PACE: Fully Parallelizable BFT from Reproposable Byzantine Agreement

Sisi Duan
Tsinghua University
duansisi@mail.tsinghua.edu.cn

Haibin Zhang
Beijing Institute of Technology
haibin@bit.edu.cn

ABSTRACT

Reaching consensus is the single most crucial problem in fault-tolerant distributed computing. This paper studies asynchronous consensus with Byzantine failures commonly known as asynchronous binary Byzantine agreement (ABA). ABA is a key component as well as a bottleneck for (almost) all known asynchronous Byzantine fault-tolerant (BFT) protocols and asynchronous multi-party computation (MPC) with guaranteed output. This paper addresses two significant problems in ABA: we propose the first common-coin based ABA with 2 steps only in a round and the first practical solution on running ABA in a fully parallelizable manner.

We first present Pillar, a new round-based ABA protocol using common coins, having 2 steps in a round. Pillar can be directly used to improve all practical asynchronous BFT protocols implemented and MPC protocols achieving guaranteed output. To demonstrate the performance of Pillar, we use it in the BEAT protocol based on the classic framework of Ben-Or, Kemler, and Rabin and HoneyBadgerBFT, and implement a new BFT protocol, called ACE, providing up to 2x the throughput of BEAT.

We go on to suggest *reproposable ABA* (RABA), a generalized ABA primitive that allows a replica to change its mind and vote twice. In contrast to conventional ABA, RABA requires the protocol to be biased towards 1, but does not require external validity (via, e.g., expensive threshold signatures). We use Pillar to build Pisa, a signature-free RABA protocol that is as efficient as Pillar. We also show how to turn some other ABA protocols (including the one by Cachin, Kursawe, and Shoup) to RABA. We then propose a novel asynchronous BFT framework built on reliable broadcast (RBC) and RABA. This leads to the first fully parallelizable asynchronous BFT protocol, allowing *all* ABA instances to run in a strictly concurrent manner. Our concrete instantiation, called PACE, consistently and significantly outperforms existing asynchronous BFT protocols in terms of all metrics, having much fewer steps than all asynchronous BFT implemented for all practically large systems. PACE provides 6x the throughput of BEAT when there are 91 replicas. Such a solution can be *directly* used as a solution to run ABA in parallel, a procedure needed not just in BFT but in interactive consistency and MPC with guaranteed output. Our result, therefore, identifies RABA as a first-class primitive in distributed computing.

Also, the PACE framework lays the foundation on information-theoretic asynchronous BFT and asynchronous BFT without public-key cryptography.

KEYWORDS

asynchronous BFT, binary consensus, blockchain, fault tolerance

1 INTRODUCTION

The consensus problem is one of the most fundamental problems in fault-tolerant distributed computing. Consensus allows processes in a distributed system to agree on a common value out of the values they propose. The consensus problem was formally introduced in the context of Byzantine failures by Lamport, Shostak, and Pease [43]. This paper focuses on the asynchronous binary consensus problem with Byzantine failures, commonly known as "asynchronous binary Byzantine agreement (ABA)." As a fundamental primitive, ABA has been extensively studied (e.g., [13, 16, 20, 24, 37, 48, 51, 53, 56, 58, 60]). ABA has recently drawn renewed attention, as it is the core component for asynchronous BFT (atomic broadcast) and BFT is deemed as the standard model for permissioned blockchains [6, 23, 57, 61, 62]. Indeed, all known efficient asynchronous BFT protocols ever implemented (except [32]) *directly* rely on ABA as a building block [19, 34, 39, 44, 49, 50]. Moreover, as demonstrated in RITAS [50], BEAT [34], and Dumbo [39], ABA is actually the major bottleneck for all these BFT systems. ABA is also the key building block in asynchronous multi-party computation (MPC) with guaranteed output [10, 12, 26, 46]. Very often, the BFT and MPC protocols mentioned above [10, 12, 26, 34, 44, 46, 49], together with protocols such as interactive consistency [11], need to run ABA in parallel. How to run ABA in a *practical* and *fully* parallelizable way is also a major open problem in the area.

This paper improves the ABA primitive itself and presents a new and practical paradigm for running ABA in a fully parallelizable manner. In particular, we propose the first ABA protocol using 2 steps in a round from regular common coins and assuming authenticated channels only. We propose a novel and generalized ABA primitive RABA ("allowing one to change its vote") and use it as a building block for parallel ABA. These results allow us to build significantly more efficient asynchronous BFT and MPC protocols and any protocols using ABA.

A close look at ABA. The ABA protocol by Mostefaoui, Moumen, and Raynal (MMR) [51] is known as the most efficient common-coin based ABA protocol. It relies on authenticated channels only, terminating in 2 rounds on average (completing within $O(r)$ rounds with probability $1-2^{-r}$). In each round, MMR ABA has 2 or 3 steps (without counting the step for common coin). Asynchronous BFT protocols, such as HoneyBadgerBFT [49] and BEAT [34], in their proceeding versions, utilize MMR ABA.

MMR ABA, however, has a liveness issue reported independently in [1, 59]. In particular, the protocol assumes perfect random coins completely independent of the state of all correct replicas at the point when they query the coin. The property cannot be guaranteed by any cryptographic coin-flipping protocols. A malicious network scheduler can force correct replicas to enter the next round of consensus with inconsistent values, causing the protocol not to terminate.

The journal version of MMR [52] does not have the issue but has 9 to 13 steps for each round. Cobalt ABA [48] also provides a solution by modifying MMR ABA, having 3 or 4 steps in each round. Recent BFT implementations including EPIC [44] and Dumbo [39] use Cobalt ABA. Crain also proposes an ABA protocol assuming authenticated channels, but uses a high-threshold common coin protocol which is less efficient to construct [4, 33] and relies on stronger assumptions [20].

