Published: 05 December 2025



RESEARCH-ARTICLE

# DAG of DAGs: Order-Fairness Made Practical

**HEENA NAGDA**, University of Pennsylvania, Philadelphia, PA, United States

**SIDHARTH SANKHE**, University of Pennsylvania, Philadelphia, PA, United States

**SAKSHI SINHA**, Stony Brook University, Stony Brook, NY, United States

**KEON ATTARHA**

**MOHAMMAD JAVAD AMIRI**, Stony Brook University, Stony Brook, NY, United States

**BOON THAU LOO**, University of Pennsylvania, Philadelphia, PA, United States

**Open Access Support** provided by:

**Stony Brook University**

**University of Pennsylvania**

# DAG of DAGs: Order-Fairness Made Practical

HEENA NAGDA, University of Pennsylvania, USA
SIDHARTH SANKHE, University of Pennsylvania, USA
SAKSHI SINHA, Stony Brook University, USA
KEON ATTARHA, Plano West Senior High School, USA
MOHAMMAD JAVAD AMIRI, Stony Brook University, USA
BOON THAU LOO, University of Pennsylvania, USA

Ensuring order-fairness in distributed data management systems deployed in untrustworthy environments is crucial to prevent adversarial manipulation of transaction ordering, particularly in unpredictable markets where transaction order directly influences financial outcomes. While Byzantine Fault-Tolerant (BFT) consensus protocols guarantee safety and liveness, they inherently lack mechanisms to enforce order-fairness, exposing distributed systems to attacks such as frontrunning and sandwiching. Previous attempts to integrate order-fairness have often introduced substantial performance overhead, largely due to limitations of the underlying consensus protocols. This paper presents DAG of DAGs (DoD), a high-performance order-fairness protocol designed on top of DAG-based BFT consensus protocols. By leveraging the high throughput and resilience of DAG-based protocols, DoD addresses the performance limitations of existing order-fairness solutions. DoD's novel DAG of DAGs architecture enables seamless integration of order fairness with BFT consensus protocols. Through concurrent block proposals and a wave-based leader election mechanism, DoD significantly improves resilience against adversarial manipulation. A prototype implementation and experimental evaluation demonstrate that DoD effectively provides order fairness with minimal performance overhead.

CCS Concepts: • **Information systems → Distributed database transactions**; • **Computing methodologies → Distributed algorithms**.

Additional Key Words and Phrases: Order-Fairness, Byzantine Fault Tolerance, DAG-based BFT Protocols

**ACM Reference Format:**
Heena Nagda, Sidharth Sankhe, Sakshi Sinha, Keon Attarha, Mohammad Javad Amiri, and Boon Thau Loo. 2025. DAG of DAGs: Order-Fairness Made Practical. *Proc. ACM Manag. Data* 3, 6 (SIGMOD), Article 312 (December 2025), 27 pages. https://doi.org/10.1145/3769777

## 1 Introduction

Distributed data management systems have historically been under the governance of central authorities, typically operating under the assumption of trusted environments and only tolerating crash failures. However, recent advancements in blockchain technology, along with significant events such as the collapse of multinational banks [65], the unexpected deactivation of user accounts [11], and glitches in major stock exchanges [57], have led to a rapid increase in the development of

This work is supported by NSF IIS-2436080: EAGER: Synthesizing and Optimizing Declarative Smart Contracts, 2025.
Authors' Contact Information: Heena Nagda, University of Pennsylvania, Philadelphia, PA, USA, hnagda@seas.upenn.edu; Sidharth Sankhe, University of Pennsylvania, Philadelphia, PA, USA, sankhe@seas.upenn.edu; Sakshi Sinha, Stony Brook University, Stony Brook, NY, USA, sakshi.sinha@stonybrook.edu; Keon Attarha, Plano West Senior High School, Plano, TX, USA, Keonattarha1@gmail.com; Mohammad Javad Amiri, Stony Brook University, Stony Brook, NY, USA, amiri@cs.stonybrook.edu; Boon Thau Loo, University of Pennsylvania, Philadelphia, PA, USA, boonloo@seas.upenn.edu.


ACM 2836-6573/2025/12-ART312
https://doi.org/10.1145/3769777

decentralized systems. These systems enable collaboration among multiple untrusting parties on data management, highlighting the prevalence of untrusted environments and shifting the focus of fault tolerance toward tolerating Byzantine failures. As a result, Byzantine fault-tolerant (BFT) protocols have become a fundamental component of distributed data management systems operating in untrusted settings [2–5, 8, 14, 29, 31, 34, 51, 53, 54, 64, 66, 66, 67, 69].

With the growth of distributed data management systems deployed in untrustworthy environments, adversarial transaction order manipulation has become a significant concern due to the lack of trusted intermediaries [10, 22, 26, 30, 39, 55, 72]. For instance, in a volatile stock exchange market where share prices fluctuate rapidly, the order of trade transactions is critical in determining profits or losses, potentially amounting to billions of dollars. This environment incentivizes honest participants to act strategically and motivates bad actors to manipulate trade orders for financial gain. To mitigate such risks, order-fairness becomes crucial in preventing adversarial manipulation of transactions. Traditional centralized stock exchanges have enforced fairness for decades through strict monitoring and prohibiting manipulative practices. As a result, order-fairness was not a significant concern in systems until recently.

Adversarial manipulation of transactions order is studied in the decentralized finance (DeFi) domain [10, 22, 26, 30, 39, 55, 72] where transaction proposers make profit by including, excluding, or reordering transactions within blocks, known as maximal extractable value (MEV) [22]. Consider an exchange transaction to buy a particular asset. A malicious proposer can perform a front-running sandwich attack by placing the buy transaction between two buy and sell transactions (initiated by the malicious proposer) to manipulate asset prices. The proposer buys assets for a lower price to let the victim buy at a higher value and then sells them again, typically at a higher price afterwards. Adversarial manipulation of transactions in Ethereum resulted in extracting more than $686M in revenue from unsophisticated users ($1.38B across all EVM-powered networks) [17]. Other than profitability, the order of transactions might affect their validity, typically when multiple transactions access limited assets, e.g., a ticket booking scenario for a highly sought-after event where the number of available tickets is limited, and through order manipulation, tickets are purchased by users who are not supposed to get one.

Distributed systems deployed in untrusted environments utilize Byzantine Fault-Tolerant (BFT) consensus protocols to tolerate malicious failures. These protocols provide safety, ensuring that transactions are executed in a consistent order, and liveness, guaranteeing that all valid transactions are eventually processed. However, they do not inherently provide order-fairness, leaving the door open for manipulation. This gap has driven a growing body of research on integrating order-fairness into BFT consensus protocols, ensuring that transactions are executed in an order that reflects their arrival times in the network [16, 36, 37, 41, 42, 50, 71].

Existing order-fairness protocols, e.g., Themis [36], suffer from considerable performance overhead, mainly due to the time needed to establish a fair order among all transactions. Even Rashnu [50], while focusing on fairness only for data-dependent transactions, still faces performance overhead due to the limitations of the underlying consensus protocol. In untrustworthy environments like blockchains, the consensus protocol commonly becomes a significant bottleneck due to the involvement of a large number of nodes in the consensus process.

In recent years, there has been significant interest in developing high-performance BFT consensus protocols to address the communication overhead of traditional leader-based consensus protocols caused by their sequential transaction processing approach, where a leader node sequentially disseminates transaction data while proposing their execution order. Recent studies suggest that allowing multiple nodes to propose blocks concurrently could substantially enhance overall system performance. This shift creates a Directed Acyclic Graph of blocks, with edges representing previous

blocks' references, resulting in a family of DAG-based BFT protocols, e.g., DAG-Rider [35], Narwhal and Tusk [23], Bullshark [62], Shoal++ [7], Autobahn [28], and Shardag [18].

DAG-based protocols achieve high throughput by allowing concurrent block proposals and separating data dissemination from consensus routine. These features make them well-suited for addressing the performance limitations of existing order-fairness protocols, which are often bottlenecked by their underlying consensus protocols. Moreover, in DAG-based BFT consensus protocols, leader election occurs randomly in a wave-by-wave manner for previous rounds, preventing the leader of any given round from being known in advance. This approach enhances resilience against attacks by reducing opportunities for malicious nodes to manipulate the order of transactions.

Ensuring order-fairness in DAG-based systems, however, remains challenging. The inherent flexibility of DAG structures, which allows concurrent block proposals, introduces new attack surfaces for order manipulation. Unlike traditional consensus protocols, where transactions are sequentially ordered, DAGs enable more complex inter-block relationships. This complexity makes DAG-based protocols particularly vulnerable to inter-block reordering attacks, where adversaries strategically position their blocks ahead of victims' transaction blocks. Without strong order-fairness guarantees, the advantages of DAGs in terms of throughput and resilience could be overshadowed by vulnerabilities to sophisticated adversarial strategies.

Our proposed system, *DoD*, addresses this challenge by introducing the *DAG of DAGs* architecture. This architecture builds upon the advantages of DAG-based consensus by layering an additional DAG structure that explicitly manages the ordering of transactions within a batch. DoD ensures fairness by organizing transactions within individual DAGs and then ordering these DAGs within a higher-level consensus DAG. This approach allows for concurrent transaction processing while maintaining fair ordering. Using wave-based leader election of the underlying DAG protocol, DoD also eliminates the risk of leader-based manipulation, further enhancing its robustness.

DoD's goal is to integrate order-fairness with DAG-based consensus protocols in order to present a high-performance fair BFT consensus protocol. Our contributions are as follows:

- We design DAG of DAGs (DoD), a novel high-performance order-fairness protocol built on top of DAG-based BFT consensus. While our prototype is built on top of Narwhal, its design allows it to be integrated with any other DAG-based BFT protocol. DoD seamlessly adds a layer to manage transaction ordering without compromising system throughput.
- We provide a transaction processing routine that aims to fairly order and execute transactions. This routine is independent of the consensus mechanism and can be adapted to various underlying DAG structures.
- We implement a prototype of DoD and evaluate its performance. Our results demonstrate that DoD imposes minimal performance overhead while effectively ensuring order-fairness in DAG-based BFT consensus systems.

## 2 Background

Modern large-scale distributed systems need to deal with untrustworthy environments, where multiple mutually distrustful entities communicate and rely on unreliable infrastructure for data storage. Byzantine fault-tolerant (BFT) protocols have emerged as a solution, empowering distributed systems in untrustworthy environments and enabling a wide range of applications.

A BFT protocol operates over a network of nodes, which may behave arbitrarily or even maliciously. It employs the State Machine Replication (SMR) algorithm [43, 58], allowing the system to provide a replicated service with its state mirrored across multiple deterministic replicas. The primary objective of a BFT SMR protocol is to assign a specific order to each client request within the global service history and to execute those requests in that designated order [60]. A BFT SMR

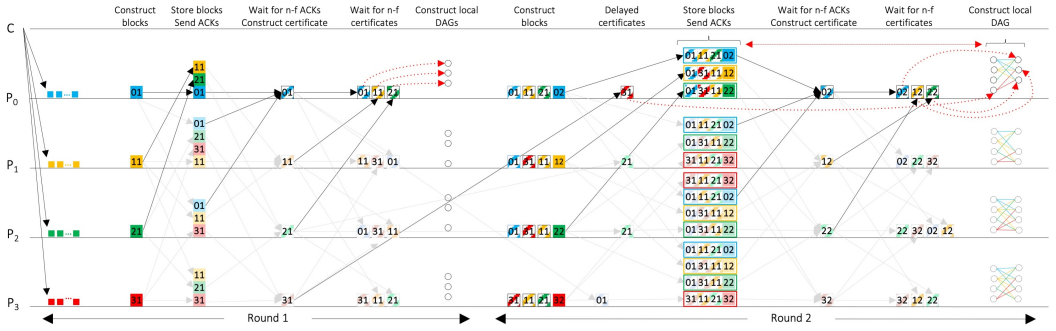


Fig. 1. DAG construction in DAG-based BFT protocols.

protocol ensures that all non-faulty replicas execute the same requests in the same order (*safety*), and that all valid requests are eventually processed (*liveness*).

Traditional BFT consensus protocols suffer from multiple shortcomings. First, BFT protocols tightly couple data dissemination with the consensus routine. Specifically, they share transaction data as an integral part of the consensus routine, creating a significant performance bottleneck.

Second, the sequential processing of transactions restricts the flow of transactions and hinders overall system throughput. Despite the implementation of efficient techniques to reduce message complexity, the consensus routine limits the system's performance.

Moreover, most such protocols adopt a leader-based approach, where a designated leader proposes a new block and broadcasts it to all replicas. Subsequently, the leader collects votes, compiles a certificate, and rebroadcasts this certificate to the network. While this method ensures consistency, it also leads to resource imbalance. The leader must handle a significant load, including storage, CPU processing, and bandwidth, while the remaining replicas are often underutilized.

