

# Chasing Lightspeed Consensus: Fast Wide-Area Byzantine Replication with MERCURY

Christian Berger  
University of Passau  
Passau, Germany  
cb@sec.uni-passau.de

Lívio Rodrigues  
LASIGE, Faculdade de Ciências,  
Universidade de Lisboa  
Lisboa, Portugal  
livio.g.j.rodrigues@gmail.com

Hans P. Reiser  
Reykjavik University  
Reykjavik, Iceland  
hansr@ru.is

Vinícius Cogo  
LASIGE, Faculdade de Ciências,  
Universidade de Lisboa  
Lisboa, Portugal  
vvcogo@ciencias.ulisboa.pt

Alysson Bessani  
LASIGE, Faculdade de Ciências,  
Universidade de Lisboa  
Lisboa, Portugal  
anbessani@ciencias.ulisboa.pt

## ABSTRACT

Blockchain technology sparked renewed interest in planetary-scale Byzantine fault-tolerant (BFT) state machine replication (SMR). While recent works predominantly focused on improving the scalability and throughput of these protocols, few of them addressed latency. We present MERCURY, a novel transformation to autonomously optimize the latency of quorum-based BFT consensus. MERCURY employs a dual resilience threshold that enables faster transaction ordering when the system contains few faulty replicas. MERCURY allows forming *compact quorums* that substantially accelerate consensus using a smaller resilience threshold. Nevertheless, MERCURY upholds standard SMR safety and liveness guarantees with optimal resilience, thanks to its judicious use of a dual operation mode and BFT forensics techniques. Our experiments spread tens of replicas across continents and reveal that MERCURY can order transactions with finality in less than 0.4s, half the time of a PBFT-like protocol (optimal in terms of number of communication steps and resilience) in the same network. Furthermore, MERCURY matches the latency of running its base protocol on theoretically optimal internet links (transmitting at 67% of the speed of light).

## CCS CONCEPTS

• **Computer systems organization** → Dependable and fault-tolerant systems and networks; • **General and reference** → Performance; • **Computing methodologies** → Distributed algorithms.

## KEYWORDS

Byzantine fault tolerance, consensus, low latency, adaptiveness, weighted replication, BFT forensics, client speculation, blockchain

## ACM Reference Format:

Christian Berger, Lívio Rodrigues, Hans P. Reiser, Vinícius Cogo, and Alysson Bessani. 2024. *Chasing Lightspeed Consensus: Fast Wide-Area Byzantine Replication with MERCURY*. In *25th International Middleware Conference (MIDDLEWARE '24)*, December 2–6, 2024, Hong Kong, Hong Kong. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3652892.3700756>

## 1 INTRODUCTION

State machine replication (SMR) is an approach to tolerate faults in distributed systems by coordinating client interactions with a set of  $n$  independent replicas [64]. Recently, many scalable (BFT) SMR protocols have emerged for blockchain infrastructures, such as HotStuff [77], SBFT [42], Tendermint [24], Mir-BFT [71], Red-BellyBC [30], Kauri [59], and the Dumbo family [38, 53]. These protocols employ some dynamically elected leader [11, 24, 42, 59, 73, 77], use multiple leaders [2, 71], or are leaderless [6, 30, 38, 78].

Nevertheless, the consensus in all these cases requires communication involving a quorum of replicas under the assumption that the adversary controls no more than a *fixed resilience threshold* of  $t = \lfloor \frac{n-1}{3} \rfloor$  replicas. Often, the quorum size for proceeding to the next protocol stage depends on this threshold. For instance, a Byzantine  $t$ -dissemination quorum requires  $\lceil \frac{n+t+1}{2} \rceil$  replicas [54]. This size equals roughly  $\frac{2}{3}$  of all replicas if an *optimal* resilience threshold is used.

Two challenges arise in optimizing end-to-end client latency for geo-replicated or planetary-scale systems like permissioned blockchains (e.g., [5, 30]) with tens of nodes distributed worldwide. First, theoretical lower bounds define that at least three communication steps are required for reaching consensus without giving up the optimal resilience [51, 57]. Second, there are physical limits that bound link transmission speed to a fraction of the speed of light (e.g.,  $0.67c$  [49]). Contrarily, improving throughput is a much more popular objective that can be achieved by parallelizing/distributing tasks (e.g., [30, 78]), improving bandwidth usage (e.g., [11, 72]), or simply by using a better infrastructure (e.g., better network links). Nonetheless, globally ordering transactions in a fraction of a second is still far from reality for existing systems [40], making near-instantaneous confirmation of transactions a missing usability feature of blockchains.

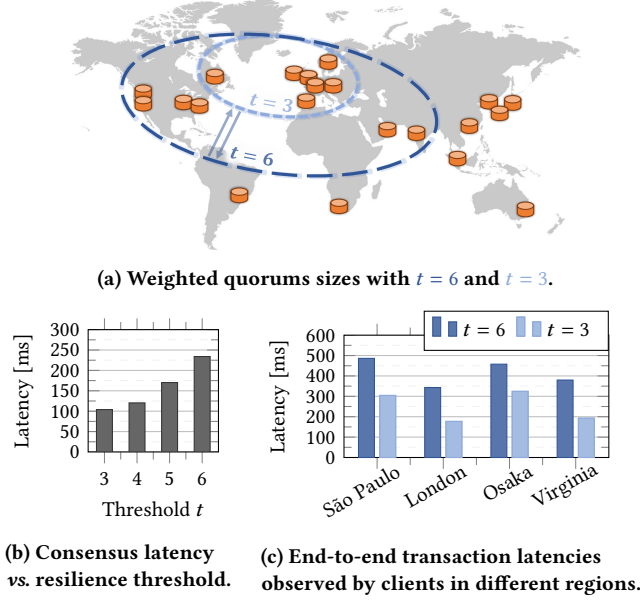
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

*Middleware '24, December 02–06, 2024, Hong Kong, China*

© 2024 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0623-3/24/12.

<https://doi.org/10.1145/3652892.3700756>



**Figure 1: Weighted quorums composition and resulting BFT SMR latency for different resilience thresholds ( $t$ ) in our  $n = 21$  setup (see details in §6).**

### 1.1 Smaller Quorums for Better Latency

We advocate that using smaller quorums of closer replicas can significantly decrease SMR latency [48, 70]. The challenge lies in ensuring these faster, smaller quorums intersect sufficiently with all other quorums of the system. Such quorums can be built using *weighted replication*, giving faster replicas more voting power.

Figure 1 illustrates how a geo-replicated system can progress faster with smaller quorums. It considers a weighted system [70] with  $n = 21$  replicas dispersed across all 21 AWS regions (see Figure 1a). When configured for maximum resilience, this system tolerates up to  $t = 6$  Byzantine replicas with  $\Delta = 2$  spare replicas. The smallest weighted consensus quorum  $Q_6$  contains 13 replicas (see §2.2 for details on the calculations), only one replica less than using non-weighted replication. Alternatively, when configured for  $t = 3$  failures, the smallest weighted quorum  $Q_3$  contains only 7 replicas, with  $\Delta = 11$ . This quorum can comprise the nearest replicas that can exchange votes with each other faster, accelerating the consensus protocol stages (see Figure 1b) and resulting in end-to-end latency improvements around the globe (see Figure 1c).

### 1.2 Challenges and the Big Picture

The problem in using a lower resilience bound  $t_{fast} < t$  is that an adversary controlling  $f$  replicas (with  $t_{fast} < f \leq t$ ) can equivocate. It means the adversary can convince *two correct* replicas to decide *different batches* of transactions for the same consensus instance, as quorums for the lower threshold  $t_{fast}$  do not necessarily overlap in at least one correct replica.

A key insight of our work is the innovative use of *BFT protocol forensics* [65, 66] as a defense against Byzantine attackers rather than post-incident forensics. Traditional BFT protocol forensics relies on clients to detect conflicting values based on replicas’ logged

messages and pinpoint equivocating replicas. In our solution, all correct replicas share the responsibility for detecting faults, enabling autonomous detection and removal of equivocating parties. In a system tolerating up to  $t_{fast}$  faulty replicas, audits detect agreement violations and identify  $t_{fast} + 1$  faulty replicas if there are no more than  $2t_{fast}$  faulty replicas [66]. Using  $t_{fast} = \lceil \frac{t}{2} \rceil$  ensures audit support for up to  $t$  faulty replicas. Our system recovers from violations by expelling the detected equivocators and rolling back the divergent decisions of correct replicas to a consistent state. Continuous auditing is important not only as a recovery mechanism but also as a *deterrent to attacks* since any perpetrator is identified and expelled from the system.

The ability to roll back decisions on replicas may lead to transaction outcomes observed by clients being undone, affecting transaction finality and durability. Consequently, we must modify the matching replies requirements on clients to ensure they can know when an operation is *finalized* in the system, ensuring consistency (i.e., linearizability [45]) and liveness as in standard SMR [25]. Another challenge we address is deriving the exact number of matching replies a client needs to expect to finalize a submitted transaction when consensus agreement violations are possible.

Minimizing consensus latency but letting clients wait for *more-than-usual* replies from all over the world to preserve linearizability counteracts our goal of reducing the end-to-end request latency. For this reason, we extend the BFT SMR programming model with *Byzantine correctables*, empowering clients with incremental consistency guarantees [41] and enabling the early confirmation of submitted transactions.

Lastly, we need to consider *SMR liveness*, as requests issued by correct clients need to be eventually completed. This property can be endangered if the protocol operates with an optimistic threshold  $t_{fast}$  and there are  $f > t_{fast}$  Byzantine replicas that stay silent, i.e., do not reply to the client or participate in consensus quorums. To deal with this scenario, we employ the idea of having two modes of operation: If the system blocks or equivocates, we stop the execution of MERCURY’s fast mode and resume the execution of the standard protocol tolerating  $t$  Byzantine replicas.

Our experimental evaluation with up to 51 replicas around the globe shows that MERCURY can order transactions with finality in less than 0.4s, which is half of the time required for BFT-SMART [19] (which implements a PBFT-like protocol) in the same network. Interestingly, our observed latencies are close to the theoretical optimum for BFT-SMART, considering the physical location of replicas and links transmitting at  $\frac{2}{3}$  of the speed of light, which is accepted as the upper bound on data transmission speed for the internet [22, 49]. Further, we achieve consensus latencies 4× smaller than recent results reported in state-of-the-art protocols targeting low-latency in similar environments (e.g., [12, 53]).

### 1.3 Contributions

MERCURY shows how to obtain a threshold-adaptive BFT protocol that strives for continuous self-optimization during runtime by tuning the resilience threshold utilized in consensus quorums. This protocol significantly reduces the latency in planetary-scale BFT SMR in the expected common case with few failures. In summary, we claim the following contributions:

- We study how to detect malicious behavior under an underestimated threshold  $t_{fast}$  by periodically auditing the system, removing faulty replicas, and repairing the correct replicas' state after an agreement violation.
- We show that it is possible to preserve the usual SMR guarantees, *linearizability* and *termination*, under the larger resilience threshold  $t$ , even if the agreement quorums are formed using a smaller threshold  $t_{fast} < t$ .
- We introduce *Byzantine correctables* to allow for client-side speculation, thus enabling a client application to minimize the observed transaction latency even further by selecting the desired consistency level of their transactions.
- We present an extensive evaluation of MERCURY in real and simulated networks, characterizing the end-to-end latency improvements of the proposed approach.
- We show that the principles underlying MERCURY can generalize to other quorum-based BFT protocols such as HotStuff [77], resulting in a greater relative latency reduction for HotStuff.