A close look at asynchronous BFT. Because of its intrinsic robustness, completely asynchronous BFT has been extensively studied [2, 12, 19, 42, 50, 54]. Having long been viewed as a "theoretical" approach, several recent asynchronous BFT systems—such as HoneyBadgerBFT [49], BEAT [34], Dumbo [39], and EPIC [44]—have shown their performance becomes comparable to their partially synchronous counterparts.

Efficient asynchronous BFT protocols may be roughly divided into two categories: the BKR (Ben-Or, Kelmer, and Rabin) paradigm [12] including HoneyBadgerBFT, BEAT, and EPIC, and the CKPS (Cachin, Kusawe, Petzold, and Shoup) paradigm [19] including SINTRA [21] and Dumbo. **Both paradigms have their benefits and drawbacks: the BKR framework is information-theoretically (IT) secure (if assuming an IT common coin protocol) and achieves quantum safety (as defined in [40]); it has an $O(\log n)$ running time. The CKPS framework is computationally secure; it has an $O(1)$ running time but a large hidden constant.**

Neglecting the security model, even just considering performance, the "common belief" that there is no "one-size-fits-all" BFT protocol has, thus far, remained true for the case of asynchronous BFT. For instance, a recent (state of the art) asynchronous BFT, Dumbo, largely follows but refines CKPS by reducing its communication complexity, at the price of four more all-to-all communications and $O(n^3)$ expensive pairing-based threshold signatures (a pairing-based signature is about 10x slower than a conventional signature). Dumbo, however, has 14 steps even in the best-case scenario (where the ABA instance terminates in one round) and on average 32 steps. The protocols following the BKR paradigm terminate in $O(\log n)$ rounds but only 6 or 7 steps in the best-case scenario. It was shown that Dumbo outperforms HoneyBadgerBFT when n is large in the WAN setting, but Dumbo is less efficient than BEAT (being more efficient than HoneyBadgerBFT) in other scenarios, for instance, when n is small or when there is no contention. Moreover, while one could make protocols in the BKR paradigm adaptively secure as shown in EPIC [44], it is challenging to build practical BFT protocols with adaptive security from CKPS or Dumbo. In particular, the CKPS paradigm uses expensive threshold cryptography extensively, and it would be inefficient to replace these cryptographic operations using much more expensive adaptively secure cryptography [8]. In addition, it is well known that intractability problems in groups with bilinear pairings (used in HoneyBadgerBFT, Dumbo, and EPIC) are weaker than the conventional discrete logarithm problems in elliptic curve groups (used in BEAT).

This work improves upon the BKR framework and all protocols along this line of research (e.g., all BKR descendants, all multi-party computation using BKR, asynchronous distributed key generation [33, 41], interactive consistency).

This work enables asynchronous BFT without public-key cryptography, information-theoretic asynchronous BFT, highly efficient asynchronous distributed key generation, and many other high-level Byzantine-resilient protocols.

1.1 Our Technical Contributions

A nearly optimal common-coin based ABA. We first present Pillar, a new ABA protocol using authenticated channel and regular threshold common coins. Pillar has 2 or 3 steps in a round, in contrast to the state-of-the-art Cobalt ABA protocol requiring 3 or 4 steps or Crain's ABA requiring high-threshold common coins [31] (which, as we will argue, are more difficult to construct). We develop new techniques of building ABA. As demonstrated in prior works, ABA is the bottleneck for all BFT implementations available. Pillar, therefore, can be used to improve all these protocols, including HoneyBadgerBFT, BEAT, Dumbo, and EPIC. Pillar can also be used to improve asynchronous MPC protocols known [10, 12, 26, 46].

Note that it is fair to say Pillar and MMR ABA have 2 steps, as the additional step may or may not be triggered. The situation, especially for the message pattern, is the same as that of Bracha's broadcast [16]: Bracha's broadcast has 3 or 4 steps, but we often say it is a 3-step protocol. It is easy to show that ABA with 1 step per round is unachievable assuming optimal resilience, so the best one can hope for is a 2-step ABA.

RABA. We suggest *reproposable ABA* (RABA), a new ABA primitive allowing replicas to change votes if needed. We utilize Pillar to build Pisa, a RABA protocol as efficient as Pillar. We view RABA as a first-class and powerful primitive, one is essential to improved BFT in this paper and other applications we briefly introduce shortly. We also show in the Appendix how to transform some other ABA protocols (e.g., [20]) to RABA protocols.

A new framework for parallel ABA, BFT, interactive consistency, and MPC. The paper uses RBC and ABA as a black box to build a new paradigm for parallel ABA.

- The framework leads to the first fully parallelizable asynchronous BFT protocol, allowing *all* ABA instances to run in a strictly concurrent way. The improvement can increase the throughput dramatically but also reduce the latency *concretely*. In fact, if calculating the concrete time complexity (exposing all hidden terms), we find the instantiations from our framework outperform existing protocols ever implemented for practically large n's, in both best-case and average-case scenarios, and in terms of the number of cryptographic operations (Table 1 and Table 5 in Appendix A).
- The framework can be directly applied to interactive consistency [11] without modification.
- The framework improves the state-of-the-art asynchronous common subset (ACS) framework used in MPC or asynchronous distributed key generation. Note that the CKPS framework cannot be used in these works.

We rigorously prove the correctness of all our protocols.