To reduce the chance of Byzantine faults, these protocols employ leader rotation mechanisms to shift the leadership role among replicas. However, since any replica could become the leader at any time, all replicas must be over-provisioned to handle the potential load. This over-provisioning leads to inefficient resource utilization, as only one replica fully utilizes its capacity at any given time, while others remain largely idle.

## 2.1 DAG-based BFT Consensus Protocols

DAG-based consensus protocols have recently been proposed to address the shortcomings of traditional BFT protocols. DAG-based protocols leverage the inherent parallelism of the DAG structure to enable concurrent processing of multiple transaction blocks. The key idea is to decouple transaction dissemination from the consensus routine. This process involves two steps: *DAG construction* and *total ordering*. In the DAG construction phase, transactions are disseminated and organized into a DAG. Each block within this DAG contains a list of transactions and a causal history, indicating its relationship to blocks from previous rounds. The DAG builds asynchronously to the total order. For communication, a reliable broadcast mechanism ensures all participants receive transaction data reliably and efficiently. Each individual participant maintains a local view of the DAG. In the total ordering phase, the protocol ensures a consistent view of this DAG across all parties. Each party can then independently determine a total order of the constructed DAG, without needing an additional communication phase, as all the necessary information is already contained within the locally constructed DAG. While the complexity of various consensus protocols built around the DAG may differ, the fundamental advantage lies in consensus no longer hindering

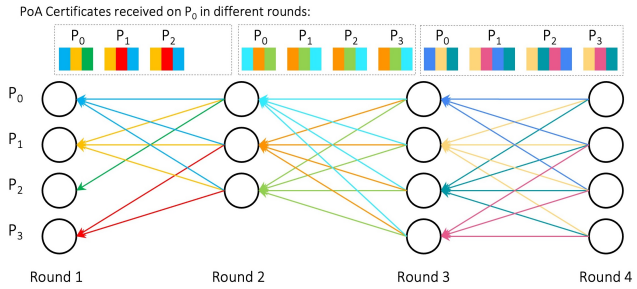


Fig. 2. DAG construction on party $P_0$ over four rounds. As shown in Figure 1 Party $P_0$ receives a PoA certificate from parties $P_0$, $P_1$, and $P_2$ in round 1, resulting in the creation of vertices for only these three parties in round 2. The back edges connecting round 2 to round 1 are determined by the causal history of blocks stored by $P_0$. Being the initial round, round 1 has no back edges.

the flow of transactions, and thus improves throughput. Additionally, this approach keeps every machine utilized at its maximum capacity consistently.

Figure 1 illustrates the DAG construction process involving 4 parties across 2 rounds. In this figure, DAG-BFT blocks are represented by solid-colored squares, while PoA certificates are denoted by squares with a hashed pattern. In each round, every party receives a unique set of client transactions and independently forms a transaction block. This block incorporates the party's transactions, a record of the causal history through preceding block certificates, and the party's digital signature. Consider round 2 as an example. Party $P_0$ constructs a block that includes three certificates, each representing a block in its causal history. This newly created block is then broadcast to all the parties ($P_0$ through $P_3$), who each validate it. In the preceding round 1, $P_0$'s block was received by $P_0$, $P_1$, and $P_2$. Upon successful validation, each receiving party stores the block and acknowledges its receipt by signing a digest of the block, along with the current round number and the identity of the block creator. Once the creator (e.g., $P_0$ in round 2) gathers $n - f$ distinct acknowledgments (for instance, from $P_0$, $P_1$, and $P_2$), it aggregates these signatures into a Proof of Availability (PoA) certificate. This certificate contains the block digest, the current round number, the creator's identity, and the $n - f$ signatures. The creator then broadcasts this PoA certificate to all parties, who will include it in their subsequent block constructions. The protocol ensures that all parties with a given certificate have an identical causal history. The creator then waits to receive at least $n - f$ such certificates for the current round before constructing its vertices for that round and proceeding to the next. In round 2, $P_0$ receives certificates from $P_0$, $P_1$, and $P_2$, and thus creates corresponding vertices for these parties in round 2 of its local DAG. The back edges connecting blocks from the current round (e.g., round 2) to the previous round (e.g., round 1) are determined by the causal history (PoA certificates) included in the blocks received and stored by the party constructing the DAG (e.g., $P_0$). As round 1 is the initial round, it contains no back edges since no prior round certificates have been received.

A PoA certificate includes $n - f$ signatures, i.e., at least $f + 1$ honest validators have checked and stored the block. Thus, the block is available for retrieval when needed to sequence transactions. Further, since honest validators have checked the conditions before signing the certificate, quorum intersection ensures availability and integrity (i.e., prevents equivocation) of each block. Since a block contains references to certificates from previous rounds, we get by an inductive argument that all blocks in the causal history are certified and available, satisfying causality. As depicted in Figure 2, this DAG construction phase prioritizes the efficient dissemination of transaction data and the establishment of a shared, though locally maintained view of the order in which transactions were proposed, without requiring immediate global consensus.

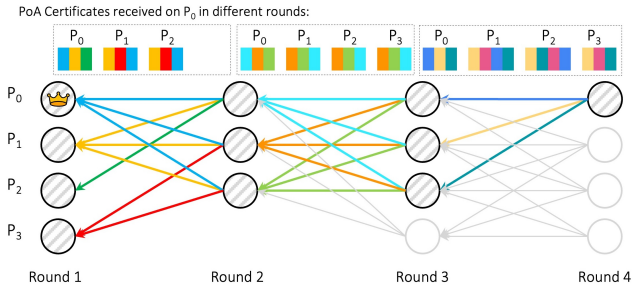


Fig. 3. The process of committing transactions in a DAG-based consensus protocol over four rounds on a party $P_0$ where $P_0$ is elected as the leader in the fourth round for the first round of the wave. The highlighted vertices represent the blocks that are included in a total order within a wave, determined by the existence of paths between leader vertices of consecutive waves and subsequently ordered using a topological sort. The gray vertices indicate blocks that are not yet part of the total order in the current wave (but might be ordered in later waves with the new elected leaders).

In the subsequent phase, as presented in Figure 3, a consensus routine is applied to the locally constructed DAGs to establish a consistent, global order for all transactions. Each replica independently determines this total order by processing its local view of the DAG, leveraging the embedded causal history. This allows replicas to commit transactions in the same sequence without requiring additional communication during the final commitment stage. This local processing during the ordering phase is a key advantage, minimizing communication overhead and potential bottlenecks.

The inherent parallelism of DAG-based consensus, where each replica independently builds and processes its local DAG, allows all replicas to operate concurrently at their full capacity. This contrasts with traditional leader-based protocols, which often centralize load on a single leader, leaving other replicas underutilized. This decentralized nature not only enhances throughput but also significantly improves the system's resilience by eliminating single points of failure (i.e., even if the current elected leader is faulty, it does not stop the system from processing transactions).

DAG-based BFT consensus protocols employ different methods to achieve the total ordering of all the vertices (after the local DAG construction stage). The total order occurs in each wave, where each wave consists of multiple rounds, with the number of rounds fixed depending on the specific protocol. In the last round of each wave, a random leader is elected to oversee the first round of the subsequent wave. If there exists a path from the last-round vertex of the leader party in the current wave to the last-round vertex of the leader party in the previous wave, then all the vertices between these two vertices are ordered using a topological sort. This ensures that the transactions are consistently ordered while maintaining the efficiency of the DAG structure.

There is a large body of recent DAG-based consensus protocols, e.g., DAG-Rider [35], Narwhal and Tusk [23], Bullshark [62], Shoal [61], Mahi-Mahi [32], Shoal++ [7], Autobahn [28], BBCA-CHAIN [47], Shardag [18], Sailfish [59], and Wahoo [21] (some are surveyed in [56]). The key innovation of these protocols is the elimination of communication overhead during the DAG ordering phase. Their operation is structured around rounds where each replica independently constructs a local view of the DAG. Communication between replicas during this DAG construction phase relies on reliable broadcast. Subsequently, the final ordering of DAG vertices is determined locally by each party, requiring no further communication. To ensure liveness despite potentially malicious asynchronous adversaries, these protocols incorporate randomness for leader election.

We focus on DAG-Rider [35], Narwhal and Tusk [23], and Bullshark [62], as they are more relevant to our work in this paper. In DAG-Rider, within round $r$, each vertex has at least $n - f$ strong edges and up to $f$ weak edges. Strong edges point to vertices in the immediately preceding

round, $r-1$ while weak edges point to vertices in earlier rounds ($r' < r-1$) that might otherwise be unreachable in DAG. These edges are ignored by the consensus mechanism but are used to ensure validity. Since a new round advances once at least $n-f$ vertices are added, strong edges might not connect to all the vertices from the previous round. Weak edges address this by ensuring that all vertices are eventually included in the total order constructed from the DAG, helping slower replicas to propose their transactions. However, this necessitates infinite storage as every received block must be kept due to potential future weak edges. While DAG-Rider's strong fairness is theoretically achievable, its infinite storage requirement makes it impractical.

Narwhal is a scalable DAG-based mempool for data dissemination using a primary-worker model, and Tusk is an asynchronous consensus that orders metadata from Narwhal's DAG in three-round waves. Tusk removes weak edges to allow garbage collection, reinjecting transactions from uncommitted garbage-collected blocks into new blocks in subsequent rounds. Tusk piggybacks the first round of each wave with the third round of the previous wave, achieving lower round latency compared to DAG-Rider. Moreover, Tusk employs quorum-based reliable broadcast, contrasting with DAG-Rider's classic reliable broadcast.

Bullshark is a hybrid protocol blending DAG-Rider's and Tusk's features, operating in synchronous and asynchronous phases. It utilizes weak edges during synchronous periods, similar to DAG-Rider. However, like Tusk, it performs garbage collection and omits weak edges during asynchronous periods. Consequently, fairness is provided only in synchronous phases. Bullshark's synchronous periods, which utilize weak edges like DAG-Rider, introduce latency because the protocol must wait for participants, contributing to its lower throughput compared to Tusk.

### 2.2 Order-Fairness

Adversarial manipulation of transaction order is studied in the decentralized finance (DeFi) domain [10, 22, 26, 30, 39, 55, 72] where proposers make profit by including, excluding, or reordering transactions, known as maximal extractable value (MEV) [22]. Early studies focus on censorship resistance [49] where the goal is to ensure that correct transactions are eventually ordered, i.e., not censored. However, reordering transactions, e.g., sandwich attacks, is still possible. Similarly, reputation-based systems [9, 20, 40, 45] only detect unfair censorship.

Recently, the notion of *order-fairness* is presented to address the manipulation of transaction ordering [16, 36, 37, 41, 42, 71]. There are mainly three definitions of fairness: blind order-fairness, participation fairness, and time-based order-fairness.

The notion of *blind order-fairness* [46] is designed to protect the order of transactions by keeping the contents of transactions hidden until the commit phase. Blind order-fairness can be achieved using threshold encryption [9, 15, 49, 63], where transactions are encrypted, and their content is revealed once their order is fixed. This technique suffers from (1) metadata leakage and (2) collusion attacks between clients and the leader, where the leader becomes aware of a client transaction before ordering it and manipulates its order [36, 37, 41].

*Participation fairness* [48] ensures that the final total order of transactions is contributed by a sufficiently large number of honest parties. In particular, random leader (committee) election provides opportunities for every replica to propose and commit its transactions, e.g., by becoming the proposer [1, 9, 23, 27, 35, 38, 45, 48, 52, 62, 68]. However, a malicious proposer can still order transactions unfairly in its turn.

Finally, *time-based order-fairness* [48] ensures that the order in which transactions are sent to or received from parties is protected. Time-based fairness can be achieved by (1) adding timestamps at the client side when sending transactions or (2) measuring network latency to calculate propagation time. However, clients might assign timestamps maliciously and measuring network latency for each transaction is difficult due to the asynchronous nature of the network, which may introduce

arbitrary delays. An alternative method is to measure the arrival time of each transaction and order transactions based on their arrival time. Time-based order-fairness based on arrival time is known as *receive order-fairness*. It ensures that the transactions within a block are committed in the same order as they arrive in the network.

This paper adopts the notion of receive order-fairness in DAG-based BFT consensus. Specifically, receive order-fairness is parameterized by an order-fairness parameter $\gamma$, which represents the fraction of replicas that receive transactions in a particular order. We formally define receive order-fairness as follows. Given a set of $n$ parties, if at least $\gamma n$ parties receive a transaction $t_1$ before $t_2$, then no honest party will commit transaction $t_2$ before $t_1$.

