI_{\min}$, implying that d_4 may be assigned a lower *AOI* than d_3 (Lines 38-43).

6 FAIRDAG-RL

FairDAG-RL is a **relative** fairness protocol. It constructs a dependency graph where each node represents a transaction digest, and directed edges are added based on committed local orderings, representing dependencies between nodes. The final transaction ordering is extracted by computing a Hamiltonian path through the graph, satisfying *γ-Batch-Order-Fairness* under the condition $\mathbf{n} > \frac{\mathbf{f}(2\gamma+1)}{2\gamma-1}$ for $\frac{1}{2} < \gamma \leq 1$.

6.1 Transaction Dissemination

As in FairDAG-AB, clients broadcast transactions to all replicas to prevent proposer bias. Upon meeting the DAG protocol's conditions, 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 8 Lines 7-8). For simplicity, we omit the ordering indicators in Figure 9.

We define the *committed local ordering* of a replica R as the sequence of transaction digests contained in the vertices proposed by R that have been committed in the DAG layer. The committed local orderings contained in A_r will be input to the fairness layer after L_r is committed (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. A new dependency graph is constructed each time a leader vertex is committed:

First, transaction digests from A_r are added as graph nodes if not previously added. Each transaction digest d is associated with a node that stores:

```
State Variables
$graphs := []$
$current\_round, replica\_id$
$counter := 0$

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

DAG Layer Thread (forming the DAG) :
event On propose vertex do
    $v := DAGVertex(replica\_id, current\_round)$
    $v.dgs := dgs$,   $v.ois := ois$
    $dgs.clear()$,   $ois.clear()$
    Reliably broadcast $v$

event On commit($L_r$) do
    Send $A_r$ to Ordering Layer
```

Figure 8: FairDAG-RL: Transaction dissemination and DAG vertex proposal.

- *type*: the node type, initialized as *blank* and updated upon insertion into a graph.
- *committed_ois*: a vector of committed ordering indicators for d, initialized to ∞, indexed from 1 to $\mathbf{n}$.
- *committed_rounds*: the corresponding commitment rounds, initialized to ∞, indexed from 1 to $\mathbf{n}$.
- G: the graph that the node is added into.

Second, pairwise ordering preferences are aggregated into a weight function: $weight(d_1, d_2)$ denotes the number of committed local orderings in which d_1 precedes d_2, representing the counts of ordering preference. Each dependency graph contains the following information:

- *nodes*: the set of transaction digest nodes.
- *weight* : mapping from node pairs to their weights.
- *edges*: directed edges representing inferred ordering constraints between 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 no 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 faulty replicas, relative fairness protocols should leverage as many local orderings as possible. Prior protocols such as Themis [30] and Rashnu [36] rely on a leader-based consensus layer, where a Byzantine leader may 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 the faulty replicas ignore it, which increases both the susceptibility to ordering manipulation and the minimum node requirement (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. Accordingly, FairDAG-AB raises the thresholds used in dependency graph construction 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 elaborate on the details below.

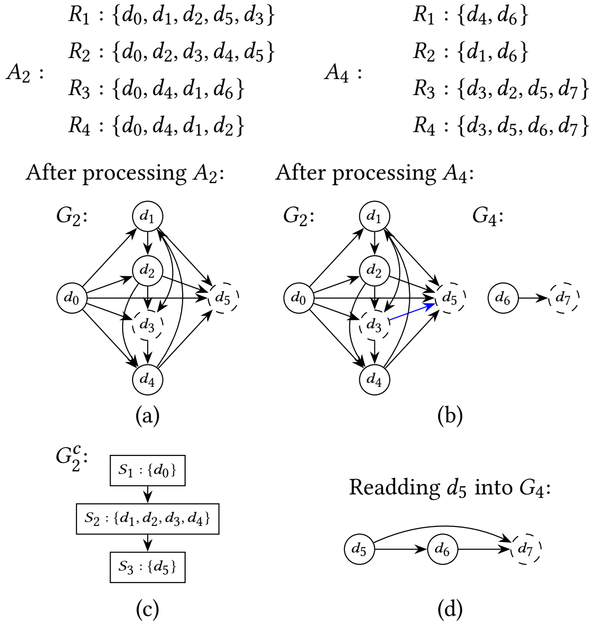

A_2 :
$R_1 : \{d_0, d_1, d_2, d_5, d_3\}$
$R_2 : \{d_0, d_2, d_3, d_4, d_5\}$
$R_3 : \{d_0, d_4, d_1, d_6\}$
$R_4 : \{d_0, d_4, d_1, d_2\}$
A_4 :
$R_1 : \{d_4, d_6\}$
$R_2 : \{d_1, d_6\}$
$R_3 : \{d_3, d_2, d_5, d_7\}$
$R_4 : \{d_3, d_5, d_6, d_7\}$
After processing A_2:
After processing A_4:
G_2:
d_0 d_1 d_2 d_3 d_4 d_5
G_2:
d_0 d_1 d_2 d_3 d_4 d_5
G_4:
d_6 d_7
(a)
(b)
G_2^c:
$S_1 : \{d_0\}$
$S_2 : \{d_1, d_2, d_3, d_4\}$
$S_3 : \{d_5\}$
Readding d_5 into G_4:
d_5 d_6 d_7
(c)
(d)

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

Adding nodes

For each transaction digest d in A_r, let $node(d)$ denote a global variable that represents its dependency graph node. Its *committed_ois* and *committed_rounds* are updated using information contained in A_r, and the node is added to a temporary set *updated_nodes* (Figure 10 Lines 3-10) [1].

Then, we check if any transaction digest can be added as nodes into the new dependency graph of round r. We define $ap(d, r)$ as the number of replicas from which an ordering indicator for d is committed by R by round r:

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

Each node in *updated_nodes* that is still *blank* 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 $ap(d, r) < \frac{\mathbf{n}-\mathbf{f}}{2}$.

If the *type* of a *blank* node changes to *solid* or *shaded*, the node is added to G_r (Lines 11-18). Since classification is only applied to previously *blank* nodes, each digest is inserted into at most one dependency graph.