We prove the correctness of MERCURY in the extended version of the paper [16], and all the code employed in our experiments is available online [17].

## 2 BACKGROUND

In this section, we first highlight some fundamental concepts of BFT SMR and then revisit *weighted quorums* in BFT replication.

### 2.1 Byzantine State Machine Replication

Assuming a deterministic replicated service in which all replicas start at the same state [64], a BFT SMR service aims to satisfy two fundamental properties [14, 25]:

- (1) *SMR Safety*: It behaves as a centralized service executing atomic operations, one at a time (linearizability [45]).
- (2) *SMR Liveness*: All operations issued by correct clients eventually complete.

A common way to satisfy SMR Safety is to employ a consensus protocol for executing all operations/transactions in total order in all replicas, creating a replicated *decision log* abstraction where every log position  $i$  contains at most one decided operation (or batch of operations, as in blockchains). Many Byzantine consensus protocols have been proposed in the literature for implementing BFT SMR. In this paper, we are mostly interested in the ones that provide optimal resilience ( $t < \frac{n}{3}$ ) and best-case latency (3 communication steps [1]), such as PBFT [25] (described next).

Alternatives, such as speculative protocols like Zyzzyva [50] and “fast” consensus variants [51, 57], do not perform satisfactorily in geo-replicated settings. The main reason for this inadequacy lies in their network environment requirements and quorum formation rules. For instance, *speculative execution*, as used in Zyzzyva, demands a predictable and stable network environment, which is uncommon in geo-distributed deployments. Specifically, Zyzzyva's performance degrades because it necessitates responses from *all* replicas within a *strictly configured time window* to complete a request in a single phase [68]. On the other hand, fast Byzantine protocols run consensus in two communication steps but use proportionally *larger* quorums, e.g.,  $4t - 1$  out of  $n = 5t - 1$  [51]. While

theoretically faster in homogeneous networks, this approach also results in increased latency in geo-replicated settings [48].

*PBFT*. The Practical Byzantine Fault Tolerance (PBFT) SMR algorithm [25] is considered the first practical method for implementing BFT services, optimal in terms of resilience and best-case latency, ensuring safety under asynchrony and requiring a weak form of synchrony for liveness. PBFT orders requests by relying upon a stable leader that assigns sequence numbers to request batches. If the leader is correct and the system is sufficiently synchronous, PBFT executes only its normal case operation. This pattern represents a *Byzantine agreement/consensus instance* and consists of the leader proposing a batch of operations to all replicas (PRE-PREPARE), followed by two phases of all-to-all message exchanges (PREPARE and COMMIT), in which replicas use quorums to commit/decide the messages with a given sequence number. These quorums are sufficiently large such that any intersection of two quorums  $Q$  and  $Q'$  contains at least one correct replica, i.e.,  $|Q \cap Q'| \geq t + 1$ . If the protocol stalls (e.g., the leader is faulty), a *view change* sub-protocol is triggered when  $t + 1$  replicas suspect the leader. During a view change (alias leader change or synchronization phase [19]), the newly-elected leader collects the current status from a quorum of replicas and takes consistent decisions for pending requests.

### 2.2 Weighted Quorums in BFT Replication

WHEAT [70] improves PBFT-like SMR for geographically dispersed deployments by reducing client latency using  $\Delta$  additional replicas, which does not affect the resilience threshold, i.e.,  $n = 3t + 1 + \Delta$ . Instead of the egalitarian Byzantine majorities of replicas used in most BFT works (e.g., [25, 73, 77]), WHEAT uses weighted replication to obtain proportionally smaller quorums, enabling the assignment of high weights to well-connected replicas for accelerating the protocol. This approach maintains system availability, as votes from low-weight replicas are still used in case of failures.

BFT systems typically probe a Byzantine dissemination quorum containing  $\lceil \frac{n+t+1}{2} \rceil$  replicas [54], as shown in Figure 2a. With  $n = 21$  replicas, quorums of size 14 intersect in at least  $t + 1$  replicas (for  $t = 6$ ). Achieving the *same intersection property with smaller quorums* is possible by leveraging replicas with more voting power (*weight*), as shown in Figure 2b. A fast quorum with 13 replicas (voting weight 17) is smaller than an egalitarian quorum while preserving intersection due to including all replicas with high voting power. Consider the scenario where the fastest, geographically closest replicas constitute that quorum. These replicas can progress the voting phases of consensus more swiftly, as they need to wait less time for vote collection. This acceleration in WHEAT also decreases the overall latency of BFT SMR [70].

Unlike traditional protocols like PBFT that wait for responses from a strict number of replicas, WHEAT waits for a sum of votes. WHEAT uses a bimodal scheme where the  $2t$  best-connected replicas have a voting power of  $V_{max} = 1 + \frac{\Delta}{t}$  while the remaining replicas have a voting power of 1. As a result, the number of votes required for a quorum is  $Q_v = 2tV_{max} + 1$ . This approach ensures the ability to form quorums even if  $t$  best-connected replicas fail. In this scheme, all quorums contain between  $2t + 1$  and  $n - t$  replicas. The size of the *smallest* quorum only depends on the chosen

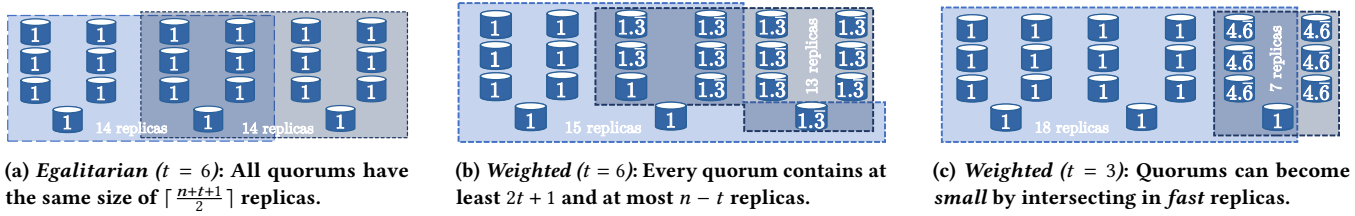


Figure 2: Overview over BFT quorum systems for  $n = 21$  replicas.

threshold  $t$ , not on the actual size of the system  $n$ . For instance, in Figure 2c, a fast quorum comprises 7 replicas for  $t = 3$ .

*AWARE*. Distributing voting weights is difficult due to the complexity of determining the optimal weight configuration for given network characteristics. *AWARE* [15] addresses this challenge by enabling geo-replicated state machines to self-optimize dynamically with automated weight tuning and leader placement, supporting the emergence of fast quorums in the system.

In *AWARE*, each replica monitors latencies to other replicas by measuring response times to protocol messages. These latency measurements are disseminated by each replica with total order, allowing all correct replicas to consistently update a latency matrix [15]. This matrix serves as input to a deterministic prediction model, which computes the expected consensus latency by simulating the protocol run for several weight distributions and leader locations, thus finding the optimal system configuration. *AWARE* automatically chooses the fastest configuration given a fixed resilience threshold  $t$ . When the latency matrix is updated and changes are detected, the system reconfigures the weight distribution and/or leader location to better suit the current network conditions.

### 3 SYSTEM MODEL AND DESIGN

*System Model*. This work employs the same system model used in BFT protocols such as HotStuff [77] and PBFT [25]. Ensuring liveness requires a weak *partial synchrony* [33] where the system may initially behave asynchronously and, after an *unknown GST* (*Global Stabilization Time*), some upper bound holds for all message transmission delays. We consider an adaptive adversary capable of corrupting up to  $t < n/3$  Byzantine replicas and an unbounded number of Byzantine clients. Byzantine entities can behave arbitrarily and collude under the control of the adversary.

*System Design*. *MERCURY* is a self-optimizing protocol transformation that adapts the resilience threshold of a BFT protocol and tunes replica voting weights to enable the emergence of smaller quorums for low-latency transaction ordering. Our approach seeks to balance between maintaining maximum resilience and continuously striving for faster consensus execution by optimizing the system for the common case with few or no failures. Specifically, we aim to design a fast BFT SMR approach for wide-area deployments that satisfies the standard SMR safety and liveness (see §2.1) for the optimal resilience bound  $t = \lfloor \frac{n-1}{3} \rfloor$  and can tune itself to achieve fast commit latency when there is a stable, correct leader and no more than  $t_{fast} = \lceil \frac{t}{2} \rceil$  faulty replicas.

This approach is implemented using two modes of operation (see Figure 3), as done in the past for improving performance [9] and

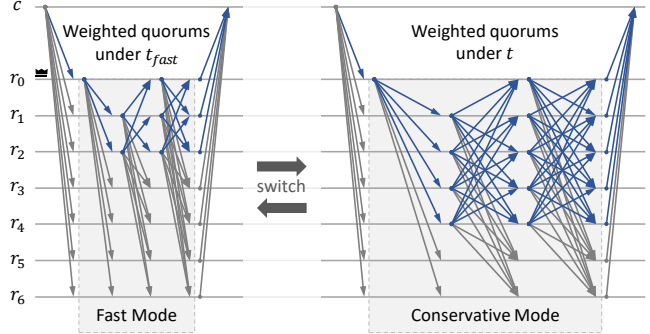


Figure 3: *MERCURY* two modes of operation ( $t = 2, t_{fast} = 1$ ).

resource efficiency [32]. The system starts in *conservative mode* by running instances of a quorum-based consensus protocol tolerating  $t$  failures. Periodically, after a number of consensus instances are decided, the system attempts to switch to the optimistic *fast mode* that tolerates only  $t_{fast}$  failures. While the leader is correct and the number of actual failures  $f$  does not surpass  $t_{fast}$ , *MERCURY* stays in this configuration and uses smaller quorums to accelerate consensus. If latency improvements do not match expectations, the leader is suspected to be faulty, or correct replicas detect equivocations, *MERCURY* switches back to the conservative mode.

*Building Blocks*. The weighted replication scheme introduced by *WHEAT* [70] distributes voting power to enable the existence of small quorums, ensuring that all quorums intersect in at least one correct replica (consistency) and that there is always at least one quorum available (availability). These requirements ensure that there is at least one minimal quorum with  $2t_{fast} + 1$  replicas, regardless of  $n$ .

Another hurdle when employing weighted replication is how to (re)assign the weights in accordance with the current system conditions. In our case, we must select the  $2t_{fast}$  replicas that will receive maximal voting power, so these can comprise the compact quorums. We resort to the latency measurement and weight reassignment scheme of *AWARE* [15] to (re)assign weights (select the  $2t_{fast}$  replicas that will receive maximal voting power) and focus instead on the problem of running the protocol optimistically, tolerating a few failures. We remark, however, that the influence of malicious replicas on these measurements is bounded by *AWARE*'s sanitization strategy and the verification of a replica reported values using its geo-location, as done in secure location services [49].

*Open Design Challenges.* While building upon AWARE's [15] features, which enables small quorums with reconfigurable weights based on the observed latency between replicas, the design of MERCURY faces several non-trivial challenges. First and foremost, we need mechanisms to detect and diagnose the system when there are more than  $f > t_{fast}$  failures. Second, a robust reconfiguration mechanism must safely abort the fast mode and switch to the conservative one in such situations. Finally, the client-replica contract must ensure linearizability in our dual fault-threshold approach.