## 3 DoD Model

DoD is deployed in an asynchronous network consisting of a set of $n$ known parties (replicas) where at most $f$ of them are Byzantine (malicious) at each time. In the Byzantine failure model, faulty replicas may exhibit arbitrary, potentially malicious, behavior. A strong adversary can coordinate malicious replicas and delay communication. However, the adversary is computationally bounded and cannot subvert standard cryptographic assumptions. Replicas are connected with point-to-point bi-directional communication channels, and each client can communicate with all replicas. Network links are pairwise authenticated, which guarantees that a malicious replica cannot forge a message from an honest replica. For communication between replicas, we assume the presence of digital signatures and public-key infrastructure (PKI). A collision-resistant hash function $D(.)$ is also used to map a message $m$ to a constant-sized digest $D(m)$.

BFT protocols require the number of replicas $n > 3f$ to guarantee safety with at most $f$ malicious replicas [6, 12, 13, 19, 25, 44]. However, fair ordering of transactions requires a larger $n$ [36, 50]. The order-fairness is further parameterized by an order-fairness parameter $\gamma$ representing the fraction of replicas that receive transactions in a particular order. In DoD, similar to Themis [36] and Rashnu [50], $n$ needs to be larger than $\frac{4f}{2\gamma-1}$. Intuitively, since $f$ out of $n$ replicas might be faulty, the protocol can rely on a quorum of $n-f$ replicas to generate the final order. Among these $n-f$ replicas, $f$ might be malicious, i.e., $f$ replicas not participating in the quorum are honest slow replicas. As a result, out of the quorum of $n-f$ replicas, only $n-2f$ replicas are guaranteed to be honest. To realize order-fairness if $\gamma n$ replicas receive transactions in a particular order, the final ordering must reflect that order. Since only $n-2f$ replicas within the quorum are guaranteed to be honest, the output of order-fairness must be the same as $\gamma n$ even with $\gamma n - 2f$ replicas broadcasting a particular order. On the other hand, a majority of replicas must agree with the order. Otherwise, we could end up with conflicting fair orders suggested by different sets of $\gamma n$ replicas. More precisely, to ensure that only one of the two different orders $t \prec t'$ or $t' \prec t$ is captured between two transactions $t$ and $t'$, $\gamma n - 2f > \frac{n}{2}$. As a result $n > \frac{4f}{2\gamma-1}$.

Note that the definition of order-fairness asserts that when at least a $\gamma$ fraction of nodes receive two transactions in a particular order, that order must be maintained. However, it places no restrictions on scenarios where fewer than a $\gamma$ fraction of nodes receive the transactions in any order. As a result, in an unstable, jittery network environment where no partial order between two given transactions may surpass the $\gamma n$ threshold, DoD will still include these transactions in the final order, with the flexibility to determine any relative order for them.

In an asynchronous network, replicas might receive transactions in different orders. Hence, as demonstrated by the *Condorcet paradox*, defining a fair order among all transactions becomes impossible, even if all replicas are honest.The Condorcet paradox states that even if the local order of each replica is transitive, there might be situations that lead to non-transitive collective voting preferences. Consider a simple example with three replicas $r_1$, $r_2$ and $r_3$ that receive three

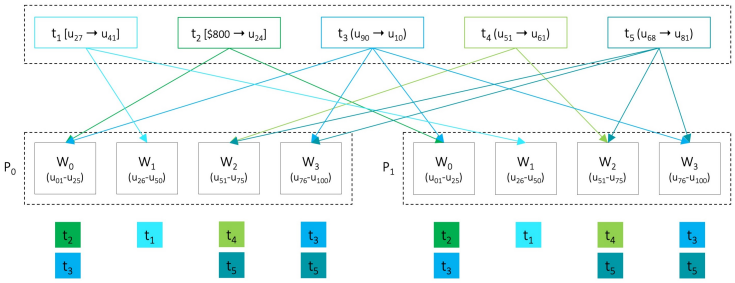


Fig. 4. A DoD deployment with two parties, each consisting of four workers

transactions $t_1$, $t_2$ and $t_3$ in the following order: on $r_1 : t_1 \rightarrow t_2 \rightarrow t_3$, on $r_2 : t_2 \rightarrow t_3 \rightarrow t_1$, and on $r_3 : t_3 \rightarrow t_1 \rightarrow t_2$. Here, a majority (2 out of 3) of replicas received $t_1$ before $t_2$ (replicas $r_1$ and $r_3$). Similarly, $t_2$ is received before $t_3$ by a majority of replicas ($r_1$ and $r_2$ ). However, $t_3$ is also received by replicas $r_2$ and $r_3$ before $t_1$. The collection of these orders results in a cyclic global ordering. To address this challenge, transactions involved in a Condorcet cycle can be sent to replicas concurrently within the same batch [37]. Specifically, given transactions $t$ and $t'$ where $t$ is received by sufficiently many replicas before $t'$; while strong order-fairness requires the leader to send $t$ before $t'$, *batch-order-fairness* relaxes this requirement by saying $t$ should be delivered no later than (before or at the same time as) $t'$. Batch-order-fairness does not specify the order of transactions within a batch and respects a fair order up to this limit. Hence, if a total order among all transactions is required, a deterministic total ordering for transactions in the same cycle, e.g., alphabetical, needs to be specified.

In DoD, we utilize the *data-dependent* order fairness notion presented in Rashnu. This notion recognizes that fair ordering is crucial when transactions access shared resources, and it ensures that if a sufficient number of replicas receive data-dependent transactions in a particular order, that order is preserved in the execution. More precisely, given two data-dependent transactions $t$ and $t'$; If $\gamma$-fraction of replicas receive $t$ before $t'$, no honest replica outputs $t'$ before $t$. Focusing on ordering only those transactions that have data dependencies, data-dependent order-fairness avoids the performance overhead associated with ordering all transactions. Furthermore, considering only data-dependent transactions minimizes the likelihood of Condorcet cycles.

While DoD can be bootstrapped from any DAG-based BFT protocols, we particularly choose Narwhal and Tusk as our base protocol in preference to DAG-Rider and Bullshark for multiple reasons. Firstly, DAG-Rider's dependence on weak links to assure the eventual inclusion of blocks originating from slower replicas introduces unacceptable latency, rendering it not fully practical for our performance objectives. Secondly, while Bullshark represents a hybrid protocol integrating aspects of both DAG-Rider and Tusk, its synchronous phases continue to employ weak links, consequently resulting in a notable reduction in throughput when compared to Tusk. In contrast, our DoD system implements a different transaction dissemination strategy wherein the client broadcasts transactions to all replicas, which is needed to guarantee order-fairness. This eliminates the requirement for weak links and the associated delays incurred by awaiting slower replicas. As a result, DoD achieves both efficient consensus and fair ordering with $n - f$ replica batches. Thirdly, by building upon Narwhal, DoD benefits from its robust and fast mempool management and the use of sharding among workers, which helps in achieving high throughput.

Our protocol builds on the concept of the mempool and uses sharding. Workers with the same ID across all parties are responsible for the same shard, and this shard assignment is predetermined and known to both clients and workers. Figure 4 presents the shard assignment of DoD in a deployment

with two parties ($P_0$ and $P_1$) and four workers ($W_0$ to $W_3$) within each party. As shown, each worker is responsible for processing transactions on a separate data shard (of a simple data set with user IDs $u_1$ to $u_{100}$). Clients submit transactions $t_1$ to $t_5$. Each transaction is processed by the workers maintaining the corresponding data shards. Transactions are either single-shard or cross-shard, e.g., cross-shard transaction $t_3$ is processed by workers $W_0$ and $W_3$ of all parties.

## 4 Transaction Processing in DoD

This section gives an overview of the entire transaction processing routine of DoD and then discusses local ordering and global ordering stages in more detail. The core contribution of the transaction processing routine is its ability to order and execute transactions fairly through distinct phases: local ordering, global ordering, and order finalization. While DoD is built on top of the Narwhal protocol, its ordering layer can be easily integrated with any other DAG-based BFT protocol. Although some communication phases are adapted to fit the underlying DAG structure, the core process of achieving local or global order can be done regardless of the specific consensus mechanism used, as DoD manages the ordering of transactions within each batch.

Figure 5 presents the communication flow of the protocol in a deployment with five replicas ($P_0$ to $P_4$) each containing a primary ($\pi$) and a single worker ($W_0$). In DoD, and in order to provide order-fairness, each client sends its transactions to their designated workers determined by the shard assignment at *every* party (in Figure 5, all transactions go to $W_0$). This is needed because if only a single party receives a client transaction, no other party can validate the proposed order.

Each worker collects a block of transactions, constructs a local dependency graph based on the received order, and sends this local order graph as a local-order message to all the workers with the same ID on other parties (i.e., workers who were supposed to receive the same set of transactions). As can be seen, workers might receive transactions in different orders or even receive different sets of transactions due to the asynchronous nature of the network.

Upon receiving $n - f$ different local order graphs, each worker generates a global dependency graph that ensures fair ordering of the transactions and disseminates a global-order message including the graph to the workers with the same ID across all parties. This communication step is required to construct the DAG of global order graphs on each party. Each worker then waits to receive $n - f$ global-order messages and sends the digest of these global order graphs to the party's primary. At the same time, if a worker does not have the content of a transaction in a received global order graph, the worker asynchronously contacts the sender to receive the transaction content (e.g., in Figure 5, worker $W_0$ of party $P_0$ requesting transaction 4 from a sender party worker, i.e., $P_1W_0$).

Upon receiving a global order message, DoD performs a similar process as shown in Figure 1. A worker validates the message, stores the associated data, and sends an acknowledgment (signatures) to the worker that created it. Once a quorum of acknowledgments is gathered, the cryptographic hash of the batch and $n - f$ signatures are shared with the primary for inclusion in a new block. The primary then collects these digests until either a predefined maximum block size is reached or a timeout occurs. The primary constructs a PoA certificate using the received $n - f$ signatures, ensuring that the data within the block is accessible. This allows the primary to advance to the next round and construct the DAG. While each of these steps is detailed in Figure 1, they are intentionally omitted from Figure 5 for simplicity, represented instead by three dots to indicate these additional underlying operations.

In the consensus stage, transactions are ordered by processing the DAG in waves, which consists of three rounds. Replicas commit to a leader block if $f + 1$ votes in their local view reference it, and then order its causal history by recursively checking for paths to previously committed leaders. If such a path exists, the replica orders the previous leader before the current candidate and repeats

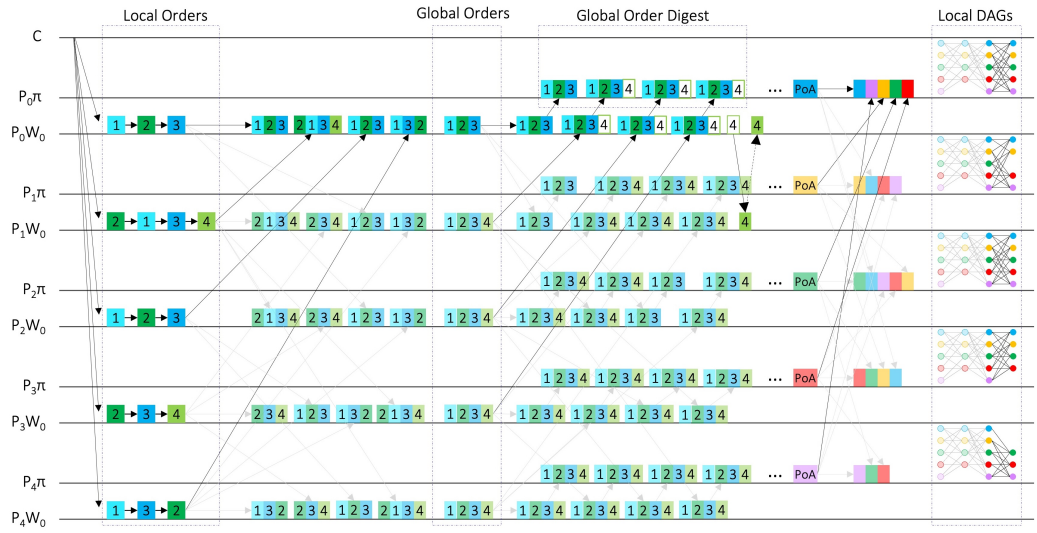


Fig. 5. The communication flow of DoD

this process. This mechanism guarantees that all honest replicas eventually agree on the same sequence of committed leader blocks.

Figure 6 presents local and global ordering stages of DoD in a simple example. We use this example throughout this section. It illustrates worker $W_0$ of four parties $P_1W_0$ to $P_4W_0$ (assuming $f = 1$, $n = 5$ ($n = 4f + 1$), $n - f = 4$ replicas in the quorum). A set of transactions $t_1$ to $t_9$ is received from clients, where each transaction accesses a subset of data objects $A, B, C, D, E$ (shown at the bottom). These interactions are either read operations ($R(obj)$) or write operations ($W(obj)$).