	IT secure (using IT common coin)?	quantum safety?	cryptographic assumption	steps $n = 7$		steps $n = 31$		steps $n = 91$		steps $n = 301$	
				best	avg	best	avg	best	avg	best	avg
HoneyBadgerBFT [49]	√	√	pairing	6	22.38	6	34.45	6	43.61	6	53.91
BEAT [34]	√	√	CDH	6	22.38	6	34.45	6	43.61	6	53.91
EPIC [44]	√	√	pairing	6	22.38	6	34.45	6	43.61	6	53.91
Dumbo [39]			pairing	15	32	15	32	15	32	15	32
Dumbo-Pillar (Pillar this paper)			pairing	14	26	14	26	14	26	14	26
ACE (this paper)	√	√	CDH	5	15.92	5	23.96	5	30.07	5	36.94
PACE (this paper)	√	√	CDH	4	11.48	4	15.62	4	18.69	4	22.13

Table 1: Some asynchronous BFT protocols. Dumbo-Pillar is identical to Dumbo, expect that Dumbo-Pillar uses our more efficient Pillar ABA as the underlying ABA. IT security means information-theoretic security. HoneyBadgerBFT and BEAT, strictly speaking, do not attain IT security even if assuming IT secure common coins; they can be made IT secure if following the technique of EPIC in selecting transactions, so we have "yes" for them for IT security. Quantum safety is defined in DAG-Rider [40]; quantum security = quantum safety + quantum liveness. We illustrate the number of steps with a system of n nodes using several examples. "Best" means the best possible case, while "avg" represents the expected number of steps. We show in Appendix A that the expected number of steps for PACE is larger than that of Dumbo-Pillar only when $n > 1{,}000$.

1.2 Our Practical Contributions

On the practical side, we implement two new efficient asynchronous BFT systems—ACE and PACE. ACE follows the classic framework of Ben-Or, Kemler, and Rabin [12] and is identical to BEAT except that the ABA phase now uses Pillar. ACE provides up to 2x the throughput of BEAT. PACE uses our new framework and Pisa. PACE significantly outperforms existing asynchronous BFT protocols in terms of *all* metrics (latency under no contention, throughput, latency vs. throughput, and scalability).

It is evident that when n is small, PACE has much fewer steps than Dumbo and others. Even when $n = 91$, PACE needs 19 steps on average, while Dumbo and BEAT have 32 and 44 steps, respectively. In fact, Dumbo, even if using our improved ABA, would have fewer steps only when n >1,000. The significant improvement is due to the more efficient ABA protocol and our new framework that removes the BKR phase bottleneck. Also, PACE is pairing-free, while Dumbo requires 8.54×10^5 expensive pairings operations. In particular, PACE provides 6.1x the throughput of BEAT for the case of $n = 91$ in WANs.

Remarkably, both ACE and PACE rely on a well-studied, standard, and pairing-free cryptographic assumption (Computational Diffie-Hellman for elliptic curves), achieving standard 128-bit security. Both of them can be easily made adaptively secure at relatively low cost as in EPIC [44] and made quantum safe (but not quantum live) [40]. These features are in sharp contrast to constructions derived from the CKPS paradigm relying heavily on expensive, computationally secure threshold signatures.

1.3 Derived and Up-Coming Works

We view RABA and the new BFT framework from RBC and RABA as important contributions to fault-tolerant distributed computing. This work is the first of several we have:

- The BFT framework introduced here can be used to build the first practical asynchronous BFT protocol—WaterBear, which can be implemented and achieve information-theoretic security [35]. We also provide a quantum secure protocol– WaterBear-QS, which only relies on authenticated channels and hash functions. The two protocols also achieve adaptive security, assuming no dealers or trusted setup. They offer comparable performance to state-of-the-art asynchronous BFT protocols.
- The aforementioned protocols can be used to construct practical asynchronous distributed key generation (ADKG) protocols. Implementations are underway.
- The RABA protocol itself is of particular interest and can be used to construct various new high-level protocols with either asymptotically or concretely better efficiency.

2 SYSTEM MODEL AND DEFINITIONS

We consider distributed computing protocols, where f out of n replicas may fail arbitrarily (Byzantine failures). The protocols we consider in this work (ABA, BFT, and RBC) assume $f \leq \lfloor \frac{n-1}{3} \rfloor$, which is optimal. We consider completely asynchronous systems making no timing assumptions on message processing or transmission delays. A (Byzantine) *quorum* is a set of $\lceil \frac{n+f+1}{2} \rceil$ replicas. For simplicity, we may assume $n = 3f + 1$ and a quorum size of $2f + 1$. In our protocols, we may associate each protocol instance with a unique session identifier *sid*, tagging each message in the protocol with *sid*; we may omit these identifiers when no ambiguity arises.

Asynchronous (binary) Byzantine agreement (ABA). An ABA abstraction is specified by *propose* and *decide*. Each replica proposes an initial binary value (*vote*) for consensus and replicas will decide on some value. ABA should satisfy the following properties:

- **Validity**: If all correct replicas *propose* v, then any correct replica that terminates *decides* v.
- **Agreement**: If a correct replica *decides* v, then any correct replica that terminates *decides* v.
- **Termination**: Every correct replica eventually *decides* some value.
- **Integrity**: No correct replica *decides* twice.

Validated Byzantine agreement (VBA). Conventional ABA implements a binary decision. CKPS [19] considers validated Byzantine agreement (VBA) and uses VBA to build efficient Byzantine atomic broadcast protocols. VBA generalizes ABA in two aspects. First, decisions in ABA can be a value from an arbitrarily large set. Second, VBA has a validity condition, *external validity*, encompassing the standard validity condition as a special case. In VBA, each replica adds some validation data π to the proposed value v and the decided value (v, π) satisfies a global predicate Q known to all parties such that $Q(v, \pi)$ holds. Such a validation proof, in practice, can be constructed using, e.g., threshold signatures. A VBA with predicate Q satisfies the following properties:

- **External validity**: Every correct replica that terminates *decides* v validated by π such that $Q(v, \pi)$ holds.
- **Agreement**: If a correct replica *decides* v, then any correct replica that terminates *decides* v.
- **Termination**: Every correct replica eventually *decides* some value.
- **Integrity**: If all replicas follow the protocol, and if a replica *decides* v validated by π, then some replica *proposed* v validated by π.

As VBA explicitly assumes a computationally bounded adversary, CKPS additionally bounds the running time of the adversary by defining the following property:

- **Efficiency**: The communication complexity of the protocol is probabilistically uniformly bounded [19].[1]

CKPS also defines for VBA on binary values (namely, validated ABA) an additional and *optional* validity property, "ABA biased towards 1," which we simply call *biased validity*.