## 4 MERCURY: THRESHOLD-ADAPTIVE BFT

In this section, we describe MERCURY, a self-optimizing protocol transformation that adapts the resilience threshold of a BFT protocol to enable the emergence of smaller quorums for faster transaction ordering. The consolidated algorithm of MERCURY is presented in Figure 4. The algorithm starts with a client submitting a request  $o$  to all replicas (C1). By Rule S1, replicas start a *timer* for each request, and the leader creates a batch of unordered requests and proposes it through the normal case operation of the underlying base protocol AWARE or AWARE★ if the system runs in *fast* mode. The rest of the algorithm summarizes all the required extensions on AWARE to accommodate the novel mechanisms of MERCURY. We describe these mechanisms in the context of the challenge they address in our protocol design.

### 4.1 Challenge #1: Dealing with $f > t_{fast}$ Failures

A key challenge in devising MERCURY is ensuring safety and liveness when the system runs in fast mode and  $f > t_{fast}$ .

**4.1.1 Safety.** When running in fast mode, the adversary can control more than  $t_{fast}$  replicas and cause equivocations in the system. This might lead correct replicas to decide different transaction batches in a consensus instance since fast mode's smaller quorums are not guaranteed to overlap in one or more correct replicas.

*Periodic Checkpoint.* To handle this scenario, the system must detect if the actual number of faults ( $f$ ) surpasses the resilience threshold of the fast mode ( $t_{fast}$ ) and, if necessary, revert the system to the conservative mode. We need to check the state of the replicas through periodic *checkpoint messages* (as in PBFT [25]) to ensure they are consistent and detect faults. By Rule S2, on every  $k$  completed consensus instances, each replica takes a snapshot of the service state and broadcasts to all replicas a signed message with the digest of this snapshot  $h$  and the highest consensus instance  $i$  that affected it. Each replica waits for  $n - t$  matching checkpoint hashes for the same consensus instance to define the checkpoint as *stable*. During this process, if a correct replica, which we call *the auditor*, detects non-matching checkpoints, it runs the lightweight forensics procedure of Figure 5 to identify and obtain a non-repudiable *Proof-of-Culpability* (PoC) for the protocol violators.

This protocol can identify equivocating replicas in the system if there are no more than  $2t_{fast}$  faulty replicas. In fact, Sheng et al. show that it is impossible to identify misbehaving replicas if there are more than  $2t$  Byzantine replicas in a system tolerating  $t$  failures [66]. This limitation leads MERCURY to use  $t_{fast} = \lceil \frac{t}{2} \rceil$ .

If the auditor receives  $n - t$  matching checkpoint hashes during this execution, it stops the forensics procedure. This action prevents

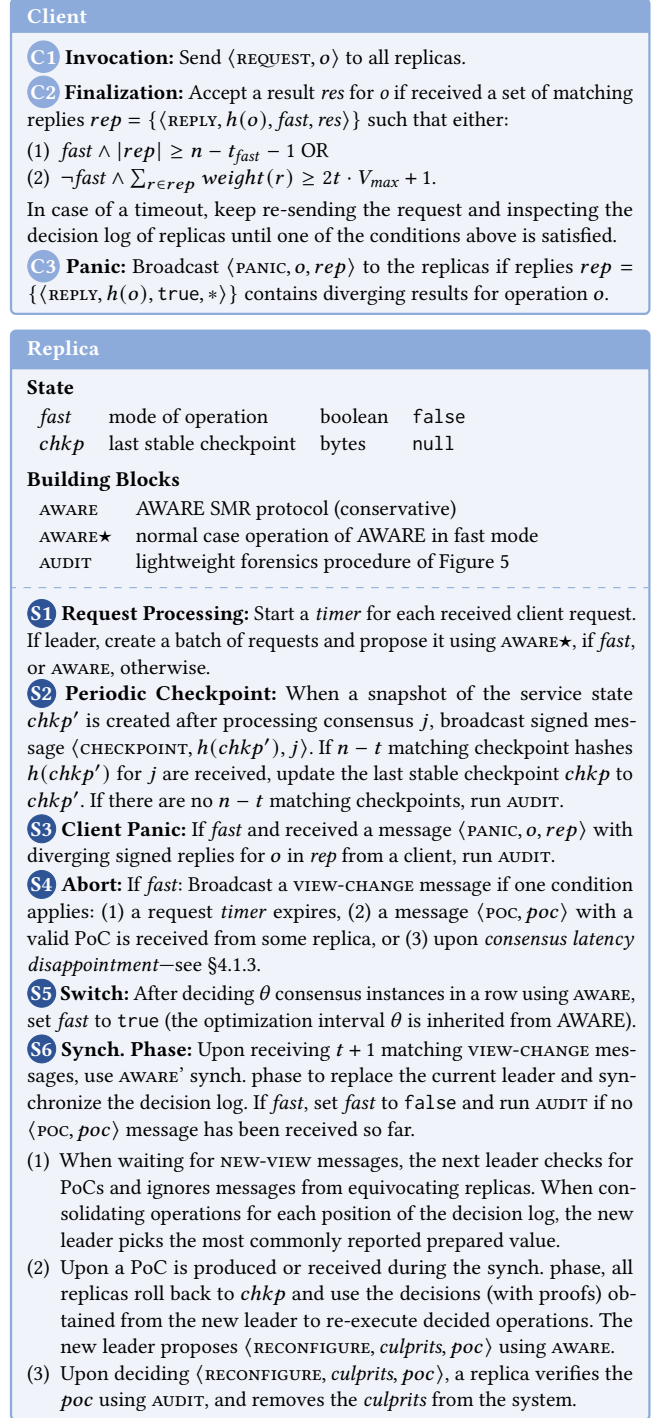
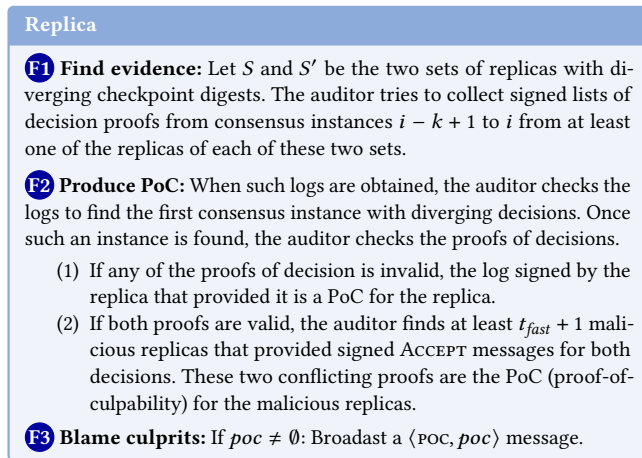


Figure 4: A summary of MERCURY.

faulty replicas from blocking correct replicas in forensics procedures since a faulty replica can send a non-matching checkpoint but never send its corresponding log. After concluding the lightweight forensics procedure, if a PoC is produced for one or more



**Figure 5: Lightweight forensics procedure.**

replicas, the auditing replica broadcasts this PoC to all replicas, forcing the system to switch to the conservative mode and expel the misbehaving replicas (described in §4.2).

*Client Panic.* The lightweight forensics protocol also triggers if a client detects non-matching signed replies for a request (C3). When this happens, the client sends a PANIC message containing the conflicting replies to the replicas. By Rule S3, a replica receiving a PANIC message with correctly signed conflicting replies starts the lightweight forensics protocol, but fetches logs from the last checkpoint until the consensus instance that decided the problematic request.

**4.1.2 Liveness.** Besides equivocations,  $f > t_{fast}$  replicas controlled by an adversary can stay silent and negatively affect the liveness of the system. In such situations, there will be fewer than  $n - t_{fast}$  correct replicas in the system, violating a key liveness assumption of the consensus protocol designed for no more than  $t_{fast}$  failures. This can lead to two unfavorable situations. First, client requests might not be ordered, triggering a timeout and initiating a view change. As explained in the next section, this sub-protocol reverts the system to the conservative mode that tolerates up to  $t$  failures.

The second situation is more complex: the request may be ordered, but faulty replicas might not reply to the client, preventing result consolidation. In this case, the client could send a *panic* message to the replicas asking them to switch to the conservative mode, which might trigger a leader change to switch the system's mode. However, this must be done carefully, as a malicious client could abuse this mechanism to prevent the system from operating in fast mode, a known weakness inherent to optimistic protocols [9]. This mechanism could make MERCURY optimization fragile, as a single malicious client can undermine latency improvements. Thus, we propose an alternative approach for dealing with this situation.

*Finalization.* By Rule C2, if the client does not receive the required number of replies, it periodically checks the decision log until the next checkpoint to see where its operation appears in the finalized decision log. This procedure is similar to how blockchain clients inspect the blockchain until their requests are included in a

block several blocks away from the blockchain head. This approach ensures that clients can benefit from MERCURY's low latency as long as there are no more than  $t_{fast}$  faulty replicas in the system.

**4.1.3 Performance Degradation.** To ensure MERCURY does not degrade performance compared to the conservative mode, all replicas periodically monitor their observed performance and compare it with expectations they have on the conservative mode. Replicas retrieve their expectations from AWARE's underlying latency prediction model [15]. Using this model, replicas can predict their consensus latency for the conservative mode using the network latency map and set their *consensus latency expectation threshold*.<sup>1</sup> If replicas find that the latency they currently observe exceeds this threshold, they stop their execution and ask for a view change (S4). When  $t + 1$  replicas ask for a view change, the system switches to the conservative mode (see next section). In the end, MERCURY only runs in fast mode if replicas observe the consensus latency to be lower than the expected latency in the conservative mode.

## 4.2 Challenge #2: Reconfiguration of the System

As explained before, MERCURY operates in two regimes: fast, in which smaller quorums are used and  $t_{fast}$  failures are tolerated, and conservative, in which standard-size quorums are employed and  $t$  failures are tolerated.

*Switch.* The system starts in the conservative mode, and by Rule S5 after finishing a predefined number of  $\theta$  consecutive consensus instances, it switches to the fast mode. Such reconfiguration is very simple because it is done deterministically at a certain point in the execution, i.e., after a certain consensus instance is decided. At this point, it simply requires each replica to locally change the fault threshold to  $t_{fast}$  and recalculate its quorum size before executing the next consensus instance.

*Abort.* A replica stays in the fast configuration until the underlying algorithm view change is triggered by Rule S4. This can be done deliberately due to either safety or liveness issues, as discussed in previous subsections. In both cases, we require the participation of  $t + 1$  replicas to start the view change, which always runs considering threshold  $t$ , not  $t_{fast}$  (which might already be violated).

*Synchronization Phase.* During the synchronization phase (*view change* in PBFT parlance), the newly elected leader receives from  $n - t$  replicas the log of the decided instances since the last checkpoint and verifies it for diverging decisions using the lightweight forensics procedure (Figure 5). Rule S6 is applied in subsequent steps:

- (1) If multiple decisions for the same slot exist, the new leader selects the most commonly reported one to consolidate the decision log.
- (2) The first transaction of the newly elected leader's regency, after repairing the system to a single transaction history, is a reconfiguration request [19]. This request aims to remove the Byzantine replicas involved in the equivocation and contains

<sup>1</sup>In AWARE, replicas run consensus to create a uniform view of latency measurements, which are recorded in the decision log. These can be accessed by each replica to deterministically compute an expectation value for consensus latency by feeding the measurements into a model that simulates the protocol run in conservative mode. The obtained expectation value serves as an indication to determine if an enabled optimization actually accelerates consensus or not.