### 4.1 Local Ordering

In the local ordering phase, each worker constructs a dependency graph as it processes transactions. During any given DoD round $i$, a worker may encounter three distinct types of transactions:

(1) **Unseen** transactions that have never been observed by this worker in prior rounds.
(2) **Unprocessed** transactions that were received in a prior round $j (j < i)$ and included in the global order graph but were not executed due to missing edges in the graph.
(3) **Processed** transactions that were previously executed in an earlier round $j (j < i)$ but, due to network delays, are now being received by the worker.

As outlined in Algorithm 1, upon receiving a valid signed request message $m = \langle \text{REQUEST}, t, \tau, c \rangle_{\sigma_c}$ from client $c$ to execute transaction $t$ with timestamp $\tau$, each worker stores the transaction in an ordered list, maintaining the sequence of arrival. This list, referred to as a block, is sealed when it reaches its maximum capacity or when the timer expires, whichever occurs first.

We define a weight function $w : E \mapsto [0, n - f]$ to represent the number of different workers that observe a specific edge. Given a set of $n - f$ local dependency graphs and two transactions $t$ and $t'$ in round $i$, $w(t, t')$ indicates how many local-order graphs include an edge from $t$ to $t'$.

At the beginning of round $i$, each worker $w$ initializes an empty graph $G_w = (T_w, E_w)$. Each worker $w$ also maintains missing edges and their current weight in a missing edges store $M_w$. An edge $(t, t')$ is a missing edge if $t$ and $t'$ are data-dependent and both $w(t, t')$ and $w(t', t)$ are smaller than $n(1 - \gamma) + f + 1$. Each worker keeps updating the weight of the missing edges in its missing edges store until the weight becomes greater than $n(1 - \gamma) + f$ (as shown in the global ordering).

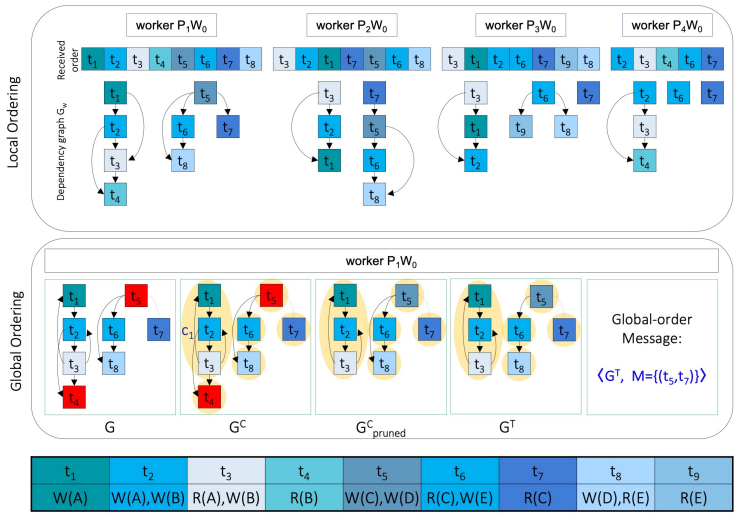


Fig. 6. Local and global ordering in DoD

**Algorithm 1** Local ordering on a worker $w$

**Input:** a set of incoming client requests in *DoD* round $i$

Initiate an empty graph $G_w = (T_w, E_w)$
**while** round $i$ has not finished **do**
  **for** every valid received client request $m = \langle \text{REQUEST}, t, \tau, c \rangle_{\sigma_c}$ **do**
    **if** $t$ is already **Processed then**
      continue
    **else if** $t$ is **Unprocessed then**
      **if** any missing pair $(t', t)$ exists in the missing edge store $M_w$ where $t'$ is already received **then**
        $w(t', t) = w(t', t) + 1$
    **else if** $t$ is an **unseen** transaction **then**
      Add $t$ to $T_w$
      **for** every vertex $t' \in T_w$ **do**
        **if** $t$ and $t'$ are data-dependent **then**
          Add $(t', t)$ to $E_w$
Send $\langle \text{LOCAL-ORDER}, i, G_w \rangle_{\sigma_w}$ to the workers of other replicas with the same ID

Workers discard any Processed transactions. When a worker receives an Unprocessed transaction $t$, it checks the missing edges set for any undirected edges involving $t$. If a missing pair $(t', t)$ is found and the replica has already received $t'$, the worker adds $(t', t)$ to the set to establish the order between $t'$ and $t$. Since DoD employs a DAG-based BFT consensus, where each replica maintains a local DAG of vertices, updated edges do not need to be communicated to other replicas. This approach differentiates DoD fundamentally from Rashnu. Following this, the worker integrates all Unseen transactions into $G_w$. For each transaction $t'$ that precedes transaction $t$ and shares a data dependency with $t$, an edge $(t', t)$ is inserted into $G_w$. The worker ensures that an edge $(t', t)$ is added if transaction $t'$ was received before $t$ (*i.e.*, $t' \prec t$) and both transactions are data-dependent. At the end of round $i$, each worker $w$ sends a signed local-order message $\langle \text{LOCAL-ORDER}, i, G_w \rangle_{\sigma_w}$ to the workers of other replicas with the same ID. These local-order messages are then utilized by workers in the global ordering phase to confirm that this set of requests has been received.

Figure 6 (top part) presents the local ordering phase on different workers (with the same ID). As an example, worker $P_3W_0$ receives transactions in this order: $t_3 \prec t_1 \prec t_2 \prec t_6 \prec t_7 \prec t_9 \prec t_8$ and

generates a local dependency graph consisting of data dependency relations, e.g., there is an edge from $t_1$ to $t_2$ because $t_1$ was received before $t_2$ and both write to the data object $A$.

### 4.2 Global Ordering

The global ordering phase of DoD, in contrast to Rashnu, uses sharding where different workers of the same replica receive disjoint sets of transactions while workers with the same ID across different replicas receive the same set of transactions, enabling DoD to provide order fairness. As presented in Algorithm 2, during the global ordering phase, each worker receives local-order messages from workers with the same ID across different parties. Since up to $f$ parties may be faulty and fail to send their local-order messages, each worker collects a quorum of $n-f$ local-order messages from different workers (including itself) to establish the global order.

In DoD, transactions are classified into two categories: fixed and pending. A transaction $t$ is labeled as fixed if it appears in at least $n-2f$ local-order messages, ensuring that it has been received by a sufficient number of replicas and can be safely included in the final order. Since only up to $f$ local-order messages may originate from faulty replicas, a transaction must be observed by at least $n-2f$ honest replicas to be considered reliably ordered. On the other hand, a transaction is labeled as pending if it is present in at least $n(1-\gamma)+f+1$ but fewer than $n-2f$ local-order messages. These transactions have not yet reached enough replicas to be fully finalized. However, dependencies can still form between pending and fixed transactions. To maintain consistency in ordering, pending transactions are kept if an edge exists from pending transactions to fixed transactions.

For instance, in Figure 6, since $\gamma = 1$ and $f = 1$ ($n = 5$), a transaction $t$ is fixed if it appears in 3 or 4 local orderings, e.g., transactions $t_1$, $t_2$, $t_3$, $t_6$, $t_7$ and $t_8$ (in blue), while transaction $t$ is pending if only 2 local orderings include $t$, e.g., transactions $t_4$ and $t_5$ (in red). Transaction $t_9$ is not included in the global order graph because it is received by only 1 party.

**Global dependency graph construction.** At the beginning of round $i$, each worker collects local-order messages from at least $n-f$ distinct workers with the same worker ID across different parties and initializes an empty graph $G = (V, E)$. The worker then adds all transactions classified as fixed or pending to the vertecis of $G$. To determine dependencies, the worker examines each pair of transactions $t$ and $t'$ in $V$, calculating both $w(t, t')$ and $w(t', t)$. If the stronger of the two values meets or exceeds $n(1-\gamma)+f+1$, an edge is inserted into $G$ to reflect the ordering relationship.

This process ensures that transactions are included in a way that preserves fairness. The fairness rule states that if a $\gamma$ fraction of replicas observe transaction $t$ before transaction $t'$, then $t'$ cannot be placed before $t$ in the final order. By requiring at least $n(1-\gamma)+f+1$ replicas to report a consistent ordering, the protocol guarantees that any conflicting minority cannot override the correct sequence. Since only $n-2f$ replicas in a quorum of $n-f$ are guaranteed to be honest, this threshold prevents fairness violations, even if all remaining replicas report a reversed order.

For instance, if $\gamma = 1$, fairness demands that if all honest replicas within the quorum (at least $n-2f$) receive transaction $t$ before $t'$, then $t'$ cannot be placed before $t$. If at least $f+1$ replicas confirm the order $t \prec t'$, the protocol safely enforces it. This is because it becomes impossible for a quorum of $(n-2f)$ honest replicas to have observed $t'$ before $t$. If there is no clear majority order, meaning both $t \prec t'$ and $t' \prec t$ are supported by more than $f$ replicas, the protocol resolves the ambiguity by selecting the order with the higher weight.

In Figure 6, a directed edge is added between two transactions if the edge appears in at least $n(1-\gamma)+f+1 = 2$ local graphs. For instance, while two local order graphs ($P_1W_0$ and $P_2W_0$) include both $t_5$ and $t_7$, the workers receive $t_5$ and $t_7$ in different orders, i.e., $t_5 \prec t_7$ in $P_1W_0$ while $t_7 \prec t_5$ in $P_2W_0$. Hence, no global order can be established and $(t_5, t_7)$, is recorded in a missing edges set $\mathcal{M}$.

While Byzantine replicas may attempt to introduce false dependencies, their impact is limited, as each local-order message is verified using certificates sent by the originating replicas. This ensures

that no potentially invalid edges will be included in the final graph. The complexity of constructing the global dependency graph is $O(n(|V| + |E|))$, as it involves traversing $n$ local graphs.

**Condensation graph generation.** The global dependency graph may contain cycles due to the Condorcet paradox. However, the final graph must be acyclic to determine the transaction order. To achieve this, DoD applies the graph condensation technique. Given a graph $G$, to generate its condensation graph $G^C$, DoD first identifies the strongly connected components (SCCs) of $G$. Each SCC intuitively represents either a single transaction or a cycle in the graph. Formally, a strongly connected component $C$ is a maximal subset of vertices such that any two vertices within this subset are reachable from each other.

The condensation of graph $G$ is a new graph $G^C = (V^C, E^C)$, where each vertex $v^C \in V^C$ corresponds to a strongly connected component (SCC) of $G$. An edge $(u_i^C, v_j^C) \in E^C$ exists if and only if there are vertices $u \in u_i^C$ and $v \in v_j^C$ such that $(u, v) \in E$. In the condensation graph, a vertex $v^C$ is classified as fixed if it contains at least one fixed transaction; otherwise, it is classified as pending. The resulting graph $G^C$ is guaranteed to be acyclic. As shown in Figure 6, graph $G$ is condensed into an acyclic graph $G^C$ where vertices are SCCs. Here, we have 6 vertices: singletons $t_4$, $t_5$, $t_6$, $t_7$, $t_8$ and $C_1 = \{t_1, t_2, t_3\}$ (forming a cycle). Vertices $t_4$ and $t_5$ are pending because no fixed transaction is included in those vertices. All other vertices ($t_6$, $t_7$, $t_8$ and $C_1$) are fixed. The condensation graph can be generated using Kosaraju's algorithm for finding strongly connected components, which runs in $O(|V| + |E|)$ time.

**Graph pruning.** The next step involves eliminating pending transactions that do not have any outgoing edges to a fixed transaction. These pending transactions are excluded because they are not yet received by enough replicas and do not play a role in determining the order of fixed transactions. Although they were initially included in the graph with the possibility of a connection to a fixed transaction, once it is confirmed that no such connection exists, they can be safely removed. Consider two transactions, $t$ and $t'$, where $t$ is fixed, $t'$ is pending, and there exists an edge $(t, t') \in E^C$. Since at least $n(1 - \gamma) + 1$ replicas have received $t$ before $t'$, removing $t'$ does not violate the fairness property of the system. In Figure 6 pending vertices $t_4$ and $t_5$ are considered for removal. However, $t_5$ is kept due to outgoing edges to the fixed vertices $t_6$ and $t_8$, while $t_4$ is pruned (will include later when it appears in a sufficient number of local order graphs). The pruning process (removing unnecessary pending transactions) can be done in $O(|V| + |E|)$ time complexity.

