Not eXactly Byzantine: Efficient and Resilient TEE-Based State Machine Replication

Marc Leinweber, Hannes Hartenstein
KASTEL Security Research Labs
Karlsruhe Institute of Technology (KIT)
marc.leinweber@kit.edu, hannes.hartenstein@kit.edu

***Abstract*—We propose, implement, and evaluate NxBFT, a resilient and efficient State Machine Replication protocol using Trusted Execution Environments (TEEs). NxBFT focuses on a "Not eXactly Byzantine" (NXB) operating model as a middle ground between crash and Byzantine fault tolerance. NxBFT's consensus layer is asynchronous, graph-based, leaderless, and optimized for the NXB operating model, enabling load-balancing of requests between replicas and, in fault-free cases, two network round trips between decisions. We identify fundamental issues with crash recovery due the use of TEEs in asynchrony that only can be circumvented by relying on synchrony for liveness. We provide a throughput-latency trade-off analysis of NxBFT, Chained-Damysus (rotating leader), and MinBFT (static leader) for up to 40 replicas and network round trip latencies up to 150 ms. NxBFT achieves the highest throughput in all scenarios. When small latencies are required, MinBFT and Damysus are at an advantage with Damysus benefiting from the NXB model in terms of throughput for small deployments. In contrast to leader-based approaches, NxBFT's performance is almost not impacted when actual crash faults occur.**

***Index Terms*—Directed Acyclic Graph (DAG), SMR, Atomic Broadcast, Trusted execution, Asynchrony, Recovery**

I. Introduction

In this paper, we focus on federations in the permissioned blockchain model, where a set of operators (federation members) are running a State Machine Replication (SMR) protocol [1] to provide a common service. We assume that the federation consists, at most, of tens of operators and typically is spread over a country or continent. We also assume a main motivation of this operating model to be sovereignty reasons, that is, to have equal power and rights. Still, we assume that the federation members run a Byzantine atomic broadcast for resilience reasons. However, we deviate from classical Byzantine fault tolerant (BFT) SMR by assuming that a replica does not show Byzantine behavior in its interaction with its clients. We consider this assumption valid for use cases where the federation members are highly regulated and manipulation towards the client is directly observable and correctable, for example when logging check-ins and check-outs as part of a ticketing system for mobility-as-a-service federation [2], [3]. In this example, operators can be mobility providers or public parties, e.g., states of a federal state or member states of a supranational union. This scenario also generalizes to similar transaction processing tasks for electric vehicle charging or smart grid scenarios [4] and, potentially, even to some Central Bank Digital Currency [5], [6] scenarios. These use cases require high throughput, at least in the tens of thousands of transactions per second, high availability, and resilience. Thus, we are interested in understanding how such a "Not eXactly Byzantine" (NXB) fault model can be exploited by protocol design and, correspondingly, to evaluate gains and trade-offs.

We propose an approach called NxBFT that makes use of Trusted Execution Environments (TEEs) as well as asynchronous and graph-based atomic broadcast. TEEs are chosen since they are able to significantly increase efficiency and performance [7], [8], [9]. Asynchronous, graph-based atomic broadcast is chosen since it has demonstrated resilience and impressive throughput [10]. NxBFT is based on TEE-Rider [11], an asynchronous and graph-based atomic broadcast protocol with a Byzantine fault tolerance of $n > 2f$, where n denotes the number of replicas and f the number of faulty nodes. TEE-Rider itself is based on DAG-Rider [12]. To increase *resilience, throughput* and *practicality*, we extend the NXB fault model to an operating model: While, in general, operating in a Byzantine and asynchronous environment, we assume that operators (1) do not attack their TEE themselves, (2) do not manipulate the business logic of their application, and (3) enable sufficiently long phases of synchrony required for "maintenance work". The NXB model allows maximum utilization of the inherent parallelism of TEE-Rider, thereby increasing throughput significantly, and to overcome fundamental issues with crash recovery.

The main contribution of this paper is to show that the *combination* of the NXB operating model and the NxBFT approach is highly beneficial in terms of efficiency, crash resilience, and throughput. In particular, our contributions are as follows:

Co-Design of Assumptions and Algorithms: With NxBFT, we designed a performance and resilience-oriented SMR protocol with crash recovery optimized for the NXB model. NxBFT requires only half a network round trip for a logical round. NxBFT uses a small TEE for non-equivocation and a common coin; the goal is to reduce costly cryptography and context switches. We exploit the NXB model such that each message exchanged between replicas at the same time distributes new client requests and drives the consensus process.

Practicality in the Light of Fundamental Challenges: We observed that the intricacies of the combination of TEE and asynchrony have not yet been fully explored. Recovery of a crashed replica p_c requires resetting the TEE of p_c and

agreement between all correct replicas on the message history of p_c, i.e., agreement on a checkpoint. Asynchrony makes it impossible for a replica to be sure that it has received the same of p_c's messages as other replicas, denying quorum-based checkpoint establishment. We discuss these issues and a corresponding workaround relying on synchrony for liveness. Additionally, we propose a pragmatic setup procedure.

High Throughput and Resilience: We implemented NxBFT and provide a throughput-latency trade-off analysis in comparison to Chained-Damysus [13] (rotating leader) and MinBFT [14] (static leader) for up to 40 replicas and network round trip latencies up to $150\,\mathrm{ms}$[1]. NxBFT outperforms MinBFT and Damysus in terms of throughput in all settings. For $n = 40$, NxBFT achieves a throughput of $178\,\mathrm{kOp/s}$ for datacenter deployments and $20\,\mathrm{kOp/s}$ for world-wide deployments. The throughput-oriented optimizations based on the NXB model have a latency penalty that we deem reasonable. When small latencies are required, MinBFT and Damysus are at an advantage, with Damysus benefiting from the NXB model in terms of throughput for small deployments. In contrast to MinBFT and Damysus, NxBFT's performance is almost not affected by actual crash faults. Please note that a general comparison between TEE-based and non-TEE-based approaches is not a goal of this paper.

The remainder of this paper is structured as follows. We give an overview on related work on hybrid fault models, asynchronous consensus, and recovery procedures in Sec. II. In Sec. III, we define the NXB model and give an overview on NxBFT. We describe the NxBFT design in Sec. IV in detail. Our experimental setup and empirical findings are presented in Sec. V. We discuss the implications of our approach and future research directions in Sec. VI and conclude with Sec. VII.

II. Background and Related Work

Since NXB is a variant of a hybrid fault model and NxBFT is constructed and optimized using atomic broadcast based on directed acyclic graphs (DAGs), we present corresponding related work. Furthermore, since recovery is particularly challenging when TEEs are used, we also address this specific aspect. We assume the reader is familiar with the concepts of State Machine Replication (SMR) [1] and of classical Byzantine fault tolerant SMR approaches like PBFT [15].

A. Hybrid Fault Models

The core idea of hybrid fault models, which have been investigated for at least two decades [16], is to equip replicas with a trustworthy subsystem that is assumed to only fail by crashing and enforcing non-equivocation: A replica cannot send two messages with different contents to different peers in the same context without being noticed [17], [18], [19], [20]. The non-equivocation property saves rounds of communication and allows operating BFT SMR with a fault tolerance of $n > 2f$. With the advent of hardware-based Trusted Execution Environments (TEEs) like Intel SGX [21], researchers were able to propose practical systems using a trusted subsystem in the partially synchronous model adopting ideas from PBFT [22], [14], [7], [23]. These works rely on a TEE to provide confidential execution of code. Additionally, a TEE must also be able to remotely attest to the integrity of itself and code running on it. The TEE implements a signature service that maintains an increment-only counter. When broadcasting a message, a replica has to attach a signature and a corresponding counter. Receiving replicas will only accept one message for each possible counter value for each replica. Under the assumption that the TEE will assign each counter value only once, receiving replicas can be sure that they all see the same message for the same counter value. Recent work focuses on the rotating leader paradigm [13], [9], the asynchronous [24], [11], [25] timing model, and alternative trusted subsystems [26]. We show, in contrast to [27]'s claim, that trusted subsystems allow a highly parallelized and efficient consensus protocol design as it is also stated by [28].

B. DAG-Based Atomic Broadcast

Asynchronous consensus protocols promise increased resilience and throughput while at the same time being less complex than their partially synchronous counterparts [29], [10]. Hashgraph [30] and DAG-Rider [12] were among the first asynchronous consensus protocols that use DAGs to encode the logical chronology of messages exchanged and derive the total order of transactions using a deterministic graph traversal. In contrast to Hashgraph (and its TEE variant [24]), DAG-Rider simplifies the consensus derivation and the message exchange by building the graph deterministically. With Narwhal [10], the messages exchanged were reduced from $O(n^3)$ to $O(n^2)$ compared to DAG-Rider. TEE-Rider [11] showed that DAG-Rider can be compiled to withstand Byzantine faults with $n > 2f$ replicas using a TEE-based signature service. Ladelsky and Friedman [31] generalize the proof of [11] and analyze the impact of quorum sizes on termination properties. Yanadmuri et al. [32] showed how to reach quadratic communication complexity in the worst case using a small TEE. NxBFT builds upon TEE-Rider for increased throughput and for its less complex implementation; it further improves ideas from Narwhal to reach quadratic communication complexity. In contrast to Fides [25], which pursues similar ideas to TEE-Rider, NxBFT improves both communication complexity and throughput and minimizes the size of the TEE-deployed code. We comment on parallel work on leaderless atomic broadcast in *partial* synchrony, in contrast to the work at hand, in Sec. VI.