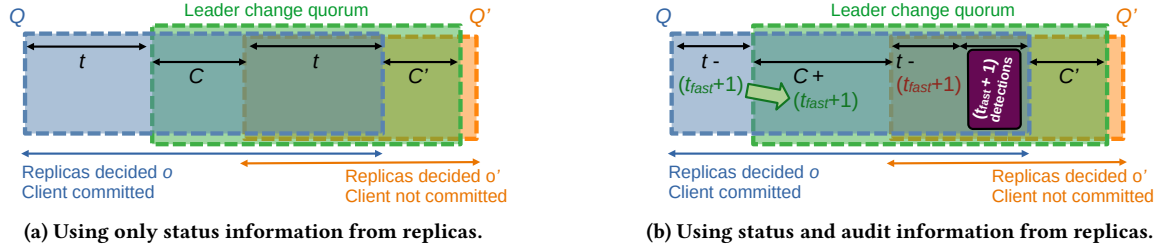


Figure 6: Quorum reasoning in MERCURY.

the PoC generated during the forensics procedure. Correct replicas remove these compromised replicas from the system after processing reconfiguration requests with valid PoCs.

- (3) Further, if the replica was subject to equivocation and the new leader decided differently from it, it might need to roll back its state to a previous *stable checkpoint* (see §4.1.1), reapplying the correct transaction history as defined by the new leader on this state.

### 4.3 Challenge #3: Ensuring Linearizability

In typical BFT SMR systems, a client waits for  $t + 1$  matching replies to ensure the replicated system perfectly emulates a centralized server, satisfying linearizability [45]. This quorum size becomes  $\lceil \frac{n+t+1}{2} \rceil$  if avoiding consensus for read-only operations [14, 25]. While in conservative mode, these quorum sizes are still valid in MERCURY. However, when the system is in fast mode, the existence of equivocations and the possibility of divergent decisions (that will be later detected and punished) requires revisiting the number of matching replies expected by clients in **Rule C2**.

Figure 6 illustrates the scenario where two clients received replies for operations  $o$  and  $o'$  from two different quorums  $Q$  and  $Q'$ , respectively, for consensus instance  $i$  (ignore the leader change quorum for now in the Figure 6a). Even if  $t$  malicious replicas are present in the intersection of the quorums, a client can assume that its request has been committed and will not be rolled back by waiting for  $n - t$  matching replies. This holds because the intersection  $(n - t) + (n - t) - n > t$  when  $n > 3t$ . It means that two quorums with  $n - t$  replicas intersect in more than  $t$  of them, ensuring the presence of at least one correct replica in this intersection. By waiting for  $n - t$  matching replies, the responses accepted by clients will never be rolled back, even with divergent decisions for consensus instance  $i$ , as long as there are no leader changes.

Now, consider the scenario in which the replies for  $o$  were deemed final by the client, but a leader change occurred, and the new leader must define the result of consensus instance  $i$ . The elected leader waits for  $n - t$  replicas to inform their status as indicated in the leader change quorum of Figure 6a, receiving replies from every replica but  $t$  slow replicas that decided operation  $o$ .

In this setting, the decision of  $o$  will be preserved as long as such value is the majority value among the ones informed by replicas, i.e.,  $C > C' + t$  in the figure. Considering  $n = 2t + C + C'$  and  $|Q| = C + 2t$  (both directly from the highlighted variables in the figure), we can reach that  $C > \frac{n}{3}$ , leading to  $|Q| = n$ . Therefore, waiting for  $n - t$  matching replies is insufficient to ensure a value will

never be rolled back during a leader change that switches the system to the conservative mode. The only quorum big enough to ensure this is waiting for matching replies from *all* replicas.

Fortunately, integrating a continuous lightweight BFT forensics procedure in the view change sub-protocol (view change in PBFT or synchronization phase in BFT-SMART) enables the use of smaller quorums. More specifically, we observe that to produce equivocations that lead some correct replicas to decide  $o$  and  $o'$ , and later force a committed value to be rolled back, the  $t_{fast} + 1$  equivocators must participate in the three quorums (for  $o$ ,  $o'$ , and leader change). Therefore, if the new leader executes the forensics protocol during the leader change, it detects  $t_{fast} + 1$  equivocators. Consequently, it can discard the contributions of these malicious replicas and wait for messages from  $t_{fast} + 1$  additional replicas. This situation is illustrated in Figure 6b.

In this scenario, instead of assuming  $C > C' + t$ , we have

$$C + (t_{fast} + 1) > C' + t - (t_{fast} + 1) \quad (1)$$

By developing this in inequality like before, we find that *by waiting for  $n - t_{fast} - 1$  matching replies in fast mode, a client knows the result of its operation is finalized, ensuring the durability and linearizability of the replicated service*, allowing us to prove the following theorem in the extended version of the paper [16]:

**Theorem 1.** *If an operation  $o$  is finalized in  $i$ -th position of the decision log, then no client observes an operation  $o' \neq o$  in this position of the decision log.*

*Detailed derivation of  $Q = n - t_{fast} - 1$ :* Considering  $n = 2t + C + C' > 3t$  (from Figure 6a), it follows that  $C + C' > t$  and thus  $C' > t - C$ . Replacing  $C'$  with this value in inequality (1) yields:

$$C + (t_{fast} + 1) > \overbrace{t - C}^{C'} + t - (t_{fast} + 1) \quad (2)$$

$$2C > 2t - 2(t_{fast} + 1) \quad (3)$$

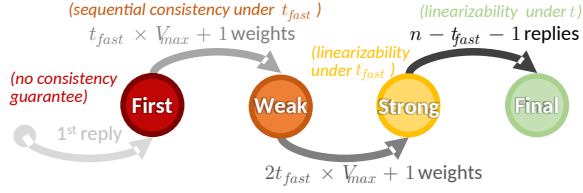
$$C > t - t_{fast} - 1 \quad (4)$$

Now, we calculate the response quorum  $Q = 2t + C$  (also from Figure 6a) using the inequality of (4):

$$Q > 2t + \overbrace{t - t_{fast} - 1}^C \quad (5)$$

$$Q > 3t - t_{fast} - 1 \quad (6)$$

Which can be generalized to  $Q = n - t_{fast} - 1$ .



**Figure 7: Incremental consistency levels obtained through the Byzantine correctable programming interface.**

## 5 IMPLEMENTATION AND OPTIMIZATIONS

*Implementation.* MERCURY was implemented on top of the AWARE prototype [15], which is based on BFT-SMART [19]. We stress that all mechanisms employed in AWARE (e.g., latency measurement and weights reassignment) could be implemented in any quorum-based SMR protocol. This implementation uses TLS to secure all communication channels and the elliptic curve digital signature algorithm (ECDSA) and SHA256 for signatures and hashes, respectively. Most of our modifications are related to the switching between two modes of operation (with different resilience thresholds) and implementing BFT forensics.

*Improving Latency with Speculation.* Operating in fast mode requires clients to collect  $n - t_{fast} - 1$  matching replies to preserve linearizability. This finalization quorum is considerably bigger than the  $t + 1$  replies typically required in BFT SMR and is expected to negatively impact clients’ observed latencies in fast mode. MERCURY can further improve the end-to-end latency by using client-side speculation. For this purpose, we implemented correctables [41] in the client shim of our BFT protocol. A correctable is a programming abstraction that allows a client application to work with incremental consistency guarantees and accelerates the application by allowing it to speculate with intermediate results. For example, this enables fast confirmation for low-value transactions such as time-sensitive micropayments even before linearizability under  $t$  failures is ensured. The state of a correctable can be updated multiple times, depending on the replies received by the client, strengthening the consistency guarantee each time until it reaches the final state, which corresponds to the strongest consistency guarantee. MERCURY’s *Byzantine correctables* follow two principles:

- (1) Ensure the same safety guarantee as traditional BFT SMR: the final consistency guarantee must satisfy linearizability under the resilience threshold  $t$ .
- (2) Less safe consistency guarantees may relax either the assumptions on the number of Byzantine replicas or trade linearizability for a weaker consistency model.

We define incremental consistency levels for MERCURY as follows (see Figure 7). *First* is the speculative result a client can access as soon as the first response arrives, which does not provide any correctness guarantee. *Weak* demands replies from replicas totaling  $t_{fast} \times V_{max} + 1$  votes, being  $V_{max}$  the maximum weight assigned to a replica, and thus must have been confirmed by at least one correct replica if  $f \leq t_{fast}$ . This result can be stale, satisfying only sequential consistency [8] under  $t_{fast}$  failures. *Strong* demands  $Q_v = 2t_{fast} \times V_{max} + 1$  votes and satisfies linearizability if  $f \leq t_{fast}$ . Lastly, the *Final* level satisfies linearizability under  $t$  (just like any typical SMR

with the read-only optimization enabled) by waiting for  $n - t_{fast} - 1$  replies, as explained in the previous section. Since classical SMR preserves linearizability [25], only *Strong* and *Final* give the typical safety guarantee for their respective resilience thresholds.

## 6 EVALUATION

We evaluate the latency of the MERCURY prototype on the AWS cloud as well as a simulated network of 51 replicas based on real data from the internet. Parts of our experiments were conducted in our local data center using high-fidelity tools for network emulation and simulation [39, 47]. We validated the fidelity of these emulated/simulated setups with additional experiments reported in the extended version of this paper [16]. Our experiments focus on measuring latency, which is fundamentally limited by the distance between nodes and quorum formation rules in WANs.

### 6.1 AWS Data-centers

To begin with, we investigate the potential performance gains of MERCURY, comparing it to AWARE and BFT-SMART as baselines. Later, we reason about MERCURY’s runtime behavior (particularly its adaptiveness) in the presence of failures or malicious replicas.

*Setup.* We deploy a client and a replica in each of the  $n = 21$  AWS cloud regions using c5.xlarge instances (depicted in Figure 1a). All clients send 400-bytes requests simultaneously and continuously to the replicas (2000 requests per client) until each client has finished its measurements. A request arriving at the leader may wait for inclusion in a batch when there is currently a consensus instance running. We employ synchronous clients that block until a result is obtained and send the next request after randomly waiting for up to 1s. Finally, *request latency* is the average end-to-end protocol latency computed by a client after finalizing all operations.

*6.1.1 MERCURY Acceleration.* For a better exposition, we group the 21 clients’ results by the continent they are located in, reporting only their regional averages (see Figure 8). First, we observe that MERCURY significantly accelerates consensus, leading to a speedup of  $3.57\times$  for reaching decisions<sup>2</sup> (Figure 8a). This result also surpasses the speedup of  $2.29\times$ , achievable if the speed of the links employed by BFT-SMART/PBFT approaches the speed of light.<sup>3</sup> Second, accelerating consensus decisions also leads to faster request latencies observed by clients worldwide (Figure 8b). Averaged over all client regions, MERCURY leads to a speedup of  $1.87\times$  over BFT-SMART for clients’ observed end-to-end request latencies with *Final* consistency (AWARE with the same resilience leads to  $1.33\times$  only).