Additionally, for any pair of transactions $t$ and $t'$, if they are not already connected by an edge and are not part of the same condensation graph vertex (i.e., they do not belong to the same strongly connected component), the worker adds this pair to the set of missing edges $\mathcal{M}$. Keeping track of these missing edges is crucial, as the determination of their order may create new cycles or extend existing ones. Therefore, a transaction cannot be finalized until all its predecessor transactions have their order determined. It is important to note that missing edges between transactions within the same cycle do not need to be tracked, as they do not impact the overall ordering of transactions.

**DAG transitive reduction.** To optimize the generated graph, we perform transitive reduction, which removes redundant edges while preserving the reachability between vertices. This process simplifies the graph by ensuring that only the essential edges, directly contributing to the order of transactions, remain. Given a directed graph $G^C = (V^C, E^C)$, the transitive reduction results in a graph $G^T = (V^T, E^T)$ where: (1) $V^C = V^T$, meaning the vertices remain unchanged. (2) For every pair of vertices $u$ and $v$, there is a path from $u$ to $v$ in $G^T$ if and only if there is a path from $u$ to $v$ in $G^C$. This ensures that all reachability relationships in $G^C$ are maintained in $G^T$, but without the redundant edges that can be inferred through other paths (i.e., don't contribute new information about the transaction order). In Figure 6, edge $(t_5, t_8)$ is removed due to the indirect path $t_5 \rightarrow t_6 \rightarrow t_8$, simplifying the graph while preserving reachability.

**Algorithm 2** Global ordering on a worker $w$

**Input:** $n - f$ received local-order messages $\langle \text{LOCAL-ORDER}, i, G_{w'} \rangle_{\sigma_{w'}}$

▷ Transaction labeling
**for** every transaction $t$ in some local-order graph $G_w$ **do**
  **if** $t$ appears in at least $n - 2f$ local-order graphs **then**
    Label $t$ as fixed
  **else if** $t$ appears in at least $n(1-\gamma) + f + 1$ local-order graphs **then**
    Label $t$ as pending
▷ Global dependency graph generation
Initiate an empty graph $G = (V, E)$
Initiate an empty set of missing edge pairs $\mathcal{M} = \phi$
**for** every fixed or pending transaction $t$ **do**
  add a vertex $t$ to $G$
**for** each pair of vertices $t$ and $t'$ **do**
  **if** $(t', t) \in M_w$ **then** ▷ The edge is already in the missing edges store
    Increase the weight of $(t', t)$ by 1
  **if** $w(t, t') > w(t', t)$ & $w(t, t') \geq n(1-\gamma) + f + 1$ **then**
    Add $(t, t')$ to $E$
  **else if** $w(t', t) > w(t, t')$ & $w(t', t) \geq n(1-\gamma) + f + 1$ **then**
    Add $(t', t)$ to $E$
  **else if** $w(t, t') > 0$ or $w(t', t) > 0$ **then**
    Add $(t, t')$ pair into $\mathcal{M}$
    **if** $w(t, t') > 0$ & $(t, t') \notin M_w$ **then**
      Add $((t, t'), w(t', t))$ to $M_w$
    **if** $w(t', t) > 0$ & $(t', t) \notin M_w$ **then**
      Add $((t', t), w(t, t'))$ to $M_w$
▷ Condensation graph generation
$G^C = (V^C, E^C) \leftarrow\!\!\shortmid \textsc{Condensation}(G)$
▷ Graph pruning
**for** every pending vertex $u$ **do**
  **if** there is no fixed vertex $v$ such that $(u, v) \in E^C$ **then**
    Remove $u$ from $V^C$
    Remove all edges involving $u$ from $E^C$
**for** every pair of data-dependent transactions $t$ and $t'$ with no edges **do**
  **if** $t$ and $t'$ are in two different vertices of $V^C$ **then**
    Add $(t, t')$ to $\mathcal{M}$
▷ DAG transitive reduction generation
$G^T = (V^T, E^T) \leftarrow\!\!\shortmid \textsc{TransitiveReduction}(G^C)$
Send $\langle \text{GLOBAL-ORDER}, G^T, \mathcal{M} \rangle_{\sigma_w}$ to the workers of other replicas with same ID
▷ Global order synchronization
**if** Receiving valid global-order messages **then**
  **for** every transaction $t$ in some $G^T$ **do**
    **if** $t$ does not exist in the local storage of worker $w$ **then**
      send an INFO-REQ(t) message to the workers with the same ID
**if** Receiving INFO-REQ(t) from worker $w'$ & $t$ exists in the local storage **then**
  send transaction $t$ to $w'$
**for** every $(t, t')$ pair in some $\mathcal{M}$ **do**
  Update the missing edge store $M_w$ with $(t, t')$
**for** every $(G^T, \mathcal{M})$ in the receiving global-order messages in a round $r$ **do**
  Create digest $d = D((G^T, \mathcal{M}))$
  Update the local store with $(d, (G^T, \mathcal{M}))$ mapping
  **if** $(G^T, \mathcal{M})$ is coming from worker $w'$ of another party **then**
    Send an acknowledgment $\langle \text{ACK}, d, r, w' \rangle_{\sigma_w}$ to $w'$.
  send $d$ to the Primary
**for** every $\langle \text{ACK}, d, r, w \rangle_{\sigma_{w'}}$ receiving from a worker $w'$ **do**
  send $\langle \text{ACK}, d, r, w \rangle_{\sigma_{w'}}$ to the Primary

Since $G^C$ is acyclic, the transitive reduction $G^T$ is unique and forms a subgraph of $G^C$, meaning it is the minimal graph that still maintains the essential structure needed for ordering transactions. The complexity of generating the transitive reduction of a directed acyclic graph is $O(|V|^2)$, where $|V|$ is the number of vertices. After the transitive reduction, the resulting graphs, along with a set of missing edges in these graphs, are broadcast to the same ID workers across all the parties. This ensures that all workers can access the necessary information for the final BFT-DAG construction.

**Global order synchronization.** The global order synchronization process initiates by receiving global-order messages (including global-order graphs and their associated sets of missing edges) and consists of two main steps. First, synchronizing transactions across all workers, and second, updating the local missing edges store.

Clients multicast the same transactions to workers with the same ID across all replicas. However, due to network delays or message losses, some transactions may not reach all workers. As a result, in any DoD round, the global order observed by workers with the same ID may not be exactly the same, as shown in Figure 5. The global order contains only transaction IDs instead of full transactions to reduce communication overhead.

As each worker processes the global-order graphs, it checks whether each transaction ID is already present in its local storage. If a transaction ID $t$ is not present in its local store, the worker requests the full transaction corresponding to the unknown transaction ID from the sender worker by sending an info-req(t) message. Upon receiving this request, the sender asynchronously responds with the required transaction, allowing other workers to update their local stores. Importantly, this communication step is fully asynchronous, ensuring that the requesting worker does not wait for the response during the DAG construction or the consensus phase, as the requested information is only needed during the order finalization stage. By efficiently synchronizing transaction data while preserving asynchronous operations, the protocol ensures that all replicas maintain a consistent set of transactions, even if some were initially delayed or lost.

In the second step, each worker $w$ updates its missing edge store $M_w$ to maintain accurate dependency relationships between transactions. As it processes the sets of missing edges, the worker examines each pair of transactions along with their weight (i.e., occurrence count). If a dependency between two transactions is identified, the missing edge store is updated accordingly.

Finally, for each received $\langle \text{GLOBAL-ORDER}, G^T, \mathcal{M} \rangle_{\sigma_w}$ message, the worker generates a digest $d = D((G^T, \mathcal{M}))$. This digest, along with the global-order message, is stored in the local storage of the worker. To enable the construction of the BFT-DAG, the worker sends the digest to the Primary. Additionally, it sends an $\langle \text{ACK}, d, r, w' \rangle_{\sigma_w}$ message to the worker $w'$ that originally created this global-order message, unless the message originated from the current worker itself.

### 4.3 Order Finalization

Order finalization is managed by the primary within each party. As presented in Algorithm 3, the primary receives digest $d = D(G^T, \mathcal{M}))$ of global-order messages from workers until the maximum block size is reached or a timeout occurs. It also gathers $n - f$ acknowledgment messages from workers, which originally created this global order. Upon receiving both the digests and the quorum of acknowledgment messages, the primary initiates the Narwhal algorithm to incorporate new vertices into the BFT-DAG and advance the current round. As part of this Narwhal process, the primary first generates a proof of availability certificate and multicasts it to all other replicas. Upon receiving $n - f$ PoA certificates from other replicas, the primary constructs its local DAG by adding digests as vertices along with the back edges (obtained from PoA certificates), following the Narwhal DAG construction algorithm.

Once workers receive the total order from the primary, they place the ordered global-order messages into an execution queue. The execution queue is traversed starting from the head. For

each graph in the queue, worker $w$ first verifies if the directions of any missing edges are finalized in its missing edge store $M_w$. If finalized, the worker updates the global-order graph with these edges and removes the corresponding pairs from the set of missing edges. If no missing edges remain for the graph, the worker executes the graph and moves to the next element. The worker stops traversing when it encounters a global-order message with unresolved missing edges.

**Algorithm 3** Order Finalization

▷ BFT-DAG Construction and Total Ordering on primary $\pi$
**if** primary receiving $n - f$ $\langle \text{ACK}, d, r, w \rangle_{\sigma_{w'}}$ for $w$ and a $d = D((G^T, \mathcal{M}))$ in round $r$ **then**
  use Narwhal to construct BFT-DAG
**if** BFT-DAG vertices are totally ordered on primary using Tusk **then**
  Send each totally ordered vertex (digest $d$) to the corresponding worker
▷ Transaction execution on worker $w$
**while** worker receives totally ordered digest $D((G^T, \mathcal{M}))$ from primary **do**
  Insert $D((G^T, \mathcal{M}))$ into execution queue $Q$
  **while** $Q$ is not empty **do**
    $d = D((G^T, \mathcal{M}))$ Head of $Q$
    Retrieve $(G^T, \mathcal{M})$ from local storage using $d = D((G^T, \mathcal{M}))$
    Check missing edge store for directions of edges in $\mathcal{M}$
    **if** Any missing edge direction is finalized **then**
      Update $G^T$ with the finalized edge
      Remove the corresponding pair from $\mathcal{M}$
    **if** $\mathcal{M}$ is empty **then**
      Execute each transaction in $G^T$ using Khan's algorithm
      Remove $G^T$ from $Q$
    **else**
      **break** ▷ Stop as current $G^T$ has unresolved missing edges

The global-order graph is represented as a directed acyclic graph, allowing concurrent execution of transactions using Kahn's algorithm [33]. To optimize performance, DoD employs a thread pool for efficient resource utilization. The number of threads can be configured as a system parameter based on the hardware specifications, enabling adaptable scalability.

A transaction might access multiple objects. If all objects within a transaction $t$ belong to the same shard, the client simply sends $t$ to the corresponding worker within each party. However, if the objects span multiple shards, the client distributes the objects to the relevant workers across all parties. As a result, some workers may receive objects that are outside their designated shard. During order finalization, each worker updates only the objects associated with its shard. Since the other workers within the same party will update their respective objects, consistency is preserved. This design ensures *eventual consistency*, allowing all replicas to reach a consistent state after executing a block.

### 4.4 Correctness Arguments

We briefly discuss the order-fairness, safety, and liveness of DoD.