Updating weights between nodes

[1] 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:    $G_r := NewGraph(), \quad graphs.push(G_r)$
3:    ▷Find nodes updated with $A_r$
4:    $updated\_nodes := \{\}$
5:    for $v \in A_r$ do
6:      $i := v.replica\_id$
7:      for $(d, oi) \in (v.dgs, v.ois)$ do
8:        $node(d).committed\_ois[i] := oi$
9:        $node(d).committed\_rounds[i] := r$
10:       $updated\_nodes.insert(d)$
11:   ▷Add new non-blank nodes to $G_r$
12:   for $d \in updated\_nodes$ do
13:     if $node(d).type = blank$ then
14:       $ap(d, r) := |\{i \mid node(d).committed\_rounds[i] \leq r\}|$
15:       if $ap(d, r) \geq$ n − f then
16:         $node(d).type := solid; \quad G_r.nodes.add(node(d))$
17:       else if $ap(d, r) \geq \frac{n-f}{2}$ then
18:         $node(d).type := shaded; \quad G_r.nodes.add(node(d))$
19:   ▷Find all candidate edges in all existing graphs $G_r$
20:   $addable\_edges := \{\}$
21:   for $v \in A_r$ do
22:     $i := v.replica\_id$
23:     for $(d, oi) \in (v.dgs, v.ois)$ do
24:       $G' := node(d).G$
25:       $d\_oi := node(d).committed\_ois[i]$
26:       for $node(d_2) \in G'.nodes$ 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 $\frac{n-f}{2}$ then
32:           $addable\_edges.insert(d, d_2)$
33:   ▷Add edge if weight reaches threshold
34:   for $(d, d_2) \in addable\_edges$ do
35:     $G := node(d).G$
36:     if $G.weight[(d, d_2)] \geq G.weight[(d_2, d)]$ 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 10: Dependency graph construction in FairDAG-RL.

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 $node(d)$ with all other nodes in the same graph G using their values in $committed\_ois[i]$. For each pair (d, d_2) such that $node(d_2) \in G$, we: increment $G.weight[(d, d_2)]$ if $node(d).committed\_ois[i] < node(d_2).committed\_ois[i]$; otherwise, increment $G.weight[(d_2, d)]$ (Lines 21-30).

During this process, we maintain a set $addable\_edges$ to track candidate edge insertions. If $G.weight[(d, d_2)]$ or $G.weight[(d_2, d)]$ reaches the threshold $\frac{n-f}{2}$, the corresponding pair is added to $addable\_edges$ (Lines 31-32).

```
Fairness Layer Thread (Ordering transactions) :
1:  function OrderFinalization() do
2:    while $G_r := graphs.Front()$ do
3:      if $G_r$ is a tournament then
4:        $graphs.Pop()$
5:        $G_r^c := Trajan\_SCC(G_r)$
6:        $[S_1, S_2, ..., S_s] := Topologically\_Sorted(G_r^c)$

7:        $last := \max\{j \mid \exists node \in S_j, node.type = solid\}$
8:        for $j = 1, 2, ..., last$ do
9:          $p_j := Hamilton\_Path(S_j)$
10:         Append $p_j$ to final ordering
11:         for $node(d) \in p_j$ do
12:           $ordered\_nodes.add(node(d))$

13:       $G_{r'} := graphs.Front()$
14:       for $j = last + 1, last + 2, ..., s$ do
15:         for $node(d) \in S_j$ then
16:           $ap(d, r') := |\{i \mid node(d).committed\_rounds[i] \leq r'\}|$
17:           if $ap(d, r') \geq$ n − f then
18:             $node(d).type = solid$
19:           else if $ap(d, r') \geq \frac{n-f}{2}$ then
20:             $node(d).type = shaded$
21:           $G_{r'}.nodes.add(node(d))$
22:           for $node(d_2) \in G_{r'}.nodes$ do
23:             Calculate $G_{r'}.weights[(d, d_2)]$
24:             Calculate $G_{r'}.weights[(d_2, d)]$
25:             if $G_{r'}.weights[(d, d_2)] \geq \frac{n-f}{2}$
                $\vee G_{r'}.weights[(d_2, d)] \geq \frac{n-f}{2}$ then
26:               if $G_{r'}.weights[(d, d_2)] \geq G_{r'}.weights[(d_2, d)]$
                  then
27:                 $G_{r'}.edges.add(e(d, d_2))$
28:               else
29:                 $G_{r'}.edges.add(e(d_2, d))$
30:     else
31:       break
```

Figure 11: Finalizing Transaction Ordering in FairDAG-RL.

Adding edges

For each pair $(d, d_2) \in addable\_edges$, if no edge currently exists 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 $G_2.weight[(d_3, d_5)]$ nor $G_2.weight[(d_5, d_3)]$ reaches the threshold $\frac{n-f}{2} = \frac{3}{2}$. 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

A *tournament* graph is a graph such that there is an edge between each pair of nodes. After adding edges, we check if the graphs are *tournaments* in a round-increasing order. If a graph is a *tournament*, we finalize the ordering of transactions within it as follows and then check the next graph until encountering a *non-tournament* graph.

Condensing dependency graph

In graph theory, a *strongly connected component (SCC)* is a maximal subset of nodes such that for every pair of nodes $(node_1, node_2)$ in the subset, there exists a directed path from $node_1$ to $node_2$ and a directed path from $node_2$ to $node_1$.

For a *tournament* dependency graph G_r, we condense it into G_r^c by finding all *SCCs* with *Tarjan's SCC algorithm*. Figure 9 (c) shows G_2^c the condensed graph of G_2.

Ordering finalization

It is guaranteed that after condensation, there will be one and only one topological sorting of the *SCCs* in G_r^c, which we denote by $S_1, S_2, ..., S_s$, where S_j represents the j-th *SCC* in the topological sorting.

We denote by the S_{last} the last *SCC* that contains a *solid* node. Then, for each S_j such that $j \leq last$, we find a *Hamilton path* p_j of nodes in S_j and append p_j to the final transaction ordering. In Figure 9 (c), S_2 is S_last of G_2^c.

Readding shaded nodes

For the *SCCs* behind S_j, the nodes are all *shaded* nodes. We add these nodes into the next graph $G_{r'}$. For each $node(d)$, we run the following steps:

(1) Classify $node(d)$ based on $ap(d, r')$;
(2) Calculate the weights between $node(d)$ and all existing nodes in $G_{r'}$.
(3) Add edges between $node(d)$ and other nodes if the weight reaches threshold $\frac{\mathbf{n-f}}{2}$.

In Figure 9 (d), $node(d_5)$ is readded into G_4, being classified as a *solid* node. And two edges pointing from $node(d_5)$ are added.

6.4 Ordering Dependency

After illustrating how to construct dependency graphs and finalize transaction ordering in FairDAG-RL, we now discuss *ordering dependencies* in FairDAG-RL. For any transaction T_1 and T_2 with digests d_1 and d_2, either T_1 is dependent on T_2 or T_2 is dependent on T_1. As the transactions are ordered based on the edges in the dependency graphs, intuitively, if an edge $e(d_1, d_2)$ exists, then T_2 is dependent on T_1.

