FIDES: Secure and Scalable Asynchronous DAG Consensus via Trusted Components

Shaokang Xie
University of California, Davis
skxie@ucdavis.edu

Dakai Kang
University of California, Davis
dakang@ucdavis.edu

Hanzheng Lyu
University of British Columbia, Okanagan Campus
hzlyu@student.ubc.ca

Jianyu Niu*
City University of Hong Kong
njianyu@gmail.com

Mohammad Sadoghi*
University of California, Davis
msadoghi@ucdavis.edu

ABSTRACT

DAG-based BFT consensus has attracted growing interest in distributed data management systems for consistent replication in untrusted settings due to its high throughput and resilience to asynchrony. However, existing protocols still suffer from high communication overhead and long commit latency. In parallel, introducing minimal hardware trust has proven effective in reducing the complexity of BFT consensus.

Inspired by these works, we present FIDES, an asynchronous DAG-based BFT consensus protocol that, to our knowledge, is among the first to leverage TEEs to enhance both scalability and efficiency. FIDES tolerates a minority of Byzantine replicas and achieves $O(\kappa n^2 + n^3)$ metadata communication complexity through a customized TEE-assisted Reliable Broadcast (*T-RBC*) primitive with linear communication complexity in one-step broadcast. Building on T-RBC, FIDES redefines the DAG construction rules by reducing the reference requirement from $2f+1$ to $f+1$ between consecutive vertices. This new structure weakens DAG connectivity and invalidates traditional commit rules, so we formally abstract the problem and derive new theoretical bounds on liveness. We further propose a *four-round commit rule* that achieves the theoretically minimal commit latency. In addition, we design two primitives, *T-RoundCert* and *T-Coin*, to efficiently certify DAG references and replace the costly cryptographic common coin used in prior protocols. Comprehensive evaluations on geo-distributed and local testbeds show that FIDES substantially outperforms state-of-the-art protocols, including Tusk, Bullshark, Mysticeti, Shoal++, RCC, Damysus, Achilles and HybridSet, achieving lower latency and higher throughput while preserving strong safety and liveness guarantees.

PVLDB Reference Format:
Shaokang Xie, Dakai Kang, Hanzheng Lyu, Jianyu Niu, and Mohammad Sadoghi. FIDES: Secure and Scalable Asynchronous DAG Consensus via Trusted Components. PVLDB, 19(11): 3302 - 3316, 2026.
doi:10.14778/3836663.3836690

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

*Jianyu Niu and Mohammad Sadoghi are corresponding authors.

Proceedings of the VLDB Endowment, Vol. 19, No. 11 ISSN 2150-8097.
doi:10.14778/3836663.3836690

1 INTRODUCTION

Modern distributed data management systems rely on replication to keep data consistent and available despite failures. As deployments scale to unreliable, geo-distributed, and even untrusted environments, Byzantine fault tolerance (BFT) consensus has gained significant interest as a backbone for fault-tolerant replicated databases. Notable examples include the rapid rise of blockchains [7, 17, 76] and Web3 applications [44, 64], which require BFT to order transactions in decentralized and Byzantine networks. However, geo-distributed deployments with fluctuating network conditions [10, 20, 23, 49] pose new challenges for classic BFT protocols, such as PBFT [19], which are typically designed for partially synchronous networks [31]. Prior studies have shown that these protocols may fail to guarantee liveness (*i.e.*, the sequence of agreed transactions is ever-growing) in fully asynchronous environments [1, 19, 31, 59].

To remove the reliance on network synchrony, a series of fully asynchronous protocols [1, 26, 27, 32, 43, 47, 50, 59, 73] have been proposed to ensure consensus under arbitrary message delays. Among them, asynchronous *Directed Acyclic Graph* (DAG)-based protocols [26, 27, 43, 47, 73] have gained particular prominence due to their efficiency, robustness, and conceptual simplicity. They form a DAG of proposals from all replicas, avoid single-leader bottlenecks, commit transactions directly from the DAG without extra communication, and preserve asynchronous liveness with optimal amortized communication complexity. Representative examples include Tusk [27] and Bullshark [73].

Asynchronous DAG-based protocols require $n=3f+1$ replicas to tolerate up to f Byzantine ones (*i.e.*, behaving arbitrarily). They proceed in waves, each comprising multiple rounds. In each round, all replicas use reliable broadcast (RBC) to disseminate proposals (*i.e.*, vertices), each carrying pending transactions and references to at least $2f+1$ proposals from the previous round, thereby forming a DAG. At the end of each wave, replicas invoke a cryptographic common coin to collectively select a leader vertex from the first round of that wave, and decide to commit the leader vertex and its referencing proposals according to the commit rules. This enables replicas to produce a consistently ordered sequence of transactions.

Despite their promising features, DAG-based BFT protocols still struggle to scale in large-scale decentralized applications. *First*,

the *large system size* requirement of n=3f+1 replicas causes performance (*e.g.*, throughput) to degrade sharply once the number of replicas reaches several tens (see Section 9). *Second*, these protocols incur *high communication complexity*: each round runs n instances of reliable broadcast (RBC), each with $\Theta(n^2)$ communication cost, and requires validating $O(n^2)$ references per round (up to n vertices) [14, 27, 47]. *Third*, they experience *long commit latency*, especially under adversarial asynchronous networks. Table 1 shows that asynchronous DAG protocols guarantee commitment but still require either many message steps or larger committees, while partially synchronous DAG protocols such as Shoal++, Sailfish, and Mysticeti fail to ensure progress under asynchrony.

In this paper, we introduce Fides, an asynchronous DAG-based BFT consensus protocol that, to our knowledge, is among the first to leverage Trusted Execution Environments (TEEs) to enhance both scalability and efficiency. However, using TEEs in DAG protocols is *not plug-and-play*. The key challenge is to *identify the specific bottlenecks in DAG-based consensus* and address them with trusted components built atop TEEs by following the minimized Trusted Computing Base (TCB) principle [13, 28, 29, 37, 48, 62]. To this end, we employ Monotonic Counter (MC), a trusted component to prevent message equivocation by binding each message to a unique counter value. With it, we design a *TEE-assisted Reliable Broadcast (T-RBC)* primitive with linear communication complexity in one-step broadcast. By incorporating T-RBC, Fides can tolerate a minority of Byzantine replicas and achieve $O(\kappa n^2 + n^3)$ communication complexity.

T-RBC also enables us to relax the DAG construction rule: each vertex now needs only f+1 references to the previous round, instead of 2f+1. However, this relaxation creates a *new liveness pitfall* that is specific to TEE-assisted DAG protocols and is absent from prior TEE-assisted (non-DAG) BFT designs [9, 13, 29, 37, 48, 62, 79, 86]. In Tusk's wave-based commitment [27], progress implicitly relies on sufficient overlap between the reference sets of consecutive rounds. With only f+1 references, the overlap between two such sets can drop to one honest replica in the worst case (rather than at least f+1 under the original 2f+1 rule). An adversary that controls message scheduling can then indefinitely delay that replica's messages, suppressing the only bridging references and causing the protocol to lose liveness under adversarial asynchrony.

To resolve this pitfall, we abstract the wave-level commit conditions as a k-iteration common core problem (Section 7.1), building on insights from prior DAG protocols [27, 47, 73]. The parameter k is the number of rounds per wave. Using this abstraction, we derive that the minimum $k = 3$ suffices for liveness, while setting $k = 4$ is necessary to achieve constant-round commit latency. The gap stems from the fact that, with $k = 3$, commits can require $\Theta(f)$ additional rounds in the worst case; in contrast, increasing to $k = 4$ yields constant-round commits and achieves the optimal expected commit latency within our model.

We propose the Round Advancement Certifier (RAC), a trusted component that validates a vertex's collected references to the previous round and issues a compact cryptographic certificate when the references are valid. Building on RAC, we design *TEE-assisted Round Certifier (T-RoundCert)* so that replicas can quickly verify references and advance to the next round. Besides, we propose *TEE-assisted Common Coin (T-Coin)* that utilizes TEEs' confidential

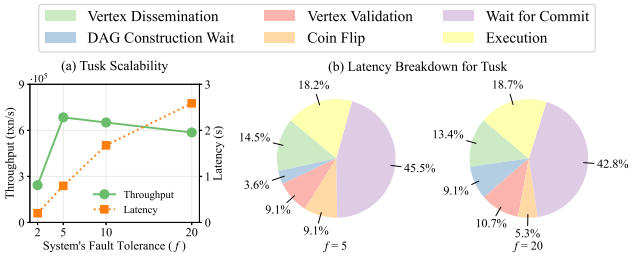

Figure 1: The Scalability and Latency Breakdown for Tusk.

property to provide a lightweight Common Coin, replacing costly threshold-cryptographic constructions.

We built end-to-end prototypes of Fides atop the open-source Apache ResilientDB platform [6, 36] and used the Open Enclave SDK [63] to develop trusted components on *Intel SGX* [24]. We conducted extensive experiments in the public cloud to evaluate and compare Fides with several representative protocols, including Tusk [27], Bullshark [73], Mysticeti [11], Shoal++ [8], RCC [34], Damysus [28], Achilles [62], and HybridSet [86]. Evaluation results in geo-distributed and local environments show that Fides achieves peak throughput of 400k tps and 810k tps, respectively, the highest among all compared protocols. We also examine its runtime overhead through fault-injection tests and a detailed stage-by-stage and component-based performance breakdown.

Contributions. We propose Fides, to our knowledge, one of the first asynchronous TEE-assisted DAG-based consensus protocols:

- **TEE-assisted DAG with n=2f+1 replicas and $O(\kappa n + n^2)$ bits per proposal.** We reduce the replica requirement of asynchronous DAG protocols from n=3f+1 to n=2f+1 while tolerating f Byzantine faults, enabled by *T-RBC*. Additionally, *T-RBC* allows Fides to reduce dissemination and referencing overhead, improving message complexity to $O(\kappa n + n^2)$ bits per proposal.
- **Reduced-quorum liveness and commitment.** We identify a liveness pitfall introduced by relaxing cross-round references and develop a four-round commit rule, achieving constant (expected) commit latency under adversarial asynchrony.
- **Lower computation overhead.** Fides avoids crypto-heavy components (notably threshold-cryptographic common coins) via *T-RoundCert* (powered by *RAC*) and a lightweight TEE-backed common coin (*T-Coin* with *RNG*).
- **Implementation and evaluation.** We implement Fides and evaluate it against state-of-the-art protocols, attributing costs and quantifying the performance gains of each trusted component.

2 BACKGROUND AND MOTIVATION

2.1 Trusted Execution Environment

Trusted Execution Environments (TEEs) are hardware-enforced, isolated execution environments that protect code and data even from a malicious OS or hypervisor. For example, in Intel SGX, the isolated environments are known as enclaves, whose memory is transparently encrypted and isolated. Besides, there are two key mechanisms of TEEs: attestation, which enables TEEs to prove remotely that a particular enclave with a known code identity is

running securely, and sealing, which encrypts enclave state for persistent storage tied to enclave identity [38, 39, 41].