Lemma 4.1. *On a replica, if transaction $t$ is* fixed*, then $t$ is included in the global order for execution.*

Proof. DoD leverages Narwhal's Certificate of Availability. When a local-order message is created, it is broadcast to other replicas. Once $n-f$ replicas acknowledge the local-order message, a certificate of availability is formed. This certificate includes the local order's hash, ensuring that the local order is available and can be retrieved. When a replica needs to retrieve the content of a block, it uses the hash from the certificate of availability to request the block from other replicas. Since at

least $n - 2f$ honest replicas have stored the block, the requesting replica can retrieve the block by querying other replicas. The Certificate of Availability provides a reliable way for replicas to determine which local-order messages other replicas received in the previous round. A replica then constructs a global-order graph based on $n - f$ local-order messages, which are automatically verified since the local-order messages have already been validated. The replica includes all fixed and pending transactions in its global-order graph. During the pruning step, pending vertices (those with no outgoing paths to fixed vertices) are removed from the condensation graph. Since a vertex in the condensation graph is classified as pending only if it contains no fixed transactions, all fixed transactions are guaranteed to be included by the replica. □

Lemma 4.2. *For any data-dependent transactions $t$ and $t'$, the global order will establish either $(t, t')$ or $(t', t)$ as their ordering.*

Proof. The DoD protocol ensures that at least $n - 2f$ replicas in the quorum eventually transmit both transactions to a receiving replica. Given that $n - 2f$ exceeds $2(n(1-\gamma) + f)$, it follows that at least one of $w(t, t')$ or $w(t', t)$ meets or surpasses the threshold of $n(1-\gamma) + f + 1$. The protocol enforces a rule where only the higher-weight edge is retained in cases where both $w(t, t')$ and $w(t', t)$ exceed this threshold. Consequently, the final global-order graph contains precisely one of the two possible edges, ensuring a deterministic ordering between $t$ and $t'$. □

Lemma 4.3. *The graph $G^T$ remains acyclic.*

Proof. Although the original graph $G$ may contain Condorcet cycles, each such cycle is encapsulated within a single vertex, representing a strongly connected component in the condensation graph $G^C$. Since neither the graph pruning process nor the transitive reduction step introduces additional edges, the resulting graph maintains its acyclic nature. □

Lemma 4.4. *Given two order-dependent transactions $t$ and $t'$ received in round $i$, where $t$ is* fixed*, if a global order includes only $t$ on any valid replica, then at least $n(1-\gamma) + 1$ honest replicas must have received $t$ before $t'$.*

Proof. If transaction $t'$ were a fixed transaction, it would necessarily be included in the global order, so $t'$ is not fixed. If $t'$ is not a pending transaction, it has been received by at most $n(1-\gamma) + f$ replicas. Since $t$ is fixed, it appears in at least $n - 2f$ local-order messages. This means that the number of replicas that ordered $t$ before $t'$ is at least $p = (n - 2f) - (n(1-\gamma) + f) = \gamma n - 3f$. Given that $n > \frac{4f}{2\gamma - 1}$, it follows that $p > n(1-\gamma) + f$. Since at most $f$ of these replicas may be faulty, at least $n(1-\gamma) + 1$ honest replicas must have received $t$ before $t'$. If $t'$ is a pending transaction and is not included in the global order, there is no path from $t'$ to any fixed transaction, including $t$. This leads to two possible cases: either $w(t', t) \leq n(1-\gamma) + f$, or $n(1-\gamma) + f + 1 \leq w(t', t) \leq w(t, t')$. In the second case, at least $n(1-\gamma) + 1$ honest replicas must have received $t$ before $t'$. In the first case, since $t$ is fixed, $w(t, t') \geq \gamma n - 3f$, which is greater than $n(1-\gamma) + f$ given that $n > \frac{4f}{2\gamma - 1}$. Hence, in all cases, at least $n(1-\gamma) + 1$ honest replicas received $t$ before $t'$. □

Lemma 4.5. *Given two order-dependent transactions $t$ and $t'$, if a valid replica includes only $t$ in its global-order (and $t'$ was not included in an earlier proposal), and if $t'$ is received before $t$ by $\gamma n$ replicas, then $t$ and $t'$ form part of the same Condorcet cycle.*

Proof. Since $t$ appears in the global-order, it must be either fixed or pending. If $t$ is a fixed transaction, then at least $n(1-\gamma) + 1$ honest replicas must have received $t$ before $t'$ (by lemma 4.4), contradicting the assumption that $t'$ was received before $t$ by $\gamma n$ replicas. Therefore, $t$ must be pending. Because $t$ is pending, there must exist a path $t, t_1, t_2, ..., t_k$ leading to some fixed transaction

$t_k$ in the proposal. Since $t_k$ is fixed, lemma 4.4 ensures that at least $n(1-\gamma)+1$ honest replicas received $t_k$ before $t'$. Given that $t'$ was received before $t$ by $\gamma n$ replicas, the sequence $t, t_1, t_2, ..., t_k, t'$ forms a Condorcet cycle. □

Theorem 4.1. DoD guarantees data-dependent order-fairness.

Proof. Given two data-dependent transactions $t$ and $t'$ where $\gamma n$ replicas receive $t$ before $t'$. Four cases can happen in the final ordering. First, $t$ and $t'$ are proposed in the same valid block and in different vertices of the graph $G^T$ (then, following Algorithm 2, $t'$ is not ordered before $t$ since $\gamma n$ replicas receive $t$ before $t'$). Second, $t$ and $t'$ are proposed in the same valid block and in the same vertex of the graph $G^T$ (then, $t'$ is not ordered before $t$ – both are part of the same cycle). Third, $t'$ is proposed in a later valid block than $t$ (then, $t'$ is obviously not ordered before $t$), and fourth, $t$ is proposed in a later valid block than $t'$ (then, based on lemma 4.5, $t$ and $t'$ are in the same Condorcet cycle and $t'$ is not ordered before $t$). Hence, DoD guarantees data-dependent order-fairness. □

Theorem 4.2. DoD guarantees safety.

Proof. The safety of DoD is a direct consequence of the safety of Narwhal and Tusk [23], as DoD does not modify any aspects that could affect the safety of the protocol. □

Theorem 4.3. DoD guarantees liveness.

Proof. DoD assumes that a correct transaction $t$ will be (eventually) received by all replicas. As a result, $t$ will appear in at least $n-2f$ local-order messages (either in the same round or different rounds), and become fixed. Since Narwhal ensures that once a block is written and certified, it remains available for future reads, a fixed transaction will be proposed by a leader. The execution of $t$ only depends on missing edges between previously proposed order-dependent transactions, and as soon as such edges are added (i.e., corresponding transactions appear in $n-2f$ local-order messages), $t$ can be executed. However, the order of transaction $t$ does not depend on any transaction that has not been proposed; consequently, Condorcet cycles can not be chained and violate liveness. □

## 5 Experimental Evaluation

The main goal of our evaluation is to measure the efficiency of DoD compared to other fair BFT consensus protocols and the overhead introduced by order-fairness relative to Tusk [23], the underlying (unfair) DAG-based BFT consensus protocol of DoD. To the best of our knowledge, this is the first work to incorporate order-fairness into a DAG-based BFT protocol. For a well-rounded evaluation, we also benchmark our protocol against Themis [36] and Rashnu [50], two leading order-fairness protocols built on the HotStuff [70] consensus, and against HotStuff itself.

We examine how the performance of DoD is affected by different parameters, including: transaction batch size (Section 5.1), geo-distributed environment (Section 5.4), degree of contention (Section 5.2), and order-fairness parameter (Section 5.3).

The DoD implementation is built upon the Narwhal and Tusk protocol. We began with Narwhal's open-source codebase and modified its mempool worker to enforce fair transaction ordering. Instead of broadcasting Batches as Narwhal does, workers use Narwhal's mempool to disseminate local-order graphs. The global order is then determined locally and processed similarly to how Tusk handles Batches. During execution, DoD processes a global-order graph instead of a Batch, which maintains order fairness throughout the system. Another difference lies in the round progression mechanisms of the two systems. In Narwhal, a round advances with each block, and each block contains multiple batches. Each Narwhal batch is a list of transactions. By contrast, a DoD round advances with each batch, and each DoD batch contains a global dependency graph. This distinction enables

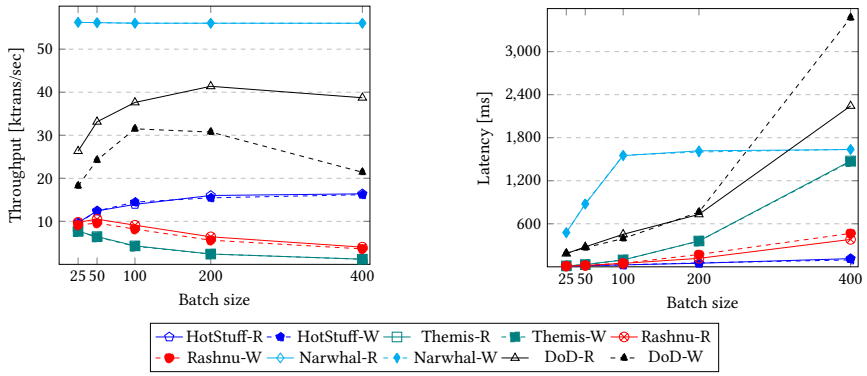


Fig. 7. Impact of batch size

DoD to establish a consistent transaction order at a finer granularity, independently of Narwhal's block-level progression.

We carry out all of our experimental evaluations across all 5 systems (Hotstuff, Themis, Rashnu, Narwhal, and DoD) utilizing the SmallBank benchmark. Smallbank consists of $n$ users, each having two types of accounts: checking and savings. There are seven types of transactions in total to read and update account balances, six of which modify account values and one that simply reads the account values based on predefined rules. We begin by populating the system with $10,000$ records and then evaluate each protocol under both read-heavy ($P_w = 0.05$) and write-heavy ($P_w = 0.95$) workloads, with DoD-R and DoD-W representing DoD in the respective read-heavy and write-heavy scenarios. Here $P_w$ is the probability of a transaction being a write (modifying) transaction, with its value ranging from 0 to 1.

The accounts accessed by each transaction are determined based on a Zipfian distribution, which can be adjusted for skewness, where $s = 0$ corresponds to a uniform distribution.

We run our experiments on a set of c6220 bare-metal machines on CloudLab [24], each with two Xeon E5-2650v2 processors (8 cores each, 2.6Ghz), 64GB RAM and two 1TB SATA 3.5" 7.2K rpm hard drives. These machines are connected by two networks, each with one interface: (1) a 1 Gbps Ethernet control network; (2) a 10 Gbps Ethernet commodity fabric. We report latency and throughput. The results reflect end-to-end measurements from the clients.

### 5.1 Performance with Different Batch Sizes

We first analyze how varying batch sizes affect the performance of different protocols across multiple workloads. By tuning the batch size, we aim to determine the optimal configuration that maximizes performance while balancing efficiency and fairness.

We evaluated five batch sizes: 25, 50, 100, 200, and 400. For this experiment, the system is configured with 5 replicas (assuming $f = 1$ in the $4f + 1$ model), the order-fairness parameter $\gamma = 1$, and accounts are selected uniformly at random. Figure 7 presents the results for all five protocols, with solid lines denoting read-heavy workloads and dashed lines representing write-heavy ones.

Across both read-heavy and write-heavy workloads, DoD consistently delivers better throughput compared to other fair protocols (Themis and Rashnu), reaching a peak throughput of over 41 ktps for read-heavy and 30 ktps for write-heavy workloads with a batch size of 200. Compared to HotStuff, DoD offers a significantly higher throughput while guaranteeing order fairness alongside the consensus protocol. With a batch size of 100, DoD-R and DoD-W demonstrate approximately 170% and 117% higher throughput, respectively.

Against Themis and Rashnu, the throughput advantage of DoD is even more pronounced, especially as the transaction batch size increases. At a batch size of 100, DoD-R achieves over 312%

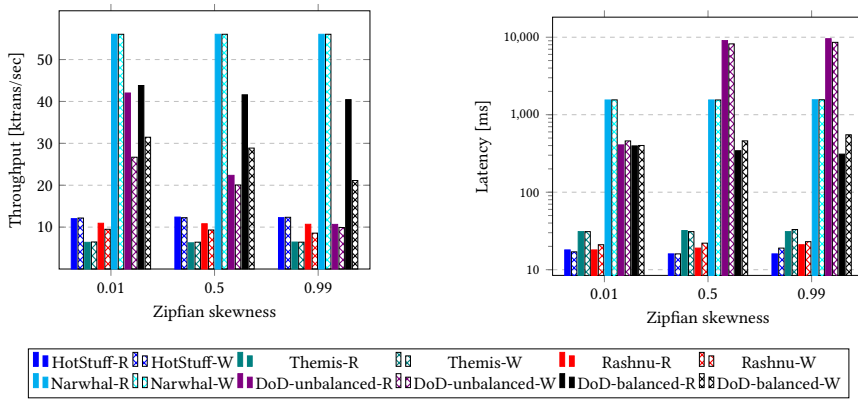


Fig. 8. Impact of workload contention

higher throughput compared to Rashnu, and approximately 775% higher throughput compared to Themis. This gap keeps increasing as the batch size increases. This is due to DoD's efficiency in processing transactions, which helps offset the overhead associated with generating and managing dependency graphs with higher batch sizes. As the batch size increases to 400, DoD experiences a slight decrease in throughput due to the added overhead of generating dependency graphs. However, it still delivers a significant improvement over other fair protocols.

Compared to Narwhal, however, DoD provides 28% lower throughput, which is the cost of providing order-fairness in DoD. In fact, in DoD, every single transaction needs to be received by all replicas. While in Narwhal, each replica receives a disjoint set of transactions. As a result, DoD has to process duplicate transactions, which leads to lower throughput compared to Narwhal. Notably, this high throughput in Narwhal comes at the cost of high latency. DoD-R, while showing lower throughput than Narwhal, offers a latency reduction of over 70% (453 vs. 1500 ms). This is because DoD leverages parallel transaction execution to reduce latency.