However, for two transactions that are in different replicas, deciding *ordering dependency* is more complicated. We denote by $weight[(d_2, d_1)]_{max}$ the maximal number of the local orderings in which T_2 is ordered before T_1. Even though $node(d_2)$ is in an earlier dependency graph, it is possible that $weight[(d_2, d_1)]_{max} < \frac{\mathbf{n-f}}{2}$, which implies that edge $e(d_2, d_1)$ cannot exist even if the two nodes are in the same dependency graph. Then, if $weight[(d_2, d_1)]_{max} < \frac{\mathbf{n-f}}{2}$, T_2 is dependent on T_1.

To endorse the transactions that are added as nodes earlier, if T_1 is in a earlier dependency graph and $weight[(d_1, d_2)]_{max} \geq \frac{\mathbf{n-f}}{2}$, which implies that edge $e(d_1, d_2)$ could exist, we also say that T_1 is also dependent on T_2.

Definition 6.2. We say that T_2 with digest d_2 is dependent on T_1 with digest d_1 if:

- $weight[(d_2, d_1)]_{max} < \frac{\mathbf{n-f}}{2}$; **or**
- edge $e(d_1, d_2)$ exists; **or**
- $weight[(d_1, d_2)]_{max} >= \frac{\mathbf{n-f}}{2}$ and $node(d_1)$ is in a earlier dependency graph.

7 COMPARING FAIRNESS PROTOCOLS

In this section, we demonstrate how FairDAG-AB and FairDAG-RL outperform Pompe [57] and Themis [30] 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 [15] and HotStuff [54], where a single leader is responsible for collecting local orderings from a quorum of replicas.

Beyond the difference in underlying consensus protocols, Pompe assigns non-overlapping intervals to consensus rounds, permitting only transactions whose *AOI* falls 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 rather than being broadcast, rendering 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 filtering out unfavorable local ordering. Additionally, we demonstrate how FairDAG-AB and FairDAG-RL mitigate these vulnerabilities, ensuring a more resilient and fair transaction ordering process.

In Pompe, a malicious replica can exploit the assigned ordering indicators broadcast by other replicas to selectively choose $\mathbf{n-f}$ local ordering indicators for the transactions it manages. This selective inclusion allows the adversary to manipulate the relative order of transactions with assigned ordering indicators that are close.

Furthermore, if at least $\mathbf{n} - 2\mathbf{f}$ correct replicas do not receive the assigned ordering indicator for a transaction T before a malicious leader replica collects assigned ordering indicators, the leader can exclude T from the new block. Since the leader is only required to collect responses from $\mathbf{n} - \mathbf{f}$ replicas, it can selectively collect from $\mathbf{n} - 2\mathbf{f}$ correct replicas and $\mathbf{f}$ malicious replicas, thereby delaying the final ordering position of T.

While In FairDAG-AB, the *Validity* property of the underlying DAG protocols guarantees that all local ordering indicators from correct replicas will eventually be delivered, and the round-robin or random leader rotation reduces the chance of selectively collecting local ordering indicators. Thus, FairDAG-AB is more resilient against order manipulation than Pompe.

In Themis, for one pair of transactions, there might be only $\mathbf{n}-2\mathbf{f}$ local orderings containing them, where a Byzantine leader could exclude $\mathbf{f}$ local orderings from correct replicas and include $\mathbf{f}$ local orderings from Byzantine replicas that do not contain the transactions. Thus, the threshold of edge direction is $\frac{\mathbf{n}-2\mathbf{f}}{2}$. To guarantee the *γ-Batch-Order-Fairness*, we require that the votes of the opposite direction cannot reach the threshold: $\mathbf{f}+(1-\gamma)(\mathbf{n}-\mathbf{f}) < \frac{\mathbf{n}-2\mathbf{f}}{2}$, 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 received by all replicas. Then, for each transaction pair, there will be at least $\mathbf{n}-\mathbf{f}$ local orderings containing them. Thus, to guarantee the *γ-Batch-Order-Fairness*, we require i.e., $\mathbf{f}+(1-\gamma)(\mathbf{n}-\mathbf{f}) < \frac{\mathbf{n}-\mathbf{f}}{2}$, i.e., $\mathbf{n} > \frac{\mathbf{f}(2\gamma+1)}{2\gamma-1}$. When $\gamma = 1$, FairDAG-RL requires $\mathbf{n} > 4\mathbf{f}$.

7.3 Delaying Transaction Dissemination

In Pompe, since each client transaction is sent to a single replica, a malicious replica can execute a back-running attack by intentionally delaying its dissemination to other replicas. Consequently, the transaction receives a higher assigned ordering indicator, causing it to be placed later in the final ordering.

FairDAG-AB effectively mitigates this issue, as transactions are broadcast to all replicas, and each replica independently generates and disseminates its local ordering indicators.

7.4 Crashed Leader and Asynchronous Network

In Pompe and Themis, if the leader crashes, a recovery mechanism must be initiated to elect a new leader, 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 falling within that time slot cannot be committed or ordered, resulting in 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 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 their leaderless design, inherent to DAG-based consensus protocols, and the Reliable Broadcast Communication (RBC) mechanism, which ensures the *Validity* property even in asynchronous settings.

8 CORRECTNESS PROOF

In this section, we will prove the safety, liveness and fairness properties of FairDAG-AB and FairDAG-RL. Derived from [18, 29, 46], the DAG-Layer has the following properties:

- **Agreement**: if a correct replica commits A_r of leader vertex L_r, then every other correct replica eventually commits A_r.
- **Total Ascending Order**: if a correct replica commits A_r before $A_{r'}$, then $r < r'$ and no correct replica commits $A_{r'}$ before A_r.
- **Validity**: if a correct replica broadcasts a DAG vertex v, then eventually each correct replica will commit a leader vertex L_r such that $v \in A_r$.