2.2 Asynchronous DAG-Based Consensus

Asynchronous DAG-based consensus protocols [27, 47, 73] operate round-by-round. In each round, every replica proposes a block containing a batch of transactions, and each block references at least $2f+1$ blocks (*i.e.*, $\Theta(n)$) from the previous round. These blocks act as *vertices*, and the reference links between them serve as *edges*, collectively forming a DAG. As a result, a replica has to verify $\Theta(n^2)$ references given at most n proposals in a round during the DAG construction of one round. Each reference is a κ-bit hash [14, 27, 47]. After vertex validation, the proposed blocks then go through *reliable broadcast (RBC)* to be delivered by replicas. The agreement property of RBC ensures at most one vertex from a replica is produced for each round (Appendix A.1 [83]). The above process is also referred to as the dissemination phase.

Rounds are grouped into consecutive waves, each wave comprising a specific number of rounds. The rounds in a wave collectively form a *common core* problem, determining how many vertices from the first round can be safely committed [27, 47, 73]. At the end of each wave, a vertex is randomly elected as the *leader vertex* using a *global common coin* (Appendix A.2 [83]), which produces the same random value across all replicas. If the leader vertex is in the common core, replicas can commit the vertex and its referenced vertices (*i.e.*, transactions) in a consistent order. The commitment requires no additional communication overhead to achieve transaction ordering.

2.3 Dissecting DAG Consensus

We dissect the performance of Tusk, a representative asynchronous DAG protocol that is extended by many later DAG protocols [27, 72, 73]. We evaluate throughput and latency of Tusk under varying numbers of nodes in a LAN setting, as shown in Figure 1. The experimental results reveal the following findings.

- **Large system size leads to poor scalability.** Throughput decreases and latency increases as the system scales, reflecting the potential inherent cost of a large quorum.
- **RBC dominates latency.** RBC accounts for one of the largest parts of end-to-end latency (15% in LAN and more than 50% in WAN), making it a primary scalability bottleneck.
- **Referencing overhead at scale.** Encoding $\Theta(n)$ references to previous-round vertices inflates per-vertex metadata by $\Theta(n)$, slowing propagation and validation at large n.
- **Common coin primitives are costly and fragile.** Tusk's threshold-signature-based coin adds about 5–10% to end-to-end latency. Worse, recent work on cross-block front-running attacks reveals the coin's potential vulnerabilities [70].

These insights generalize across DAG protocols. Building on them, Fides employs TEE-assisted services to efficiently address the first four bottlenecks (Section 5):

- We enhance fault tolerance to $n{=}2f{+}1$ via *T-RBC* to improve scalability, while systematically analyzing and resolving the *liveness challenges* arising from the quorum reduction.
- *T-RBC* further optimizes dissemination complexity and latency.
- *T-RoundCert* cuts crypto referencing complexity and overhead.
- *T-Coin* accelerates secure randomness generation.

By utilizing the above four ideas in Fides, Table 1 further summarizes how Fides performs compared to representative DAG protocols in different aspects.

3 SYSTEM MODEL AND GOALS

3.1 System Model

Replicas with TEEs. We consider a network Π consisting of $n{=}2f{+}1$ replicas, denoted by $\Pi = \{p_i\}_{i=1}^{n}$. Each replica p_i has a unique identity known to all others within the network. Each replica runs on an SGX-enabled machine that provides a Trusted Execution Environment (TEE), whose integrity and confidentiality are guaranteed by the underlying hardware. Up to f replicas may be Byzantine and behave arbitrarily. The remaining replicas are honest and strictly follow the protocol. In addition to replicas, there is an unbounded set of clients that create transactions.

Threat Model. We adopt the threat model used in prior TEE-assisted BFT protocols [9, 13, 28, 62, 79, 85]. We assume an adversary that controls up to f Byzantine replicas and any number of clients. The adversary has root-level access to corrupted replicas except for TEEs, enabling control over the network. In other words, all components outside the TEEs are considered vulnerable to compromise. In contrast, the TEE itself is assumed to remain secure and untampered with, maintaining its core trusted functionalities. We also assume that TEEs support secure remote attestation, providing unforgeable cryptographic evidence that a specific program is running inside enclaves [30, 41]. Following prior TEE-assisted BFT work [9, 13, 28, 79, 80, 85], we do not consider most TEE-specific attacks (*e.g.*, transient execution [18, 21], micro-architectural side channels [15, 66, 77, 78], fault injection [60], and rollback [4, 57, 61, 65]), as these can be mitigated by microcode patches [42], hardened cryptographic libraries [84], and state-continuity mechanisms [4, 57, 61, 65, 80], respectively. Moreover, Fides's minimal TCB (only tiny components in the enclave) further reduces exposure to residual side channels compared to designs that place full protocol logic in enclaves.

Each replica possesses a public/private key pair established through a Public-Key Infrastructure (PKI) and uses it to sign consensus messages (*e.g.*, votes). The private keys are securely sealed and managed within TEEs, while the corresponding public keys are known to all replicas. The adversary is computationally bounded and cannot compromise standard cryptographic primitives such as digital signatures or collision-resistant hash functions. All outputs generated by the enclave are cryptographically signed by the TEE, ensuring that they cannot be forged, modified, or tampered with by any replica, even if the host system is Byzantine.

Asynchronous Network. We assume an asynchronous network model [22], where messages between replicas may experience arbitrary but finite delays, i.e., messages are eventually delivered and never lost, but there are no guarantees on delivery time or order.

3.2 DAG-Based Consensus Goals

DAG-based BFT consensus protocols should allow replicas to agree on a sequence of client transactions in the presence of Byzantine

Table 1: Comparison of DAG-based BFT Protocols (Normalized to Message Delays δ).

	Fault Tolerance	Network Assumption	LV Commit[(1)]	Async LV Commit	NLV Commit[(2)]	Bits per round[(3)]
DAG-Rider [47]	$3f+1$	Async	18δ (6 RBCs)	18δ (6 RBCs)	$+7.5\delta$	$O(\kappa n^4)$
Tusk [27]	$3f+1$	Async	13.5δ (4.5 RBCs)	21δ (7 RBCs)	$+4.5\delta$	$O(\kappa n^3)$
Bullshark [73]	$3f+1$	Async & Partial Sync	6δ (2 RBCs)	18δ (6 RBCs)	$+4.5$–7.5δ	$O(\kappa n^3)$
GradedDAG [26]	$3f+1$	Async	5δ (1 RBC & 1 CBC)	7.5δ (1.5 RBC & 1.5 CBC)	$+5\delta$	$O(\kappa n^3)$
Mahi-Mahi [43]	$3f+1$	Async	5δ (5 broadcasts)	$5\delta/p^{*\ddagger}$	$+0\delta$	$O(\kappa n^3)$
Shoal++ [8]	$3f+1$	Partial Sync	$\mathbf{4}\delta$ (1 RBC & 1 broadcast)	∞	$+2$–3δ	$O(\kappa n^3)$
Sailfish [71]	$3f+1$	Partial Sync	4δ (1 RBC & 1 broadcast)	∞	$+3\delta$	$O(\kappa n^3)$
Mysticeti [11]	$3f+1$	Partial Sync	$\mathbf{3}\delta$ (3 broadcasts)	∞	$+3\delta$	$O(\kappa n^3)$
Fides	$2f+1$	Async	4δ (4 T-RBCs)	8δ (8 T-RBCs)	$+2.5\delta$	$O(\kappa n^2+n^3)$

(1) LV = leader vertex. In leader–anchored commit rules (e.g., Bullshark [73], Shoal++ [8], Sailfish [71]), blocks are decided with respect to the wave leader of that wave (Section 2).
(2) NLV = non-leader vertex. In those leader-based commit rules, non-leader vertices are indirectly committed and typically need a few extra DAG rounds beyond leaders.
(3) We count metadata only (parents/certificates/signatures), not payload. In recent DAG designs, each node emits one vertex to $\Theta(n)$ replicas per round with n certified parents ($\Theta(\kappa)$ for Fides, while $\Theta(\kappa n)$ for others). For DAG-Rider, the dissemination (RBC) complexity is $\Theta(n^2)$, while $\Theta(n)$ for all others. In total $\Theta(n)$ nodes yield $O(\kappa n^3)$ bits per round (constants depend on the signature scheme).
‡ Under an async adversary, Mahi-Mahi commits probabilistically with $p^*=\ell/(3f+1)$ per wave (ℓ = leader slots/round). The expected async commit latency grows with n.

behaviors (see the above threat model). Each replica continuously proposes blocks containing batches of transactions, with each block b assigned a consecutive round number r. A block is also a vertex in a DAG, and we use them interchangeably in this paper. The round number r is used to distinguish between blocks proposed by the same replica. Each block references at least $f+1$ vertices from the previous round number, thus creating a DAG.

Replicas then use common coins to select the leader vertex in a wave and further commit all blocks referenced by the leader vertex according to certain deterministic algorithms. Once a block is committed by a correct replica, it is assigned a unique sequence number sn, and replicas send the corresponding replies to clients. The consensus protocol should satisfy the following properties:

- **Safety:** If two correct replicas commit two blocks b and b' with the same sequence number sn, then $b = b'$.
- **Liveness:** A transaction tx created by a correct client will eventually be committed to a block b by correct replicas.

4 FIDES OVERVIEW

Figure 2 overviews Fides, which tolerates f Byzantine faults with $n=2f+1$ replicas. Figure 3 illustrates an ordering-layer execution. Specifically, Fides operates in two distinct layers:

- **Dissemination layer:** In this layer (Section 6), replicas construct their local views of DAG by a prescribed construction procedure. With the support of TEEs, replicas utilize the *T-RBC* service (Section 5.1) to reliably disseminate vertices. To ensure that replicas have consistent local views of the DAG, the *T-RoundCert* service (Section 5.2) is employed to validate references between vertices.
- **Ordering layer:** In this layer (Section 7), each replica independently interprets its local DAG to deterministically establish an order of vertices. To ensure that all honest replicas derive a consistent order, they follow a four-round commit rule (Section 7.2). With the support of TEEs, we introduce the *T-Coin* mechanism (Section 5.3) to generate shared randomness that determines the vertex order consistently across replicas, which could eliminate the additional coordination required in prior DAG protocols.

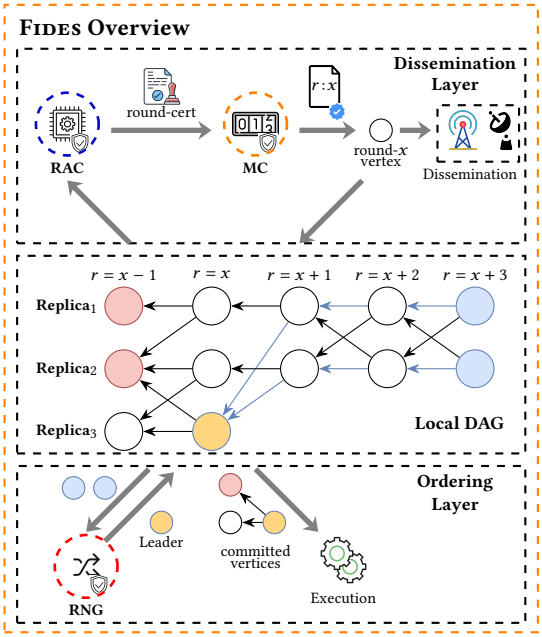

Figure 2: Architecture Overview of Fides.

Consensus Flow. We now present a high-level processing flow.