In general, DAG-BFT protocols incur much higher latency compared to traditional leader-based consensus protocols. This is because DAG-based protocols decouple DAG construction from the total ordering process. These protocols operate in waves, and each wave consists of multiple rounds where the leader is elected at the end of a wave, and only at this point, the constructed DAG in the wave is totally ordered, and transactions are executed. While this approach improves throughput, there is a significant delay between receiving transactions by a replica and their eventual execution, leading to considerable latency. Rashnu and Themis (bootstrapped from HotStuff) demonstrate lower latency compared to DoD (bootstrapped from Narwhal and Tusk). Furthermore, DoD shows a significant advantage in lower latency over Narwhal for batch sizes up to 200 due to its parallel transaction execution. At a batch size of 400, while its latency is higher, this is attributed to the overhead of dependency graph construction becoming the dominant factor. Finally, DoD's latency increases with batch size in both read-heavy and write-heavy workloads. This increase is more pronounced in write-heavy workloads at higher batch sizes due to the overhead associated with constructing, managing, and processing larger dependency graphs.

## 5.2 Varying the Degree of Contention

We next examine the impact of varying Zipfian skewness ($\alpha$) on the performance of different protocols. We change the skewness from $s = 0.01$ (uniform distribution) to $s = 0.5$ and $s = 0.99$ (contentious workload). The batch size is set to 100, with $n = 5$ replicas and the order-fairness parameter $\gamma = 1$. For this experiment, we configured DoD with two different shard configurations: DoD-balanced and DoD-unbalanced. DoD-unbalanced refers to the unbalanced load on each worker,

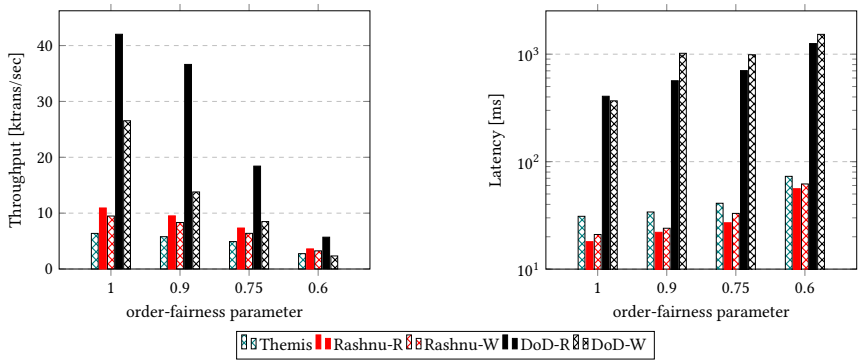


Fig. 9. Impact of the order-fairness parameter

where shard assignment to each worker is uniform, meaning an equal number of accounts (data items) are assigned to each worker. Consequently, a skewed workload leads to an unbalanced load across workers. Conversely, DoD-balanced aims for a balanced load on each worker. In this configuration, we manually assigned shards to workers to ensure that even with highly skewed data, the incoming transaction load is uniformly distributed. This allows different workers to process an equal number of transactions, potentially while maintaining an unequal number of accounts.

As shown in Figure 8, the performance of Narwhal, HotStuff, and Themis is not affected by increasing the workload skewness since they do not construct dependency graphs.

Under low contention ($\alpha = 0.01$), DoD-balanced-R achieves a throughput of 43.8 ktps, significantly outperforming HotStuff-R (12.0 ktps), Themis-R (6.4 ktps), and Rashnu-R (10.9 ktps) by roughly 265%, 587%, and 301%, respectively. Similarly, for write operations, DoD-balanced-W achieves 31.5 ktps, surpassing HotStuff-W, Themis-W, and Rashnu-W by about 158%, 385%, and 232%. In contrast, the DoD-unbalanced configuration shows a slightly lower throughput (42.0 ktps for reads and 26.7 ktps for writes), suggesting a less optimal initial shard assignment even under uniform load.

As contention increases ($\alpha = 0.5$), the impact of load imbalance becomes more pronounced for the unbalanced configuration of DoD, with throughput dropping to 22.4 ktps for reads and 20.1 ktps for writes. In comparison, DoD-balanced-R experiences a smaller decrease to 41.6 ktps, still significantly higher than HotStuff-R, Themis-R, and Rashnu-R by approximately 235%, 560%, and 285%, respectively. Similarly, DoD-balanced-W decreases to 28.9 ktps, still outperforming the other protocols except Narwhal-W. Under high contention ($\alpha = 0.99$), the throughput disparity between the balanced and unbalanced DoD configurations becomes even more evident, highlighting the role of shard assignment in handling increased workload skewness.

As DoD uses sharded workers, when the data is more skewed, a few workers receive most of the transactions, leading to more costly dependency graph generation. However, if the shard assignment for each worker is adjusted to better distribute the load, the throughput can increase even in the presence of more skewed data. Therefore, it is crucial to analyze the contention characteristics of the system to determine the appropriate shard assignment, ensuring an even load distribution among workers right from the initialization stage.

The impact of shard assignment is also evident in latency. As contention increases, the latency for balanced DoD-R remains relatively stable around 300 ms, and balanced DoD-W exhibits a moderate increase from 400 ms to 550 ms. In contrast, the unbalanced configuration suffers a dramatic surge in latency, reaching 9575 ms for reads and 8590 ms for writes at $\alpha = 0.99$. This extreme increase in latency for the unbalanced case shows that overloaded workers not only reduce throughput but also significantly increase the processing time per transaction, leading to higher latency.

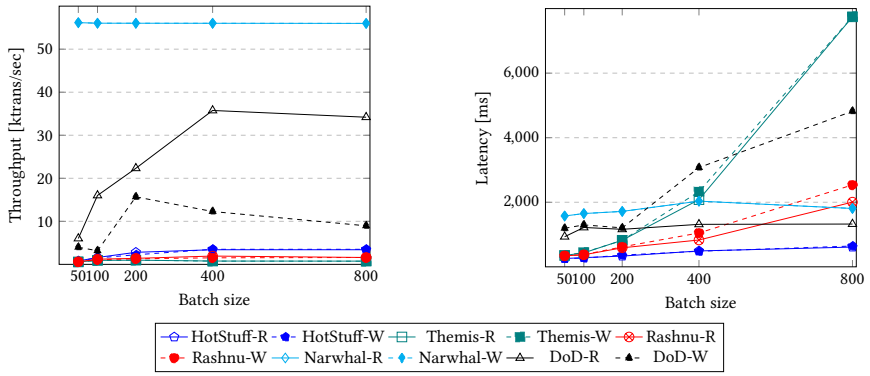


Fig. 10. Impact of batch size in a geo-distributed setting

### 5.3 Varying the Order-fairness Parameter

We next analyze the performance of DoD, Rashnu, and Themis under varying order-fairness parameters, as shown in Figure 9. The network size is dictated by the order-fairness parameter, following the formula $n = \frac{4f}{2\gamma-1} + 1$. A decrease in $\gamma$ leads to an increase in the required network size, $n$. We evaluate protocol performance with $f = 1$ at various $\gamma$ values: $\gamma = 1$ ($n = 5$), $\gamma = 0.9$ ($n = 6$), $\gamma = 0.75$ ($n = 9$), $\gamma = 0.6$ ($n = 21$), and $\gamma = 0.55$ ($n = 41$). All experiments are conducted using a batch size of 100 and uniform account selection.

Figure 9 illustrates the throughput and latency across protocols for different $\gamma$ values. In terms of throughput, DoD consistently outperforms both Rashnu and Themis across most settings. At $\gamma = 1$, DoD-R achieves approximately 42.0 ktps throughput, while DoD-W reaches 26.5 ktps. As $\gamma$ decreases, the network size needs to be expanded. This increase in network size causes communication overhead to become more significant than the cost of fair ordering, which eventually leads to a gradual drop in throughput and an increase in network latency of DoD. However, it still performs better than the other protocols throughout the entire range.

### 5.4 Geo-distributed Setting

In this set of experiments, we measure the performance of protocols in a geo-distributed setup. We repeat the first set of experiments (Section 5.1) with an additional 50 ms latency (injected by Linux `netem`) for sending messages between any pair of replicas. The results are shown in Figure 10.

As expected, the throughput of all protocols except Narwhal decreases once network latency is introduced. Interestingly, DoD, Rashnu and Themis achieve peak throughput in distributed settings at larger blocks compared to the local setup (Figure 7). Specifically, DoD-R and Rashnu reach their highest throughput at a block size of 400, while DoD-W and Themis peak at 200. In contrast, in the local setting, DoD-R and DoD-W achieve their best throughput at block sizes of 200 and 100, respectively. This shift highlights a trade-off between communication latency and fair ordering: when latency is high, the cost of communication outweighs the overhead of fair ordering, making larger block sizes more beneficial. Conversely, with low communication latency, the overhead of fair ordering dominates, and thus, smaller blocks yield better performance.

At its peak throughput (block size of 400), DoD-R experiences a $\sim$ 38% throughput reduction compared to Narwhal (compared to 26% reduction in the local setting with block size 200): high network latency has a lower impact on DoD-R's throughput for read-only transactions. The contrast is more pronounced for DoD-W, which sees a 70% reduction at block size 200 in the local setting, versus a 40% reduction at block size 100 in the distributed setting. This wider gap arises due to the additional overhead of graph construction and the need to wait for missing edges in DoD-W.

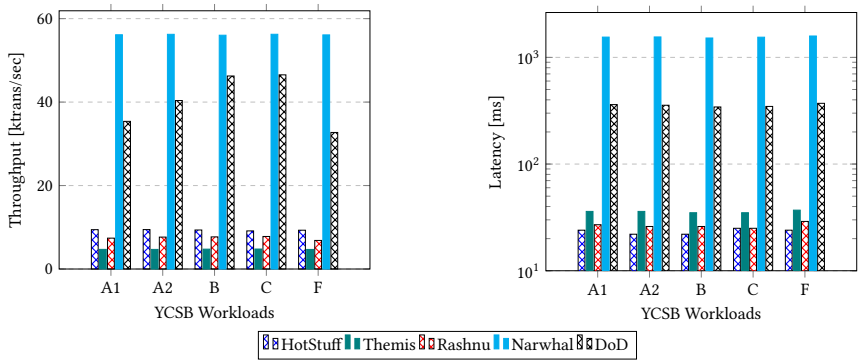


Fig. 11. Performance under YCSB workload

Despite the reduction in throughput, DoD-R achieves a peak throughput 10× higher than HotStuff, while DoD-W reaches 4.2× HotStuff's throughput. Regarding latency, DoD-R demonstrates stable performance across varying batch sizes. Although its latency is higher than that of HotStuff, Themis, and Rashnu for smaller blocks, specifically those under 200, it becomes significantly lower than Themis and Rashnu for larger blocks exceeding 200. In contrast, DoD-W's latency increases with larger block sizes due to the computational overhead of dependency graph construction.

### 5.5 Performance with YCSB Workloads

In the last set of experiments, we study the performance of different protocols under YCSB workloads. We have chosen 5 different (most relevant) workloads from the YCSB benchmark: A: update heavy workload (A1: R/W = 5/95 and A2: R/W = 50/50), B: read-mostly workload (R/W = 95/5), C: read-only workload (R/W = 100/0), and F: read-modify-write workload. In this set of experiments, the batch size is 100, $n = 5$, $\gamma = 1$, and Zipfian skewness is $s = 0.01$ (uniform distribution). As shown in Figure 11, DoD demonstrates lower throughput compared to Narwhal, while its throughput is much higher than Themis, Rashnu, and Hotstuff under different workloads. Remarkably, DoD demonstrates a throughput that is 275% to 410% higher than the state-of-the-art, traditional BFT consensus protocol, HotStuff, across various workloads. As expected, DoD demonstrates its best performance under the read-only workload (C). Overall, both the throughput and the latency results are consistent with the results of the SmallBank benchmark.

## 6 Conclusion

This paper introduces DoD, a high-performance order-fairness protocol. DoD effectively addresses the performance limitations associated with existing ordering fairness protocols by leveraging the high-performance DAG-based BFT consensus protocol, Narwhal and Tusk, and by incorporating the concept of data-dependent order fairness to reduce the costs associated with establishing a global transaction ordering. DoD achieves fairness through the organization of transactions within individual DAGs, subsequently ordering these DAGs within a higher-level BFT consensus. This methodology facilitates concurrent transaction processing while preserving a fair ordering of transactions. Experimental results highlight DoD's ability to balance the trade-off between fairness and performance effectively, achieving consistently high throughput and low latency across a range of configurations and workloads.

## 7 Acknowledgments

This work was supported in part by NSF IIS-2436080: EAGER: Synthesizing and Optimizing Declarative Smart Contracts, 2025.

## References


[1] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. 2017. Solida: A Blockchain Protocol Based on Reconfigurable Byzantine Consensus. In *Int. Conf. on Principles of Distributed Systems (OPODIS)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[2] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2021. SharPer: Sharding Permissioned Blockchains Over Network Clusters. In *SIGMOD Int. Conf. on Management of Data*. ACM, 76–88.