C. Crash Recovery and Reconfiguration

When replicas experience unintended faults, e.g., in the case of hardware failures or simply when doing maintenance, they miss state updates and cannot participate anymore. A recovery procedure allows a replica to catch up and be able to produce valid messages and to decide the validity of received messages [33]. PBFT [15] implements a checkpointing mechanism usable for recovery. A recovery mechanism can be extended

[1]All implementations are available at https://blinded.for/review

to support reconfiguration: replicas may join or leave the peer-to-peer network based on the ongoing consensus algorithm. A TEE-aware recovery procedure requires the establishment of a new signature secret and it has to bring all correct replicas "on the same page" concerning the messages broadcast by the recovering replica. To the best of our knowledge, CCF [34] and Achilles [35] are the only TEE-based SMR systems offering recovery. Both are leader-based and operate in partial synchrony. We show that such an approach is incompatible with the asynchronous nature of NxBFT's atomic broadcast.

III. NxBFT: Overview

We explicate our assumptions of the NxB operating model and provide an overview of the protocol structure of NxBFT.

A. Assumptions: NxB Operating Model

We consider a federation of n predefined operators offering a common service using State Machine Replication (SMR) for an arbitrary number of clients. Each operator operates a replica; $f < \frac{n}{2}$ replicas and all clients may behave Byzantine. We assume replicas to execute the same non-manipulated state machine limiting operators to omission faults when communicating with clients. Each replica is equipped with a Trusted Execution Environment (TEE) capable of running arbitrary code (called enclave) that offers confidential and integrity protected computing as well as remote attestation to verify the authenticity of the TEE and its enclave. The TEE can confidentially and authentically export (partial) enclave state ("sealing"). The TEE is assumed to only fail by crashing; its internal state is lost when crashing. If not explicitly stated, code is not executed inside the TEE. Replicas and clients communicate via secure point-to-point links. Messages can be reordered and arbitrarily delayed but not dropped. Setup and recovery require synchrony for liveness.

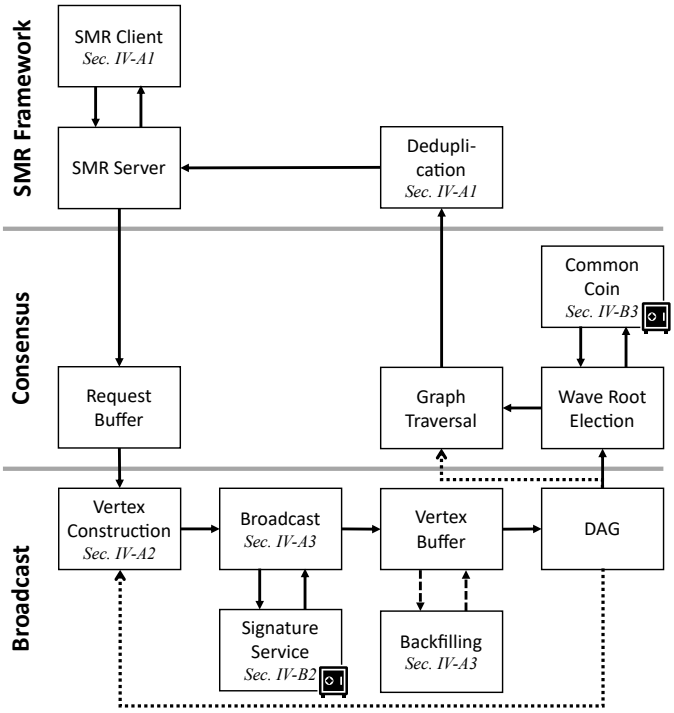


Fig. 1. NxBFT components for normal operation (setup and recovery left out). The safe icon marks components executed inside the TEE. Solid arrows show the flow for a client request to be successfully ordered and executed. After being received by the replica, a request is buffered and eventually broadcast as a vertex payload using TEE-based signatures. A vertex is added to the DAG when its ancestry was already added (optional backfilling, dashed arrows). Each added vertex will eventually be ordered by a graph traversal; the corresponding root is selected using a common coin. After request deduplication, the request is output to the application layer and the response sent to the client. The dotted arrows highlight components using the DAG as input. Serif italic font names the section covering the component description.

B. Protocol: NxBFT

NxBFT makes use of the NxB operating model: each replica uses its TEE for a signature service and a common coin. NxBFT builds on TEE-Rider [11] that maintains a DAG that is structured in rounds. With the exception of setup and recovery, NxBFT operates asynchronously. NxBFT is structured in three layers (see Figure 1): SMR framework, consensus, and broadcast. In the following, we describe a full Request–Broadcast–Consensus–Response cycle.

a) Request: When a client wants to use the federated service, it composes a request containing the command and a sequence number and sends the request to a single replica (cf. Sec. IV-A1). A receiving replica buffers the request until it can be included in the payload of a broadcast round.

b) Broadcast: In each round r of the broadcast layer, each replica has to propose a new DAG vertex containing client requests to be ordered (cf. Sec. IV-A2). Vertices v are bound to a round $v.r$ and broadcast in a best effort fashion. NxBFT uses a TEE-based signature service (cf. Sec. IV-B2) to prevent equivocation on the broadcast layer limiting faulty replicas to omission faults. The counter of the signature service is used to enforce a FIFO ordering, i.e., a message with lower counter value must be delivered before a message with higher counter value can be delivered. If received vertices contain an already processed counter value or an invalid signature, they are dropped. A vertex needs to reference $\lfloor \frac{n}{2} \rfloor + 1$ vertices of the previous round as edges. Correct replicas will always reference their own vertex of the previous round as an edge. Due to asynchrony, vertices for rounds $v.r < r$ or $v.r > r$ may be received any time. Received valid vertices are buffered until they can be added to a replica's local DAG. A vertex can be moved from buffer to DAG if the replica already transitioned to the vertex' round and the replica's DAG contains all referenced vertices. By transitively applying this requirement, the *complete* ancestry of a newly added vertex must already be part of the DAG. If vertices of the ancestry are missing, the replica will ask its peers to retransmit said vertices (backfilling). DAG construction and backfilling-based reliable broadcast (cf. Sec.IV-A3) in combination build a causal order broadcast [36, Module 3.9]; the DAG encodes the causal order of exchanged messages. A replica completes round r and transitions to round $r + 1$ when it added at least $\lfloor \frac{n}{2} \rfloor + 1$

valid vertices for round r to the DAG.

c) Consensus: Every disjoint four consecutive broadcast rounds are grouped into a *wave*: round r belongs to wave $w = \lceil \frac{r}{4} \rceil$. When a wave w is completed, that is, its fourth round is completed, the consensus layer is invoked: It tries to derive agreement on a total order of requests of wave numbers smaller than w for all correct replicas. The wave construction works as follows. Assume a wave w is complete. A vertex of the first round of wave w is selected as wave root; since the TEE-based common coin (cf. Sec. IV-B3) is used for selection, all correct nodes select the same vertex. If the wave root is part of a replica's local DAG when it is selected and at least $\lfloor \frac{n}{2} \rfloor + 1$ vertices of w's fourth round have a path to it ("direct commit rule"), the replica can commit the wave: The wave root is then used as starting point for a pre-defined deterministic graph traversal to order all not yet ordered requests of previous waves. If the wave root is not part of the DAG, the wave cannot be committed yet. A replica will check during direct commits of future waves if it can retrospectively commit a wave. For a retrospective commit, a single path between old and new wave root is sufficient. The construction of a wave ensures that the root of the next wave (and all future waves) has a path to a previous root. For proofs of correctness, we refer the reader to [11, Lemma 3] and [12, Proposition 2]. However, it is important to note that the consensus layer works solely on the local DAG without any additional communication.

d) Response: The requests are executed by the server-side application in the order they were added to the vertex by the proposing replica. The sequence number of a request is used for deduplication (cf. Sec. IV-A1): the combination of client id and sequence number is output exactly once to the application. After execution, the result is sent as a response to the initially requesting client.

IV. NxBFT: Design Considerations

We discuss various design considerations in detail and refer the reader to Appendix A for arguments of correctness.

A. Increasing Throughput

1) NxB *Client Model:* Clients select a replica at random and unicast their request accompanied with a client id and a sequence number. Once the consensus layer decided, all correct replicas will input the request to the state machine and the output of the (application logic) state machine is sent as a response. In the NxB model, a single response suffices for the client to complete the request. This mechanism allows each replica to propose a disjoint set of requests in each round scaling up throughput with n.

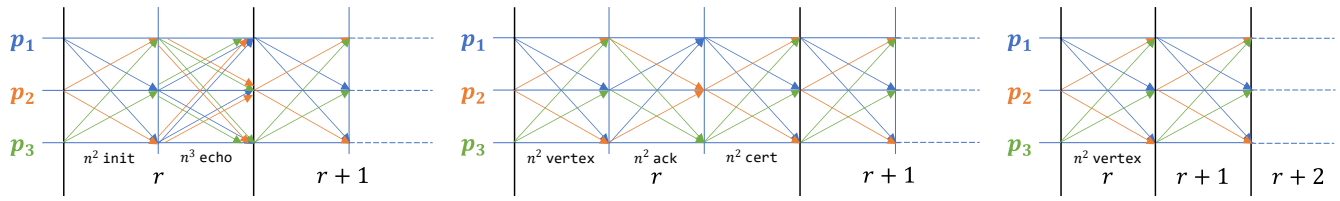
When the client issues its request, it starts a timer. If the selected replica does not answer within the time interval, the client selects a different replica for its request (and so forth). The client is guaranteed that its request will eventually be ordered, however, a request can appear in the DAG up to n times. NxBFT employs a simple deduplication logic: a correct client will only have one unanswered request at a time. The replicas store for each client the client's last executed sequence number. A request is only input to the state machine when its sequence number is greater than the stored sequence number. If a client issues more than one request simultaneously or equivocates on sequence numbers, it is possible that some of its requests will not be answered.