Combining the **Agreement** and **Total Ascending Order**, we have the following lemma:

Lemma 8.1. *If a correct replica R commits a series of leader vertices $L\_list^R = L^R_{r_j}, L^R_{r_{j+1}}, \ldots$ and every other correct replica R' will eventually commit $L\_list^{R'} = L^{R'}_{r_j}, L^{R'}_{r_{i+1}}, \ldots,$ such that $\forall j, r_j < r_{j+1}$; $L^R_{r_j} = L^{R'}_{r_j}$; and $C^R_{r_j} = C^{R'}_{r_j}$.*

8.1 Safety

Lemma 8.2. *In* FairDAG-AB*, if a correct replica R assigns transaction T with digest d* assigned ordering indicators $d.AOI^R$*, then every other correct replica R' will eventually assign T with $d.AOI^{R'}$ such that $d.AOI^R = d.AOI^{R'}$.*

Proof. According to FairDAG-AB algorithm, we know that a correct replica deterministically calculates $d.AOI$ based on C_r, the causal history of the lowest-round leader vertex L_r such that after the replica commits L_r, there are at least $\mathbf{n}-\mathbf{f}$ *committed_ois* of d in C_r.

Then, we prove the lemma by contradiction. We assume that R calculates $d.AOI^R$ based on the causal history C^R_r of leader vertex L^R_r. If R' never assigns T with an *assigned ordering indicator*, then R' never commits C^R_r. If R' assigns T with an *assigned ordering indicator* different from T', then R' must have committed some different $C'^{R'}_r$. Both cases contradict 8.1. □

Lemma 8.3. *In* FairDAG-AB*, after A_r is committed and processed, for a transaction T with digest d that has no* assigned ordering indicator *but a* lowest possible assigned ordering indicator $d.LPAOI_r$*, if T eventually gets an* assigned ordering indicator *$d.AOI$, then $d.AOI \geq d.LPAOI_r$.*

Proof. We denote by $highest\_oi\_list_r$ and $d.seen\_ois_r$, respectively, the $highest\_oi\_list$ and $d.seen\_ois$ after A_r is committed. According to the FairDAG-AB algorithm, $d.LPOAI^r$ is the $(\mathbf{f}+1)$-th lowest value of lp_ois_r where $lp\_ois_r[i] :=$ $min(d.seen\_ois_r[i], highest\_oi\_list_r[i])$, $1 \leq i \leq \mathbf{n}$. As the DAG grows, each new $d.committed[i]$ will be not smaller than $min(d.seen\_ois_r[i], highest\_oi\_list_r[i])$. Thus, as $d.AOI :=$ $sorted(d.committed\_ois[i])[\mathbf{f{+}1}]$, where $\forall i, d.committed\_ois[i] \geq$ $lp\_ois_r[i]$, it holds true that $d.AOI \geq d.LPAOI_r$. □

Lemma 8.4. *In* FairDAG-AB*, for any two transactions T_1 and T_2 with digests d_1 and d_2, if $d_1.AOI < d_2.AOI$, then d_1 will be ordered before d_2 in the final ordering.*

Proof. We denote by or_j the round such that transaction digest d_j is ordered, getting an *assigned ordering indicator* lower than $LPAOI_{min}$, i.e., $d_j.AOI < LPAOI_{min}$.

Now we prove by the lemma by contradiction. Assuming that d_2 is ordered before d_1, then obviously, $or_1 \geq or_2$.

- If $or_1 = or_2$, then according to FairDAG-AB algorithm, d_1 and d_2 are ordered based on assigned ordering indicator. Thus, with a lower AOI, d_1 would be ordered before d_2, contradicting with our assumption.

- If $or_1 > or_2$, then by the definition of $LPAOI_{min}$ we know that $d_2.AOI < LPAOI_{min} \leq d_1.LPAOI_{or_2}$. Also, from Lemma 8.3 we know that eventually d_1 will get $d_1.AOI \geq d_1.LPAOI_{or_2}$. Thus, we have $d_1.AOI > d_2.AOI$, which contradicts the fact that $d_1.AOI < d_2.AOI$.

In summary, it holds true that if $d_1.AOI < d_2.AOI$, then d_1 will be ordered before d_2 in the final ordering. □

Theorem 8.5. *(*FairDAG-AB *SAFETY) In* FairDAG-AB*, if a correct replica orders transaction T at position p in the final ordering, then every correct replica will eventually order T at position p.*

Proof. From Lemma 8.2 we know that every correct replica will eventually assign the same *assigned ordering indicator* to T. From Lemma 8.4 we know that all transactions with *assigned ordering indicators* are ordered in an ascending order of AOI. Combining the two claims above, we conclude that every correct replica will eventually order T at the same position in the final ordering. □

Theorem 8.6. *(*FairDAG-RL *SAFETY) In* FairDAG-RL*, if a correct replica orders transaction T at position p in the final ordering, then every correct replica will eventually order T at position p.*

Proof. After committing a leader vertex, each correct replica uses a deterministic method to construct dependency graphs and finalize transaction order. Combining this with Lemma 8.1, we know that safety holds for FairDAG-RL. □

8.2 Liveness

We claim the following assumption holds, which is necessary for the liveness property.

Assumption. If a correct replica R receives transaction T, then every correct replica will eventually receive transaction T.

Lemma 8.7. *In* FairDAG-AB*, if a transaction T with transaction d is received by correct replicas, then T will eventually get an* assigned ordering indicator.

From Assumption 8.2 we know that all correct replicas will receive T will propose a DAG vertex containing d and a corresponding ordering indicator. From the **Validity** of DAG layer, we know that all these DAG vertices will be committed. Thus, d will get at least $\mathbf{n} - \mathbf{f}$ *committed_ois* from correct replicas and then get an *assigned ordering indicator.*

Lemma 8.8. *For transactions T_1 and T_2 with digests d_1 and d_2, if only T_1 has an* assigned ordering indicator *and T_2 has an LPAOI lower than $d_1.AOI$, i.e., $d_2.AOI = \infty \wedge d_2.LPAOI < d_1$, then eventually,*

- *either T_2 gets an* assigned ordering indicator*;*
- *or $d_2.LPAOI$ becomes larger than $d_1.AOI$.*

Proof. If T_2 is received by correct replicas, from Lemma 8.7, we know that eventually T_2 gets an *assigned ordering indicator.*

Next, we discuss the case that T_2 is received by only faulty replicas. Since $d_2.LPAOI$ is the $(\mathbf{f} + 1)$-th lowest value of lp_ois, $\mathbf{n} - \mathbf{f}$ values of which are the *highest_oi_list* values from correct replicas, thus, $d_2.LPAOI$ is not greater than *highest_oi_list* value from at least one correct replica. Due to the **Valifity** property of the DAG layer, DAG keeps growing and the *highest_oi_list* value from each correct replica will eventually be higher than $d_1.AOI$. □

Theorem 8.9. *(*FairDAG-AB *Liveness) If a transaction T with digest d is received by correct replicas, then T will eventually be ordered.*

Proof. From Lemma 8.7 we know that T will eventually get an $d.AOI$. From Lemma 8.8 we know that eventually there will be no other transaction that has an $LPAOI$ lower than $d.AOI$. Thus, eventually it will be satisfied that $d_1.AOI < LPAOI_{min}$, and then T will be ordered. □

Lemma 8.10. *In* FairDAG-RL*, each dependency graph G will eventually become a tournament.*

Proof. Since only *solid* and *shaded* nodes can be added into a dependency graph, for each $node(d)$ in G, there are at least $\frac{\mathbf{n}-\mathbf{f}}{2} > \mathbf{f} + 1$ local orderings that contain d. Thus, d will be received by all correct replicas. And then eventually, due to the **Validity** property of the DAG layer, there will be a round r such that $ap(d, r) \geq \mathbf{n} - \mathbf{f}$.