❶ **Client Transaction Submission.** Clients submit transactions to replicas, which verify and mark the transactions as pending for inclusion into vertices.

❷ **Vertex Construction.** Fides progresses in a round-by-round manner, where each replica proposes one vertex per round. Upon entering a new round and collecting at least $f+1$ vertices from the previous round, a replica constructs a new vertex that includes a batch of pending transactions, a validity proof, and relevant metadata. The validity proof is generated by the *T-RoundCert* service based on $f+1$ references to vertices from the preceding round.

❸ **Vertex Dissemination.** Once the vertex is constructed, it is disseminated to all replicas through the efficient *T-RBC* service.

❹ **DAG Construction.** Upon receiving a vertex through the *T-RBC* service, a replica first verifies the validity of the vertex. If any

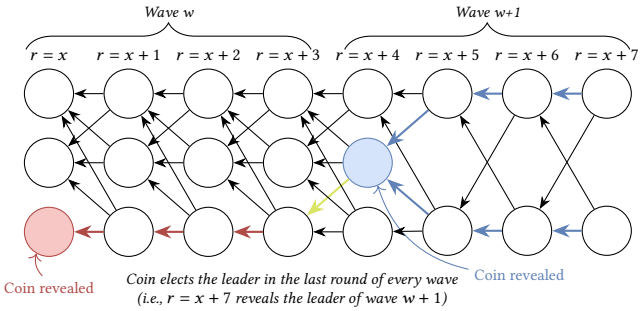

Figure 3: Example of Commit Rule in Fides. If a leader has less than $f+1$ references from its fourth round (red in the figure), it is ignored; otherwise (blue in the figure), the algorithm searches the causal DAG to commit all preceding leaders (dark yellow and red in the figure) and then orders the remaining DAG by the predefined rule afterward.

referenced vertices in the validity proof are missing, the replica waits until they are received and validated. Once all dependencies are resolved, the vertex is appended to the local DAG.

❺ **Wave Leader Election.** Each replica interprets its local DAG to determine the order of vertices in a wave-by-wave manner. In the fourth round of each wave, replicas leverage the shared randomness provided by the *T-Coin* service to collaboratively elect a leader vertex (i.e., a vertex from the first round) for that wave.

❻ **Vertex Ordering and Commitment.** Once a leader vertex is elected and satisfies the commit rule—specifically, when at least $f+1$ vertices from the fourth round of the same wave have a path to it—the leader vertex, together with all vertices in its causal history, is committed in a predetermined order.

Specifically, we observed that reducing the system size to $n=2f+1$ using the *T-RBC* service introduces a potential *liveness* issue such that the leader vertex can never satisfy a traditional two-round commit rule, which drives us to develop the four-round wave and commit rule. This four-round design and the corresponding commit rule preserve both safety and liveness while maintaining low commit latency, especially under adversarial asynchronous network conditions (see Sec. 8 for details).

❼ **Transaction execution.** Replicas execute the transactions contained in committed vertices and return the results to the client.

Steps ❷–❹ form the Dissemination Layer (Section 6). Steps ❺–❻ form the Ordering Layer (Section 7).

5 TRUSTED COMPONENTS WITHIN TEES

5.1 TEE-Assisted Reliable Broadcast

Trusted Component - MC. The *Monotonic Counter (MC)* is a trusted component designed to prevent message equivocation in RBC design. In each round, the *MC* gives its vertex a unique, verifiable identifier implemented by a monotonically increasing counter in TEE, ensuring that each replica can generate at most one valid vertex per round. Its interface is as follows:

- $\langle counter \rangle \leftarrow$ GetCounter*(round-cert, v)*: Given a vertex v and a round certificate *round-cert* which certifies that the replica has received at least $f+1$ valid vertices from the previous round (see Section 5.2), this function returns a fresh counter value, attaches it to v along with a cryptographic proof, and atomically increments the counter by one. The counter value is set as 0 at initialization.

Reliable Broadcast with *MC*. Traditional RBC protocols (*e.g.*, Bracha's RBC [16]) require three rounds of all-to-all communication and $n=3f+1$ replicas to tolerate f Byzantine faults, resulting in quadratic message complexity and significant communication overhead (Section 2.3). In contrast, Fides integrates *MC* to construct a *TEE-assisted RBC (T-RBC)* where non-equivocation can be guaranteed with the cryptographically signed *MC* value in each vertex. Thus, without the need to collect support from most honest replicas, *T-RBC* has linear communication complexity and a lower system size of $n=2f+1$.

Since Fides 's RBC operates on top of a DAG, we further leverage the causality of the DAG to resolve potential issues in which a vertex is received only by part of the honest replicas. In the DAG, vertices reference vertices from earlier rounds. When vertices are missing from the causal history of a newly received vertex v, a replica proactively requests them by sending a catch-up request message to the sender of v, with linear message complexity. The complete workflow of T-RBC is detailed in Appendix A.1.1 [83].

We formally prove T-RBC's correctness in Appendix B.1 [83]. Overall, T-RBC achieves linear communication complexity, one-step communication latency in the good case, and operates with a reduced system size of $n=2f+1$ replicas.

5.2 TEE-Assisted Round Certifier

Trusted Component - RAC. The *Round Advancement Certifier (RAC)* accelerates round certification and vertex validation in the Dissemination Layer. When a replica generates a vertex, it must reference a quorum of at least $f+1$ vertices from the previous round (Section 6). To validate the references, RAC checks whether each is delivered through *T-RBC* by verifying the *MC* value within the vertex. Upon successful validation of $f+1$ references to vertices from the previous round, RAC issues and cryptographically signs a *round-cert* that is encoded as an n-bit bitmask, justifying its eligibility to propose a vertex in the current round.

Upon receiving a vertex, a replica can verify its validity by verifying the attached round certificate, ensuring that the vertex refers to $f+1$ vertices of the previous round.

This allows the replica to replace the $n-f$ hash references of size $\theta(\kappa n)$ with a compact certificate of size $\theta(\kappa+n)$ and reduce the number of cryptographic verifications from $\theta(n)$ to $\theta(1)$, substantially reducing CPU overhead. RAC provides the following interface:

- $\langle round\text{-}cert \rangle \leftarrow$ ValidateVertices*(vertexList)*: Given a list of referenced vertices, this function verifies whether the vertices have been delivered through *T-RBC*, ensures that there are at least $f+1$ vertices, and validates their metadata correctness. If verified, it returns a round certificate $\langle round\text{-}cert \rangle$. The certificate is an n-bit mask indicating which vertices from the previous round are referenced (bit $i = 1$ iff the i-th vertex is included).

5.3 TEE-Assisted Common Coin

Trusted Component - RNG. In the Ordering Layer, the random number generator (RNG) replaces the cryptographic common coin with a lightweight, TEE-assisted alternative for leader vertex selection. Previous DAG protocols implement the common coin via

Data Structures (for replica p_i)

Vertex v:

$v.round$ — the round index of v in the DAG

$v.source$ — the replica that broadcasts v

$v.txns$ — the batch of transactions

$v.strongEdges$ — a bitmask (*round-cert*) indexed by vertices in round $v.round-1$ representing *strong edges*

$v.weakEdges$ — a list of hashes of vertices in rounds $< v.round-1$ representing *weak edges*

$v.counter$ — the signed *MC* value

In-Memory Variables:

DAG_i — the local view of DAG of p_i, a map from round number r to the set of vertices known to replica p_i in round r

$txnsToPropose$ — a queue of client transactions that p_i received and has not proposed

1: **function** PATH(v, u) **do**
2: **return** true if there exists a path of v to u in DAG_i.

3: **function** STRONGPATH(v, u) **do**
4: **return** true if there exists a path of v to u in DAG_i such that all edges on the path are strong edges.

Figure 4: Data Structures and Basic Utilities for Replica p_i.

DAG Construction (running at each replica$_i$) **:**

Local variables: $r := 0$; $buffer := \{\}$

Suppose each replica has a genesis vertex v_0

5: **event** $Deliver_i(v, round, p_k)$ **do** ▷ Receive v from T-RBC
6: **if** VERIFYVERTEX(v) **then** ▷ Verify the vertex
7: **if** $\forall v' \in v.strongEdges \cup v.weakEdges : v' \in \bigcup_{k\geq 1} DAG_i[k]$ **then**
8: $DAG_i[v.round] := DAG_i[v.round] \cup \{v\}$
9: **else**
10: DEFER v until any new vertex committed

11: **event** $|DAG_i[r]| \geq f+1$ **do**
▷ The quorum is adjusted to $f+1$ out of $2f+1$
12: **if** $r > 1 \wedge (r-1)\ mod\ 4 = 0$ **then**
13: WAVEREADY $(\frac{r-1}{4})$ ▷ Each wave contains 4 rounds
14: *round-cert* := *RAC*.VALIDATEVERTICES($DAG_i[r]$)
15: $r := r + 1$; $v :=$ CREATENEWVERTEX(r, *round-cert*)
16: TBCAST$_i(v, r)$

17: **function** CREATENEWVERTEX(r, *round-cert*) **do**
18: $v.txns := txnsToPropose$
19: $v.strongEdge := DAG_i[r-1]$
20: SETWEAKEDGES(v,r)
21: **return** v

22: **function** SETWEAKEDGES(v,r) **do**
23: Add every prior vertex not reachable in DAG as weak edges

Figure 5: Pseudocode for DAG Construction.

threshold cryptography, which is costly [82]. *RNG* instead uses a shared seed inside each TEE to produce identical, verifiable leader vertices across correct replicas. At startup, a DKG protocol [69] seeds *RNG* consistently across all TEEs.

The leader vertices are generated wave by wave, where each wave consists of four rounds, and the leader vertex is in the first round. To request the random leader vertex value of a wave, a valid round certificate ⟨*round-cert*⟩ for the fourth round, issued by the *RAC*, is required. This design ensures that the random leader vertex becomes available only after a quorum of at least $f+1$ replicas have proposed vertices in the fourth round.

The four-round commit rule determines commitment by checking whether a sufficient number of fourth-round vertices have paths to the leader vertex (Section 7.2). If Byzantine nodes were able to know the leader vertex in advance, the adversary could undermine liveness by deliberately excluding the leader vertex from their causal histories and manipulating message delivery times to reduce the likelihood of its commitment. *RNG* thus preserves the security and unpredictability of the traditional common coin while reducing computation to a single trusted call. Its correctness is formally proved in Appendix B.2 [83]. The *RNG* has the following interfaces:

- $\langle \rho_r \rangle \leftarrow$ RAND(*round-cert*): Given a valid *round-cert* for round $r+3$, returns the corresponding leader vertex ρ_r of round r if the *round-cert* is verified.

6 DISSEMINATION LAYER

We first introduce the basic structure of the DAG, and then describe its construction, which forms the dissemination layer of FIDES.

6.1 DAG Vertex Structure

Figure 4 shows the pseudocode for the DAG data structure and its basic utilities. Each vertex in the DAG contains: (i) associated metadata (*e.g.*, round number, source), (ii) a batch of transactions, (iii) references to vertices in previous rounds[1]. The references are divided into two types:

- **Strong Edges:** a bitmask (*round-cert*) indexed by vertices showing references to at least $f+1$ vertices from the immediately preceding round $v.round-1$.
- **Weak Edges:** a list of hashes showing references to vertices from rounds lower than $v.round-1$.