2) Vertex Construction: Asynchronous algorithms typically rely on valid messages from peers to make progress (see, e.g., [12, Algorithm 2]). In the case of TEE-Rider, a replica needs at least $\lfloor \frac{n}{2} \rfloor + 1$ valid vertices for its current round to transition to the next round and to be allowed to broadcast its next vertex. To increase the achievable throughput, a replica holds back its vertex until it can propose at least a configurable amount of client request as the vertex' payload. If now only a few clients issue requests, the system would come to a halt as the quorum for a round to complete would need a lot of time to be established. The NxB client model worsens this problem as requests are equally distributed among the replicas. We address this issue by working with (local) timers on the side of the replica: When a replica is not able to broadcast a vertex containing enough client request within a pre-defined time interval, the replica will broadcast the vertex anyway. This allows other replicas to complete the round.

3) Backfilling-Based Reliable Broadcast: Inspired by Narwhal [10] and in contrast to TEE-Rider [11], correct replicas do not echo vertices by default. Instead, if an ancestor is unknown, i.e., it was neither already added to the DAG nor buffered, a request for the missing vertex is sent to all replicas. Due to asynchrony, correct replicas cache vertex requests if they cannot answer them directly: It is possible that the corresponding vertex arrives delayed and the replica receiving the vertex request is the only correct replica that receives the requested vertex due to faults by the sender. A different option would be for replicas to repeat their request after some time trading potentially unbounded memory consumption with algorithmic complexity. TEE-Rider uses a proactive single echo broadcast leading to $n^2 + n^3$ messages per broadcast round (Figure 2(a)). Narwhal uses a three-way handshake to produce quorum certificates that certify the availability of vertices. NxBFT can save a constant communication factor as we do not rely on quorum certificates: In the common case, Narwhal needs $3n^2$ messages per broadcast round (Figure 2(b)), whereas NxBFT proceeds with one n^2 communication step (Figure 2(c)). In the worst case, both send up to n backfilling requests to all replicas and receive n replies, leading to $5n^2$ messages for Narwhal vs. $3n^2$ messages for NxBFT per broadcast round.

B. Small Enclave for Efficiency and Resilience

The goal of our enclave design is a small trusted computing base and the reduction of costly context switches for efficient and resilient execution. The enclave of NxBFT combines the well-known concept of a TEE-based signature service to prevent equivocation [22], [14] with a common coin based on a cryptographically secure pseudorandom number generator (PRNG) and a pragmatic setup procedure. To be able to toss a coin, a replica has to convince the coin implementation that it



(a) TEE-Rider: $n^2 + n^3$ (b) Narwhal: $3n^2$ (c) NxBFT: n^2

Fig. 2. Common case communication patterns of TEE-Rider [11], Narwhal [10] and NxBFT (this work). Blue vertical lines delimit communication rounds, black vertical lines delimit rounds of the broadcast layer. The colors of the arrows signify the payloads' originator. NxBFT proceeds with one round of communication per broadcast round.

made sufficient progress on the consensus layer by providing sufficient signature service signatures from peering replicas. In the following, we use the term (public/secret) enclave keys to refer to the keys used for the signature service in contrast to long-living (public/secret) replica keys.

1) Setup: During setup, NxBFT enclaves mutually attest and verify their integrity, reach consensus on the deployed enclave code as well as the peers' public enclave keys to prevent simulation attacks, and establish the authenticated channels between replicas. The second objective of the setup protocol is to initialize the PRNG underlying the common coin with a collaboratively generated random seed. Due to the required participation of all n replicas in both of the above tasks, NxBFT cannot tolerate but reliably detect faults during setup. We construct the setup protocol using n synchronous authenticated reliable broadcast instances with a fault tolerance of $n > f$ [37] to preempt equivocation by faulty replicas. We describe the setup protocol of NxBFT as a one-way handshake between replicas p_i and p_j. A replica p_i also performs this setup handshake with itself.

Upon initialization, the enclave of p_i initializes its state variables and generates both a new enclave key pair and a random seed share for the later initialization of the common coin PRNG. Replica p_i now uses a synchronous authenticated single-echo reliable broadcast to disseminate its identifier and its signed attestation certificate carrying its public enclave key. Replica p_j waits to receive n validly signed echo messages carrying consistent and valid attestation certificates. Replica p_j then sends its encrypted seed share p_i (using p_i's public enclave key), thereby completing its part of the handshake. Replica p_i then invokes its enclave with the received encrypted seed share. The enclave decrypts the seed share and XOR's it with the current seed value. Once a replica has completed all n handshakes, it has completed the setup protocol and broadcasts a ready message to all replicas. Replicas start a timer for each handshake, the expiration of which raises an error and leads to an abortion of the entire setup. Similarly, conflicting or invalid messages and certificates raise errors and lead to an abort.

2) Signature Service: The signature service is used to enforce non-equivocation on messages. Each signature is accompanied with a unique counter value. Receiving replicas accept only one message per counter value. On initialization, each enclave generates an asymmetric key pair used exclusively for the signature service and sets its counter value c to 0. Attestation certificates generated by an enclave contain its public enclave key and are signed with a replica key during setup and recovery in order to establish a binding between these two key pairs. Thus, enclaves learn the public enclave keys of each other when verifying attestations ensuring the key stems from a running enclave. When a replica requests a signature for a message, the enclave signs the combination of message and current counter value with its secret enclave key, increments its counter by one, and returns signature and corresponding counter. Attestation certificates and signatures can be verified outside of enclaved execution as well.

3) Common Coin: Asynchronous consensus protocols rely on randomness to circumvent the FLP impossibility [38], [39]. NxBFT uses the common coin to select a wave root, i.e., it tosses a number in $\{0, \ldots, n-1\} \subset \mathbb{N}_0$. The common coin must be fair, it must produce the same output for every replica and wave, it must not be predictable until at least $\lfloor \frac{n}{2} \rfloor + 1$ replicas tossed for a wave, and it has to be revealed when at least $\lfloor \frac{n}{2} \rfloor + 1$ replicas toss for a wave. Typically, common coins are implemented using threshold signatures [40], [41]. We argue that, when already using a TEE, it seems rational to avoid the use of rather expensive cryptographic computations. The random bit string established by XOR'ing the local seed shares during setup is used to seed a PRNG. The coin implementation allows a coin toss whenever it can be convinced that four rounds of NxBFT's broadcast layer were successfully completed by the replica. Vertices have to be from the last round of the wave that is being tossed for. The implementation maintains a variable that is initially set to four and resembles the expected round in which a coin toss is allowed. A replica has to provide at least $\lfloor \frac{n}{2} \rfloor + 1$ validly signed vertices of different replicas and the expected round, proving the wave was completed by at least one correct replica. If the replica supplied sound evidence, the expected round is incremented by four and the next value of the PRNG returned.

C. Limits of and Approach to Crash Recovery

Faults like hardware failures, human configuration errors, and maintenance require that a replica can be put back into the condition to validate received messages and produce valid messages. We are interested in an algorithmic solution that can recover replicas while allowing the system to continuously

operate. Such a recovery procedure must ensure that the recovery does not allow (1) equivocation and (2) to learn coin values beforehand. For NxBFT in particular, the recovery procedure has to be aware of the TEE and its state. We first present two challenges one is facing in the design of a recovery procedure for NxBFT. Then we present a recovery procedure that is Byzantine safe under asynchrony but requires participation of *all* replicas for liveness.

1) Impossibility Argument: NxBFT's only *active* agreement primitive is an asynchronous TEE-based reliable broadcast with a fault tolerance of $n > f$. From the reliable broadcast message history and state, all decisions are passively derived. Thus, recovering a NxBFT replica is, for the most part, recovering the reliable broadcast module leading to two issues: (1) the recovery procedure cannot use the ongoing asynchronous consensus for coordination and (2) it is impossible to use a quorum-based decision for recovery.

We explain the first impossibility with the following example. Assume three replicas Alice, Bob, and Charlie with Charlie trying to recover and broadcasting a specially crafted request containing its new public enclave key authenticated with Charlie's public replica key. Alice and Bob both receive Charlie's request, propose it for ordering, and eventually reach consensus. Due to asynchrony, after reaching consensus on the recovery, Alice receives a vertex from Charlie signed with the old enclave key. Consequently, she will reject it. Bob, however, received the same vertex before reaching consensus with Alice and accepted it. If both now sort their graphs, they will get different orderings breaking the consensus layer's safety. Even when requesting the vertex from Bob because he references it in his vertex, Alice has no possibility to judge on Bob's and Charlie's honesty. Thus, if multiple identities of a peer's enclave exist simultaneously, equivocation can happen.

Recovery requires all correct replicas to reach agreement on the valid vertices of the recovering replica. Additionally, to prevent the necessity for rollbacks, the input of all correct replicas to such a decision has to be honored. This forbids a quorum-based approach, as any quorum reached in asynchrony may outvote a correct replica: In a TEE-based reliable broadcast with a fault tolerance of $n > f$, a receiving replica can, on successful signature verification, deliver immediately. Therefore, a correct replica Alice at an arbitrary but fixed point in time may be the only correct replica having delivered a certain value, e.g., of Bob. This is not in conflict with the totality property as Alice will relay the value and eventually all correct replicas will deliver as well. If Bob now crashes and recovers, all correct replicas need to agree on what Bob sent and what was potentially delivered. Any quorum smaller than n has a chance to outvote Alice: due to asynchrony, there is neither a guarantee that all votes building the quorum are cast by correct replicas nor that the votes used for a quorum are the same for all correct replicas. Thus, there exist traces in which Alice would need to "undeliver" Bob's value breaking the reliable broadcast totality property [36, Module 3.12]. If now any precondition is lifted, i.e., asynchrony, prevention of rollbacks, or TEEs preventing equivocation, this impossibility does not hold anymore.