Thus, for each pair of nodes $node(d_1)$ and $node_2$ in G, there will be a round r such that $ap(d_1, r) \geq \mathbf{n} - \mathbf{f}$ and $ap(d_2, r) \geq \mathbf{n} - \mathbf{f}$. Thus, at least one of $G.weights[(d_1, d_2)]$ and $G.weights[(d_2, d_1)]$ will reach the threshold $\frac{\mathbf{n}-\mathbf{f}}{2}$. Then, eventually, there will be an edge between each pair of nodes in G, i.e., G will be a tournament. □

Theorem 8.11. *(*FairDAG-RL *Liveness) If a transaction T with digest d is received by correct replicas, then T will eventually be ordered.*

Proof. Due to the **Validity** property of the DAG layer, there will eventually be a round r such that $ap(d, r) \geq \mathbf{n} - \mathbf{f}$, and then $node(d)$ will be added into a dependency graph G.

From Lemma 8.10 we know that G will eventually be a tournament. If $node(d)$ is *solid*, the T will be ordered. If $node(d)$ is *shaded*, the T might be ordered. Even if *shaded* $node(d)$ has to be added into the next dependency graph, eventually $node(d)$ will be added as a *solid* node and T will be ordered. □

8.3 FairDAB-AB: Ordering Linearizability

To prove ordering linearizability, we introduce the following denotations:

- $d.\_ois^C$: the ordering indicators of d from correct replicas.
- $d.low\_oi^C$: the lowest value in $d.\_ois^C$.
- $d.high\_oi^C$: the highest value in $d.\_ois^C$.

Lemma 8.12. *For each transaction T with digest d, $d.low\_oi^C \leq d.AOI \leq d.high\_oi^C$.*

Proof. As the $d.AOI$ is the $(\mathbf{f} + 1)$-th lowest value of a subset of $d.committed\_ois$ with at least $\mathbf{n} - \mathbf{f} \geq 2\mathbf{f} + 1$ values. Among the $\mathbf{f} + 1$ lowest values of the subset, at least one is in $d.\_ois^C$. Thus, $d.low\_oi^C \leq d.AOI$. Similarly, among the $\mathbf{f} + 1$ highest values of the subset, at least one is from a correct replica. Thus, $d.AOI \leq d.high\_oi^C$. □

Theorem 8.13. *(Ordering Linearizability) For two transactions T_1 and T_2 with digests d_1 and d_2, if $d_1.high\_oi^C < d_2.low\_oi^C$, then T_1 will be ordered before T_2 in the final ordering.*

Proof. From Lemma 8.12 we know that $d_1.AOI \leq d_1.high\_oi^C$ and $d_2.low\_oi^C \leq d_2.AOI$. Thus, $d_1.AOI < d_2.AOI$. From Lemma 8.4 we know that transactions are ordered based on *AOI* values. Thus, T_1 will be ordered before T_2 in the final ordering. □

8.4 FairDAG-RL: γ-Batch-Order-Fairness

Lemma 8.14. *For any two transactions T_1 and T_2 with digests d_1 and d_2, if $\gamma(\mathbf{n}-\mathbf{f})$ correct replicas receive T_1 before T_2, then $\frac{\mathbf{n}-\mathbf{f}}{2} > weight[(d_2, d_1)]_{max}$.*

Proof. As $\gamma(\mathbf{n}-\mathbf{f})$ correct replicas receive T_1 before T_2, then $weight[(d_2, d_1)]_{max} \geq \mathbf{f}+(1-\gamma)(\mathbf{n}-\mathbf{f})$. As $\mathbf{n} > \frac{(2\gamma+1)\mathbf{f}}{(2\gamma-1)}$, $\frac{1}{2} < \gamma \leq 1$, we have:

$$\mathbf{n} > \frac{(2\gamma+1)\mathbf{f}}{(2\gamma-1)} \iff (2\gamma-1)(\mathbf{n}-\mathbf{f}) > 2\mathbf{f} \iff$$
$$(2\gamma-2)(\mathbf{n}-\mathbf{f})+\mathbf{n}-\mathbf{f} > 2\mathbf{f} \iff \mathbf{n}-\mathbf{f} > 2\mathbf{f}+(2-2\gamma)(\mathbf{n}-\mathbf{f}) \iff$$
$$\frac{\mathbf{n}-\mathbf{f}}{2} > \mathbf{f}+(1-\gamma)(\mathbf{n}-\mathbf{f}) \iff \frac{\mathbf{n}-\mathbf{f}}{2} > weight[(d_2, d_1)]_{max}$$

□

Combing Lemma 8.14 with the definition of *ordering dependency*, we have:

Lemma 8.15. *For any two transactions T_1 and T_2 with digests d_1 and d_2, if $\gamma(\mathbf{n}-\mathbf{f})$ correct replicas receive T_1 before T_2, then T_2 is dependent on T_1, i.e., $T_1 \longrightarrow T_2$.*

Lemma 8.16. *For any two transactions T_1 and T_2 with digests d_1 and d_2 in a tournament dependency graph G, if $\gamma(\mathbf{n}-\mathbf{f})$ correct replicas receive T_1 before T_2, then after condensing G and topologically sorting the* SCCs, *$node(d_1)$ is in the same or an earlier* SCC *than $node(d_2)$.*

Proof. According to graph theory, we know that a tournament dependency graph G can be condensed into a graph with multiple *SCCs*, and after topologically sorting the *SCCs*, there is a unique list $S_1, S_2, ..., S_s$. It holds that for any two *SCCs* S_a and S_b, if $a < b$, then $\forall node(d_a) \in S_a, \forall node(d_b) \in S_b$, edge $e(d_a, d_b)$ exists in G.