6.2 DAG Construction

FIDES proceeds in a round-by-round manner. Once a replica p_i receives $f+1$ valid vertices from round r, it advances to round $r+1$, forming a new vertex v referencing these vertices using strong edges and including a *round-cert* of round r, and then broadcasts v, incrementally constructing the DAG.

Figure 5 illustrates the DAG construction in FIDES, including vertex creation, verification, and integration into the DAG. Replicas begin with the genesis vertices and progress round by round. A replica advances to the next round once it has received at least $f+1$ vertices in the current round (line 11).

[1]We ignore the *MC* value here as it is calculated during *T-RBC*.

Fides (running at each replica p_i) :

Local variables:
$decidedWave := 0$; $deliveredVertices, leadersStack := \{\}$

24: **event** On receiving a block b from clients **do**
25: $txnsToPropose$.Enqueue(b)

26: **function** WaveReady(w) **do**
27: v := GetWaveVertexLeader(w)
28: **if** $v = \perp \vee |\{v' \in DAG_i[round(w,4)] : \text{StrongPath}(v', v)\}| < f+1$ **then**
29: **return**
30: $leadersStack$.Push(v)
31: **for** wave w' from w - 1 down to $decidedWave$ + 1 **do**
32: v' := GetWaveVertexLeader(w')
33: **if** $v' \neq \perp \wedge$ StrongPath(v, v') **then**
34: $v := v'$; $leadersStack$.Push(v')
35: $decidedWave := w$
36: OrderVertices($leadersStack$)

37: **function** GetWaveVertexLeader(w) **do**
38: $j := RNG$.Rand$_i$(v.round-cert)
39: **if** $\exists v \in DAG_i[round(w,1)]$ $s.t.$ $v.source = p_j$ **then**
40: **return** v
41: **return** $\perp$

42: **function** OrderVertices($leadersStack$) **do**
43: **while** $\neg leadersStack$.IsEmpty() **do**
44: $v := leadersStack$.Pop()
45: $ToDeliver := \{v' \in \bigcup_{r>0} DAG_i[r] \mid \text{Path}(v, v') \wedge v' \notin deliveredVertices\}$
46: **for every** $v' \in ToDeliver$ in predefined order **do**
47: **output** CommitBlock$_i$(v')
48: $deliveredVertices := deliveredVertices \bigcup \{v'\}$

Figure 6: Pseudocode of Fides's Byzantine Consensus.

To ensure correctness, Fides employs the *T-RoundCert* mechanism. Before moving to the next round, a replica p_i invokes *RAC* to validate quorum criteria (line 14), specifically that $f+1$ vertices from distinct replicas have been delivered. Upon successful validation, *RAC* issues a round certificate (*round-cert*), authorizing p_i to advance. Then, p_i generates a new vertex (line 15) and disseminates it to all replicas via *T-RBC* (line 16). The TEE-assisted services ensure that all vertices meet the vertex structure requirements and are delivered reliably, preserving the causal structure of the DAG.

Each replica maintains its own local view of the DAG. Upon receiving a vertex, the replica fetches all missing referenced vertices before adding the new vertex (line 7). Although temporary divergences may occur due to network asynchrony, *T-RBC* guarantees that all non-faulty replicas eventually receive the same set of vertices (Lemma 2 in Appendix C [83]). The differences in delivery order do not affect correctness because the DAG's topological order is determined by vertex references, as shown in the example below.

7 ORDERING LAYER

Fides requires no additional communication for transaction commitment, consistent with existing DAG-based protocols [27, 47, 73]. Vertices are organized into *waves*, each consisting of k *rounds* of vertex generation. Within each wave, the rounds collectively form a *common core* problem—determining how many vertices from the first round can be safely committed. To ensure liveness, at least one vertex must satisfy the commitment condition. Moreover, the number of such vertices determines the probability that a randomly chosen leader vertex can be successfully committed, which directly impacts the expected latency. The optimal number of rounds per wave, k, is derived analytically in Section 7.1.

At the final round of each wave, Fides invokes a lightweight *T-Coin* to randomly select a leader vertex among those generated in the first round. Each replica independently verifies whether the leader vertex belongs to the *common core*. If the condition holds, all replicas deterministically order the vertices within the wave and commit the corresponding transactions. The complete commit rule, including how replicas apply the coin outcome to resolve ties and maintain consistent views, is detailed in Section 7.2.

7.1 Dissecting Liveness with $f+1$ Quorums

We now derive the optimal number of rounds k that minimizes the expected commit latency. First, as said above, the rounds in a wave are abstracted as a *k-iteration common core problem*, which generalizes the original common core formulation used in prior DAG consensus protocols (e.g., Tusk [27], where $k = 2$). Notably, the original common core was designed for standard Byzantine settings, where the threshold is $2f+1$ among $3f+1$ replicas. In contrast, Fides leverages TEE-assisted trust to reduce the threshold to $f+1$ among $2f+1$ replicas. This change invalidates a direct reuse of existing results from the original common core formulation.

7.1.1 k-Iteration Common Core Abstraction. We first use x_i to denote the input of replica p_i, *i.e.*, the vertex it generates in the first round of a wave in Fides. Let N denote the total number of participants in the common core instance, equal to the number of replicas in Fides. Throughout, we fix $N{=}2f{+}1$ as in Section 3. The protocol proceeds in k conceptual rounds, where each round builds upon the aggregation results of the previous one.

1. **Initial Broadcast ($r = 1$).** Each party broadcasts its input x_i using the Reliable Broadcast protocol (RBC).
2. **Recursive Aggregation (for $r = 2, 3, \ldots, k$).** Each party i maintains a set A_i^r representing its aggregation result at round r.
   (1) For $r = 2$: define $A_i^2 = \{(j, x_j)\}$. Once $|A_i^2| \geq f+1$, send A_i^2 to all parties.
   (2) For $r > 2$: upon receiving A_j^{r-1} from party j, accept it after receiving all broadcasts x_k for every $(k, x_k) \in A_j^{r-1}$. After accepting $f+1$ such sets, compute and broadcast $A_i^r = \bigcup_j A_j^{r-1}$.
3. **Output.** After completing k aggregations, output $U_i = A_i^k$.

The k-iteration common core protocol satisfies three properties:

- **Common Core.** There exists a core set S^* ($|S^*| > 0$) such that every non-faulty party includes S^* in its output set.
- **Validity.** If a non-faulty party includes a pair (j, x_j) in its output and party j is non-faulty, then x_j must be j's original input.

- **Agreement.** All parties that include a pair for some sender j must agree on its value. Formally, if two non-faulty parties include (j, x) and (j, x') in their outputs, then $x = x'$.

The agreement and validity properties can be satisfied by the RBC protocol (Appendix A [83]). The number of rounds k determines the size of S^*, and the maximum size of S^* is bounded by $f+1$, since the f adversaries can maliciously avoid generating vertices in certain rounds.

Bounds of $|S^*|$. To ensure system liveness, we must guarantee that $|S^*| \geq 1$. Since only the leader vertex is selected from S^*, a wave can be committed only if this condition holds. If $|S^*| = 0$ under some adversarial scenario, the expected latency diverges to infinity, indicating that no wave can ever be committed.

Expected Latency. The size of S^* determines the probability of successful commitment, further influencing the expected commit latency. We now derive the expected commit latency of Fides.

Lemma 1 (Expected Latency)**.** *The expected latency of committing a wave is $\frac{k \times N}{|S^*|}$, where $\frac{|S^*|}{N}$ denotes the expected probability of committing a single wave.*

Proof. In the k-iteration common core abstraction, each round involves communication and aggregation among replicas. The randomly elected leader (Section 5.3) in a wave can be committed only if it belongs to S^*. Hence, the probability of successfully committing a wave is $p = \frac{|S^*|}{N}$. Given that each wave proceeds sequentially through k rounds, the total expected latency (measured in the number of communication rounds) is:

$$\text{Expected Latency} = k \times \frac{1}{p} = \frac{k \times N}{|S^*|}.$$ □

7.1.2 Theoretical Analysis. We now determine the number of rounds k in Fides that guarantees the common-core property (*i.e.*, liveness) and yields low commit latency. Before that, we first introduce some notations for analysis. We use $R_1, \ldots, R_k$ to denote the vertices in the rounds of a wave. Besides, for a vertex $x \in R_1$, we use $supp_k(x)$ to denote the number of round-k vertices that are (causally) carried from x. As shown in Appendix D.2.2 [83], any lower bound on the number of first-round vertices satisfying $supp_k(x) \geq f+1$ in the DAG directly implies the same lower bound on $|S^*(k)|$, so we can equivalently reason in the DAG domain.

We next analyze feasibility by incrementally trying k=2—the wave shape adopted by early DAG protocols—to larger k.

Liveness Violation for k=2. The appendix [83] demonstrates that an asynchronous adversary can schedule a two-round wave so that no first-round vertex reaches $\text{supp}_2(x) \geq f+1$, i.e., $|S^*(2)|$=0. Hence, the per-wave success probability p_2=0, and liveness cannot be guaranteed in adversarial networks.

Liveness Exists, but Not Enough for k=3. The appendix [83] shows that *exactly two* first-round vertices reach the threshold in the worst case, so $|S^*(3)|$=2. We have

$$p = \frac{2}{2f+1} \quad \Rightarrow \quad \mathbb{E}[\text{rounds}] = 3 \cdot \frac{2f+1}{2} = 3f + \tfrac{3}{2}.$$

Liveness holds, but the expected commit time grows linearly with f, so there is *no constant-round guarantee*. We thus have the theorem:

Theorem 1 (Minimum Rounds (k)). *The k-iteration common core property holds when $k \geq 3$.*

Sketch. By the appendix proof [83] and the DAG⇒Common-Core bridge, an adversary can realize $S^*(2) = \emptyset$; thus k=2 fails liveness. For k=3, exactly two first-round vertices achieve $\text{supp}_3(\cdot) \geq f+1$, hence $|S^*(3)|$=2 > 0 via the bridge. Therefore, a non-empty common core exists if and only if $k \geq 3$. □

Optimal Constant-Time Threshold for k=4. With four rounds, our worst-case construction shows that $|S^*(4)| \geq f+1$ always holds. Combined with the upper bound in the appendix [83], this yields tightness: $|S^*(4)| = f+1$ in the worst case. Then

$$p = \frac{f+1}{2f+1} \quad \Rightarrow \quad \mathbb{E}[\text{rounds}] = 4 \cdot \frac{2f+1}{f+1} \leq 8,$$

establishing a *constant* worst-case expected commit latency.