2) Recovery Protocol: The recovery protocol is a variation of classical interactive consistency [37]. The recovery protocol assumes correct operators to create a backup of state and enclave at least after setup and after every successful recovery. The enclave backup uses sealing and contains all state except the secret enclave key and the counter value. To circumvent the introduced impossibilities, the protocol requires the input of *all* replicas to reach a recovery decision. Based on the backed-up state, a recovering replica p_c requests a recovery consensus of all replicas by broadcasting a *RecoveryRequest* message. Receiving replicas initialize a consensus procedure to achieve agreement among all n replicas on p_c's message history.

We assume that at any time, a number of $r < \lfloor \frac{n}{2} \rfloor + 1$ replicas request to recover. First, the recovering replica p_c will try to recover the enclave by providing an encrypted enclave state export. The replica p_c broadcasts a *RecoveryRequest* with the new attestation certificate including p_c's new public enclave key. A receiving replica p_i will verify the attestation, delete all buffered vertices that belong to p_c, and identify all vertices of p_c in its graph, i.e., p_c's message history. Finally, p_i broadcasts a *RecoveryProposal* for p_c with p_c's new attestation certificate and the message history as payload. As soon as n RecoveryProposals with equal attestation certificates are accepted, a correct replica p_i will identify the proposal with the longest valid vertex chain and broadcast it along with p_c's new attestation certificate as a *RecoveryCommit*. Each replica signs a RecoveryCommit with their secret replica key. On the receipt of $\lfloor \frac{n}{2} \rfloor + 1$ consistent RecoveryCommits, a replica p_i is able to complete the recovery procedure. Replica p_i will add all unknown vertices from the commit to the vertex buffer and set the expected counter value for p_c to 0. Additionally, p_i will provide the new attestation certificate of p_c and the signatures from the collected RecoveryCommits to its enclave that, on successful verification, will exchange the public enclave key of p_c; p_i can now process new messages of p_c. The recovering replica p_c will derive the first round it is allowed to propose a new vertex for from the RecoveryCommits; missed vertices are fetched using the backfilling mechanism.

V. Practical Evaluation

We investigate the impact of the NxB client model, scaling behavior of NxBFT with varying payload sizes and network conditions, and the behavior under faults in comparison to MinBFT [14, Sec. 4] and Chained-Damysus [13, Sec. 7]. We also conduct measurements of the recovery procedure.

A. Experiment Setup and Implementation

We use the ABCperf framework [42] as the basis for our experiments. ABCperf orchestrates the experiment, implements the full communication stack, handles performance measurement, and emulates client behavior. ABCperf ships a no-op application with random byte payloads and a MinBFT implementation; we add NxBFT and Chained-Damysus (own implementation based on [13, Sec. 7]) as atomic broadcast modules. We run our experiments on a cluster of 25 servers

of which we use one as an orchestrator, 20 for replicas (all Intel E-2288G, 64 GB main memory) and four for client emulation (AMD EPYC 9274F, 128 GB main memory) connected with 10 GBit interfaces. ABCperf can emulate an increased network round trip latency. ABCperf emulates 50% of the configured latency in each direction. The base network round trip latency of our cluster is $\sim 0.15\,\mathrm{ms}$. The actual number of replicas is equally distributed among the 20 replica hosts.

A single experiment run has the following pattern: the replicas are setup and perform the setup phase of the atomic broadcast algorithm. Then, we start to invoke client requests following a configurable constant frequency ("request rate"). A client has at most one open request. Now, whenever a request is to be made, the client emulator checks if it has an idle client, i.e., a client with no pending request, and issues the request according to the NxB (random selection with fallback, one valid answer suffices) or BFT (broadcast, $\lfloor \frac{n}{2} \rfloor + 1$ valid answers required) client models. To prevent crashes of the emulation and experiment framework due to resource exhaustion, we have to limit the overall number of clients to two million. After a pre-defined warm-up phase, the measurement phase starts. We measure the end-to-end latency of each request as the elapsed time, observed by a client, between sending a request and getting a valid answer. The achieved throughput is the number of successfully completed requests per second.

To ease implementation and, foremost, testing, the atomic broadcast state machines do not use any form of parallelism. We deem this to be reasonable as we are not interested in actual numbers that can be expected for production deployments but the relative behavior of the three algorithms at investigation. ABCperf's communication stack and client emulation, however, are highly parallelized. All enclave code is executed with Intel SGX [21]. Although we are aware of the limitations and weaknesses of SGX (e.g., [43]), in our experiments SGX serves as a viable way to account for the overhead of enclaved execution. To ensure a correct implementation of all algorithms, we employ a combination of unit and randomized integration tests.

The algorithm parameters that are fixed for all experiments are as follows and brought the best performance in a manual sensitivity analysis. MinBFT requires a block to contain at least 10k requests, Chained-Damysus and NxBFT require 100 requests. All algorithms propose at least every $0.1\,\mathrm{s}$ a new block. MinBFT uses an exponentially increasing timer for timeouts. Chained-Damysus uses the exponential increase and linear decrease as described in the paper [13, Sec. 3]. MinBFT and Chained-Damysus have an initial view timeout value of $3\,\mathrm{s}$. The fallback timeout of an NxB client is $5\,\mathrm{s}$.

B. NxB *Client Model Impact and Throughput Scaling*

To evaluate the impact of the client model and the algorithms' scaling behavior, we run NxBFT, Chained-Damysus, and MinBFT with request rates between $50\,\mathrm{kOp/s}$ and $800\,\mathrm{kOp/s}$ (step size: $50\,\mathrm{kOp/s}$) for $n \in \{3, 10, 20, 40\}$ replicas with the BFT and the NxB client model with eight repetitions of $60\,\mathrm{s}$ each. To prevent measuring the effects of network speed and cryptographic operations and, instead, investigate the protocols' overhead, the requests have a zero byte payload and no additional network latency is emulated.

TABLE I
MAXIMUM SUSTAINED THROUGHPUT (STEP SIZE $50\,\mathrm{kOp/s}$) IN DEPENDENCE ON CLIENT MODEL ($0\,\mathrm{B}$ PAYLOAD, $0.15\,\mathrm{ms}$ NETWORK ROUND TRIP LATENCY)

	MinBFT		**Damysus**		**NxBFT**	
	BFT	NxB	BFT	NxB	BFT	NxB
$n = 3$	150	150	100	350	150	350
$n = 10$	100	100	100	600	150	700
$n = 20$	50	50	50	150	100	450
$n = 40$	50	50	–	50	50	400

Table I shows the maximum request rates for which a stable system state was observed. Using the NxB client model, MinBFT achieves its peak performance for $n = 3$. Chained-Damysus' and NxBFT's throughput peaks at $n = 10$. When increasing n to 20, the throughput of Chained-Damysus decreases by a factor of ~ 4. The throughput of NxBFT drops by a factor of ~ 1.5. The table shows the maximum request rates for the BFT client as well. MinBFT achieves the same throughput for both client variants. Chained-Damysus' and NxBFT's throughput is at least halved when using the BFT client model. For $n = 40$, Chained-Damysus did not stabilize for a request rate of $50\,\mathrm{kOp/s}$.

Figure 3 shows the corresponding end-to-end latencies for the NxB client model. Before saturation, MinBFT achieves latencies of $0.1\,\mathrm{s}$ for $n < 20$ and $2.5\,\mathrm{s}$ for $n \geq 20$ (omitted for readability). For NxBFT and Chained-Damysus, the latency increases when increasing the number n of replicas. NxBFT achieves the best latency for $n = 3$ and a request rate of $50\,\mathrm{kOp/s}$ with $\sim 0.12\,\mathrm{s}$ and stays for all configurations, before saturating, below $1\,\mathrm{s}$. Chained-Damysus achieves the best latency for $n = 3$ and a request rate of $350\,\mathrm{kOp/s}$ with $\sim 0.01\,\mathrm{s}$. For $n \geq 20$, Chained-Damysus does not achieve latencies faster than $1\,\mathrm{s}$.

Figure 4 compares the achieved end-to-end latencies of the two client models under investigation. To increase readability, we select $n = 10$. Chained-Damysus and NxBFT achieve a speedup when using the BFT client model (factor ~ 24 for Chained-Damysus and factor ~ 6 for NxBFT). MinBFT does not benefit from using the BFT client in terms of latency.