We prove the lemma by contradiction. Assuming that $node(d_2)$ is in an earlier *SCC* than $node(d_1)$, then $e(d_2, d_1)$ exists in G. However, it contradicts Lemma 8.14, from which we know $\frac{\mathbf{n}-\mathbf{f}}{2} > weight[(d_2, d_1)]_{max}$, i.e., $e(d_2, d_1)$ cannot exist. □

Lemma 8.17. *$\forall node(d_1) \in G_{r_1}, \forall node(d_1) \in G_{r_2}, r_1 < r_2$, if $node(d_1)$ is* solid, *then transaction T_2 is dependent on T_1.*

Proof. If $node(d_2)$ was not added as a node into G_{r_1}, then $ap(d_2, r_1) < \frac{\mathbf{n}-\mathbf{f}}{2}$ when $ap(d_1, r_1) \geq \mathbf{n}-\mathbf{f}$. Thus, there are more than $\frac{\mathbf{n}-\mathbf{f}}{2}$ committed local orderings in which T_1 is before T_2. Then $weight[(d_1, d_2)]_{max} \geq \frac{\mathbf{n}-\mathbf{f}}{2}$, then, combining with the fact that $node(d_1)$ is in an earlier dependency graph, T_2 is dependent on T_1.

If $node(d_2)$ was added as a node into G_{r_1} but readded into G_{r_1}, then it implies that $node(d_2)$ was in an *SCC* later than the last *SCC* that contains a *solid* node. Thus, solid $node(d_1)$ has an edge to $node(d_2)$, i.e., $e(d_1, d_2)$ exists, and then T_2 is dependent on T_1. □

Theorem 8.18. *(γ-Batch-Order-Fairness) For any two transactions T_1 and T_2 with digests d_1 and d_2, if $\gamma(\mathbf{n}-\mathbf{f})$ correct replicas receive T_1 before T_2, then T_1 will be ordered no later than T_2.*

Proof. We denote by G_{r_1} and G_{r_2} in which T_1 and T_2 are ordered, respectively. If $r_1 < r_2$, then obviously T_1 is ordered before T_2 in the final transaction ordering. If $r_1 = r_2$, then from Lemma 8.17 we know that $node(d_1)$ is in the same or an earlier *SCC* than $node(d_2)$. Thus, T_1 is ordered no later than T_2 in the final transaction ordering.

If $r_1 > r_2$, then $node(d_2)$ is *shaded*. Otherwise, assuming $node(d_2)$ is *solid*, then there are at least $\mathbf{n}-\mathbf{f}$ *committed local ordering* containing d_2 when $node(d_2)$ is added into G_{r_2}. From Lemma 8.14 we know that $G_{r_2}.weights[(d_2, d_1)] < \frac{\mathbf{n}-\mathbf{f}}{2}$, then $G_{r_2}.weights[(d_1, d_2)] > \frac{\mathbf{n}-\mathbf{f}}{2}$, and edge $e(d_1, d_2)$ exists. Therefore, $node(d_1)$ should be ordered in G_{r_2} as it has a path to a *solid* node, contradicting the fact that $r_1 > r_2$. Thus, $node(d_2)$ must be *shaded* in G_{r_2}. Hence, there is some *solid* node $node(d_s)$ such that there is a path p_2 from $node(d_2)$ to $node(d_s)$. We denote by p_1 the path in G_{r_1} from the first ordered node to $node(d_1)$. Combining the following information:

- from Lemma 8.17 we know that all transactions on p_1, including $T-1$, are dependent on transaction T_s of $node(d_s)$;
- from Lemma 8.15 we know that T_2 is dependent on T_1.
- along path p_2, each transaction is dependent on the previous one, from T_s to T_2.

Thus, the transactions on p_1 and p_2 form a *cyclic dependent batch* b. For the transactions that are ordered in G_{r_2} later than T_s, they are in the same *SCC* as T_s and then can be added into b. Therefore, even if $r_1 > r_2$, T_1 and T_2 are in the same *cyclic dependent batch* in the final ordering, i.e., T_1 is ordered *no later* than T_2. □

9 EVALUATION

This section evaluates FairDAG-AB and FairDAG-RL by comparing their performance with other baseline protocols. We implement the protocols [25] using Apache ResilientDB (Incubating) [1, 22]. 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 runs on a separate machine. The setup of geo-distributed experiments can be found in Section 9.1.

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

- Pbft [15]: A single-proposer consensus protocol without fairness guarantees, $\mathbf{n} \geq 3\mathbf{f}+1$.
- Pompe [57]: an absolute fairness protocol running on top of Pbft, $\mathbf{n} \geq 3\mathbf{f}+1$.
- Themis [30]: a relative fairness protocol running on top of Pbft, $\mathbf{n} > \frac{\mathbf{f}(2\gamma+2)}{2\gamma-1}$ (see node requirement explanation in Section 7.2).
- RCC [22]: a multi-proposer protocol that runs concurrent Pbft instances without fairness guarantees, $\mathbf{n} \geq 3\mathbf{f}+1$.
- Tusk [18], a multi-proposer DAG-based consensus protocol without fairness guarantees, $\mathbf{n} \geq 3\mathbf{f}+1$.

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 modified Tusk with weak edges.

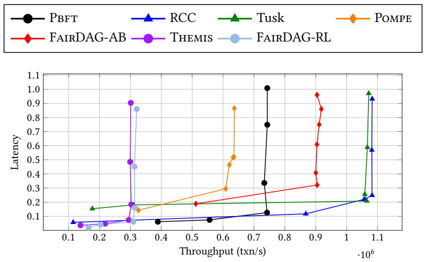


Figure 12: Throughput vs latency with f = 8.

9.1 Scalability

In the scalability experiments, we mainly measure two performance metrics:

(1) *Throughput* – the maximum number of transactions per second for which the system completes consensus and orders.

(2) *Client Latency* – the average duration between the time a client sends a transaction and the time the client receives **f** + 1 matching responses.

We compare the performance of different protocols with varying **f**, the maximal number of faulty replicas allowed. The value of **f** varies from 5 to 8. With the same **f**, different protocols have different replica numbers. For example, when **f** = 5, THEMIS has **n** = 21 replicas while other protocols have **n** = 16 replicas.

The two performance metrics are highly related to the workload that the replicas process. As shown in Figure 12, where we set **f** = 8, as the workload increases, throughput increases until the pipeline is fulfilled by the transactions. Then, after the throughput reaches the peak, latency increases as the workload increases. We define by *optimal point* the point with the lowest latency while maintaining the highest throughput. The scalability experiments are conducted at the *optimal points* of each protocol with varying **f**.