- **Biased validity**: If at least $f + 1$ correct replicas *propose* 1, then all correct replicas that terminate *decide* 1.

This paper does not use VBA but uses a new ABA primitive, called RABA, which we will introduce in Sec. 7.1. We will compare VBA and RABA in detail.

BFT. We use BFT and (Byzantine) atomic broadcast interchangeably. Syntactically, in BFT, a replica *a-delivers* (atomically deliver) *transactions*, each *submitted* by some client. The client computes a final response to its submitted transaction from its responses from replicas. The correctness of a BFT protocol is specified as follows:

- **Agreement**: If any correct replica *a-delivers* a transaction tx, then every correct replica *a-delivers* tx.
- **Total order**: If a correct replica *a-delivers* a message tx before *a-delivering* tx', then no correct replica *a-delivers* a message tx' without first *a-delivering* tx.
- **Liveness**: If a transaction tx is *submitted* to all correct replicas, then all correct replicas eventually *a-deliver* tx.

RBC. We review the definition of Byzantine reliable broadcast (RBC). A RBC protocol is specified by *r-broadcast* and *r-deliver* such that the following properties hold:

- **Validity**: If a correct replica p *r-broadcasts* a message m, then p eventually *r-delivers* m.
- **Agreement**: If some correct replica *r-delivers* a message m, then every correct replica eventually *r-delivers* m.
- **Integrity**: For any message m, every correct replica *r-delivers* m at most once. Moreover, if a replica *r-delivers* a message m with sender s, then m was previously *r-broadcast* by replica s.

The paper uses AVID RBC of Cachin and Tessaro [22] that achieves $O(n|m| + \lambda n^2 \log n)$ communication, where λ is a security parameter.

Common coins and thresholds. The common coin protocol outputs a binary value at each correct replica [53]. These protocols can be divided into common coins with the regular thresholds ($f + 1$) and common coins with the high thresholds ($2f + 1$). Protocols from both regular coins [48, 51] and high-threshold coins [20, 30, 31] have been proposed.

It is preferred to use regular common coin protocols over high-threshold common coin protocols (from both theoretical perspective and practical perspective): first, without assuming a trusted setup, it is more expensive to build decentralized key generation protocols for high-threshold common coins than for regular common coins, as high-threshold asynchronous verifiable (complete) secret sharing is more expensive than the regular one [4, 33]. Second, even assuming the trusted setup model, regular common coin protocols are easier to construct. For instance, the most efficient common coin protocols due to Cachin, Kursawe, and Shoup [20] include a regular common coin protocol using the Computational Diffie-Hellman (CDH) assumption and a high-threshold common coin protocol that must use the (stronger) Decisional Diffie-Hellman (DDH) assumption.

3 REVIEWING EFFICIENT ASYNCHRONOUS BFT AND IDENTIFYING THEIR BOTTLENECKS

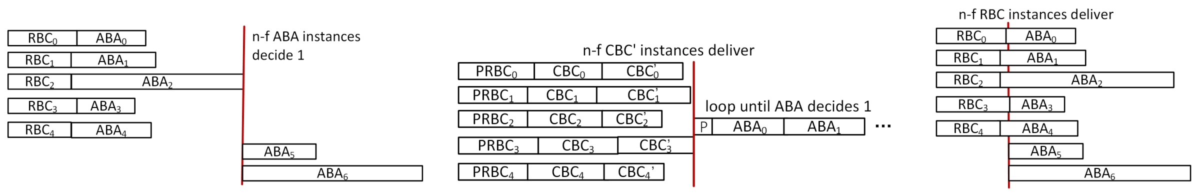
We roughly divide efficient asynchronous BFT approaches into two categories: the BKR (Ben-Or, Kelmer, and Rabin) paradigm [12] and the CKPS (Cachin, Kusawe, Petzold, and Shoup) paradigm [19]. HoneyBadgerBFT [49], BEAT [34], and EPIC [44] fall into the BKR paradigm, while SINTRA [21] and Dumbo [39] rely on the CKPS paradigm. Both approaches proceed in epochs. In each epoch, all replicas can propose transactions from their transaction pool as a proposal. They then agree on a set containing the union of the proposals proposed by replicas.

From a technical perspective, however, the two methods are *significantly* different, leading to BFT protocols with *significantly* different performance and security features.

3.1 The BKR Paradigm

The paradigm reduces asynchronous BFT to RBC and ABA. In each epoch, all replicas first run an RBC phase to reliably broadcast their proposals. Then they run an ABA phase, where n parallel ABA instances are invoked. The i-th ABA instance agrees on whether the proposal of replica p_i has been delivered in the RBC phase: Upon RBC delivery of a proposal from p_j, the replica proposes 1 to the j-th ABA instance. If a correct replica p_j decides 1 for the i-th ABA instance, the proposal from p_i is delivered. Otherwise, the proposal is not included.

[1]Let X be a random variable. We say X is probabilistically uniformly bounded (by T), if there exist a fixed polynomial $T(k)$ and fixed negligible functions $\delta(l)$ and $\epsilon(k)$ such that for all $l, k \geq 0$, $\Pr[X > lT(k)] \leq \delta(l) + \epsilon(k)$.



(a) BKR (HoneyBadgerBFT, BEAT, EPIC). **(b) CKPS, Dumbo, Dumbo-Pillar.** **(c) Our new paradigm.**

Figure 1: Visualizing asynchronous BFT. Assume there are 7 replicas (2 of which may have failed). RBC, CBC, and PRBC stand for reliable broadcast, consistent broadcast, and provable reliable broadcast, respectively. RBC and CBC have three steps, and PRBC has four steps. The BKR paradigm has two ABA subphases: replicas have to wait for the slowest ABA in the first subphase to terminate before invoking the second subphase. Dumbo and Dumbo-Pillar (built on top of CKPS) have one PRBC phase (4 steps), two CBC phase (2× 3 steps), one permutation step ("P"), and three sequential distribution (1 step) and ABA (2 for Pillar or 3 for Cobalt ABA) instances. (So Dumbo and Dumbo-Pillar have on average 32 and 26 steps, respectively.)