Our results also show that even higher speedups can be achieved by employing the incremental consistency levels of MERCURY’s correctables. The *Strong* consistency level, which guarantees linearizability if  $f < t_{fast}$ , achieves a speedup of  $2.38\times$ , while the speculative levels *Weak* and *First* achieve speedups of  $2.76\times$  and  $2.90\times$ , respectively (results are averaged over all regions).

<sup>2</sup>To put these results in perspective, the MERCURY latency (100ms) is more than  $4\times$  faster than the results reported for a recent optimistic protocol in a similar network (see Table 1 in [53] and Figure 4 in [12]).

<sup>3</sup>Which is impossible: in practice, it is accepted that the maximum speed internet links can reach is  $0.67c$  [22, 49].



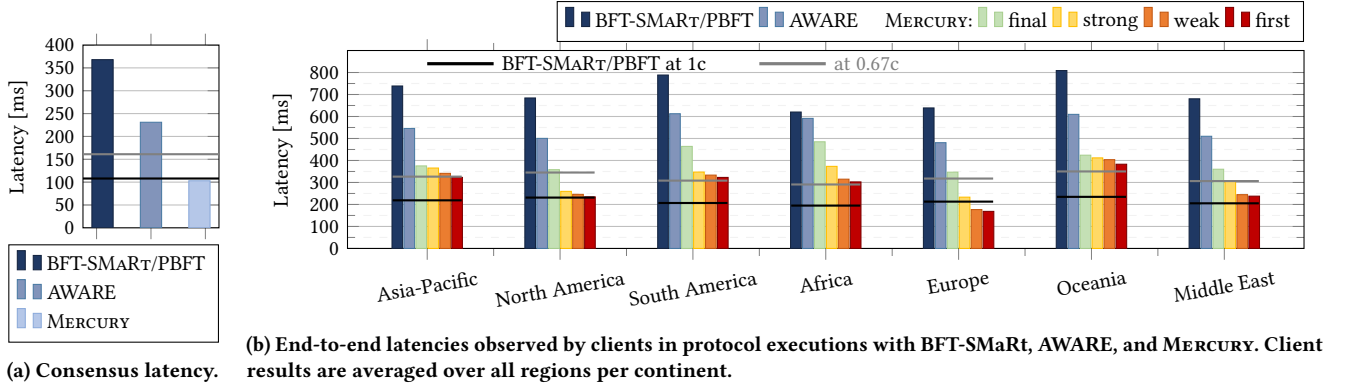


Figure 8: Achievable latency improvements for the  $n = 21$  AWS setup.

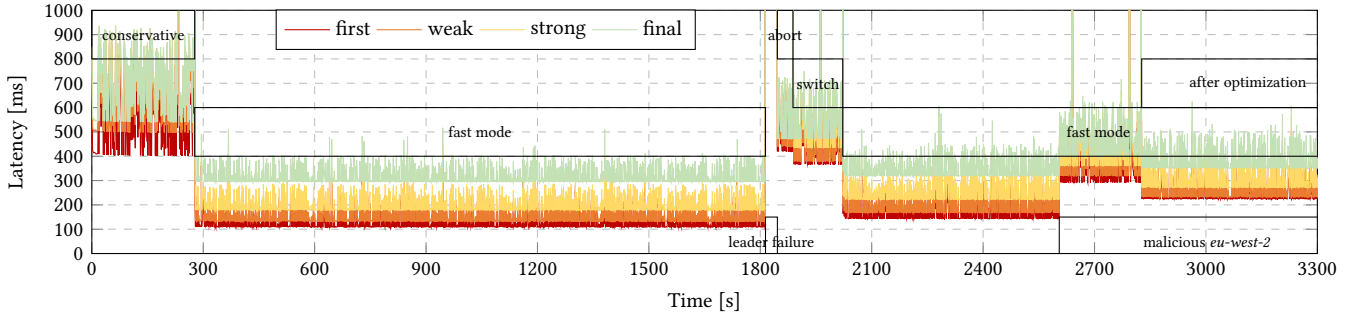


Figure 9: Runtime behavior of MERCURY under induced failures.

**6.1.2 Runtime Behavior under Failures.** For this experiment, we create an emulation of the AWS network in our local cluster to be more flexible with the induction of events. The emulated network uses latency statistics from *cloudping*,<sup>4</sup> and, the Kollaps network emulator, which was validated for realistic WAN experimentation with BFT-SMaRT and WHEAT [39]. We launch MERCURY in the same  $n = 21$  AWS regions to observe its runtime behavior during the system’s lifespan. Noticeably, clients’ request latencies show high variations, which are caused by a random waiting time of a request at the leader before getting included in the next batch, which takes a varying time depending on how shortly the request arrives before the next consensus can be started. Moreover, we induce the following events to evaluate MERCURY’s reactions and plot the latency observed by a representative correct client in Figure 9.

**Configuration switch.** MERCURY starts in the conservative configuration, displaying a latency similar to a “normal” PBFT-like protocol. Later, around time 277s, the system switches to the fast configuration (such switches are attempted every  $k = 400$  consensus instances), leading to a significant latency improvement.

**Silent leader.** At the time 1814s, the leader stops participating in the protocol and remains silent (this could be either seen as an attempt to impede the system to progress or the effect of a common crash failure). Subsequently, replicas perform a leader change and abort (at 1846s), switching back to the conservative

mode. This blocking time is similar to what a client experiences when discovering an equivocation. At 1889s, after finishing another 400 consensus instances, replicas return to the fast mode, yielding only a modest performance improvement because the weights are not yet optimized. At 2022s, the system self-optimizes to its best weight distribution and leader placement (using the mechanisms leveraged from AWARE), reaching again latencies almost as low as experienced before.<sup>5</sup>

**Malicious leader.** At the time 2640s, we artificially let the current leader eu-west-2 conduct a *pre-prepare delay attack* [3], purposely delaying sending its proposal to degrade the system’s performance. After about 185s, the time required for a measurement round and self-optimization [15], MERCURY detects it is running in a sub-optimal configuration and changes replicas’ weights, moving the leader to us-east-1 to accelerate performance.

**6.1.3 Throughput.** Although MERCURY aims to optimize latency, we conducted a simple 0/0-microbenchmark in our emulated AWS network with an increasing number of clients evenly distributed among all AWS regions while measuring the throughput of BFT-SMaRT, AWARE ( $t = 6$ ), AWARE ( $t = 3$ ), and MERCURY. In this experiment, we used 0-byte requests and replies to avoid the saturation of links bandwidth. Figure 10 presents the results, which show

<sup>4</sup><https://www.cloudping.co/grid>.

<sup>5</sup>After the leader fails, there are fewer replicas and thus less flexibility in quorum formation. Since the failed leader was part of the best clique of well-connected replicas, the following configuration (after re-optimization) is slightly slower.

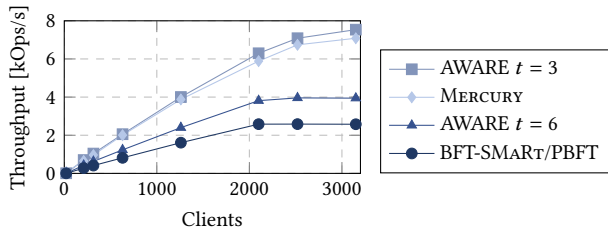


Figure 10: Throughput comparison for  $n = 21$  replicas.

two main insights. First, faster consensus instances can achieve higher throughput, provided the available network bandwidth is not exhausted. Second, MERCURY displays a modest performance decrease compared to AWARE ( $t = 3$ ), which uses the same quorums but offers *half of our resilience*. This difference stems from the costs of additional signatures needed to integrate lightweight forensics support into MERCURY.

## 6.2 Larger Deployments

In this experiment, we assess if the performance gains of MERCURY are sustained in a different scenario with a larger number of  $n = 51$  replicas, approaching an expected permissioned blockchain deployment. Since this number exceeds the available AWS regions, we sample locations from a publicly available dataset provided by *Wonderproxy*.<sup>6</sup> We distributed a replica in each of the chosen 51 locations and deployed 12 clients, as depicted in Figure 11.

*Setup.* For simulating this network, we use *Phantom* [47], which employs a hybrid simulation-emulation architecture, where real, unmodified applications execute as native Linux processes within this network simulator. Previous research has shown that Phantom can be used to faithfully evaluate the performance of BFT protocol implementations [18]. For validity, we repeated the  $n = 21$  AWS experiment depicted in the previous section in this simulator and observed a marginal deviation with the results obtained from the real AWS network and Kollaps emulator [16].

In our experiment, we measure the latency speedup that MERCURY achieves in direct comparison with BFT-SMART. We measure both consensus latency and client end-to-end latencies using the same method described before. Phantom bootstraps replicas and clients in their host locations with the initial protocol leader in Cape Town. As before, clients run simultaneously and send requests with a 400-bytes payload with a randomized waiting interval of up to 4s between two requests. When running MERCURY, replicas are started in a configuration with optimal resilience threshold ( $t = 16$ ), but MERCURY optimizes this threshold to  $t_{fast} = 8$  before clients collect their measurement samples.

*Results.* Figure 12 shows similar latency improvements as in the previous experiment for MERCURY when compared with BFT-SMART. The consensus latency (not shown) decreases from 350ms in BFT-SMART to only 88ms in MERCURY, corresponding to a consensus execution speedup of 3.98 $\times$ . For request latencies with *Final* consistency, the highest speedup observed was in Frankfurt (1.95 $\times$  from

<sup>6</sup><https://wonderproxy.com/blog/a-day-in-the-life-of-the-internet/>.

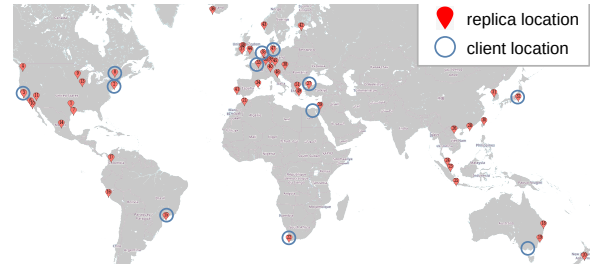


Figure 11: Map showing the locations of the 51 replicas used in our larger deployment.

615ms down to 314ms), and the lowest speedup was observed in Cape Town (1.41 $\times$  from 618ms to 440ms).

Further, the speedup increases when using the incremental consistency levels of the correctable. For instance, in Paris, the *First* level achieves a speedup of 5.52 $\times$ , while the *Strong* level still provides a speedup of 4.03 $\times$ . The average speedup across all client locations from BFT-SMART to MERCURY *Final* level is 1.83 $\times$  and becomes incrementally higher for the speculative levels *Strong* (2.70 $\times$ ), *Weak* (2.99 $\times$ ) and *First* (3.23 $\times$ ). For comparison, the speedup that BFT-SMART would achieve if the speed of network links approximated the speed of light is roughly 2.5 $\times$ . These results show that using smaller weighted quorums effectively reduces latency, with values similar to the expected latency of an optimal protocol (BFT-SMART/PBFT) using speed-of-light network links.