Throughput. Figure 13 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 **f** = 5 and **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 **f** = 5 and **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 **f** = 5 and **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 **f** = 5. As **f** grows, FAIRDAG-RL has a lower latency than THEMIS, which is 20.9% lower when **f** = 8. FAIRDAG-RL achieves a lower latency because THEMIS needs **f** more correct replicas to guarantee fairness, which causes higher overhead for both consensus and ordering. By comparing the latency of THEMIS with **f** = 6 and FAIRDAG-RL with **f** = 8, we can verify this claim. Then, with the same replica number **n** = 25, FAIRDAG-RL achieves a 4.6% higher latency than THEMIS.

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 **f** = 8 and distributed the replicas evenly across the regions. Figure 13 (c, d) show that in the geo-distributed setting, the latencies of all the protocols are high and increase with the number of regions. This is due to the higher inter-region network latency. Furthermore, FairDAG has higher throughputs than the other fairness protocols.

We observe that for all protocols except the relative ordering protocols, FAIRDAG-RL and THEMIS, increasing the batching parameters allowed us to achieve throughput comparable to that in the single-region setting. In FAIRDAG-RL and THEMIS, larger batch sizes lead to an increase in the overhead of the fairness layer, which grows quadratically with the number of transactions per round. 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 under geo-scale settings with limited bandwidth and higher message delays.

Data-dependent fairness. RASHNU [36] proposes a technique to reduce the overhead of the fairness layer in relative fairness protocols by computing edge directions only for data-dependent transactions. This design is orthogonal to both FAIRDAG-RL and THEMIS. We implement two RASHNU-enhanced variants, called THEMIS-RASHNU and FAIRDAG-RL-RASHNU, and compare them to the Themis and FairDAG-RL.

In this experiment, we implemented a transaction workload with keys following a Zipfian distribution. We fixed **f** = 8 and varied the skewness parameter *s* from 0.01 to 0.99, where higher skewness indicates greater data dependency among transactions. The results, presented in Figure 14, 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 computing edge directions when transactions are less interdependent. Notably, FAIRDAG-RL benefits more significantly from RASHNU than THEMIS does, highlighting the performance potential of DAG-based consensus protocols once the fairness-layer bottleneck is mitigated.

9.2 Tolerance to Byzantine Behavior

In the following part, we will discuss the impact of Byzantine behaviors on the performance and final transaction ordering of the fairness protocols.

Faulty leader. In this experiment, we make the leader replica in PBFT faulty, which would trigger a view-change to replace the faulty leader. While for TUSK, as the leader vertices are randomly selected and there is no stable leader, we make a replica faulty. Figure 15 shows how the faulty leader affects the performance of

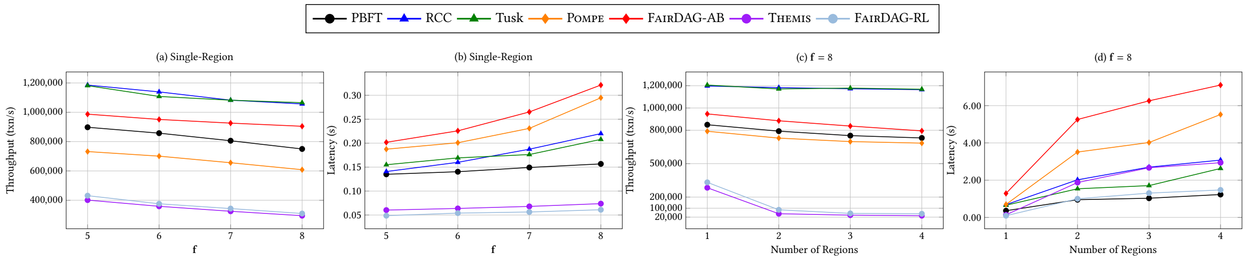


Figure 13: Performance of FairDAG and baseline protocols, with varying f (a,b), and varying number of regions (c,d).

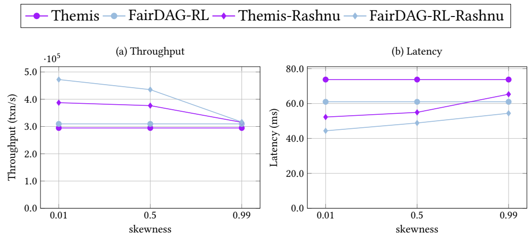


Figure 14: Performance of Rashnu-enhanced variants vs. relative fairness protocols.

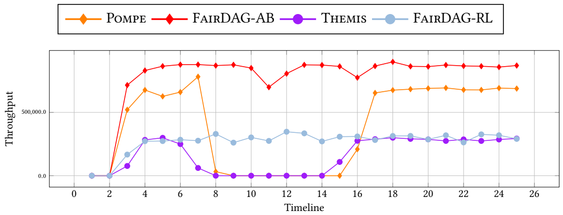


Figure 15: Real-Time Throughput with a Faulty Leader.

Themis and Pompe. At time 7, the leader becomes faulty and stops participating in consensus. With a period without progress, Pbft timers are out within other replicas, and then 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, due to the random leader vertex selection of Tusk, running on top of it, the performance of FairDAG-RL and FairDAG-AB is not affected.

Adversarial Manipulation. We conduct two experiments in which malicious replicas attempt to manipulate transaction ordering. We evaluated the fairness quality of the fairness protocols under two malicious behaviors: (1) Reversing order: in each round, the faulty replicas reverse the local ordering indicators of the received transactions. (2) Targeted delay: the faulty replicas assign large values to the local ordering indicators of the targeted transactions.

In the experiments, we measure the ratio of correctly ordered pairs across different *Dist* values. For two transactions T_1 and T_2, we say that they are correctly ordered if T_1 is ordered before T_2 and $weight[(d_1, d_2)] > \frac{n}{2}$, and vice versa. The value of $Dist(T_1, T_2)$ is computed as $|weight[(d_1, d_2)] - weight[(d_2, d_1)]|$. For the *reversing order* attack, we evaluate all transaction pairs; for the *targeted delay* attack, we consider only the transaction pairs that involve the victim transactions.

We conduct the experiments with $\mathbf{f} = 10$, and vary $\mathbf{f}_a$, the actual number of malicious replicas, from 0 to 10. For example, Themis-7 denotes Themis with $\mathbf{f}_a = 7$. Since Themis and FairDAG-RL differ in their total number of replicas **n**, we normalize the x-axis values accordingly to ensure comparability. The results of FairDAG-RL and Themis are shown in Figure 16 (a-d), while the results of FairDAG-AB and Pompe are in Figure 16 (e,f).