To guarantee throughput, the BKR paradigm requires that if a replica has not received some proposals during the RBC phase, the replica *abstains* from proposing 0 until $n-f$ ABA instances terminate with 1. This method ensures that proposals from at least $n-f$ replicas are delivered. Otherwise, the throughput could be 0. The BKR paradigm breaks the parallelism of the "RBC+ABA" structure, a performance bottleneck repeatedly pointed out by prior works [9, 11, 49]. As shown in Figure 1a, the ABA phase has two subphases: replicas have to wait until at least $n-f$ ABA instances terminate with 1 and then invoke the remaining ABA instances with 0.

HoneyBadgerBFT [49]. HoneyBadgerBFT can be regarded as the first practical instantiation of the BKR paradigm. As mentioned by the authors, their protocol "refute(s) the prevailing wisdom that such (asynchronous BFT) protocols are necessarily impractical." HoneyBadgerBFT cherrypicks a bandwidth-efficient RBC, the AVID broadcast of Cachin and Tessaro [22], and the MMR ABA [51]. In particular, HoneyBadgerBFT uses a pairing-based threshold signature scheme [14] to generate common coins for MMR. HoneyBadgerBFT uses a pairing-based threshold encryption scheme to achieve liveness. The strategy is more efficient than the conventional FIFO strategy used, for instance, by CKPS [19]. Bulk data is carried only in the RBC phase, which dominates the communication complexity; the ABA phase has two subphases and dominates the running time, i.e., $O(\log n)$.

BEAT [34]. BEAT is a collection of five asynchronous BFT protocols following the BKR paradigm and being built on top of HoneyBadgerBFT. BEAT0 eliminates the usage of expensive pairing cryptography, thereby providing better latency and throughput. The remaining four BEAT protocols essentially explore trade-offs using different RBC primitives in the BKR framework to attain various performance trade-offs. BEAT1 and BEAT2 explore how to optimize latency at the price of having lower throughput. BEAT3 and BEAT4 are BFT storage protocols that keep all transaction copies in the form of erasure coding; while BEAT3 and BEAT4 significantly outperform BEAT0, they are not conventional BFT state machine replication protocols. Therefore, subsequent protocols compare themselves with BEAT0 only.

EPIC [44]. EPIC is the first practical asynchronous BFT protocol with adaptive security, where the adversary can choose to corrupt replicas at any moment during the execution of the protocol. Prior protocols, such as SINTRA, HoneyBadgerBFT, and BEAT, achieve static security only, where the adversary needs to choose the set of corrupted replicas before the execution of the protocol. EPIC is built on top of BEAT0 yet with two significant differences: first, EPIC uses a hybrid transaction selection approach removing the need for threshold encryption used in BEAT; second, EPIC leverages a common-coin protocol with adaptive security [8]. However, the adaptively secure common coin protocol relies on expensive pairing-based cryptography. While achieving reasonable performance, EPIC is much less efficient in terms of both latency and throughput than BEAT; the fact, once again, substantiates the well-established view of favoring regular cryptography (e.g., elliptic curve) over pairing-based cryptography.

3.2 The CKPS Paradigm

The CKPS paradigm reduces asynchronous BFT to (multi-valued) validated Byzantine agreement (VBA). As described in Sec. 2, VBA generalizes ABA in the sense that VBA allows decisions on a value from an arbitrarily large set and has external validity. CKPS shows that VBA can be built using (verifiable) consistent broadcast (CBC) and ABA. The CKPS protocol terminates in a constant expected number of rounds, but there are two bottlenecks. First, the CKPS protocol has a large $O(n^3|m|)$ communication complexity [19] for the VBA component. Second, it uses a significantly large number of threshold signatures; unlike other threshold cryptosystems, the instantiations for threshold signature either rely on expensive pairing-based cryptography [14] or RSA problems for the same security level [55].

SINTRA [21]. SINTRA includes an atomic broadcast implementation of the CKPS paper. It uses Shoup's RSA threshold signature scheme and chooses to implement a simpler atomic broadcast protocol in CKPS that does not terminate in an expected constant number of rounds. Being the first CKPS instantiation, SINTRA is not optimized for high throughput.

Dumbo [39]. Dumbo includes two asynchronous BFT protocols—Dumbo1 and Dumbo2. Dumbo2 performs consistently better than Dumbo1, so we focus on Dumbo2 only. Dumbo is motivated by the fact that the VBA construction in CKPS requires a large bandwidth. Instead of directly applying the bulk data to VBA, Dumbo introduces an additional provable RBC (PRBC) phase such that only the PRBC phase carries bulk data, and the VBA phase takes as input data fingerprints only. Hence, Dumbo has four more steps from the PRBC phase, including three expensive n to n communication and an additional $O(n^3)$ pairing operations, incurring higher latency than CKPS.

Dumbo shares similar performance as HoneyBadgerBFT when n is small or medium in WANs but outperforms HoneyBadgerBFT when n becomes larger. Dumbo is less efficient than BEAT when n is small in WANs, as BEAT consistently outperforms HoneyBadgerBFT. Dumbo did not provide the evaluation in the LAN setting where cryptographic overhead, instead of communication, may dominate. In Appendix A, we evaluate the popular BN curve to understand the latency for Dumbo: the latency for the pairing operations alone, without counting any communication or any other operations, is larger than the entire latency of BEAT.

The overhead incurred by expensive pairing operations in Dumbo cannot be mitigated using the elliptic curve cryptography as in BEAT, as Dumbo requires the transferability of threshold signatures. Besides, Dumbo cannot be efficiently made adaptively secure as in EPIC because Dumbo uses much more threshold cryptographic operations than BEAT and it would be prohibitively expensive to replace all threshold cryptography using adaptively secure cryptography.