## 6.3 MERCURY-flavored HotStuff

The principle of improving quorum sizes introduced in MERCURY is general and can be used in other BFT SMR protocols to decrease latency. For example, our techniques can be directly applied to speedup agreement in multi-leader protocols [72], as long as the leaders are selected only among the best-connected replicas, or even in protocols providing additional guarantees such as fair ordering [79]. Here, we experimentally demonstrate this aspect by applying the MERCURY transformation to HotStuff [77].

Compared to BFT-SMART and PBFT, HotStuff uses an agreement pattern with one extra phase, achieving linear communication complexity by letting the leader collect and distribute quorum certificates in each phase. It results in 7 communication steps per consensus instance instead of 3 in BFT-SMART/PBFT. This design makes the system’s latency even more sensitive to how fast agreement can be achieved, which depends on the speed at which a HotStuff leader succeeds in collecting quorum certificates. MERCURYHotStuff selects a configuration where the leader communicates fast with a set of well-connected replicas granted high voting power.

*Setup.* We use a prediction model of the HotStuff protocol<sup>7</sup> to simulate MERCURYHotStuff and the original HotStuff protocol running on the same latency map of  $n = 51$  replicas used before (Figure 11). These simulations compute the achievable latencies for different consistency levels of MERCURYHotStuff, the latency of the HotStuff, and the hypothetical latencies of HotStuff with speed-of-light links.

<sup>7</sup>This model is similar to the one AWARE uses to anticipate the effect of weight and leader changes during its self-optimization [15].

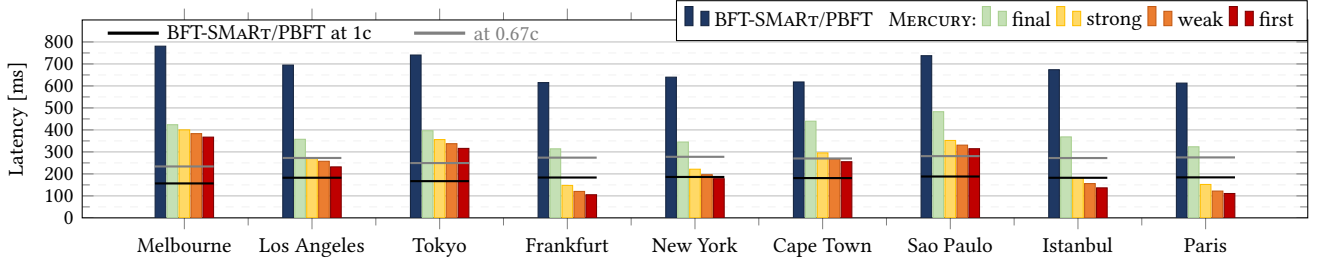


Figure 12: Latencies of BFT-SMaRt and MERCURY for  $n = 51$  replicas, observed from different client locations.

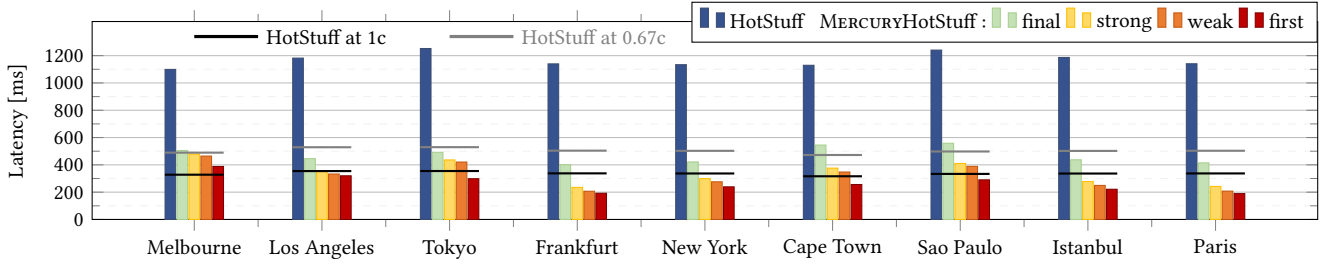


Figure 13: Latencies of HotStuff using MERCURY techniques for  $n = 51$  replicas.

**Results.** Our results show that MERCURYHotStuff significantly optimizes HotStuff’s consensus latency (see Figure 13) from 853ms to only 177ms in MERCURYHotStuff. It corresponds to a consensus execution speedup of 4.82 $\times$ , achieved by incorporating weights, optimal leader placement, and the use of smaller quorums. For client request latencies with *Final* consistency, the highest speedup observed was in Frankfurt (2.84 $\times$ ) and lowest in Cape Town (2.07 $\times$ ).

Like before, the speedup increases with lower consistency levels. For instance, in Paris, the *First* level achieves a speedup of 6.03 $\times$  (from 1141 ms down to 189 ms), while the *Strong* level still provides a speedup of 4.71 $\times$  (242 ms). The average speedup across all client locations from HotStuff to MERCURY HotStuff’s *Final* level is 2.56 $\times$  and becomes incrementally higher for the speculative levels *Strong* (3.69 $\times$ ), *Weak* (4.05 $\times$ ) and *First* (4.74 $\times$ ).

## 7 RELATED WORK

**Adaptivity in BFT SMR.** Making BFT protocols adaptive to their environment has been studied in multiple works [10, 13, 23, 26, 34, 52, 53, 61, 67]. RBFT monitors system performance under redundant leaders to prevent a faulty leader from degrading performance [10]. Other approaches propose optimizing the leader selection (e.g., [34, 52]), adaptively switching consensus algorithms [13, 23, 53], strengthening the protocol by reacting to perceived threat level changes [67], being network-agnostic (tolerating a higher threshold in synchronous networks) [20], or adapting the state transfer strategy to the available network bandwidth [26]. Bolt-Dumbo [53] runs a fast quorum-based protocol in synchronous periods and falls back to an asynchronous consensus otherwise. In contrast, MERCURY techniques apply to most BFT protocols and accelerate planetary-scale Byzantine consensus by making both system configuration (leader and replica weights) and threshold adaptive without impacting resilience through the integration of BFT protocol forensics.

**Geographically-distributed SMR.** Various works studied the improvement of SMR for WANs [4, 29, 35, 36, 44, 55, 56, 60, 62, 74–76]. Mencius, one of the earliest of these works, optimizes WAN performance using a rotating leader scheme that allows clients to pick their geographically closest replica as its leader [56], but tolerating only crash faults. EBAWA uses the same rotating leader technique together with trusted components on each replica to tolerate Byzantine replicas in a protocol with the same number of communication steps as Mencius [74]. Steward proposes a hierarchical, two-layered replication architecture. Regional groups within a system site run Byzantine agreements, and these replication groups are then connected with a crash-tolerant protocol [4]. Fireplug [60] later adapts this hierarchical architecture for efficient geo-replication of graph databases by composing multiple BFT-SMaRt groups. GeoBFT [44] assumes regional clusters and employs hierarchical consensus to first replicate a client transaction in its local replication group, and afterward, the transactions of all local groups are shared globally, then ordered and executed. A similar approach was used in [76], which employs the Damysus protocol [31] to decide on superblocks in the upper consensus layer. In hierarchical approaches, the number of failures is assumed to be bounded for each cluster.

WHEAT optimizes BFT SMR latency by incorporating weighted replication and tentative executions [70], while AWARE enriches WHEAT through self-monitoring capabilities and dynamic optimization by adjusting weights and leader position [15]. By integrating lightweight forensics, MERCURY can safely use smaller consensus quorums and accelerate Byzantine consensus even further than AWARE, thus mastering the resilience-performance trade-off that limits AWARE’s performance.

**Fast or Speculative BFT.** It has been shown that having additional redundancy (and using a less-than-optimal resilience threshold) can

be efficiently utilized to develop “fast” consensus variants, i.e., two-step Byzantine consensus [46, 51, 57], or even one-step asynchronous Byzantine consensus for contention-free scenarios [37, 69]. DuoBFT [7] uses the hybrid and Byzantine fault models where clients can choose the favored model for each command. Since hybrid commits take fewer communication steps and use smaller quorums than BFT commits, clients benefit from low-latency commits in the hybrid model. In comparison, MERCURY extends a latency- and resilience-optimal protocol (PBFT) to significantly improve latency without requiring more than  $3t + 1$  replicas.

Some BFT protocols propose mechanisms based on speculation to accelerate the overall protocol when running in a “common case” scenario (often assuming the absence of failures or congestion) [25, 43, 50, 75]. A form of server-side speculation technique was initially proposed by PBFT as *tentative execution*, in which replicas execute and respond to requests directly after the PREPARE stage [25]. In the Proof-of-Execution protocol, server-side speculation after the PREPARE stage was revisited, formally specified, and proved correct [43]. *Tentative executions* are orthogonal to the techniques explored in this paper and are prototypically implemented and explored by the WHEAT protocol [70], which AWARE and MERCURY are extending.

Zyzyva proposes a form of *speculative execution* in which replicas execute requests directly after receiving a proposal from the leader [50]. Yet Zyzyva requires a predictable and stable network that is uncommon in geo-distributed deployments and shows quickly degrading performance in case of failures, as it necessitates collecting responses from *all* replicas within some time window to complete a request in a single phase [68]. PBFT-CS refines PBFT with client-side speculation. Clients send subsequent requests after predicting a response to an earlier request without waiting for the earlier request to commit—however, clients need to track and propagate the dependencies between requests [75].

In contrast, MERCURY incorporates a novel technique for client-side speculation to allow applications to work on correctable results (as first proposed in [41]) obtained from the replicated state machine by proposing *Byzantine correctables* which offer increasing consistency guarantees to clients.

*Accountability in BFT.* BFT forensics analyzes safety violations in BFT protocols after they happened [66], yielding results such as that at least  $t + 1$  culprits can be identified in case of an equivocation (with the accountability of up to  $\frac{2n}{3}$  replicas that may be Byzantine). Polygraph is an accountable Byzantine consensus algorithm tailored for blockchain applications that allows the punishment of culprits (e.g., via stake slashing) in case of equivocations [27]. A simple transformation to obtain an accountable Byzantine consensus protocol from any Byzantine consensus protocol has been proposed in [28]. The Basilic class of protocols solves consensus with  $n \leq 3t + d + 2q$  replicas tolerating  $t$  general Byzantine failures,  $d$  deceitful failures (that violate safety—the ones that forensics can identify), and  $q$  benign failures [63]. Basilic’s resilience has been proven optimal. FireLedger [21] proposes a high-throughput blockchain consensus protocol in which the last  $t + 1$  blocks of every replica’s blockchain are considered *tentative* and replicas verify the correctness (finality) of these blocks later on. A malicious proposer can be detected using a proof and is removed from the system

through a recovery procedure. IA-CCF [65] shows that logging all messages exchanged in a PBFT-based blockchain makes it possible to identify any misbehaving replicas in case of equivocations.

MERCURY does not differentiate failures or require a blockchain, employing instead a light version of the BFT forensics protocol of [66] to identify a limited number of equivocators in fast mode.

## 8 CONCLUSION

MERCURY accelerates planetary-scale Byzantine consensus by combining weighted replication with lightweight BFT forensics to safely underestimate the resilience threshold, using faster quorums to drive consensus decisions. We showed how to obtain MERCURY from AWARE by utilizing BFT forensics techniques in a novel way: as a protective countermeasure against attacks. Notably, MERCURY always achieves linearizability and liveness under the optimal resilience threshold, even when quorums are formed using the fast threshold. Our evaluation results indicate that latency benefits are substantial, i.e., achieving a speedup of  $1.87\times$  over BFT-SMART/PBFT. Our methodology is a transformation applicable in other BFT protocols, improving their speed under geographical dispersion. We showed that if a protocol’s agreement pattern consists of more communication steps (like in HotStuff), it results in even greater benefits ( $2.56\times$  speedup).