As shown in Figure 16, FairDAG-RL and FairDAG-AB consistently demonstrate superior resilience against adversarial ordering manipulation across all experimental settings, compared to Themis and Pompe, respectively. These results substantiate our claim that FairDAG effectively mitigates manipulation of quorum selection through the properties inherent in the DAG-based consensus layer.

10 RELATED WORK

Fairness in BFT

In traditional Byzantine Fault Tolerance (BFT) research, protocols are designed to ensure both safety and liveness in the presence of malicious replicas. While these protocols do not explicitly guarantee fair transaction ordering, they mitigate unfair ordering to some extent. Protocols such as HotStuff [54], which employ leader rotation in a round-robin manner [3, 19, 20, 26, 34, 48], provide each participant with the opportunity to propose a block. Multi-proposer approaches, including concurrent consensus protocols [22, 27, 47] and DAG-based protocols [6, 18, 29, 45, 46], enable multiple participants to propose blocks concurrently, ordering them globally through either predetermined or randomized mechanisms. Although these protocols reduce reliance on a single leader and distribute transaction ordering authority, a malicious participant can still manipulate the ordering of transactions within the blocks it proposes.

Some protocols seek to eliminate the block proposers' oligarchy over the selection and ordering of transactions within blocks by incorporating *censorship resistance* [5, 12, 35, 53]. In these protocols, transactions are encrypted and remain indecipherable until the transaction ordering is determined. However, block proposers can still engage in censorship based on metadata, such as IP addresses, or prioritize their own transactions, since they possess knowledge of the mapping from their transactions to the encrypted ciphertexts. More importantly, these censorship-resistant protocols fail to

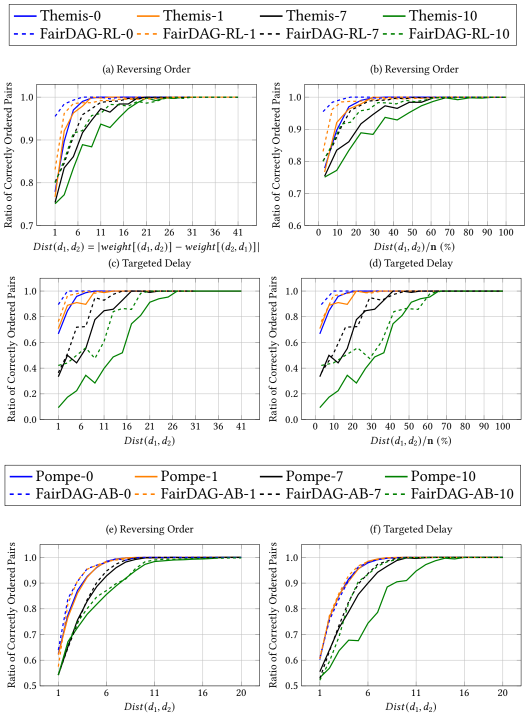


Figure 16: Fairness quality of the fairness protocols under malicious ordering manipulation attacks.

guarantee fairness, as the ordering of transactions within blocks remains fully controlled by the proposers.

Besides Pompe and Themis, there are other fairness protocols. Wendy [32] guarantees *Timed-Relative-Fairness* similar to *Ordering Linearizability*, but it relies on synchronized local clocks, which are impractical in asynchronous networks. Aequitas [31] and Quick-Order-Fairness [13] guarantee batch-order-fairness but suffer from liveness issues due to the existence of infinite *Condorcet Cycles*, which Themis solves via a *batch unspooling* mechanism. Rashnu [36] improves Themis performance by identifying data-dependent transactions that access the same resource, guaranteeing *γ-Batch-Order-Fairness* between data-dependent transactions. However, Rashnu suffers from the same issues mentioned in Section 7 as Themis because the ordering method is unchanged.

Multi-Proposer protocols

A significant amount of research [4, 6, 44, 45] has been dedicated to reducing the latency in DAG-based protocols. These works employ various techniques, including *pipelining DAG waves*, *fast commit rules*, *multi-anchor*, and *uncertified DAG*, to enhance the efficiency and speed of these protocols.

Concurrent consensus protocols [22, 27, 47] represent a distinct category of multi-proposer consensus protocols. These protocols run multiple concurrent consensus instances independently, generating a final ordering by globally ordering the committed blocks in each instance, round by round. However, these protocols face several challenges that make them unsuitable as the underlying consensus mechanisms for fairness protocols. First, the presence of a straggler instance, which lags behind the others, can substantially decrease throughput and increase system latency. Second, since the progress of different instances is not synchronized, a malicious leader of an instance could manipulate transaction ordering by delaying the block proposal until other replicas have proposed their blocks. DAG-based protocols address these issues effectively, as a replica can proceed to the next round once there are $\mathbf{n} - \mathbf{f}$ committed DAG vertices in the current round, ensuring more efficient and reliable progression.

11 CONCLUSION

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

REFERENCES

[1] 2024. Apache ResilientDB (Incubating). https://resilientdb.incubator.apache.org/

[2] 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.

[3] Ittai Abraham, Guy Gueta, and Dahlia Malkhi. 2018. Hot-stuff the linear, optimal-resilience, one-message BFT devil. *CoRR, abs/1803.05069* (2018).

[4] 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).

[5] 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.

[6] 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).

[7] 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

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

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

[10] 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/

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

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

[13] 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.

[14] 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

[15] 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

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

[17] 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).

[18] 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

[19] 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.

[20] Neil Giridharan, Heidi Howard, Ittai Abraham, Natacha Crooks, and Alin Tomescu. 2021. No-Commit Proofs: Defeating Livelock in BFT. https://eprint.iacr.org/2021/1308

[21] 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

[22] 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

[23] 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.

[24] 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

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

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

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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

[35] 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.

[36] 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.

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

[38] 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

[39] 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.

[40] 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.

[41] 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.

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

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

[44] Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, and Kartik Nayak. 2024. Sailfish: Towards Improving the Latency of DAG-based BFT. Cryptology ePrint Archive, Paper 2024/472.

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

[46] 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.

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

[48] Xiao Sui, Sisi Duan, and Haibin Zhang. 2022. Marlin: Two-Phase BFT with Linearity. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 54–66. https://doi.org/10.1109/DSN53405.2022.00018

[49] 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

[50] 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

[51] 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.

[52] 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.

[53] 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).

[54] 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

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

[56] 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

[57] 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.