4 REVIEWING ABA PROTOCOLS

Table 2 summarizes ABA protocols proposed in recent years using common coins and terminating in an expected constant number of rounds. We divide these protocols into two categories: ABA assuming authenticated channels and ABA requiring the transferability of digital signatures.

For instance, CKS ABA runs in two steps per round (essentially optimal), except its round 0 has three steps. The protocol, however, heavily relies on expensive threshold signatures. Crain-Sig20 [30], too, uses threshold signatures. Neither protocol has been implemented in any asynchronous BFT so far.

	signature needed	steps per round*	coin type
CKS05 [20]	yes	2 or 3*	high threshold
MMR14 [51] (insecure)	no	2 or 3	regular
MMR journal [52]	no	9 to 13	regular
Cobalt18 [48]	no	3 or 4	regular
Crain-Sig20 [30]	yes	1 or 2‡	high threshold
Crain20 [31]	no	2 to 3†	high threshold
Pillar (this work)	no	2 or 3	regular

Table 2: Comparison of ABA protocols using common coins. Msg denotes the message complexity. The number of steps per round does not include the steps for common coins. The expected number of steps is twice the number of steps per round for all protocols. MMR ABA has a liveness issue if instantiated using any common coins. Pillar is as efficient as MMR ABA.*Steps per round. *CKS has 3 steps in round 0. ‡Crain-Sig20 is a variant of CKS with 2 steps in round 0 and 1 step in round greater than 0. †In the 'happy' path for rounds greater than 0, Crain20 may have only 1 step.

MMR ABA uses authenticated channels only and has 2 or 3 steps per round. MacBrough [1] and Tholoniat and Gramoli [59] indepedently discovered that MMR might never terminate if using any cryptographic common-coin protocols. MacBrough proposed a solution in the Cobalt protocol [48] which has 3 or 4 steps in each round. (In Appendix B, we also describe the pseudocode of MMR ABA and Cobalt ABA and illustrate the liveness issue of MMR ABA in detail.) The journal version of MMR ABA also recognized the issue and fixed the problem with more steps than Cobalt ABA. Cobalt ABA has been implemented in Dumbo and EPIC.

This paper introduces a new ABA protocol (Pillar) that relies on authenticated channels and regular common coins. Pillar is the most efficient ABA assuming regular coins. As we argued in Sec. 2, high threshold common coins are less favorable than regular common coins, regardless of whether assuming trusted setup.

For all these ABA protocols, the expected number of steps is simply 2x the steps per round, as they all need on average 2 rounds to terminate.

5 THE PILLAR ABA PROTOCOL

We present in Figure 2 Pillar, a novel signature-free ABA protocol. We begin with some notation.

- For $b \in \{0, 1\}$, $\bar{b} = 1 - b$.
- $*$ in a message may represent $b, \bar{b}$, or $\perp$, where $b \in \{0, 1\}$. For instance, $\mathsf{aux}_r(*, b)$ may represent $\mathsf{aux}_r(b, b)$, $\mathsf{aux}_r(\bar{b}, b)$, $\mathsf{aux}_r(\perp, b)$. $\mathsf{bval}_r(b, *)$ may represent $\mathsf{bval}_r(b, b)$, $\mathsf{bval}_r(b, \bar{b})$, or $\mathsf{bval}_r(b, \perp)$. We may simply omit $*$ when there is no ambiguity; for example, we may write $\mathsf{aux}_r()$ to denote $\mathsf{aux}_r(*, *)$.
- Let $vals$ be a vector (multiset) consisting of 0, 1, and $\perp$. $\mathsf{V}_1(vals) = b$ if the only value in $vals$ is b for $b \in \{0, 1, \perp\}$.
- $\mathsf{V}_2(vals, b) = t$ if $vals$ includes b only or both b and $\perp$, where $b \in \{0, 1\}$, and the number of b's in $vals$ is t.
- $\mathsf{majority}(vals) = b$ for $b \in \{0, 1\}$, if b is a simple majority in $vals$, i.e., the number of b's is no less than $\lceil (|vals| + 1)/2 \rceil$. $\mathsf{majority}(vals) = \perp$ otherwise.

5.1 Workflow

As illustrated in Figure 2, the Pillar protocol has two message types: $\mathsf{bval}_r()$ and $\mathsf{aux}_r()$. In each round r, a replica p_i has an input est_r and an auxiliary input maj_r. The est_r value is a binary value (i.e., $est_r \in \{0, 1\}$) and $maj_r \in \{0, 1, \perp\}$. In each round, p_i first broadcasts a $\mathsf{bval}_r(est_r, maj_r)$ message. The value of maj_r is set to $\perp$ in round 0. A correct replica does not change its maj_r within a round. If p_i receives more than $f + 1$ $\mathsf{bval}_r(b)$ messages such that b is different from its input, it also broadcasts $\mathsf{bval}_r(b, maj_r)$. If a replica receives $2f + 1$ $\mathsf{bval}_r(b, *)$ messages with the same b value (b is either 0 or 1), b is added to a set bin_values_r.