No Benefit in the Worst Case for k>4. An adversary can always suppress f proposers in round 1, forcing $|R_1| = f+1$. Lemma 15 in Appendix D.2 [83] caps the worst-case common core at $|S^*(k)| \leq f+1$, independent of k. Hence, increasing the wave length beyond four cannot enlarge the worst-case core and only *increases* the expected commit latency (Lemma 16 in Appendix D.2 [83]), which yields the following theorem:

Theorem 2 (Optimal Number of Rounds). *In Fides, the optimal number of rounds per wave is $k = 4$.*

Sketch. With a global common coin, $p = \frac{f+1}{2f+1}$, so $\mathbb{E}[\text{rounds}] = 4 \cdot \frac{2f+1}{f+1} \leq 8$ (Lemma 14 in Appendix D.2 [83]), *i.e.*, constant-round commitment. Further for any $k > 4$, the same upper bound $|S^*(k)| \leq f+1$ holds while k increases, thus the worst-case expected latency strictly worsens (Lemma 16 in Appendix D.2 [83]). Therefore, $k = 4$ is optimal among constant-k designs. □

Conclusion: We summarize the choice of round number $\boldsymbol{k}$ as:

- $k = 2$: $|S^*| = 0$ — no liveness guarantee.
- $k = 3$: $|S^*| = 2$ — ensures liveness, but commit may require $\Theta(f)$ rounds.
- $k = 4$: $|S^*| = f+1$ — ensures liveness and constant-round commit.
- $k > 4$: $|S^*| = f+1$ — additional rounds only increase latency without improving commit.

Therefore, we choose $k = 4$ as the optimal rounds in a wave, achieving the best worst-case commit latency.

7.2 Commit Rule under $k = 4$ Regime

Having fixed $k = 4$ under this quorum, we now instantiate the ordering layer. Fides operates in four-round waves, and commitment is derived wave by wave directly from the DAG with no extra consensus messages—consistent with prior DAG-BFT designs. The commit rule process in Fides, as outlined in Figure 6, begins with a DKG protocol [69] at startup to set up the initial state of the TEE. A key improvement of Fides is the utilization of the lightweight *T-Coin* to generate globally consistent random values for leader election. This coin introduces negligible computational and communication overhead, ensuring efficient leader election.

Specifically by Theorem 2, we adopt a wave with four consecutive rounds to preserve liveness and optimal commit latency (as

proven in Appendix D.2 [83]). During the first round, replicas propose vertices that encapsulate their entire causal history. The second and third rounds involve replicas voting on these proposals by referencing them in subsequent vertices to enhance the connectivity of the DAG. The fourth round uses the *T-Coin* to retrospectively elect a leader vertex from the first round of the wave (line 38). A leader vertex is committed if it is referenced by at least $f+1$ vertices in the fourth round (line 28). Once a leader vertex is committed, all its preceding causal history is transitively committed in a predefined order (line 46, *e.g.*, depth-first search).

Since replicas may have divergent DAG views, not all commit a leader each wave. Fides employs a recursive mechanism (line 31):

1. Once a leader is committed in wave w, it becomes the candidate for transaction ordering.
2. The system recursively checks the preceding waves to identify the most recent wave w' with a committed leader.
3. For waves between w' and w, paths between the current candidate leader and the leaders of earlier waves are transitively committed in ascending order $(w'+1, w'+2, \ldots, w)$.

Figure 3 illustrates a recursive committing process in Fides. Consequently, an elected wave leader vertex l of wave w is committed by a non-faulty replica p through either of the following mechanisms:

1. **Direct Commit:** The leader l is directly committed when, in the replica's local view, it is referenced by at least $f+1$ vertices from the fourth round of wave w.
2. **Indirect Commit:** The leader l is indirectly committed if there exists a path from l_s to l, where l_s is the directly committed leader vertex of a wave $w_s > w$.

Fides achieves better fault tolerance (*i.e.*, tolerating f faults in an $n{=}2f{+}1$ setting) and enhanced scalability, with safety and liveness formally proven in Appendix D [83]. The appendix [83] establishes that Fides can expect to commit a leader vertex approximately every **four** rounds under random message delays, and every **eight** rounds under an asynchronous adversary.

8 CORRECTNESS ANALYSIS

We here provide a sketch of Fides's correctness, while deferring full proofs to Appendix D [83]. Before that, we first prove the correctness of two building blocks, *i.e.*, *T-RBC* and *T-Coin*.

Correctness Analysis of Building Blocks. Fides utilizes two standard primitives: *Reliable Broadcast* and *Global Common Coin*. We further instantiate them as TEE-assisted Reliable Broadcast *(T-RBC)* and TEE-assisted Common Coin *(T-Coin)*.

1. For *T-RBC*, we prove the RBC guarantees, *i.e.*, Validity, Integrity, and Agreement, even under Byzantine faults [16].
2. For *T-Coin*, we also prove the Global Common Coin properties, *i.e.*, Agreement, Termination, and Unpredictability [47].

Full proofs appear in Appendices B.1 and B.2 [83].

Safety Sketch Analysis of Fides. Safety follows from two consistency ingredients (Appendix D.1 [83]). The proof proceeds as follows:

1. *Unique leader per wave.* Any two correct replicas select the same leader for the same wave. This holds because, after converging to an identical DAG view (*Reliable Dissemination*), correct replicas feed identical inputs and seeds into RNG, thereby deterministically electing the same leader (Lemma 3 in Appendix D.1 [83]).
2. *Canonical commit order.* All correct replicas commit leader vertices in the same ascending order of waves. This is guaranteed because the algorithm commits leaders via a LIFO stack and pops them in increasing wave order (Lemma 6 in Appendix D.1 [83]).

Together, Fides guarantees safety: replicas always agree on the elected leaders and commit them in a consistent order. Hence, the causal history of each leader is committed in the same order across all replicas, ensuring that the complete sequence of committed transactions is identical (Theorem 3 in Appendix D.1 [83]).

Liveness Analysis of Fides. Liveness in Fides follows from two key properties (with full arguments in Appendix D.2 [83]), bolstered by our earlier analysis in Sec. 7.1. The outline is as follows:

1. *Non-zero per-wave commit probability via $k = 4$.* Any wave always contains at least $f+1$ first-round vertices that meet the support threshold (Sec. 7.1). Thus, each wave has a strictly positive probability of committing in both adversarial and random networks, which ensures progress in expectation.
2. *Eventual inclusion of all vertices.* When a wave leader is committed, the protocol delivers its entire causal history in a deterministic order. By utilizing weak edges and repeated commitment across waves, every vertex will be included in some committed leader's causal history. Hence, every correct replica eventually commits every vertex (Lemma 18 in Appendix D.2.3 [83]).

Together, these properties establish liveness: every wave progresses with a non-zero constant probability, and all vertices are eventually incorporated into the causal histories of committed leaders to ensure their final commitment (Theorem 4 in Appendix D.2.3 [83]).

9 EVALUATION

We compare the performance of Fides with several state-of-the-art BFT consensus protocols: Tusk [27] (an async DAG protocol), Bullshark [73] (an async DAG protocol with a partial sync fast path), Mysticeti [11] (a partial sync DAG BFT protocol), Shoal++ [8] (a partial sync DAG protocol with pipelined anchors), RCC [34] (a multi-leader BFT protocol), Damysus [28] (a TEE-assisted BFT protocol), Achilles [62] (a TEE-assisted rollback-resilient protocol) and HybridSet [86] (a TEE-assisted leaderless BFT protocol). We measure throughput and latency under varying conditions (*e.g.*, faulty nodes, network latency, batching). Our experiments aim to answer the following questions:

- **Q1:** How does Fides scale with larger f, batch sizes, and workload compared to prior DAG protocols? (Section 9.2)
- **Q2:** How well does Fides sustain performance under injected delays and leader failures? (Section 9.3)
- **Q3:** How does Fides behave under adversarial conditions, including crash faults, Byzantine attacks, and asynchrony? (Section 9.3)
- **Q4:** What is the breakdown of Fides's end-to-end latency, and how do individual and combined TEE components contribute to the overall performance? (Section 9.4)

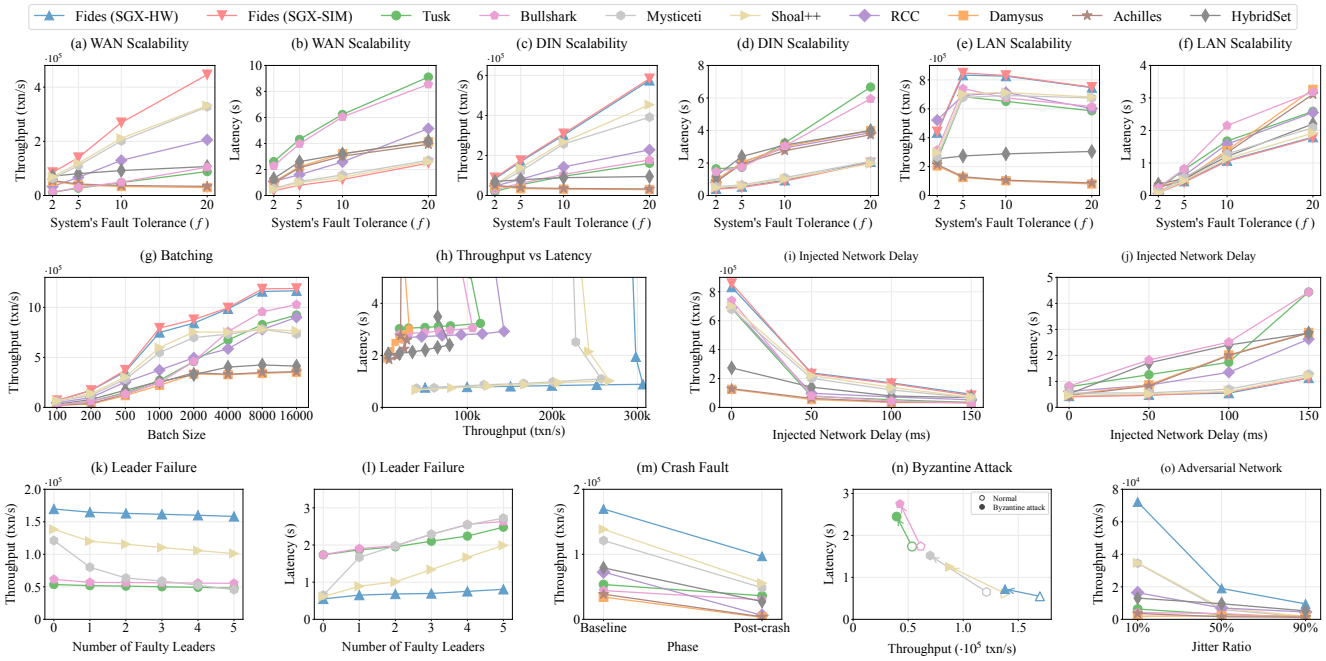

Figure 7: Results for Scalability and Resiliency Evaluation.

9.1 Implementation and Evaluation Setup

Implementation. We built Fides atop the open-source Apache ResilientDB platform [6, 36, 67], a scalable and global blockchain infrastructure. Our implementation is developed in C++ [5] and leverages Apache ResilientDB for durable data storage. We use TCP to implement reliable point-to-point communication. The core trusted components (*i.e.*, *MC*, *RAC*, and *RNG*) run within Intel SGX enclaves [24] in each replica, integrated via the Open Enclave SDK [63]. All protocols share ResilientDB's networking, storage, and cryptographic stack, differing only in the consensus layer. Reported measurements include all TEE-related overheads if applicable.