In summary, we can observe that the NxB client model allows for load balancing when having a rotating leader or no leader at all but has a latency penalty. While in MinBFT the current leader has to handle *every* client connection, Chained-Damysus and NxBFT replicas handle, in expectation, $\frac{1}{n}$th of all client connections. Additionally, NxBFT shows the benefit of each message exchanged carrying client requests: while the workload in terms of messages exchanged and, thus, cryptographic operations performed increases when increasing n, NxBFT outperforms Chained-Damysus in terms of throughput. The reduced connection load allows Chained-Damysus to scale as well, but for $n > 10$ the increasing work to be done by the atomic broadcast eliminates those benefits without producing

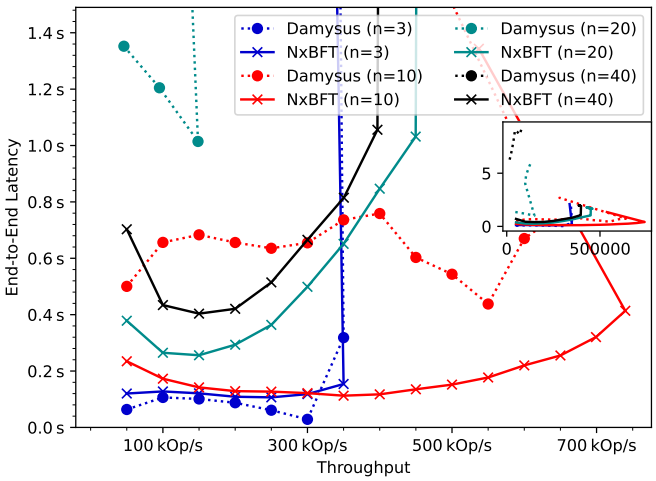


Fig. 3. End-to-end-latency of Chained-Damysus (dotted lines) and NxBFT (solid lines) for request rates between 50 kOp/s and 800 kOp/s using the NxB client model; each data point shown is an average of eight runs. The plot is limited on the y-axis; the full data is shown in the small window on the right. Table I indicates the corresponding maximum sustained request rates.

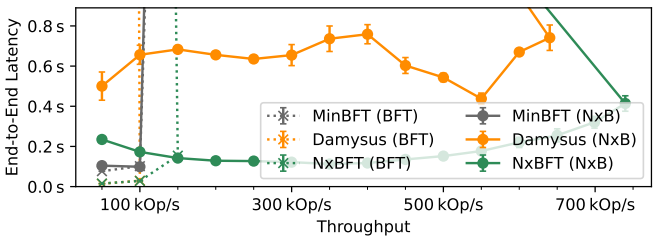


Fig. 4. Investigation of the NxB client latency penalty: end-to-end-latency of MinBFT, Chained-Damysus, and NxBFT for $n = 10$ and request rates between 50 kOp/s and 800 kOp/s using NxB (solid lines) and BFT (dotted lines) clients; each data point shown is an average of eight runs. The error bars indicate the 95% confidence interval. The plot is limited on the y-axis.

as much value as it is the case for NxBFT. However, due to the random selection of a replica by a NxB client, requests have to wait longer for block inclusion. Especially Chained-Damysus suffers significantly, as, in the worst case, a request has to wait $2n$ blocks for inclusion since Chained-Damysus requires a replica to be in charge for two blocks [44, App. B]. This maximum waiting time is reached when the client selected the replica which just proposed its second block. For MinBFT, the client choice has no impact as the static leader has to handle every request anyways.

We observe that the servers executing the replica code have on average $\sim 25\%$ total CPU usage but the process executing the atomic broadcast logic utilizes one CPU core permanently to the maximum. Suitable parallelization at each replica [7], [10], [45] may improve scaling behavior even further.

C. Impact of Network Properties and Payload

Graph-based and leader-rotating protocols are known for being highly dependent on the network latency between replicas. We investigate the impact of deployment properties by using random byte payloads of size 256 B [13], [9] and adding emulated network round trip latencies of 0 ms (same datacenter), 5 ms (same country), 35 ms (Europe), and 150 ms (World) to the physical datacenter round trip latency ($\sim$ 0.15 ms). We use the NxB client model. For each configuration and in a total of $\sim$ 2k experiments, we identify the maximum request rate that saturates the system but does not cause overload: After experiment start, the latency emulation is started and the system is stressed with a fixed request rate. We measure the achieved throughput in kOp/s for 240 s. We follow a rather conservative rejection strategy: If request rate and throughput do not match or the system shows any burst or backlog patterns, the request rate is not accepted. Please note that except for Chained-Damysus, a latency of 150 ms, and $n \geq 20$, we identify the maximum request rate with a granularity of 1 kOp/s. Additionally, we measure the time between two decisions (intermediate decision time, IDT): In MinBFT, a replica is able to decide whenever it collects $f + 1$ commits for a block (w/o faults one network round trip in expectation). In Chained-Damysus, a replica is able to decide a block when it is followed by two valid and consecutive blocks (w/o faults one network round trip in expectation). In NxBFT, a replica is able to decide when it finished a wave, i.e., it finished four consecutive broadcast rounds and the wave root selected by the common coin is part of the local DAG (w/o faults two network round trips in expectation). Table II lists the maximum request rates in bold font and kOp/s and the average IDT of 30 runs in normal font and milliseconds. IDT confidence intervals are all below 9% and left out.

MinBFT's throughput peaks for $n = 3$ and no emulated latency with 69 kOp/s and an IDT of 124 ms. MinBFT achieves the lowest throughput for $n = 40$ and a network latency of 150 ms. Chained-Damysus peaks for $n = 3$ and no latency with 223 kOp/s and an impressive IDT as small as 7 ms. For $n = 40$ and a network latency of 150 ms, the throughput drops to 0.2 kOp/s. The IDT increases with network latency and n. NxBFT shows peak performance for $n = 10$ and no latency with 584 kOp/s and 23 ms IDT. For network round trip latencies $\geq$ 5 ms, the throughput peaks for $n = 3$. For $n = 40$ and 150 ms network round trip latency, the throughput drops to 20 kOp/s. The IDT increases with network latency and n.

Although MinBFT requires at least one network round trip time between two decisions (two all-to-all broadcasts), MinBFT can stay below 150 ms IDT for 150 ms network round trip latency. This is due to MinBFT's pipelining: the leader proposes a new block whenever it has sufficient requests. Thus, as long as the system is not overloaded, MinBFT can benefit from ABCperf's parallelized network stack. This is in clear contrast to Chained-Damysus and NxBFT that both need quorums (which act as synchronization barriers) before being able to continue. The throughput of MinBFT, however,

TABLE II
MAXIMUM SUSTAINED THROUGHPUT (IN kOp/s, BOLD) AND CORRESPONDING INTERMEDIATE DECISION TIMES (IN ms) DEPENDING ON NETWORK SIZE AND LATENCY WITH 256 B PAYLOAD SIZE

Network Round Trip Latency	MinBFT				Chained-Damysus				NxBFT			
	$n=3$	$n=10$	$n=20$	$n=40$	$n=3$	$n=10$	$n=20$	$n=40$	$n=3$	$n=10$	$n=20$	$n=40$
0.15 ms	**69** 124	**28** 253	**18** 129	**11** 137	**223** 7	**132** 41	**64** 65	**36** 46	**439** 24	**584** 23	**268** 109	**178** 396
5 ms	**63** 121	**25** 184	**17** 143	**11** 149	**120** 28	**64** 69	**56** 177	**31** 123	**304** 69	**234** 247	**191** 269	**99** 486
35 ms	**49** 120	**20** 121	**13** 126	**11** 153	**29** 150	**20** 791	**14** 539	**3** 323	**91** 352	**55** 680	**45** 1 180	**28** 1 404
150 ms	**33** 111	**15** 117	**12** 123	**9** 137	**6** 1 641	**4** 1 580	**1.2** 1 218	**0.2** 766	**45** 1 299	**26** 1 924	**24** 1 625	**20** 1 680

cannot benefit: The network latency determines the minimum time a request has to wait for decision as the leader is required to collect a commit quorum. To counter this, the leader could propose bigger blocks which would increase the time a replica requires to validate the block and compute its commit. In both cases, too high request rates lead to maximally filled request queues at the side of the leader leading to dropped requests.

For network round trip latencies $\leq 5\,\text{ms}$, Chained-Damysus can take full advantage of the streamlined design, achieving IDTs clearly faster than MinBFT and NxBFT. As observed before (cf. Sec. V-B and Figure 4), Chained-Damysus severely suffers from the NXB client model when having big n. Block timeouts smaller than the selected $0.1\,\text{s}$ may circumvent this behavior. NxBFT shows the benefit of the NXB client model in terms of load balancing, significantly outperforming throughput in all configurations. While, for experiments with no payload, the load balancing allowed for higher throughput with increased n, both Chained-Damysus and NxBFT loose this ability. NxBFT, however, keeps the ability for no added latency. In fact, the load balancing increases the throughput but the increased cryptographic work, primarily driven by the payload size, masks the observed effect.

For Chained-Damysus and NxBFT, we observe that the IDT may decrease when increasing n, which indicates that the true maximum throughput may be higher. First, our step size of $1\,\text{kOp/s}$ may be too big. Second, larger networks may be more sensitive to scheduling and transmission variance when operating close to maximum performance.

D. Performance Under Faults

We investigate the algorithms' performance under faults for $n = 10$, a request rate of $50\,\text{kOp/s}$ 0B payloads using the NXB client model, and no emulated latency. An experiment lasts $400\,\text{s}$; $230\,\text{s}$ at the start are left for warm-up. After $20\,\text{s}$, replica 0 crashes, after $80\,\text{s}$, replica 3 crashes, and after $140\,\text{s}$, replica 9 crashes. We selected those replicas to crash as this pattern allows Chained-Damysus to make the most progress. To prevent unlimited increase of Damysus' timeouts, we do not crash four replicas (circumvention requires orthogonal consensus [46]). Figure 5 shows the average end-to-end latency of the experiment for ten repetitions.