We also employed client-side speculation, allowing the application to choose a relaxed consistency level from the client-replica contract on the granularity level of individual operations. This type of optimization can be a good fit for time-sensitive/low-risk transactions (e.g., micropayments) as they can benefit from up to  $6\times$  speedups. Moreover, we think there are interesting use cases for what applications can do with requests before they have been finalized. For instance, sequential consistency (called *Weak* in our model) might suffice in applications where operations work on non-shared objects (e.g., transferring a coin from an account  $A$  to  $B$  in a UTXO-based account model [58]), in which operations issued by different clients rarely conflict. Furthermore, even the most speculative level (*First*) could be used to speed up an application like a BFT/blockchain-based name resolver. Such a service could resolve a name, and then the client uses the intermediate (correctable result) to pre-fetch content from a web server. If the obtained result is incorrect, then the client would need to throw away pre-loaded content and initiate a new query to the correct server (the programming interfaces in the correctable define such behavior). Our evaluation results demonstrate that the *Strong* consistency level (which preserves linearizability under the assumption of  $f \leq t_{fast}$ ) already achieves high speedups and might present an interesting sweet spot for low-risk transaction settlement.

## ACKNOWLEDGMENTS

We thank the anonymous Middleware reviews and our shepherd, Mohammad Sadoghi, for their insightful comments. This work was supported by FCT through the ThreatAdapt (FCT-FNR/0002/2018) and SMARtChain (2022. 08431.PTDC) projects, and the LASIGE Research Unit (UIDB/00408/2020 and UIDP/00408/2020). This work was funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant number 446811880 (BFT2Chain).

## REFERENCES

- [1] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. 2021. Good-case Latency of Byzantine Broadcast: a Complete Categorization. In *Proc. of the 2021 ACM Symp. on Principles of Distributed Computing (PODC'21)*. ACM, New York, NY, USA, 331–341.
- [2] Salem Alqahtani and Murat Demirbas. 2021. BigBFT: A multileader Byzantine fault tolerance protocol for high throughput. In *Proc. of the IEEE Int. Performance, Computing, and Communications Conf. (IPCCC)*. IEEE, Piscataway, NJ, USA, 1–10.
- [3] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. 2010. Prime: Byzantine replication under attack. *IEEE Trans. on Dependable and Secure Computing (TDSC)* 8, 4 (2010), 564–577.
- [4] Yair Amir, Claudiu Danilov, Danny Dolev, Jonathan Kirsch, John Lane, Cristina Nita-Rotaru, Josh Olsen, and David Zage. 2010. Steward: Scaling Byzantine Fault-Tolerant Replication to Wide Area Networks. *IEEE Trans. on Dependable and Secure Computing (TDSC)* 7, 1 (2010), 80–93.
- [5] Elli Androulaki et al. 2018. Hyperledger Fabric: a distributed operating system for permissioned blockchains. In *Proc. of the 13th European Conf. on Computer Systems (EuroSys)*. ACM, New York, NY, USA, Article 30, 15 pages.
- [6] Karolos Antoniadis, Antoine Desjardins, Vincent Gramoli, Rachid Guerraoui, and Igor Zablotchi. 2021. Leaderless consensus. In *Proc. of the 41st IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, Piscataway, NJ, USA, 392–402.
- [7] Balaji Arun and Binoy Ravindran. 2022. DuoBFT: Resilience vs. Performance Trade-off in Byzantine Fault Tolerance. arXiv:2010.01387 [cs.DC] <https://arxiv.org/abs/2010.01387>
- [8] Hagit Attiya and Jennifer L. Welch. 1994. Sequential Consistency versus Linearizability. *ACM Trans. on Computer Systems (TOCS)* 12, 2 (May 1994), 91–122.
- [9] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. 2015. The next 700 BFT protocols. *ACM Trans. on Computer Systems (TOCS)* 32, 4 (2015), 1–45.
- [10] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. 2013. RBFT: Redundant Byzantine fault tolerance. In *Proc. of the 33rd IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, Piscataway, NJ, USA, 297–306.
- [11] Diogo Avelãs, Hasan Heydari, Eduardo Alchieri, Tobias Distler, and Alysson Bessani. 2024. Probabilistic Byzantine Fault Tolerance. In *Proc. of the 43rd Symp. on Principles of Distributed Computing (PODC)*. ACM, New York, NY, USA.
- [12] Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. 2024. Mysticeti: Reaching the Limits of Latency with Uncertified DAGs. arXiv:2310.14821 [cs.DC]
- [13] Jean-Paul Bahsoun, Rachid Guerraoui, and Ali Shoker. 2015. Making BFT protocols really adaptive. In *Proc. of the 29th IEEE Int. Parallel and Distributed Processing Symp. (IPDPS)*. IEEE, Piscataway, NJ, USA, 904–913.
- [14] Christian Berger, Hans P. Reiser, and Alysson Bessani. 2021. Making Reads in BFT State Machine Replication Fast, Linearizable, and Live. In *Proc. of the 40th IEEE Int. Symp. on Reliable Distributed Systems (SRDS)*. IEEE, Piscataway, NJ, USA, 1–12.
- [15] Christian Berger, Hans P. Reiser, João Sousa, and Alysson Neves Bessani. 2022. AWARE: Adaptive Wide-Area Replication for Fast and Resilient Byzantine Consensus. *IEEE Trans. on Dependable and Secure Computing (TDSC)* 19, 3 (2022), 1605–1620.
- [16] Christian Berger, Lívio Rodrigues, Hans P. Reiser, Vinicius Cogo, and Alysson Bessani. 2024. Chasing the Speed of Light: Low-Latency Planetary-Scale Adaptive Byzantine Consensus. arXiv:2305.15000 [cs.DC] <https://arxiv.org/abs/2305.15000>
- [17] Christian Berger, Lívio Rodrigues, Hans P. Reiser, Vinicius Cogo, and Alysson Bessani. 2024. Online Codebase. [Online]. Available: <https://github.com/bergerch/Mercury>.
- [18] Christian Berger, Sadok Ben Toumia, and Hans P. Reiser. 2022. Does My BFT Protocol Implementation Scale?. In *Proc. of the 3rd Int. Workshop on Distributed Infrastructure for Common Good (DICG)*. ACM, New York, NY, USA, 1–6.
- [19] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. 2014. State machine replication for the masses with BFT-SMaRt. In *Proc. of the 44th Annu. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE, Piscataway, NJ, USA, 355–362.
- [20] Erica Blum, Jonathan Katz, and Julian Loss. 2020. Network-Agnostic State Machine Replication. arXiv:2002.03437 [cs.CR]
- [21] Yehonatan Buchnik and Roy Friedman. 2020. FireLedger: a high throughput blockchain consensus protocol. *Proc. VLDB Endow* 13, 9 (may 2020), 1525–1539.
- [22] Frank Cangialosi, Dave Levin, and Neil Spring. 2015. Ting: Measuring and Exploiting Latencies Between All Tor Nodes. In *Proc. of the ACM Internet Measurement Conf. (IMC)*. ACM, New York, NY, USA, 289–302.
- [23] Carlos Carvalho, Daniel Porto, Luis Rodrigues, Manuel Bravo, and Alysson Bessani. 2018. Dynamic adaptation of Byzantine consensus protocols. In *Proc. of the 33rd Ann. ACM Symp. on Applied Computing (SAC)*. ACM, New York, NY, USA, 411–418.
- [24] Daniel Cason, Enrique Fynn, Nenad Milosevic, Zarko Milosevic, Ethan Buchman, and Fernando Pedone. 2021. The design, architecture and performance of the Tendermint Blockchain Network. In *Proc. of the 40th IEEE Int. Symp. on Reliable Distributed Systems (SRDS)*. IEEE, Piscataway, NJ, USA, 23–33.
- [25] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine fault tolerance. In *Proc. of the 3rd Symp. on Operating Systems Design and Implementation (OSDI)*. USENIX Association, Berkeley, CA, 173–186.
- [26] Tairi Chiba, Ren Ohmura, and Junya Nakamura. 2022. Network Bandwidth Variation-Adapted State Transfer for Geo-Replicated State Machines and its Application to Dynamic Replica Replacement. *Concurrency and Computation: Practice and Experience* Early view (2022), e7408.
- [27] Pierre Civit, Seth Gilbert, and Vincent Gramoli. 2021. Polygraph: Accountable Byzantine agreement. In *Proc. of the 41st IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*. IEEE, Piscataway, NJ, USA, 403–413.
- [28] Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, and Jovan Komatovic. 2022. As easy as ABC: Optimal (A)ccountable (B)yzantine (C)onsensus is easy!. In *Proc. of the 37th IEEE Int. Parallel and Distributed Processing Symp. (IPDPS)*. IEEE, Piscataway, NJ, USA, 560–570.
- [29] Paulo Coelho and Fernando Pedone. 2018. Geographic State Machine Replication. In *Proc. of the 37th IEEE Int. Symp. on Reliable Distributed Systems (SRDS)*. IEEE, Piscataway, NJ, USA, 221–230.
- [30] Tyler Crain, Christopher Natoli, and Vincent Gramoli. 2021. Red Belly: A secure, fair and scalable open blockchain. In *Proc. of the 42nd IEEE Symp. on Security and Privacy (S&P)*. IEEE, Piscataway, NJ, USA, 466–483.
- [31] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. 2022. DAMYSUS: streamlined BFT consensus leveraging trusted components. In *Proc. of the 17th European Conf. on Computer Systems (EuroSys) (Rennes, France) (EuroSys '22)*. ACM, New York, NY, USA, 1–16.
- [32] Tobias Distler, Christian Cachin, and Rüdiger Kapitza. 2015. Resource-efficient Byzantine fault tolerance. *IEEE Trans. on Computers (TC)* 65, 9 (2015), 2807–2819.
- [33] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the presence of partial synchrony. *J. ACM* 35, 2 (1988), 288–323.
- [34] Michael Eischer and Tobias Distler. 2018. Latency-Aware Leader Selection for Geo-Replicated Byzantine Fault-Tolerant Systems. In *Proc. of the 1st Workshop on Byzantine Consensus and Resilient Blockchains (BCRB)*. IEEE Computer Society, Los Alamitos, CA, USA, 140–145.
- [35] Michael Eischer and Tobias Distler. 2020. Resilient cloud-based replication with low latency. In *Proc. of the 21st Int. Middleware Conf.* ACM, New York, NY, USA, 14–28.
- [36] Michael Eischer, Benedikt Straßner, and Tobias Distler. 2020. Low-latency geo-replicated state machines with guaranteed writes. In *Proc. of the 7th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC)*. ACM, New York, NY, USA, 1–9.
- [37] Roy Friedman, Achour Mostefaoui, and Michel Raynal. 2005. Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems. *IEEE Trans. on Dependable and Secure Computing (TDSC)* 2, 1 (2005), 46–56.
- [38] Yingzi Gao, Yuan Lu, Zhenliang Lu, Jing Xu Qiang Tang, and Zhenfeng Zhang. 2022. Dumbo-NG: Asynchronous Consensus with Throughput-Oblivious Latency. In *Proc. of the 29th ACM Conf. on Computer and Communications Security (CCS)*. ACM, New York, NY, USA, 1187–1201.
- [39] Paulo Gouveia, João Neves, Carlos Segarra, Luca Liechti, Shady Issa, Valerio Schiavoni, and Miguel Matos. 2020. Kollaps: decentralized and dynamic topology emulation. In *Proc. of the 15th European Conf. on Computer Systems (EuroSys)*. ACM, New York, NY, USA, 1–16.
- [40] Vincent Gramoli, Rachid Guerraoui, Andrei Lebedev, Chris Natoli, and Gauthier Voron. 2023. Diablo: A Benchmark Suite for Blockchains. In *Proc. of the 18th European Conf. on Computer Systems (EuroSys) (Rome, Italy)*. IEEE, Piscataway, NJ, USA, 540–556.
- [41] Rachid Guerraoui, Matej Pavlovic, and Dragos-Adrian Seredinschi. 2016. Incremental consistency guarantees for replicated objects. In *Proc. of the 12th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*. USENIX Association, Berkeley, CA, 169–184.
- [42] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2019. SBFT: a scalable and decentralized trust infrastructure. In *Proc. of the 49th Ann. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE, Piscataway, NJ, USA, 568–580.
- [43] Suyash Gupta, Jelle Hellings, Sajjad Rahnama, and Mohammad Sadoghi. 2019. Proof-of-Execution: Reaching Consensus through Fault-Tolerant Speculation. In *Int. Conf. on Extending Database Technology*. ACM, New York, NY, USA, 13 pages. <https://openproceedings.org/2021/conf/edbt/p111.pdf>
- [44] Suyash Gupta, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. 2020. ResilientDB: global scale resilient blockchain fabric. *Proc. of the VLDB Endowment* 13, 6 (2020), 868–883.
- [45] Maurice P. Herlihy and Jeannette M. Wing. 1990. Linearizability: A correctness condition for concurrent objects. *ACM Trans. on Programming Languages and Systems (TOPLAS)* 12, 3 (1990), 463–492.
- [46] Heidi Howard, Aleksey Charapko, and Richard Mortier. 2021. Fast Flexible Paxos: Relaxing quorum intersection for Fast Paxos. In *Proc. of the 22nd Int. Conf. on Distributed Computing and Networking (ICDCN)*. ACM, New York, NY, USA, 186–190.