Evaluation Setup. We ran experiments on Alibaba Cloud Elastic Compute Service [2]. SGX-enabled runs used `ecs.g7t.xlarge` in Hong Kong; others used `ecs.g7.xlarge`. Each VM provided 4 vCPUs and 16 GB of memory and ran Ubuntu 22.04. In the LAN environment, all machines were deployed within Hong Kong. For WAN experiments, due to the limited availability of SGX-enabled instances, we adopt two complementary settings:

- **Wide Area (Geo-Distributed) Network (WAN):** We deploy replicas across four data centers located in the United States, UAE, Singapore, and Germany, with inter-datacenter latencies ranging from 80 ms to 270 ms. Each replica runs trusted components in *simulated SGX* mode (*SGX-SIM*) [40], as SGX-enabled instances are unavailable in some regions; we justify this in Appendix E.4 [83].
- **Delay-Injected Network (DIN):** We enable *SGX Hardware encryption* (*SGX-HW*) but emulate WAN conditions within a region by injecting specific end-to-end delays between machines. A default delay of 100 ms is used to represent a typical WAN RTT.

The WAN deployment enables us to evaluate the impact of real network delays in wide-area settings. Meanwhile, the DIN environment allows for the measurement of the actual performance of SGX under WAN-like latency conditions.

We performed experiments in three network environments: LAN, WAN, and DIN. In each setup, each node exclusively ran one instance of Fides, with the number of clients matching the number of consensus processes to ensure that replicas were fully saturated. Clients generated a saturating request load for replicas. The benchmark was a key-value service, where each request involved a randomly generated key-value insertion with a fixed size of 50 B. We performed three runs per condition and plotted the median value.

9.2 Scalability Evaluation

9.2.1 Varying Fault Tolerance. We evaluate performance across varying fault-tolerance levels.

Performance under WAN. In WAN environments, where network latencies are higher, scalability is a critical challenge for consensus protocols. Figure 7(a) and (b) depict throughput and latency as the system's fault tolerance f increases from 2 to 20 in the WAN setting. Across the entire range, Fides demonstrates superior scalability: Its throughput remains highest across all f, exceeding Tusk by over 4× and doubling RCC at higher f. In contrast, other protocols either saturate early or suffer sharply rising latency as faults increase. Shoal++ and Mysticeti are the closest DAG competitors, yet at f=20 Fides still outperforms both by ~35%, owing to its reduced quorum size (f+1 out of 2f+1 vs. 2f+1 out of 3f+1) and lower message complexity. Damysus and Achilles exhibit significantly lower throughput because their leader-centric communication patterns suffer especially in cross-region scenarios. HybridSet performs slightly better than the other two but still lags behind Fides, as it relies on a hybrid binary agreement to decide on each proposal.

Performance under DIN. To evaluate the overhead introduced by TEE, particularly in *SGX-HW* mode, we conducted tests in DIN environments. Figure 7(c) and (d) illustrate the scalability results in this scenario. In DIN environments, Fides-HW and Fides-SIM perform

almost identically. The negligible performance gap between the HW and SIM modes demonstrates that the real hardware-based encryption in SGX-HW mode introduces minimal computational overhead. Both of them significantly outperform all other protocols—consistent with results shown under WAN.

Performance under LAN. Figure 7(e) and (f) illustrate the results in LAN, where network latencies are negligible. Even at f=20, Fides maintains ~746k txn/s throughput with ~1.80 s latency, while competing protocols fall to 80k–681k txn/s and 1.9–3.2 s latency. Although Fides's RBC optimization yields less dramatic gains in LAN, it still scales much more gracefully than alternatives.

9.2.2 Varying Batch Size. Figure 7(g) illustrates the impact of batch size on performance with a fault tolerance of f=5. All protocols improve with larger batches before saturating, but Fides consistently leads: at batch size = 4,000, Fides achieves ~985k txn/s, while others lie in the range 328k–758k txn/s. As batch size grows further, throughput growth slows, yet Fides retains its lead consistently.

9.2.3 Varying Workload. Figure 7(h) varies workload at f=10. All protocols improve until saturation, with Fides consistently leading. For instance, near saturation, Fides reaches ~300k txn/s at ~0.9 s latency, while the next-best competitors (Shoal++ and Mysticeti) achieve ~267k and ~258k txn/s but with latency above 1.0 s, and others drop below 150k txn/s.

9.3 Resiliency Evaluation

We evaluate Fides's resiliency by examining the impact of network latencies, leader failures, and adversarial network conditions.

9.3.1 Injecting Network Delay. We evaluate the performance of all protocols under network delays injected from 0 to 150 ms. Figure 7(i,j) shows Fides (HW and SIM) maintains higher throughput and lower latency than all baselines across every delay setting.

9.3.2 Varying Leader Failures. We evaluated five DAG-based protocols (Fides, Tusk, Bullshark, Mysticeti, Shoal++) under f=5 in the DIN environment. We focus on these five protocols because they share a comparable DAG architecture, enabling a consistent and fair failure definition across systems. Here, a failed leader is defined as a replica whose first-round vertex never gathers enough support (*i.e.*, never reaches the commit quorum), and is never committed.

Figure 7(k) shows that Fides delivers the best performance, with gradual declines for all as the number of failures increases from 0 to 5, whereas Tusk, Bullshark, Shoal++, and especially Mysticeti drop off more sharply. In terms of latency (Figure 7(l)), although Mysticeti performs well in the good case, it degrades quickly once failures occur. Meanwhile, Fides maintains the lowest latency and exhibits more moderate growth compared to others, showcasing its robustness and resiliency in handling node failures.

9.3.3 Dynamic Crash Faults. Since Fides's TEE-assisted design rules out equivocation and forgery, we focus on omission, crash faults, and network asynchrony. In this setting, we crash f replicas after 10 s. As shown in Figure 7(m), all protocols experience performance degradation after the crashes, but Fides remains the best, sustaining the highest post-crash throughput of ~97k txn/s. Fides benefits from its lightweight design: TEE-assisted dissemination and a four-round commit path reduce coordination and communication overhead under failures.

9.3.4 Byzantine Delivery Attack. We further add a DAG-specific Byzantine attack: up to f Byzantine replicas send each vertex to only f+1 replicas and try to suppress leader-supporting references. This creates worst-case local DAG views without equivocation or forgery. Figure 7(n) shows Fides remains the best protocol, sustaining 138.0k txn/s with 0.72 s latency under the attack. Fides's advantage comes from its TEE-assisted dissemination and four-round commit rule, which improve DAG connectivity and reduce message overhead even when Byzantine replicas suppress references.

9.3.5 Adversarial Delay. We fix the mean network delay at 500 ms and increase the jitter ratio from 10% to 90% to emulate adversarially induced network instability. Figure 7(o) shows that timeout-sensitive baselines such as Mysticeti, Shoal++, and Bullshark degrade sharply, falling to around 2k txn/s at 90% jitter, whereas Fides's fully asynchronous design maintains the highest throughput (~10k txn/s).

9.4 System Breakdown

9.4.1 Latency Breakdown. We provide a breakdown of the five DAG-based protocols in a DIN environment with f=5. For fair comparison, transaction queuing latency is excluded to mitigate the impact of clients. The latency is divided into four stages:

- **Vertex Dissemination.** Fides disseminates a vertex in 0.07 s, outperforming Tusk (0.23 s) and Bullshark (0.23 s) while remaining comparable to Mysticeti (0.1 s), enabled by its optimized T-RBC.
- **DAG Construction.** From confirmation to local DAG insertion, Fides takes 0.05 s. Streamlined vertex validation reduces this stage by 75% versus Tusk and Bullshark.
- **Commitment.** Fides achieves a commit time of 0.36 s, approximately 50% faster than other protocols.
- **Execution.** All protocols have similar execution times of about 0.03 s, as no optimizations on execution are applied.

Figure 8(a) shows the average time spent in each stage. The breakdown results indicate that Fides achieves a minimum end-to-end latency of 0.5 s, driven primarily by its accelerated T-RBC-assisted dissemination, DAG construction, and commitment. Notably, Fides achieves comparable latency to Mysticeti and Shoal++ despite using a certified DAG. These approaches converge to similar end-to-end latency through different mechanisms: Mysticeti and Shoal++ remove per-vertex certification (but require partial synchrony for liveness), while Fides makes certification cheap by replacing multi-round RBC with T-RBC's single broadcast and using RAC to reduce certificate validation overhead.

9.4.2 Trusted Component Breakdown. To analyze the performance improvement brought by each trusted component, we ran four configurations: three cases with exactly one component enabled *(MC, RNG, RAC)*, and an *All* configuration with all components active under the f=10 setting in DIN. We measured throughput and latency for each configuration and then calculated the relative improvement compared to the baseline (*i.e.*, Fides but with no trusted component active). In all configurations, Fides preserves safety and

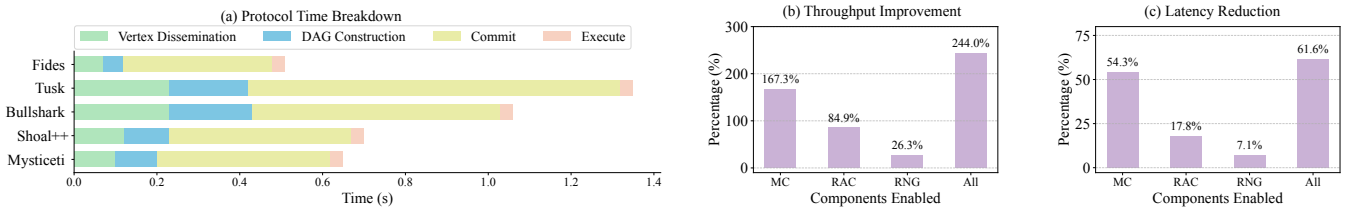

Figure 8: System Breakdown by Protocol Time and Trusted Components.

liveness. Disabling a trusted component only affects performance because the protocol falls back to corresponding untrusted compensations (e.g., extra communication or heavier cryptographic operations to tolerate Byzantine behavior).

Figure 8(b) and (c) summarize these changes. Enabling the *MC* component delivers the largest improvement, increasing throughput by 167.3% while reducing latency by 54.3%. *RAC* and *RNG* deliver additional throughput improvements of 84.9% and 26.3%, and latency reductions of 17.8% and 7.1%, respectively. When all three components are enabled simultaneously, their effects combine synergistically: throughput increases by 244.0% and latency decreases by 61.6%. This *All* configuration demonstrates the substantial performance gains achievable through our trusted-component architecture, while still preserving transaction confidentiality.

10 DISCUSSION

Hybrid fault model and TEE availability. Fides can support c honest replicas with crashed TEEs using $n{=}2f{+}2c{+}1$ replicas and quorum $f{+}c{+}1$. Under systemic TEE compromise, it falls back to $n{=}3f{+}1$ and quorum $2f{+}1$, as in FlexiTrust [37]. Beyond fault counts, *MC* and *RAC* require only TEE *integrity* [13, 48, 79], while RNG-seed leakage breaches *confidentiality*, weakening liveness but not safety.

Deployment considerations. While TEEs introduce operational complexity and vendor dependence, these costs are manageable in consortium systems, where participants can agree on hardware requirements and enforce remote attestation, mandatory microcode updates, and timely patching. TEE availability is also increasingly practical, with major cloud providers offering TEE-enabled instances [2, 3, 58].

11 RELATED WORK

We survey prior work on concurrent protocols, DAG-based BFT consensus and TEE-assisted BFT protocols. The extended version provides a more comprehensive discussion of related work [83].