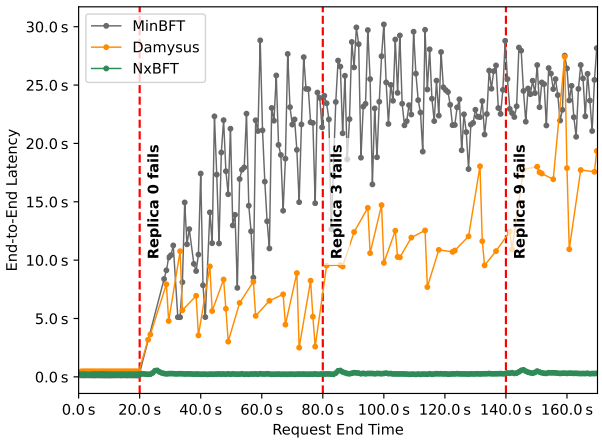


Fig. 5. End-to-end-latency of MinBFT, Chained-Damysus, and NxBFT for $n = 10$ and a request rate of $50\,\text{kOp/s}$ under faults; average of ten runs. The plot shows the average end-to-end latency for requests that are successfully answered (if any) in a time buckets of size $0.75\,\text{s}$. After $20\,\text{s}$, $80\,\text{s}$, and $140\,\text{s}$, a replica fails. In case of MinBFT, the crash of replica 0 causes a view change.

In case of MinBFT, replica 0 is the current leader and MinBFT is forced to perform a view-change. The throughput of MinBFT stalls until the inactivity of the leader is detected (i.e., $5\,\text{s}+3\,\text{s}$) and the view-change was successful. The new leader is not capable to cut the backlog down, yielding increased latencies for the remainder of the experiment. The crash of non-leader replicas (3 and 9) has no effect. For lower request rates, MinBFT would be capable of recovering fast response times within the experiment time window.

Without a recovery procedure, Chained-Damysus cannot recover the response times and each crashed replica increases the end-to-end latency clearly. Since Chained-Damysus requires a replica to be in charge for two views [44, App. B], the streamlined version of Damysus worsens this pattern.

As all client requests are uniformly distributed across all replicas, NxBFT's maximum achievable throughput degrades at most by $\frac{c}{n}$ for c crashed replicas. For the request rate of $50\,\text{kOp/s}$, our experiments show that the 7 remaining replicas

TABLE III
NXBFT RECOVERY TIMES DEPENDING ON HANDLED REQUEST COUNT (256 B PAYLOAD, 0.15 ms NETWORK ROUND TRIP LATENCY)

	125k	250k	500k
$n = 3$	$0.55 \pm 0.02\,\mathrm{s}$	$2.30 \pm 0.07\,\mathrm{s}$	$8.57 \pm 0.95\,\mathrm{s}$
$n = 10$	$0.29 \pm 0.00\,\mathrm{s}$	$0.89 \pm 0.05\,\mathrm{s}$	$3.71 \pm 0.36\,\mathrm{s}$
$n = 20$	$0.34 \pm 0.01\,\mathrm{s}$	$0.66 \pm 0.03\,\mathrm{s}$	$2.35 \pm 0.15\,\mathrm{s}$
$n = 40$	$0.43 \pm 0.01\,\mathrm{s}$	$0.79 \pm 0.02\,\mathrm{s}$	$2.45 \pm 0.07\,\mathrm{s}$

can handle the additional load: As soon as all clients identified a failed replica, the end-to-end latency recovers.

E. NxBFT Recovery Protocol

We investigate the recovery time of crashed NxBFT replicas for $n \in \{3, 10, 20, 40\}$ and request rates of $25\,\mathrm{kOp/s}$, $50\,\mathrm{kOp/s}$, and $100\,\mathrm{kOp/s}$ using the NXB client model. After $5\,\mathrm{s}$, a replica is crashed and recovered. In this time, the system handled a total of $125\mathrm{k}$, $250\mathrm{k}$, and $500\mathrm{k}$ client requests. We measure the time it takes for the crashed replica to be able to propose valid messages. Table III lists the average recovery times with the 95% confidence interval for 30 repetitions. Our experiment shows that the recovery time is primarily influenced by the number of vertices a replica sent before it crashed. The number of messages is influenced by the time until the replica crashes, the request rate and the number of replicas n. The request rate determines how many messages a crashed replica produced, n controls load balancing in the NXB client model. Regular checkpointing would limit the graph history and, thus, speedup recovery significantly.

VI. DISCUSSION AND FUTURE WORK

NxBFT's performance achievements are built upon the assumptions regarding possible attacker behavior: The NXB operating model allows reducing load between clients and replicas. Load balancing in general scales with the number of replicas, however, the quadratic overhead of the broadcast layer eventually becomes the dominating factor limiting the maximum number of replicas. The missing resilience against attacks from operators on clients can be mitigated by the application signing responses. In case of such an attack, a client can prove its correct behavior and the client state can be corrected. Instead, one could also deploy the application to a TEE as well or use off-loading technologies like CART [47].

Operators and other third parties could attack the TEE. Such an attack would not leak any additional application data in comparison to classic BFT SMR and requires, e.g., secure multi-party computation for mitigation. On the consensus layer, however, knowing coin values beforehand increases the chance for successful censorship [10], and on the broadcast layer equivocation could bring the system to a halt. We consider the probability of attacks on consensus and broadcast to be negligible in the intended deployment scenarios.

Setup and recovery require some form of synchrony to achieve liveness. We argue that, while being unconventional for secure distributed systems research, the proposed model and assumptions are well-suited for practical deployments. Federations and consortia built upon NxBFT will operate a highly resilient common service. Maintenance and recovery windows can be agreed upon beforehand and peers can recover without interrupting the service. While then formally resulting in a partially synchronous system, NxBFT operates asynchronously once set up or recovered.

As future work, garbage collection is required: With thousands of operations per second, the required memory for storing the graph grows quickly. A garbage collection that actually deletes information, e.g., based on decided waves as in [10], however, is in conflict with asynchrony *and* the backfilling strategy described above: If a replica learns rather "late" (in relation to other replicas) that it misses some information, other replicas may have reached consensus and deleted this information already. The wave counter is a local variable and gives no sufficient information on the synchronization progress with other replicas. Scheduled maintenance time slots, as discussed above, "trivially" facilitate garbage collection: Our recovery protocol can be extended to derive a full system checkpoint, but, still, depending on all replicas to cooperate. From there on, implementing a voting-based reconfiguration scheme becomes easy as well.

The wave length of four rounds in NxBFT is required to prove the atomic broadcast's agreement property [12, Proposition 3] based on the *get core* property [11, Lemma 4]. The get core property is required to derive an expectation value for a correct replica being able to commit a wave, i.e., a liveness guarantee. Related work [10], [25] proposed to shorten the wave length to three rounds. However, with a wave length of only three rounds and a fault tolerance of $n > 2f$, the get core property does not hold anymore (see Appendix B).

As our results confirm, partially synchronous protocols typically trade good latency for smaller throughput and overhead in the case of faults. Within the last two years, parallel work optimized this trade-off by adopting ideas of leaderless and DAG-based protocols in partial synchrony [48], [49], [50], [51]. Furthermore, researchers achieved impressive performance results for asynchronous protocols in non-Byzantine environments [52]. Based on their promising results, we consider the investigation of the integration of TEEs and fault model relaxations to be a promising line of future research.

VII. CONCLUSION

We presented and evaluated NxBFT, a full-fledged TEE-based State Machine Replication approach in the "Not eXactly Byzantine" operating model. NxBFT bases its safety and liveness properties of normal case operation solely on the assumption of asynchrony and is, therefore, highly resilient. Setup and recovery, however, require some form of synchrony. Using graph-based agreement and load balancing, NxBFT demonstrates competitive throughput performance while maintaining a reasonable latency trade-off. In our opinion, further optimization and increased deployability of DAG-based approaches that use TEEs represent interesting future work.

ACKNOWLEDGMENTS

This work was supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF). We would like to thank Tilo Spannagel for his expertise during the implementation work and Oliver Stengele for his feedback on setup and recovery protocols.

REFERENCES

[1] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Comput. Surv.*, vol. 22, no. 4, pp. 299–319, 1990. [Online]. Available: https://doi.org/10.1145/98163.98167

[2] J. D. Preece and J. M. Easton, "Blockchain technology as a mechanism for digital railway ticketing," in *2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019*, 2019, pp. 3599–3606. [Online]. Available: https://doi.org/10.1109/BigData47090.2019.9006293

[3] M. Leinweber, N. Kannengießer, H. Hartenstein, and A. Sunyaev, "Leveraging Distributed Ledger Technology for Decentralized Mobility-as-a-Service Ticket Systems," in *Towards the New Normal in Mobility: Technische und betriebswirtschaftliche Aspekte*. Springer Fachmedien Wiesbaden, 2023, pp. 547–567. [Online]. Available: https://doi.org/10.1007/978-3-658-39438-7_32

[4] M. B. Mollah, J. Zhao, D. Niyato, K. Lam, X. Zhang, A. M. Y. M. Ghias, L. H. Koh, and L. Yang, "Blockchain for future smart grid: A comprehensive survey," *IEEE Internet Things J.*, vol. 8, no. 1, pp. 18–43, 2021. [Online]. Available: https://doi.org/10.1109/JIOT.2020.2993601

[5] M. Raskin and D. Yermack, "Chapter 22: Digital currencies, decentralized ledgers and the future of central banking," in *Research Handbook on Central Banking*. Edward Elgar Publishing, 2018. [Online]. Available: https://doi.org/10.4337/9781784719227

[6] P. K. Ozili, "Central bank digital currency research around the world: a review of literature," *Journal of Money Laundering Control*, vol. 26, no. 2, pp. 215–226, 2023. [Online]. Available: https://doi.org/10.1108/jmlc-11-2021-0126