In addition, p_i checks the maj_r values (which form a set $majs$) in the $\mathsf{bval}_r(b, maj_r)$ messages. Specifically, $\delta_r(b)$ is set to 1 for the following conditions. If $r = 0$, $\delta_r(b)$ is set to 1 for any $b \in \{0, 1\}$. If $r > 0$, the replica compares b with the common coin value s_{r-1} in round $r - 1$. If $b = s_{r-1}$ and p_i receives only $\mathsf{bval}_r(b, b)$ and $\mathsf{bval}_r(b, \perp)$, $\delta_r(b)$ is set to 1. If $b = \bar{s}_{r-1}$ and p_i only receives $\mathsf{bval}_r(b, b)$ (i.e., $\mathsf{V}_1(majs) = b$), $\delta_r(b)$ is set to 1. After p_i receives $n - f$ $\mathsf{bval}_r()$ messages, it sends $\mathsf{aux}_r(b, b)$ when $\delta_r(b) = 1$ and $\mathsf{aux}_r(\perp, b)$ otherwise. A correct replica only sends one $\mathsf{aux}_r()$ message in each round.

```
upon event $propose(sid, v_i)$
  if $r = 0$, $est_0 \leftarrow v_i$, $maj_0 \leftarrow \perp$
round $r$
  $bin\_values_r \leftarrow \emptyset$
  $majs \leftarrow \emptyset$
  broadcast $\mathsf{bval}_r(est_r, maj_r)$                                {1st phase}
  upon receiving $\mathsf{bval}_r(v, c)$ from $j$
    $majs \leftarrow majs \cup \{c\}$
  upon receiving $f+1$ $\mathsf{bval}_r(b, *)$
    if $\mathsf{bval}_r(b, maj_r)$ has not been sent
      broadcast $\mathsf{bval}_r(b, maj_r)$
  upon receiving $n-f$ $\mathsf{bval}_r(b, *)$
    $bin\_values_r \leftarrow bin\_values_r \cup \{b\}$
    if ($r > 0$ and (($b = \bar{s}_{r-1}$ and $\mathsf{V}_1(majs) = b$) or ($b = s_{r-1}$ and $\bar{b} \notin majs$))) or
$r = 0$
      $\delta_r(b) \leftarrow 1$
    if $\mathsf{aux}_r(b, *)$ or $\mathsf{aux}_r(\bar{b}, *)$ has not been sent
      if $\delta_r(b) = 1$, broadcast $\mathsf{aux}_r(b, b)$                {2nd phase}
      else broadcast $\mathsf{aux}_r(\perp, b)$
  upon receiving $n-f$ $\mathsf{aux}_r(*, *)$ where the sets of values carried by these messages
are $vals_r$ and $avals_r$; $vals_r$ and $avals_r$ are both subsets of $bin\_values_r$.
    $s_r \leftarrow \mathsf{coin}_r$
    if $\mathsf{V}_2(vals_r, b) \geq \lceil \frac{n+f+1}{2} \rceil$          {a quorum of replicas voted for $b$}
      $est_{r+1} \leftarrow b$
      $maj_{r+1} \leftarrow b$
      if $b = s_r$, $decide(sid, b)$
    if $r > 0$ and ($\mathsf{V}_1(vals_r) = \perp$ or $\mathsf{V}_2(vals_r, b) < \lceil \frac{n+f+1}{2} \rceil$)
      if $\mathsf{V}_2(avals_r, b) \geq \lceil \frac{n+f+1}{2} \rceil$
        $est_{r+1} \leftarrow b$
        $maj_{r+1} \leftarrow b$
        if $b = s_{r-1}$ and $b = s_r$, $decide(sid, b)$
      else if $\{0, 1\} \in avals_r$ and $b = s_{r-1}$
        $est_{r+1} \leftarrow b$
        $maj_{r+1} \leftarrow b$
    if $est_{r+1}$ has not been set yet                                  {all other conditions}
      $est_{r+1} \leftarrow s_r$
      if $r = 0$, $maj_{r+1} \leftarrow s_r$
      else $maj_{r+1} \leftarrow \mathsf{majority}(vals_r)$
    $r \leftarrow r + 1$, continue to the next round
```

Figure 2: The Pillar protocol. The code for replica p_i. Broadcast in the code means best-effort broadcast.

Replica p_i only accepts an $\mathsf{aux}_r(v_1, v_2)$ message if both v_1 and v_2 are added to bin_values_r. Furthermore, a replica does not accept an $\mathsf{aux}_r(v, \bar{v})$ or $\mathsf{aux}_r(*, \perp)$ message, since these values are not allowed according to our specification. Furthermore, if a replica sets $\delta_r(b) = 1$ and receives $\mathsf{aux}_r(\bar{b}, *)$, it discards the message. Upon receiving $n - f$ $\mathsf{aux}_r()$ messages, there are three cases. 1) If p_i receives a quorum of $\mathsf{aux}_r(b, b)$ messages, p_i compares b with the common coin s_r. If $b = s_r$, p_i decides b. Otherwise p_i enters the next round with b as both est_{r+1} and maj_{r+1}. 2) If p_i only receives $\mathsf{aux}_r(b, b)$ and $\mathsf{aux}_r(\perp, b)$ and at least a quorum of the messages are of the form $\mathsf{aux}_r(*, b)$, p_i also compares b with both s_{r-1} and s_r. If $b = s_{r-1}$ and $b = s_r$, p_i decides b. Otherwise, p_i uses b for est_{r+1} and maj_{r+1} and enters the next round. 3) If p_i receives $n - f$ $\mathsf{aux}_r(\perp, b)$ and $\mathsf{aux}_r(b, b)$ and s_{r-1}, p_i compares b with $\bar{s}_r$. If $b = \bar{s}_r$, p_i uses b for est_{r+1} and maj_{r+1} and enters the next round.

If none of the cases apply and est_{r+1} has not been set yet, p_i uses the common coin value as the input (i.e., est_{r+1}) for the next round. Furthermore, p_i sets maj_{r+1} to s_r for $r = 0$ and $\mathsf{majority}(vals_r)$ for $r > 0$. Then p_i enters the next round.

5.2 Analysis

There are two key elements in our design. First, in addition to the input value est_r, every replica needs to carry a majority value from the previous round maj_r. This maj_r value is used in the $\mathsf{bval}_r()$ step and *will never change in the same round*, though a replica is allowed to broadcast $est_r = 1$ and $est_r = 0$. Furthermore, we require that a replica sends an $\mathsf{aux}_r(b, b)$ message if $\delta_r(b) = 1$. In round 0, $\delta_r(b) = 1$ for both $b = 0$ and $b = 1$. But in round $r > 0$, since the maj_r value does not change within a round, correct replicas will only have $\delta_r(v) = 1$ for at most one value. This is because if $b = \bar{s}_{r-1}$, $majs$ can only contain b for a correct replica to set $\delta(b) = 1$. If $b = s_{r-1}$, $majs$ can be either b or $\perp$, but cannot contain $\bar{b}$. Therefore, it naturally prevents a network scheduler from scheduling the messages and making some correct replicas have an est_{r+1} in the next round that is different from the common coin.