Concurrent / multi-leader BFT protocols. Leader-driven BFT remains bottlenecked by a single proposer and vote collector as the replica set grows, despite recent advances [35, 45, 54]. Concurrent designs instead use parallel leaders, as in Mir-BFT [74], or multiple instances [9, 34, 46, 52, 75], which typically require global output reconciliation. Orthrus [51] and Hydra [53] reduce confirmation latency through transaction-level concurrency, partial ordering, or avoiding global ordering. DAG-based BFT lets many replicas disseminate and reference batches concurrently; Fides adopts this approach to alleviate the single-leader bottleneck.

DAG-based BFT protocols. DAG-based BFT protocols [8, 11, 12, 25–27, 33, 43, 47, 55, 56, 71–73] organize transactions in a DAG to improve scalability and throughput. HashGraph [12] is an early asynchronous design that separates communication from ordering, while DAG-Rider [47] combines reliable broadcast with a round-by-round DAG to optimize resilience and amortized communication. Narwhal and Tusk [27] decouple data availability from ordering, and Bullshark [73] adapts this architecture to partial synchrony. Partially synchronous designs prioritize low latency in well-behaved networks but do not fully handle asynchrony [8, 11, 55, 71, 72], while recent variants explore pipelined anchors, graded reliable broadcast, and uncertified DAGs with asynchronous coins [8, 26, 43, 71]. Fides combines asynchronous DAG consensus with TEEs to improve scalability under adversarial or unpredictable networks.

Hardware-assisted BFT protocols. Hardware-assisted BFT protocols [13, 28, 29, 37, 48, 81, 86] leverage trusted hardware to improve performance and security. TrInc [48] uses a non-decreasing counter to prevent equivocation, reducing the replication threshold from $n{=}3f{+}1$ to $n{=}2f{+}1$ and consensus from three phases to two. Hybster [13] uses a trusted crash-only subsystem and reports over 1 million operations per second in an SGX prototype, while Damysus [28] uses *Checker* and *Accumulator* components to reduce communication phases.

FlexiTrust [37] adapts the replication threshold between $2f{+}1$ and $3f{+}1$ when TEE trust is uncertain. OneShot [29] uses TEEs for view-adapting consensus, Achilles [62] combines rollback-resilient recovery with chained commits for linear message complexity and low latency, and Trusted Hardware-Assisted Leaderless BFT [86] (HybridSet in the experiments) removes the leader and its associated vulnerabilities. In contrast, Fides uses TEE guarantees to streamline reliable dissemination, DAG-reference certification, and common-coin generation while keeping consensus-level DAG state outside the enclave.

12 CONCLUSION

We presented Fides, among the first asynchronous DAG-based BFT consensus protocols that integrate TEEs to tolerate a minority of Byzantine replicas. Fides adopts a customized TEE-assisted *T-RBC* and reconstructs the DAG by relaxing the vertex references from $2f{+}1$ to $f{+}1$. More importantly, this reduced quorum introduces a liveness pitfall for standard two-round DAG commit rules under adversarial asynchrony; Fides resolves it with a four-round commit rule that restores liveness and achieves optimal constant expected commit latency. Together, the three trusted components in Fides minimize network communication complexity and reduce reliance on cryptography. Comprehensive evaluations demonstrate that Fides consistently outperforms state-of-the-art protocols in both throughput and latency, validating its practicality for real systems.

ACKNOWLEDGMENTS

This work is partially funded by NSF Award Number 2245373.

REFERENCES

[1] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically Optimal Validated Asynchronous Byzantine Agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, 337–346. https://doi.org/10.1145/3293611.3331612

[2] Alibaba Cloud. 2026. Elastic Compute Service: Security Capabilities Overview. Retrieved July 21, 2026 from https://www.alibabacloud.com/help/en/ecs/user-guide/overview-of-security-capability

[3] Amazon Web Services, Inc. 2026. AWS Nitro Enclaves. Retrieved July 21, 2026 from https://aws.amazon.com/ec2/nitro/nitro-enclaves/

[4] Sebastian Angel, Aditya Basu, Weidong Cui, Trent Jaeger, Stella Lau, Srinath Setty, and Sudheesh Singanamalla. 2023. Nimble: Rollback Protection for Confidential Cloud Services. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, 193–208. https://www.usenix.org/conference/osdi23/presentation/angel

[5] Apache ResilientDB (Incubating). 2026. Fides Source Code. GitHub repository, commit fc8b9701c216f72824296b8f9cc0f37d0b464763. Retrieved July 21, 2026 from https://github.com/apache/incubator-resilientdb/tree/fc8b9701c216f72824296b8f9cc0f37d0b464763

[6] Apache ResilientDB (Incubating). n.d. Apache ResilientDB (Incubating). Retrieved July 21, 2026 from https://resilientdb.incubator.apache.org/

[7] Aptos Foundation. n.d. Aptos: The Foundation for a New Digital Economy. Retrieved July 21, 2026 from https://aptosnetwork.com/

[8] Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. 2025. Shoal++: High Throughput DAG BFT Can Be Fast and Robust!. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, 813–826. https://www.usenix.org/conference/nsdi25/presentation/arun

[9] Balaji Arun and Binoy Ravindran. 2022. Scalable Byzantine Fault Tolerance via Partial Decentralization. *Proceedings of the VLDB Endowment* 15, 9 (2022), 1739–1752. https://doi.org/10.14778/3538598.3538599

[10] Zeta Avarikioti, Lioba Heimbach, Roland Schmid, Laurent Vanbever, Roger Wattenhofer, and Patrick Wintermeyer. 2023. FnF-BFT: A BFT Protocol with Provable Performance Under Attack. In *Structural Information and Communication Complexity (Lecture Notes in Computer Science)*, Vol. 13892. Springer Nature Switzerland, 165–198. https://doi.org/10.1007/978-3-031-32733-9_9

[11] Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. 2025. Mysticeti: Reaching the Latency Limits with Uncertified DAGs. In *Proceedings 2025 Network and Distributed System Security Symposium*. Internet Society. https://doi.org/10.14722/ndss.2025.240929

[12] Leemon Baird and Atul Luykx. 2020. The Hashgraph Protocol: Efficient Asynchronous BFT for High-Throughput Distributed Ledgers. In *2020 International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE, 1–7. https://doi.org/10.1109/COINS49042.2020.9191430

[13] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. 2017. Hybrids on Steroids: SGX-Based High Performance BFT. In *Proceedings of the Twelfth European Conference on Computer Systems*. ACM, 222–237. https://doi.org/10.1145/3064176.3064213

[14] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *Advances in Cryptology – ASIACRYPT 2001 (Lecture Notes in Computer Science)*, Vol. 2248. Springer, 514–532. https://doi.org/10.1007/3-540-45682-1_30

[15] Pietro Borrello, Andreas Kogler, Martin Schwarzl, Moritz Lipp, Daniel Gruss, and Michael Schwarz. 2022. ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, 3917–3934. https://www.usenix.org/conference/usenixsecurity22/presentation/borrello

[16] Gabriel Bracha. 1987. Asynchronous Byzantine Agreement Protocols. *Information and Computation* 75, 2 (1987), 130–143. https://doi.org/10.1016/0890-5401(87)90054-X

[17] Ethan Buchman, Jae Kwon, and Zarko Milosevic. 2018. The Latest Gossip on BFT Consensus. https://doi.org/10.48550/arXiv.1807.04938 arXiv:1807.04938 [cs.DC]

[18] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 991–1008. https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

[19] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *3rd Symposium on Operating Systems Design and Implementation (OSDI 99)*. USENIX Association, 173–186. https://www.usenix.org/conference/osdi-99/practical-byzantine-fault-tolerance

[20] Tushar Deepak Chandra and Sam Toueg. 1996. Unreliable Failure Detectors for Reliable Distributed Systems. *J. ACM* 43, 2 (1996), 225–267. https://doi.org/10.1145/226643.226647

[21] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2019. SgxPectre: Stealing Intel Secrets from SGX Enclaves via Speculative Execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 142–157. https://doi.org/10.1109/EuroSP.2019.00020

[22] Florent Chevrou, Aurélie Hurault, and Philippe Quéinnec. 2016. On the Diversity of Asynchronous Communication. *Formal Aspects of Computing* 28, 5 (2016), 847–879. https://doi.org/10.1007/s00165-016-0379-x

[23] Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin, and Mirco Marchetti. 2009. Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults. In *6th USENIX Symposium on Networked Systems Design and Implementation (NSDI 09)*. USENIX Association, 153–168. https://www.usenix.org/conference/nsdi-09/making-byzantine-fault-tolerant-systems-tolerate-byzantine-faults

[24] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. Cryptology ePrint Archive, Paper 2016/086. https://eprint.iacr.org/2016/086

[25] Xiaohai Dai, Guanxiong Wang, Jiang Xiao, Zhengxuan Guo, Rui Hao, Xia Xie, and Hai Jin. 2024. LightDAG: A Low-Latency DAG-Based BFT Consensus through Lightweight Broadcast. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 998–1008. https://doi.org/10.1109/IPDPS57955.2024.00093

[26] Xiaohai Dai, Zhaonan Zhang, Jiang Xiao, Jingtao Yue, Xia Xie, and Hai Jin. 2023. GradedDAG: An Asynchronous DAG-Based BFT Consensus with Lower Latency. In *2023 42nd International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 107–117. https://doi.org/10.1109/SRDS60354.2023.00020

[27] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: A DAG-Based Mempool and Efficient BFT Consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*. ACM, 34–50. https://doi.org/10.1145/3492321.3519594

[28] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. 2022. DAMYSUS: Streamlined BFT Consensus Leveraging Trusted Components. In *Proceedings of the Seventeenth European Conference on Computer Systems*. ACM, 1–16. https://doi.org/10.1145/3492321.3519568

[29] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. 2024. OneShot: View-Adapting Streamlined BFT Protocols with Trusted Execution Environments. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 1022–1033. https://doi.org/10.1109/IPDPS57955.2024.00095

[30] Larry Dewey and Diego Gonzalez Villalobos. 2023. SEV-SNP Platform Attestation Using VirTEE/SEV. AMD Publication 58217, Revision 1.2. Retrieved July 21, 2026 from https://docs.amd.com/v/u/en-US/58217_amd-epyc-9004-ug-platform-attestation-using-virtee-snp

[31] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* 35, 2 (1988), 288–323. https://doi.org/10.1145/42282.42283

[32] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Dumbo-NG: Fast Asynchronous BFT Consensus with Throughput-Oblivious Latency. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1187–1201. https://doi.org/10.1145/3548606.3559379

[33] Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. 2024. Autobahn: Seamless High Speed BFT. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. ACM, 1–23. https://doi.org/10.1145/3694715.3695942

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

[35] Suyash Gupta, Dakai Kang, Dahlia Malkhi, and Mohammad Sadoghi. 2025. Brief Announcement: Carry the Tail in Consensus Protocols. In *39th International Symposium on Distributed Computing (DISC 2025) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 356. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 59:1–59:7. https://doi.org/10.4230/LIPIcs.DISC.2025.59

[36] 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. https://doi.org/10.14778/3380750.3380757

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

[38] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. 2013. Using Innovative Instructions to Create Trustworthy Software Solutions. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 1–1. https://doi.org/10.1145/2487726.2488370

[39] Intel Corporation. 2016. Introduction to Intel® SGX Sealing. Retrieved July 21, 2026 from https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-intel-sgx-sealing.html