[7] J. Behl, T. Distler, and R. Kapitza, "Hybrids on steroids: SGX-based high performance BFT," in *Proceedings of the Twelfth European Conference on Computer Systems, EuroSys 2017, Belgrade, Serbia, April 23-26, 2017*. ACM, 2017, pp. 222–237. [Online]. Available: https://doi.org/10.1145/3064176.3064213

[8] W. Wang, S. Deng, J. Niu, M. K. Reiter, and Y. Zhang, "ENGRAFT: enclave-guarded Raft on Byzantine faulty nodes," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*. ACM, 2022, pp. 2841–2855. [Online]. Available: https://doi.org/10.1145/3548606.3560639

[9] J. Decouchant, D. Kozhaya, V. Rahli, and J. Yu, "Oneshot: View-adapting streamlined BFT protocols with trusted execution environments," in *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2024, San Francisco, CA, USA, May 27-31, 2024*. IEEE, 2024, pp. 1022–1033. [Online]. Available: https://doi.org/10.1109/IPDPS57955.2024.00095

[10] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus," in *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*. ACM, 2022, pp. 34–50. [Online]. Available: https://doi.org/10.1145/3492321.3519594

[11] M. Leinweber and H. Hartenstein, "Brief announcement: Let it TEE: asynchronous Byzantine atomic broadcast with n $\geq$ 2f+1," in *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy*, ser. LIPIcs, vol. 281. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 43:1–43:7. [Online]. Available: https://doi.org/10.4230/LIPIcs.DISC.2023.43

[12] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is DAG," in *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*. ACM, 2021, pp. 165–175. [Online]. Available: https://doi.org/10.1145/3465084.3467905

[13] J. Decouchant, D. Kozhaya, V. Rahli, and J. Yu, "DAMYSUS: streamlined BFT consensus leveraging trusted components," in *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*. ACM, 2022, pp. 1–16. [Online]. Available: https://doi.org/10.1145/3492321.3519568

[14] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Veríssimo, "Efficient Byzantine fault-tolerance," *IEEE Trans. Computers*, vol. 62, no. 1, pp. 16–30, 2011. [Online]. Available: https://doi.org/10.1109/TC.2011.221

[15] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, 2002. [Online]. Available: https://doi.org/10.1145/571637.571640

[16] M. Correia, N. F. Neves, and P. Veríssimo, "How to tolerate half less one Byzantine nodes in practical distributed systems," in *23rd International Symposium on Reliable Distributed Systems (SRDS 2004), 18-20 October 2004, Florianpolis, Brazil*. IEEE Computer Society, 2004, pp. 174–183. [Online]. Available: https://doi.org/10.1109/RELDIS.2004.1353018

[17] B. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, "Attested append-only memory: making adversaries stick to their word," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*. ACM, 2007, pp. 189–204. [Online]. Available: https://doi.org/10.1145/1294261.1294280

[18] M. Correia, G. S. Veronese, and L. C. Lung, "Asynchronous Byzantine consensus with 2f+1 processes," in *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*. ACM, 2010, pp. 475–480. [Online]. Available: https://doi.org/10.1145/1774088.1774187

[19] A. Clement, F. Junqueira, A. Kate, and R. Rodrigues, "On the (limited) power of non-equivocation," in *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*. ACM, 2012, pp. 301–308. [Online]. Available: https://doi.org/10.1145/2332432.2332490

[20] M. F. Madsen and S. Debois, "On the subject of non-equivocation: Defining non-equivocation in synchronous agreement systems," in *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*. ACM, 2020, pp. 159–168. [Online]. Available: https://doi.org/10.1145/3382734.3405731

[21] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptol. ePrint Arch.*, p. 86, 2016. [Online]. Available: http://eprint.iacr.org/2016/086

[22] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda, "TrInc: Small trusted hardware for large distributed systems," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA*. USENIX Association, 2009, pp. 1–14. [Online]. Available: http://www.usenix.org/events/nsdi09/tech/full_papers/levin/levin.pdf

[23] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Scalable Byzantine consensus via hardware-assisted secret sharing," *IEEE Trans. Computers*, vol. 68, no. 1, pp. 139–151, 2019. [Online]. Available: https://doi.org/10.1109/TC.2018.2860009

[24] X. Fu, H. Wang, P. Shi, and X. Zhang, "Teegraph: A blockchain consensus algorithm based on TEE and DAG for data sharing in IoT," *J. Syst. Archit.*, vol. 122, p. 102344, 2022. [Online]. Available: https://doi.org/10.1016/j.sysarc.2021.102344

[25] S. Xie, D. Kang, H. Lyu, J. Niu, and M. Sadoghi, "Fides: Scalable censorship-resistant DAG consensus via trusted components," *CoRR*, vol. abs/2501.01062, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2501.01062

[26] M. K. Aguilera, N. Ben-David, R. Guerraoui, A. Murat, A. Xygkis, and I. Zablotchi, "uBFT: Microsecond-scale BFT using disaggregated memory," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023*. ACM, 2023, pp. 862–877. [Online]. Available: https://doi.org/10.1145/3575693.3575732

[27] S. Gupta, S. Rahnama, S. Pandey, N. Crooks, and M. Sadoghi, "Dissecting BFT consensus: In trusted components we trust!" in *Proceedings of the Eighteenth European Conference on Computer Systems, EuroSys 2023, Rome, Italy, May 8-12, 2023*. ACM, 2023, pp. 521–539. [Online]. Available: https://doi.org/10.1145/3552326.3587455

[28] A. Bessani, M. Correia, T. Distler, R. Kapitza, P. E. Veríssimo, and J. Yu, "Vivisecting the dissection: On the role of trusted components in BFT protocols," *CoRR*, vol. abs/2312.05714, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2312.05714

[29] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna,*

Austria, October 24-28, 2016. ACM, 2016, pp. 31–42. [Online]. Available: https://doi.org/10.1145/2976749.2978399

[30] L. Baird and A. Luykx, "The hashgraph protocol: Efficient asynchronous BFT for high-throughput distributed ledgers," in *2020 International Conference on Omni-layer Intelligent Systems, COINS 2020, Barcelona, Spain, August 31 - September 2, 2020*. IEEE, 2020, pp. 1–7. [Online]. Available: https://doi.org/10.1109/COINS49042.2020.9191430

[31] R. Ladelsky and R. Friedman, "On quorum sizes in DAG-based BFT protocols," *CoRR*, vol. abs/2504.08048, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2504.08048

[32] S. Yandamuri, I. Abraham, K. Nayak, and M. K. Reiter, "Communication-efficient BFT using small trusted hardware to tolerate minority corruption," in *26th International Conference on Principles of Distributed Systems, OPODIS 2022, December 13-15, 2022, Brussels, Belgium*, ser. LIPIcs, vol. 253. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 24:1–24:23. [Online]. Available: https://doi.org/10.4230/LIPIcs.OPODIS.2022.24

[33] T. Distler, "Byzantine fault-tolerant state-machine replication from a systems perspective," *ACM Comput. Surv.*, vol. 54, no. 1, pp. 24:1–24:38, 2021. [Online]. Available: https://doi.org/10.1145/3436728

[34] H. Howard, F. Alder, E. Ashton, A. Chamayou, S. Clebsch, M. Costa, A. Delignat-Lavaud, C. Fournet, A. Jeffery, M. Kerner, F. Kounelis, M. A. Kuppe, J. Maffre, M. Russinovich, and C. M. Wintersteiger, "Confidential consortium framework: Secure multiparty applications with confidentiality, integrity, and high availability," *Proc. VLDB Endow.*, vol. 17, no. 2, pp. 225–240, 2023. [Online]. Available: https://doi.org/10.14778/3626292.3626304

[35] J. Niu, X. Wen, G. Wu, S. Liu, J. Yu, and Y. Zhang, "Achilles: Efficient TEE-assisted BFT consensus via rollback resilient recovery," in *Proceedings of the Twentieth European Conference on Computer Systems*. ACM, 2025, pp. 193–210. [Online]. Available: https://doi.org/10.1145/3689031.3717457

[36] C. Cachin, R. Guerraoui, and L. E. T. Rodrigues, *Introduction to Reliable and Secure Distributed Programming*, 2nd ed. Springer, 2011. [Online]. Available: https://doi.org/10.1007/978-3-642-15260-3

[37] M. C. Pease, R. E. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. ACM*, vol. 27, no. 2, pp. 228–234, 1980. [Online]. Available: https://doi.org/10.1145/322186.322188

[38] M. J. Fischer, N. A. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," in *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 21-23, 1983, Colony Square Hotel, Atlanta, Georgia, USA*. ACM, 1983, pp. 1–7. [Online]. Available: https://doi.org/10.1145/588058.588060

[39] M. Ben-Or, "Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract)," in *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*. ACM, 1983, pp. 27–30. [Online]. Available: https://doi.org/10.1145/800221.806707

[40] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the Gap-Diffie-Hellman-Group signature scheme," in *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, ser. Lecture Notes in Computer Science, vol. 2567. Springer, 2003, pp. 31–46. [Online]. Available: https://doi.org/10.1007/3-540-36288-6_3

[41] M. Barbaraci, N. Schmid, O. Alpos, M. Senn, and C. Cachin, "Thetacrypt: A distributed service for threshold cryptography," *CoRR*, vol. abs/2502.03247, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2502.03247

[42] T. Spannagel, M. Leinweber, A. Castro, and H. Hartenstein, "ABCperf: Performance evaluation of fault tolerant state machine replication made simple: Demo abstract," in *Proceedings of the 24th International Middleware Conference Demos, Posters and Doctoral Symposium*. ACM, 2023, pp. 35–36. [Online]. Available: https://doi.org/10.1145/3626564.3629101