[3] Mohammad Javad Amiri, Divyakant Agrawal, Amr El Abbadi, and Boon Thau Loo. 2024. Distributed Transaction Processing in Untrusted Environments.. In *SIGMOD Int. Conf. on Management of Data*. ACM, 570–579.

[4] Mohammad Javad Amiri, Boon Thau Loo, Divyakant Agrawal, and Amr El Abbadi. 2022. Qanaat: A Scalable Multi-Enterprise Permissioned Blockchain System with Confidentiality Guarantees. *Proc. of the VLDB Endowment* 15, 11 (2022), 2839–2852.

[5] Mohammad Javad Amiri, Sujaya Maiyya, Divyakant Agrawal, and Amr El Abbadi. 2020. SeeMoRe: A fault-tolerant protocol for hybrid cloud environments. In *Int. Conf. on Data Engineering (ICDE)*. IEEE, 1345–1356.

[6] Mohammad Javad Amiri, Chenyuan Wu, Divyakant Agrawal, Amr El Abbadi, Boon Thau Loo, and Mohammad Sadoghi. 2024. The Bedrock of Byzantine Fault Tolerance: A Unified Platform for {BFT} Protocols Analysis, Implementation, and Experimentation. In *Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 371–400.

[7] Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. 2025. Shoal++: High Throughput DAG BFT Can Be Fast and Robust!. In *Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association.

[8] Balaji Arun and Binoy Ravindran. 2022. Scalable Byzantine Fault Tolerance via Partial Decentralization. *Proc. of the VLDB Endowment* 15, 9 (2022), 1739–1752.

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

[10] Carsten Baum, James Hsin-yu Chiang, Bernardo David, Tore Kasper Frederiksen, and Lorenzo Gentile. 2021. Sok: Mitigation of front-running in decentralized finance. *Cryptology ePrint Archive* (2021).

[11] Tara Siegel Bernard and Ron Lieber. [n. d.]. Banks Are Closing Customer Accounts, With Little Explanation. https://www.nytimes.com/2023/04/08/your-money/bank-account-suspicious-activity.html.

[12] Gabriel Bracha. 1984. An asynchronous [(n-1)/3]-resilient consensus protocol. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 154–162.

[13] Gabriel Bracha and Sam Toueg. 1985. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)* 32, 4 (1985), 824–840.

[14] Yehonatan Buchnik and Roy Friedman. 2020. FireLedger: a high throughput blockchain consensus protocol. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1525–1539.

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

[16] Christian Cachin, Jovana Mićić, and Nathalie Steinhauer. 2022. Quick Order Fairness. In *Int. Conf. on Financial Cryptography and Data Security (FC)*. Springer, 1–18.

[17] Chainlink. 2023. What Is Maximal Extractable Value (MEV)? https://chain.link/education-hub/maximal-extractable-value-mev.

[18] Feng Cheng, Jiang Xiao, Cunyang Liu, Shijie Zhang, Yifan Zhou, Bo Li, Baochun Li, and Hai Jin. 2024. Shardag: Scaling dag-based blockchains via adaptive sharding. In *Int. Conf. on Data Engineering (ICDE)*. IEEE, 2068–2081.

[19] Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. 2006. From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. *Comput. J.* 49, 1 (2006), 82–96.

[20] Tyler Crain, Christopher Natoli, and Vincent Gramoli. 2021. Red Belly: a secure, fair and scalable open blockchain. In *Symposium on Security and Privacy (SP)*. IEEE.

[21] Xiaohai Dai, Zhaonan Zhang, Zhengxuan Guo, Chaozheng Ding, Jiang Xiao, Xia Xie, Rui Hao, and Hai Jin. 2024. Wahoo: A dag-based bft consensus with low latency and low communication overhead. *Transactions on Information Forensics and Security* (2024).

[22] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *Symposium on Security and Privacy (SP)*. IEEE, 910–927.

[23] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *European Conf. on Computer Systems (EuroSys)*. 34–50.

[24] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, et al. 2019. The Design and Operation of {CloudLab}. In *Annual Technical Conf. (ATC)*. USENIX Association, 1–14.

[25] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)* 35, 2 (1988), 288–323.

[26] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. 2019. Sok: Transparent dishonesty: front-running attacks on blockchain. In *Int. Conf. on Financial Cryptography and Data Security (FC)*. Springer, 170–189.

[27] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 51–68.

[28] Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. 2024. Autobahn: Seamless high speed BFT. In *Symposium on Operating Systems Principles (SOSP)*. ACM SIGOPS, 1–23.

[29] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2020. ResilientDB: Global Scale Resilient Blockchain Fabric. *Proceedings of the VLDB Endowment* 13, 6 (2020), 868–883.

[30] Lioba Heimbach and Roger Wattenhofer. 2022. SoK: Preventing Transaction Reordering Manipulations in Decentralized Finance. In *Conf. on Advances in Financial Technologies (AFT)*. ACM, 1–14.

[31] Zicong Hong, Song Guo, Enyuan Zhou, Wuhui Chen, Huawei Huang, and Albert Zomaya. 2023. GriDB: Scaling Blockchain Database via Sharding and Off-Chain Cross-Shard Mechanism. *Proceedings of the VLDB Endowment* 16, 7 (2023), 1685–1698.

[32] Philipp Jovanovic, Lefteris Kokoris Kogias, Bryan Kumara, Alberto Sonnino, Pasindu Tennage, and Igor Zablotchi. 2024. Mahi-mahi: Low-latency asynchronous bft dag-based consensus. *arXiv preprint arXiv:2410.08670* (2024).

[33] Arthur B Kahn. 1962. Topological sorting of large networks. *Commun. ACM* 5, 11 (1962), 558–562.

[34] Dakai Kang, Suyash Gupta, Dahlia Malkhi, and Mohammad Sadoghi. 2025. Hotstuff-1: Linear consensus with one-phase speculation. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–29.

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

[36] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. 2023. Themis: Fast, strong order-fairness in byzantine consensus. In *SIGSAC Conf. on Computer and Communications Security (CCS)*. ACM, 475–489.

[37] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. 2020. Order-fairness for byzantine consensus. In *Annual Int. Cryptology Conf.* Springer, 451–480.

[38] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual Int. Cryptology Conf.* Springer, 357–388.

[39] Ariah Klages-Mundt and Andreea Minca. 2019. (In) stability for the blockchain: Deleveraging spirals and stablecoin attacks. *arXiv preprint arXiv:1906.02152* (2019).

[40] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *Symposium on Security and Privacy (SP)*. IEEE, 583–598.

[41] Klaus Kursawe. 2020. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *Conf. on Advances in Financial Technologies (AFT)*. ACM, 25–36.

[42] Klaus Kursawe. 2021. Wendy Grows Up: More Order Fairness. In *Int. Conf. on Financial Cryptography and Data Security (FC)*. Springer, 191–196.

[43] Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* 21, 7 (1978), 558–565.

[44] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine generals problem. *Transactions on Programming Languages and Systems (TOPLAS)* 4, 3 (1982), 382–401.

[45] Kfir Lev-Ari, Alexander Spiegelman, Idit Keidar, and Dahlia Malkhi. 2019. FairLedger: A Fair Blockchain Protocol for Financial Institutions. In *Int. Conf. on Principles of Distributed Systems (OPODIS)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[46] Zhuolun Li and Evangelos Pournaras. 2024. SoK: Consensus for Fair Message Ordering. *arXiv preprint arXiv:2411.09981* (2024).

[47] Dahlia Malkhi, Chrysoula Stathakopoulou, and Maofan Yin. 2024. BBCA-CHAIN: Low latency, high throughput BFT consensus on a DAG. In *Int. Conf. on Financial Cryptography and Data Security (FC)*. Springer, 51–73.

[48] Dahlia Malkhi and Pawel Szalachowski. 2022. Maximal Extractable Value (MEV) Protection on a DAG. In *Int. Conf. on Blockchain Economics, Security and Protocols (Tokenomics)*. 1–17.

[49] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Conf. on Computer and Communications Security (CCS)*. ACM, 31–42.

[50] 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.

[51] Faisal Nawab and Mohammad Sadoghi. 2024. Consensus in Data Management: With Use Cases in Edge-Cloud and Blockchain Systems. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4233–4236.

[52] Rafael Pass and Elaine Shi. 2017. Hybrid Consensus: Efficient Consensus in the Permissionless Model. In *Int.Symposium on Distributed Computing (DISC)*. 6.

[53] Zeshun Peng, Yanfeng Zhang, Qian Xu, Haixu Liu, Yuxiao Gao, Xiaohua Li, and Ge Yu. 2022. Neuchain: a fast permissioned blockchain system with deterministic ordering. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2585–2598.
[54] Haoyun Qin, Chenyuan Wu, Mohammad Javad Amiri, Ryan Marcus, and Boon Thau Loo. 2024. BFTGym: An Interactive Playground for BFT Protocols. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4261–4264.
[55] Kaihua Qin, Liyi Zhou, and Arthur Gervais. 2022. Quantifying blockchain extractable value: How dark is the forest?. In *Symposium on Security and Privacy (SP)*. IEEE, 198–214.
[56] Mayank Raikwar, Nikita Polyanskii, and Sebastian Müller. 2024. SoK: DAG-Based Consensus Protocols. In *Int. Conf. on Blockchain and Cryptocurrency (ICBC)*. IEEE, 1–18.
[57] Noel Randewich and Arasu Kannagi Basil. [n. d.]. NYSE glitch sparks volatility in dozens of stocks. https://www.reuters.com/markets/us/nyse-equities-investigating-reported-technical-issue-2024-06-03/.
[58] Fred B Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *Computing Surveys (CSUR)* 22, 4 (1990), 299–319.
[59] Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, and Kartik Nayak. 2024. Sailfish: Towards Improving the Latency of DAG-based BFT. In *Symposium on Security and Privacy (SP)*. IEEE, 21–21.
[60] Atul Singh, Tathagata Das, Petros Maniatis, Peter Druschel, and Timothy Roscoe. 2008. BFT Protocols Under Fire.. In *Symposium on Networked Systems Design and Implementation (NSDI)*, Vol. 8. USENIX Association, 189–204.
[61] Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li. 2024. Shoal: Improving dag-bft latency and robustness. In *Int. Conf. on Financial Cryptography and Data Security (FC)*. Springer, 92–109.
[62] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. 2022. Bullshark: Dag bft protocols made practical. In *ACM SIGSAC Conf. on Computer and Communications Security (CCS)*. 2705–2718.
[63] Chrysoula Stathakopoulou, Signe Rüsch, Marcus Brandenburger, and Marko Vukolić. 2021. Adding Fairness to Order: Preventing Front-Running Attacks in BFT Protocols using TEEs. In *Int. Symp on Reliable Distributed Systems (SRDS)*. IEEE, 34–45.
[64] Weijie Sun, Zihuan Xu, Wangze Ni, and Lei Chen. 2025. InTime: Towards Performance Predictability In Byzantine Fault Tolerant Proof-of-Stake Consensus. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–27.
[65] Catherine Thorbecke. [n. d.]. Silicon Valley Bank collapse sends tech startups scrambling. https://www.cnn.com/2023/03/10/tech/silicon-valley-bank-tech-panic/index.html.
[66] Chenyuan Wu, Mohammad Javad Amiri, Haoyun Qin, Bhavana Mehta, Ryan Marcus, and Boon Thau Loo. 2024. Towards Full Stack Adaptivity in Permissioned Blockchains. *Proc. of the VLDB Endowment* 17, 5 (2024), 1073–1080.
[67] Chenyuan Wu, Bhavana Mehta, Mohammad Javad Amiri, Ryan Marcus, and Boon Thau Loo. 2023. AdaChain: A Learned Adaptive Blockchain. *Proc. of the VLDB Endowment* 16, 8 (2023), 2033–2046.
[68] David Yakira, Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, and Ronen Tamari. 2021. Helix: A Fair Blockchain Consensus Protocol Resistant to Ordering Manipulation. *IEEE Transactions on Network and Service Management* 18, 2 (2021), 1584–1597.
[69] Hiroyuki Yamada and Jun Nemoto. 2022. Scalar DL: Scalable and practical Byzantine fault detection for transactional database systems. *Proceedings of the VLDB Endowment* 15, 7 (2022), 1324–1336.
[70] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT consensus with linearity and responsiveness. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 347–356.
[71] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. 2020. Byzantine ordered consensus without Byzantine oligarchy. In *Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 633–649.
[72] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. 2021. High-frequency trading on decentralized on-chain exchanges. In *Symposium on Security and Privacy (SP)*. IEEE, 428–445.

Received April 2025; revised July 2025; accepted August 2025