[40] Intel Corporation. 2022. Intel® Software Guard Extensions (Intel® SGX) SDK for Linux* OS: Developer Reference. Version 2.18. Retrieved July 21,

2026 from https://download.01.org/intel-sgx/sgx-linux/2.18/docs/Intel_SGX_Developer_Reference_Linux_2.18_Open_Source.pdf
[41] Intel Corporation. 2025. Attestation Services for Intel® Software Guard Extensions. Retrieved July 21, 2026 from https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html
[42] Intel Corporation. 2026. Platform Security Guidance from Intel. Retrieved July 21, 2026 from https://www.intel.com/content/www/us/en/developer/topic-technology/software-security-guidance/overview.html
[43] Philipp Jovanovic, Lefteris Kokoris-Kogias, Bryan Kumara, Alberto Sonnino, Pasindu Tennage, and Igor Zablotchi. 2025. Mahi-Mahi: Low-Latency Asynchronous BFT DAG-Based Consensus. In *2025 IEEE 45th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 549–559. https://doi.org/10.1109/ICDCS63083.2025.00060
[44] Jupiter. n.d. Jupiter: The Home of Onchain Finance. Retrieved July 21, 2026 from https://jup.ag/
[45] 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, Article 171 (2025), 29 pages. https://doi.org/10.1145/3725308
[46] Dakai Kang, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2024. SpotLess: Concurrent Rotational Consensus Made Practical Through Rapid View Synchronization. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 1916–1929. https://doi.org/10.1109/ICDE60146.2024.00157
[47] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. 2021. All You Need Is DAG. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. ACM, 165–175. https://doi.org/10.1145/3465084.3467905
[48] Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. 2009. TrInc: Small Trusted Hardware for Large Distributed Systems. In *6th USENIX Symposium on Networked Systems Design and Implementation (NSDI 09)*. USENIX Association, 1–14. https://www.usenix.org/conference/nsdi-09/trinc-small-trusted-hardware-large-distributed-systems
[49] Andrew Lewis-Pye and Ittai Abraham. 2024. Fever: Optimal Responsive View Synchronisation. In *27th International Conference on Principles of Distributed Systems (OPODIS 2023) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 286. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 14:1–14:16. https://doi.org/10.4230/LIPIcs.OPODIS.2023.14
[50] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*. ACM, 129–138. https://doi.org/10.1145/3382734.3405707
[51] Hanzheng Lyu, Shaokang Xie, Jianyu Niu, Ivan Beschastnikh, Yinqian Zhang, Mohammad Sadoghi, and Chen Feng. 2025. Orthrus: Accelerating Multi-BFT Consensus Through Concurrent Partial Ordering of Transactions. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 2615–2627. https://doi.org/10.1109/ICDE65448.2025.00197
[52] Hanzheng Lyu, Shaokang Xie, Jianyu Niu, Chen Feng, Yinqian Zhang, and Ivan Beschastnikh. 2025. Ladon: High-Performance Multi-BFT Consensus via Dynamic Global Ordering. In *Proceedings of the Twentieth European Conference on Computer Systems*. ACM, 226–242. https://doi.org/10.1145/3689031.3696102
[53] Hanzheng Lyu, Shaokang Xie, Jianyu Niu, Mohammad Sadoghi, Yinqian Zhang, Cong Wang, Ivan Beschastnikh, and Chen Feng. 2025. HYDRA: Breaking the Global Ordering Barrier in Multi-BFT Consensus. https://doi.org/10.48550/arXiv.2511.05843 arXiv:2511.05843
[54] Dahlia Malkhi and Kartik Nayak. 2023. HotStuff-2: Optimal Two-Phase Responsive BFT. Cryptology ePrint Archive, Paper 2023/397. https://eprint.iacr.org/2023/397
[55] Dahlia Malkhi, Chrysoula Stathakopoulou, and Maofan Yin. 2025. BBCA-Chain: Low Latency, High Throughput BFT Consensus on a DAG. In *Financial Cryptography and Data Security (FC 2024) (Lecture Notes in Computer Science)*, Vol. 14744. Springer, 51–73. https://doi.org/10.1007/978-3-031-78676-1_4
[56] Dahlia Malkhi and Paweł Szalachowski. 2023. Maximal Extractable Value (MEV) Protection on a DAG. In *4th International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2022) (Open Access Series in Informatics (OASIcs))*, Vol. 110. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 6:1–6:17. https://doi.org/10.4230/OASIcs.Tokenomics.2022.6
[57] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. 2017. ROTE: Rollback Protection for Trusted Execution. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 1289–1306. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/matetic
[58] Microsoft. 2024. Azure Confidential Computing. Retrieved July 21, 2026 from https://learn.microsoft.com/en-us/azure/confidential-computing/
[59] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The Honey Badger of BFT Protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 31–42. https://doi.org/10.1145/2976749.2978399
[60] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-Based Fault Injection Attacks against Intel SGX. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1466–1482. https://doi.org/10.1109/SP40000.2020.00057
[61] Jianyu Niu, Wei Peng, Xiaokuan Zhang, and Yinqian Zhang. 2022. NARRATOR: Secure and Practical State Continuity for Trusted Execution in the Cloud. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2385–2399. https://doi.org/10.1145/3548606.3560620
[62] Jianyu Niu, Xiaoqing Wen, Guanlong Wu, Shengqi Liu, Jiangshan Yu, and Yinqian Zhang. 2025. Achilles: Efficient TEE-Assisted BFT Consensus via Rollback Resilient Recovery. In *Proceedings of the Twentieth European Conference on Computer Systems*. ACM, 193–210. https://doi.org/10.1145/3689031.3717457
[63] Open Enclave SDK Project. n.d. Open Enclave SDK. Retrieved July 21, 2026 from https://github.com/openenclave/openenclave
[64] Osmosis. n.d. Osmosis. Retrieved July 21, 2026 from https://osmosis.zone/
[65] Wei Peng, Xiang Li, Jianyu Niu, Xiaokuan Zhang, and Yinqian Zhang. 2024. Ensuring State Continuity for Confidential Computing: A Blockchain-Based Approach. *IEEE Transactions on Dependable and Secure Computing* 21, 6 (2024), 5635–5649. https://doi.org/10.1109/TDSC.2024.3381973
[66] Hany Ragab, Alyssa Milburn, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2021. CrossTalk: Speculative Data Leaks Across Cores Are Real. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1852–1867. https://doi.org/10.1109/SP40001.2021.00020
[67] Sajjad Rahnama, Suyash Gupta, Thamir M. Qadah, Jelle Hellings, and Mohammad Sadoghi. 2020. Scalable, Resilient, and Configurable Permissioned Blockchain Fabric. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2893–2896. https://doi.org/10.14778/3415478.3415502
[68] Alexander Schrijver. 1986. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, UK.
[69] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (1979), 612–613. https://doi.org/10.1145/359168.359176
[70] Victor Shoup. 2025. Blue Fish, Red Fish, Live Fish, Dead Fish. In *2025 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 1–7. https://doi.org/10.1109/ICBC64466.2025.11114699
[71] Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, and Kartik Nayak. 2025. Sailfish: Towards Improving the Latency of DAG-Based BFT. In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1928–1946. https://doi.org/10.1109/SP61157.2025.00021
[72] Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li. 2025. Shoal: Improving DAG-BFT Latency and Robustness. In *Financial Cryptography and Data Security (FC 2024) (Lecture Notes in Computer Science)*, Vol. 14744. Springer Nature Switzerland, 92–109. https://doi.org/10.1007/978-3-031-78676-1_6
[73] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. 2022. Bullshark: DAG BFT Protocols Made Practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2705–2718. https://doi.org/10.1145/3548606.3559361
[74] Chrysoula Stathakopoulou, Tudor David, Matej Pavlovic, and Marko Vukolić. 2022. Mir-BFT: Scalable and Robust BFT for Decentralized Networks. *Journal of Systems Research* 2, 1 (2022). https://doi.org/10.5070/SR32159278
[75] Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolić. 2022. State Machine Replication Scalability Made Simple. In *Proceedings of the Seventeenth European Conference on Computer Systems*. ACM, 17–33. https://doi.org/10.1145/3492321.3519579
[76] Sui Foundation. n.d. Sui: The Full Stack for a New Global Economy. Retrieved July 21, 2026 from https://www.sui.io/
[77] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. 2020. SGAxe: How SGX Fails in Practice. Retrieved July 21, 2026 from https://sgaxe.com/
[78] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. 2021. CacheOut: Leaking Data on Intel CPUs via Cache Evictions. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 339–354. https://doi.org/10.1109/SP40001.2021.00064
[79] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Veríssimo. 2013. Efficient Byzantine Fault-Tolerance. *IEEE Trans. Comput.* 62, 1 (2013), 16–30. https://doi.org/10.1109/TC.2011.221
[80] Weili Wang, Sen Deng, Jianyu Niu, Michael K. Reiter, and Yinqian Zhang. 2022. ENGRAFT: Enclave-Guarded Raft on Byzantine Faulty Nodes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2841–2855. https://doi.org/10.1145/3548606.3560639
[81] Xiaoqing Wen, Quanbi Feng, Hanzheng Lyu, Jianyu Niu, Yinqian Zhang, and Chen Feng. 2025. TeeRollup: Efficient Rollup Design Using Heterogeneous TEE. *IEEE Trans. Comput.* 74, 10 (2025), 3546–3558. https://doi.org/10.1109/TC.2025.3596698
[82] Zhuolun Xiang, Sourav Das, Zekun Li, Zhoujun Ma, and Alexander Spiegelman. 2026. The Latency Price of Threshold Cryptosystem in Blockchains. In *Financial Cryptography and Data Security (FC 2025) (Lecture Notes in Computer Science)*, Vol. 15752. Springer, 291–308. https://doi.org/10.1007/978-3-032-07035-7_18

[83] Shaokang Xie, Dakai Kang, Hanzheng Lyu, Jianyu Niu, and Mohammad Sadoghi. 2026. Fides: Secure and Scalable Asynchronous DAG Consensus via Trusted Components. arXiv:2501.01062 [cs.DC] https://arxiv.org/abs/2501.01062

[84] Yuval Yarom, Daniel Genkin, and Nadia Heninger. 2016. CacheBleed: A Timing Attack on OpenSSL Constant Time RSA. In *Cryptographic Hardware and Embedded Systems – CHES 2016 (Lecture Notes in Computer Science)*, Vol. 9813. Springer, 346–367. https://doi.org/10.1007/978-3-662-53140-2_17

[85] Jiashuo Zhang, Jianbo Gao, Ke Wang, Zhenhao Wu, Yue Li, Zhi Guan, and Zhong Chen. 2022. TBFT: Efficient Byzantine Fault Tolerance Using Trusted Execution Environment. In *ICC 2022 – IEEE International Conference on Communications*. IEEE, 1004–1009. https://doi.org/10.1109/ICC45855.2022.9839169

[86] Liangrong Zhao, Jérémie Decouchant, Joseph K. Liu, Qinghua Lu, and Jiangshan Yu. 2024. Trusted Hardware-Assisted Leaderless Byzantine Fault Tolerance Consensus. *IEEE Transactions on Dependable and Secure Computing* 21, 6 (2024), 5086–5097. https://doi.org/10.1109/TDSC.2024.3357521