[43] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. USENIX Association, 2018, pp. 991–1008. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

[44] J. Decouchant, D. Kozhaya, V. Rahli, and J. Yu. (2022) DAMYSUS: streamlined BFT consensus leveraging trusted components (extended version). [Online]. Available: https://github.com/vrahli/damysus/blob/main/doc/damysus-extended.pdf

[45] A. Einarsdóttir, "Enabling parallel voting in streamlined consensus protocols," Master's thesis, Delft University of Technology, 2024. [Online]. Available: https://resolver.tudelft.nl/uuid:8b40ec25-155e-4c92-b746-290baacab577

[46] D. Malkhi and O. Naor. (2022) The latest view on view synchronization. [Online]. Available: https://blog.chain.link/view-synchronization/

[47] A. Heß, F. J. Hauck, and E. Meißner, "Consensus-agnostic state-machine replication," in *Proceedings of the 25th International Middleware Conference*. ACM, 2024, p. 13. [Online]. Available: https://doi.org/10.1145/3652892.3700776

[48] D. Malkhi, C. Stathakopoulou, and M. Yin, "BBCA-CHAIN: one-message, low latency BFT consensus on a DAG," in *Financial Cryptography and Data Security*, 2025, pp. 51–73. [Online]. Available: https://doi.org/10.1007/978-3-031-78676-1_4

[49] N. Shrestha, R. Shrothrium, A. Kate, and K. Nayak, "Sailfish: Towards improving latency of DAG-based BFT," in *2025 IEEE Symposium on Security and Privacy (SP)*, 2025. [Online]. Available: https://doi.org/10.1109/SP61157.2025.00021

[50] N. Giridharan, F. Suri-Payer, I. Abraham, L. Alvisi, and N. Crooks, "Autobahn: Seamless high speed BFT," in *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles, SOSP 2024, Austin, TX, USA, November 4-6, 2024*. ACM, 2024, pp. 1–23. [Online]. Available: https://doi.org/10.1145/3694715.3695942

[51] B. Arun, Z. Li, F. Suri-Payer, S. Das, and A. Spiegelman, "Shoal++: High throughput DAG BFT can be fast and robust!" in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, 2025, pp. 813–826. [Online]. Available: https://www.usenix.org/conference/nsdi25/presentation/arun

[52] B. Wang, S. Liu, H. Dong, X. Wang, W. Xu, J. Zhang, P. Zhong, and Y. Zhang, "Bandle: Asynchronous state machine replication made efficient," in *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*. ACM, 2024, pp. 265–280. [Online]. Available: https://doi.org/10.1145/3627703.3650091

[53] H. Attiya and J. L. Welch, *Distributed Computing - Fundamentals, Simulations, and Advanced Topics*, 2nd ed., ser. Wiley series on parallel and distributed computing. Wiley, 2004. [Online]. Available: https://doi.org/10.1002/0471478210

APPENDIX

A. Correctness Arguments

1) Backfilling-Based Reliable Broadcast: In the following, we will argue that the backfilling mechanism as described in Sec. IV-A3 transforms the best effort broadcast to a reliable broadcast. Since we use the signature service to implement a FIFO broadcast with non-equivocation, we get 'no duplication', 'integrity', and 'consistency' as reliable broadcast properties [36, Module 3.12] for free. 'Validity' and 'totality' properties require that if a correct replica broadcasts or delivers a message, every other correct replica will deliver this message; both are based on the backfilling mechanism: Correct replicas will only use vertices as edges that they already added to their DAG. This behavior implies that those edges are valid vertices with a completely known ancestry. If a replica now receives a vertex for which it does not know an edge, this is due to two reasons: (1) the edge is from a correct replica and the corresponding vertex was not yet delivered by the asynchronous channel or (2) the edge is from a faulty replica that committed a send omission fault. In case (1), our replica will receive the edge eventually and validity and totality will be preserved. In case (2), if at least one correct replica received the vertex and was able to add it to the DAG, no matter if itself

used the vertex as an edge or not, the replica will be able to answer any vertex request, thus, fulfilling totality. If no correct replica received the vertex, no correct replica will deliver the vertex, thus, not breaking totality.

2) Common Coin: The XOR construction is a variant of the straightforward $t = n$ secret sharing; the resulting seed is kept confidential by secure communication and the TEE. The enclave will not reveal a toss unless at least one correct replica requests a toss. A correct replica will always toss a coin when it completes the last round of a wave and proposed its own vertex for it, enabling every correct replica to learn the coin value as soon as possible. Note that it is not necessary to use the TEE-based signatures, as we do, to achieve the desired properties, but it makes implementation easier.

3) Recovery Protocol: Obviously, the proposed recovery protocol is only live if all replicas are eventually reactive: Byzantine faulty replicas can stop the recovery procedure by not sending a RecoveryProposal at any time (or an invalid one). In the following, we will argue that the proposed recovery protocol is Byzantine safe in an asynchronous environment. To prevent that the recovery protocol enables equivocation, the enclaves of all correct replicas must replace the recovering replica p_c's enclave key with the same new enclave public key (i.e., agreement on the p_c's new attestation certificate). To this end, a correct replica collects $\lfloor \frac{n}{2} \rfloor + 1 > r$ valid RecoveryCommits that it passes to its enclave. The enclave verifies the signatures and only accepts if all $\lfloor \frac{n}{2} \rfloor + 1$ RecoveryCommits are correctly signed. As a correct await $\lfloor \frac{n}{2} \rfloor + 1 > f$ RecoveryCommits, a correct replica will receive at least one RecoveryCommit created by a correct replica. Moreover, as a correct replica waits for n RecoveryProposals with consistent attestation certificates before sending a RecoveryCommit, the enclave logic can deduce agreement on the supplied attestation certificate and rule out equivocation of the new enclave public key of the recovering replica. Since RecoveryProposal and RecoveryCommit result in a reliable broadcast of proposals, all correct replicas apply the exact same vertices of a recovering replica p_c when they identify the longest valid vertex chain. The common coin is recovered by importing the enclave state, including the seed established during setup, and fast-forwarding the PRNG state to be directly "before" the next toss the replica would have made when not crashing. Thus, the coin implementation will only reveal new coins whenever the replica can prove the needed progress of the system (with or without the replica's contribution) to its enclave. As the common coin uses the public enclave keys, the agreement on the attestation certificates ensures that a Byzantine faulty replica cannot learn coin values in advance.

B. Impossibility of Get Core with a Reduced Wave Length

Graph-based atomic broadcast protocols of the DAG-Rider family rely for their liveness on "connectivity guarantees": After a certain amount of rounds, there is the guarantee that *all* valid vertices of a round $r' > r$ have a path to a subset of the vertices of round r. A vertex is valid if it is correctly signed, it links to at least $2f+1$ (for BFT) or $f+1$ (for hybrid fault models) vertices of the previous round, and its ancestry is known, valid, and part of the local DAG. The original DAG-Rider approach [12] requires that when selecting four consecutive and completed rounds r_1, r_2, r_3, r_4 *all* vertices of V_4 of round r_4 have a path to the *same* subset V_{core} of round r_1. Furthermore, it is required that this common subset, or common core, is at least of size $|V_{\text{core}}| \geq 2f+1$. This property can be proven by using the *get core* scheme developed by Attiya and Welch [53, Sec. 14.3.1] (sometimes also known as "Gather"). TEE-Rider showed that the property still holds when using a wave length of four, i.e., four consecutive rounds, a size of $|V_{\text{core}}| \geq f + 1$, and a fault tolerance of $n > 2f$ if it can be assumed that each possible malicious action of an attacker can be reduced to an omission fault [11, Lemma 4].

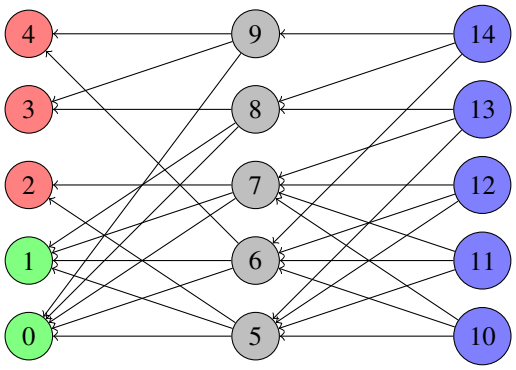


Fig. 6. Counter example for the *get core* property with $n = 5$ and a wave length of three rounds. All blue vertices (vertices of round 3) should have a path to a shared subset of round 1 (red and green vertices) of size 3. The maximum shared subset, however, is of size 2 (green vertices).

Intuition would suggest that by using TEEs the wave length could be reduced in a similar way as TEEs save a round of communication for PBFT-like protocols [14]. In Figure 6, we constructed a counter example for a wave length of 3 and $n = 5$ breaking *get core*. The *get core* property requires that each valid vertex links to at least 3 vertices of the previous round. Every vertex is valid and its ancestry is part of the graph. However, the blue vertices (i.e., the vertices of round 3) only share a subset of round 1 with size 2 (the green vertices labeled 0 and 1). If *get core* would hold for three rounds, there should be a shared subset of size 3. This is the case if a fourth round of valid vertices connecting to the blue vertices is added. If a liveness proof does not rely on *get core*, it seems possible to shorten the wave length. However, to the best of our knowledge, no such liveness proof for an asynchronous, graph-based, and Byzantine fault tolerant atomic broadcast protocol with a fault tolerance of $n > 2f$ is known yet.