- [47] Rob Jansen, Jim Newsome, and Ryan Wails. 2022. Co-opting Linux Processes for High-Performance Network Simulation. In *Proc. of the USENIX Ann. Technical Conf. (USENIX ATC)*. USENIX Association, Berkeley, CA, 327–350.
- [48] Flavio Junqueira, Yanhua Mao, and Keith Marzullo. 2007. Classic Paxos vs. fast Paxos: caveat emptor. In *Proc. of USENIX Hot Topics in System Dependability (HotDep)*. USENIX Association, Berkeley, CA, 6 pages.
- [49] Katharina Kohls and Claudia Diaz. 2022. VerLoc: Verifiable Localization in Decentralized Systems. In *Proc. of the 31st USENIX Security Symp. (USENIX Security)*. USENIX Association, Berkeley, CA, 2637–2654.
- [50] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2007. Zyzzyva: speculative Byzantine fault tolerance. In *Proc. of the 21st ACM SIGOPS Symp. on Operating Systems Principles (SOSP)*. ACM, New York, NY, USA, 45–58.
- [51] Petr Kuznetsov, Andrei Tonkikh, and Yan X Zhang. 2021. Revisiting optimal resilience of fast Byzantine consensus. In *Proc. of the 40th ACM Symp. on Principles of Distributed Computing (PODC)*. ACM, New York, NY, USA, 343–353.
- [52] Shengyun Liu and Marko Vukolić. 2017. Leader Set Selection for Low-Latency Geo-Replicated State Machine. *IEEE Trans. on Parallel and Distributed Systems (TPDS)* 28, 7 (2017), 1933–1946.
- [53] Yuan Lu, Zhenliang Lu, and Qiang Tang. 2022. Bolt-Dumbo Transformer: Asynchronous Consensus As Fast As Pipelined BFT. In *Proc. of the 29th ACM Conf. on Computer and Communications Security (CCS)*. ACM, New York, NY, USA, 2159–2173.
- [54] Dahlia Malkhi and Michael Reiter. 1998. Byzantine quorum systems. *Distributed computing* 11, 4 (1998), 203–213.
- [55] Yanhua Mao, Flavio P. Junqueira, and Keith Marzullo. 2009. Towards low latency state machine replication for uncivil wide-area networks. In *Proc. of the 5th Workshop on Hot Topics in System Dependability (HotDep)*. USENIX Association, Berkeley, CA, 6 pages.
- [56] Yanhua Mao, Flavio P. Junqueira, and Keith Marzullo. 2008. Mencius: Building Efficient Replicated State Machines for WANs. In *Proc. of the 8th USENIX Conf. on Operating Systems Design and Implementation (OSDI)*. USENIX Association, Berkeley, CA, 369–384.
- [57] Jean-Philippe Martin and Lorenzo Alvisi. 2006. Fast Byzantine consensus. *IEEE Trans. on Dependable and Secure Computing (TDSC)* 3, 3 (2006), 202–215.
- [58] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>
- [59] Ray Neiheiser, Miguel Matos, and Luis Rodrigues. 2021. Kauri: Scalable BFT consensus with pipelined tree-based dissemination and aggregation. In *Proc. of the 28th ACM SIGOPS Symp. on Operating Systems Principles (SOSP)*. ACM, New York, NY, USA, 35–48.
- [60] Ray Neiheiser, Luciana Rech, Manuel Bravo, Luis Rodrigues, and Miguel Correia. 2020. Fireplug: Efficient and Robust Geo-Replication of Graph Databases. *IEEE Trans. on Parallel and Distributed Systems (TPDS)* 31, 8 (2020), 1942–1953.
- [61] Martin Nischwitz, Marko Esche, and Florian Tschorsch. 2022. Raising the AWARENESS of BFT Protocols for Soaring Network Delays. In *Proc. of the 47th IEEE Conf. on Local Computer Networks (LCN)*. IEEE, Piscataway, NJ, USA, 387–390.
- [62] Shota Numakura, Junya Nakamura, and Ren Ohmura. 2019. Evaluation and Ranking of Replica Deployments in Geographic State Machine Replication. In *Proc. of the 38th IEEE Int. Symp. on Reliable Distributed Systems Workshops (SRDSW)*. IEEE, Piscataway, NJ, USA, 37–42.
- [63] Alejandro Ranchal-Pedrosa and Vincent Gramoli. 2023. Basilic: Resilient-Optimal Consensus Protocols with Benign and Deceitful Faults. In *Proc. of the 36th IEEE Computer Security Foundations Symp. (CSF)*. IEEE, Piscataway, NJ, USA, 15–30.
- [64] Fred B. Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)* 22, 4 (1990), 299–319.
- [65] Alex Shamis, Peter Pietzuch, Burcu Canakci, Miguel Castro, Cédric Fournet, Edward Ashton, Amaury Chamayou, Sylvain Clebsch, Antoine Delignat-Lavaud, Matthew Kerner, Julien Maffre, Olga Vrousseau, Christoph M. Wintersteiger, Manuel Costa, and Mark Russinovich. 2022. IA-CCF: Individual Accountability for Permissioned Ledgers. In *Proc. of the 19th USENIX Symp. on Networked Systems Design and Implementation (NSDI)*. USENIX Association, Berkeley, CA, 467–491.
- [66] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. 2021. BFT protocol forensics. In *Proc. of the 28th ACM Conf. on Computer and Communications Security (CCS)*. ACM, New York, NY, USA, 1722–1743.
- [67] Douglas Simões Silva, Rafal Graczyk, Jérémie Decouchant, Marcus Völz, and Paulo Esteves-Verissimo. 2021. Threat adaptive Byzantine fault tolerant state-machine replication. In *Proc. of the 40th IEEE Int. Symp. on Reliable Distributed Systems (SRDS)*. IEEE, Piscataway, NJ, USA, 78–87.
- [68] Atul Singh, Tathagata Das, Petros Maniatis, Peter Druschel, and Timothy Roscoe. 2008. BFT Protocols Under Fire. In *Proc. of the 5th USENIX Symp. on Networked Systems Design and Implementation*, Vol. 8. USENIX Association, Berkeley, CA, 189–204.
- [69] Yee Jiun Song and Robbert van Renesse. 2008. Bosco: One-step Byzantine asynchronous consensus. In *Proc. of the 22nd Int. Symp. on Distributed Computing (DISC)*. Springer-Verlag, Berlin, Heidelberg, 438–450.
- [70] João Sousa and Alysson Bessani. 2015. Separating the WHEAT from the Chaff: An Empirical Design for Geo-Replicated State Machines. In *Proc. of the 34th IEEE Int. Symp. on Reliable Distributed Systems (SRDS)*. IEEE, Piscataway, NJ, USA, 146–155.
- [71] Chrysoula Stathakopoulou, Tudor David, and Marko Vukolić. 2019. Mir-BFT: High-throughput BFT for blockchains. , 18 pages. arXiv:1906.05552v2 [cs.DC]
- [72] Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolić. 2022. State Machine Replication Scalability Made Simple. In *Proc. of the 17th European Conf. on Computer Systems (EuroSys)* (Rennes, France). IEEE, Piscataway, NJ, USA, 17–33.
- [73] Xiao Sui, Sisi Duan, and Haibin Zhang. 2022. Marlin: Two-Phase BFT with Linearity. In *Proc. of the 52nd Ann. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE, Piscataway, NJ, USA, 54–66.
- [74] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. 2010. EBAWA: Efficient Byzantine Agreement for Wide-Area Networks. In *Proc. of the 12th IEEE Int. Symp. on High Assurance Syst. Eng. (HASE)*. IEEE, Piscataway, NJ, USA, 10–19.
- [75] Benjamin Wester, James Cowling, Edmund B. Nightingale, Peter M. Chen, Jason Flinn, and Barbara Liskov. 2009. Tolerating Latency in Replicated State Machines Through Client Speculation.. In *Proc. of the 6th USENIX Symp. on Networked Systems Design and Implementation (NSDI)*. USENIX Association, Berkeley, CA, 245–260.
- [76] Wassim Yahyaoui, Joachim Bruneau-Queyreyx, Marcus Völz, and Jérémie Decouchant. 2024. Tolerating Disasters with Hierarchical Consensus. In *IEEE Int. Conf. on Computer Communications (INFOCOM 2024)*. IEEE, Vancouver, Canada, IEEE, Piscataway, NJ, USA, 1241–1250.
- [77] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proc. of the 38th ACM Symp. on Principles of Distributed Computing (PODC)*. ACM, New York, NY, USA, 347–356.
- [78] Haibin Zhang and Sisi Duan. 2022. PACE: Fully Parallelizable BFT from Reproducible Byzantine Agreement. In *Proc. of the 29th ACM Conf. on Computer and Communications Security (CCS)*. ACM, New York, NY, USA, 3151–3164.
- [79] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. 2020. Byzantine Ordered Consensus without Byzantine Oligarchy. In *Proc. of the 14th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*. USENIX Association, Berkeley, CA, 633–649.