Second, we ensure that if a correct replica decides in round r, all replicas will select the same est_{r+1} and eventually decide est_{r+1}. If a correct replica p_i decides $b = s_r$ in round r, it must have received $2f+1$ $\mathsf{aux}_r(b, b)$, or $b = s_{r-1}$. If p_i receives $\mathsf{aux}_r(b, b)$ messages from a quorum of replicas, the $\delta_r(b)$ value guarantees that all correct replicas will send either $\mathsf{aux}_r(b, b)$ or $\mathsf{aux}_r(\perp, b)$, but not $\mathsf{aux}_r(\bar{b}, \bar{b})$. If p_i only receives $\mathsf{aux}_r(b, b)$, any correct replica p_j will receive at least one $\mathsf{aux}_r(b, b)$, making it impossible for p_j to receive either a quorum of $\mathsf{aux}_r(\bar{b}, *)$ messages or $\mathsf{aux}_r(*, \bar{b})$ messages and use $est_{r+1} = \bar{b}$. Furthermore, if p_i receives $n - f$ $\mathsf{aux}_r(b, b)$ and $\mathsf{aux}_r(\perp, b)$ such that p_i decides b, we know $b = s_{r-1}$. A correct replica p_j will never receive a quorum of $\mathsf{aux}_r(*, \bar{b})$ messages so as to use $\bar{b}$ as est_{r+1}. We prove the correctness of Pillar in Appendix C.

6 THE ACE BFT PROTOCOL

It is shown in [34, 39, 50] ABA is the bottleneck for asynchronous BFT implementations. Hence, Pillar can be used to improve all practical asynchronous BFT protocols, including HoneyBadgerBFT, BEAT, Dumbo, and EPIC. To demonstrate the efficiency of Pillar, we provide a new asynchronous BFT protocol, called ACE, that follows the classic BKR framework [12]. ACE is identical to BEAT except that the ABA phase now uses Pillar instead of Cobalt ABA. We show the pseudocode of the ACE in Figure 3 that uses the *r-broadcast* and *r-deliver* primitives of RBC, and *propose* and *decide* primitives of ABA. As in BEAT, we use pairing-free, elliptic curve based threshold common coins [20] (for ABA).

ACE proceeds in epochs initialized as $e = 0$. Each epoch e includes a RBC phase, including n parallel RBC instances RBC_i for $i \in [0..n-1]$, and an ABA phase, including n ABA instances ABA_i for $i \in [0..n-1]$, where RBC_i is triggered by $p_i \in [p_0..p_{n-1}]$ to r-broadcast a proposal m_i (a batch of transactions) selected from its buffer, and ABA_i is triggered by correct replicas to decide if m_i has been r-delivered. Following the BKR paradigm, the ABA phase of ACE is not fully parallel, requiring that if a replica has not received some proposal from p_j during the RBC phase, the replica must abstain from proposing 0 for ABA_j until $n-f$ ABA instances decide 1. To isolate the primitive instances, we tag each message in the instances with e.

7 RABA AND THE PISA RABA PROTOCOL

7.1 Definition of RABA

We introduce a new distributed computing primitive, reproposable ABA (RABA). Unlike conventional ABA where replicas can vote once only, RABA allows replicas to change their votes.

Syntactically, a RABA protocol tagged with a unique identifier

```
init
    $e \leftarrow 0$
upon selecting $m_i$ from the buffer of $p_i$
    r-broadcast($[e, i], m_i$) for RBC$_i$
upon r-deliver($[e, j], m_j$) for RBC$_j$
    if ABA$_j$ has not been started
        propose($[e, j], 1$) for ABA$_j$
wait until termination of any $n - f$ ABA instances
    for all ABA$_j$ instances that have not been started
        propose($[e, j], 0$)
upon decide($[e, j], v$) for any value $v$ for all ABA instances
    let $S$ be set of indexes for ABA instances that decide 1
    wait until r-deliver($[e, j], m_j$) for all ABA$_j$ such that $j \in S$
        a-deliver($\cup_{j \in S} \{m_j\}$) in some deterministic order
        $e \leftarrow e + 1$
```

Figure 3: The ACE protocol. The code for replica p_i for an epoch e.

sid is specified by *propose*$(sid, \cdot)$, *repropose*$(sid, \cdot)$, and *decide*$(sid, \cdot)$, where the input domain is $\{0, 1\}$. For our purpose, RABA is "biased towards 1." Each replica can propose a value v at the beginning of the protocol. Each replica can propose a value only once. A correct replica that proposed 0 is allowed to change its mind and repropose 1. A replica that proposed 1, however, is not allowed to repropose 0. (That is, correct replicas should not do this.) If a replica reproposes 1, it does so at most once. A replica terminates the protocol identified by *sid* by generating a decide message. RABA (biased toward 1) satisfies the following properties:

- **Validity**: If all correct replicas *propose* v and never *repropose* $\bar{v}$, then any correct replica that terminates *decides* v.
- **Unanimous termination**: If all correct replicas *propose* v and never *repropose* $\bar{v}$, then all correct replicas eventually terminate.
- **Agreement**: If a correct replica *decides* v, then any correct replica that terminates *decides* v.
- **Biased validity**: If $f + 1$ correct replicas *propose* 1, then any correct replica that terminates *decides* 1.
- **Biased termination**: Let Q be the set of correct replicas. Let Q_1 be the set of correct replicas that propose 1 and never repropose 0. Let Q_2 be correct replicas that propose 0 and later repropose 1. If $Q_2 \neq \emptyset$ and $Q = Q_1 \cup Q_2$, then each correct replica eventually terminates.
- **Integrity**: No correct replica decides twice.

RABA has a slightly different validity property modified for our RABA syntax. It implies validity for two cases: 1) all correct replicas propose 1 (and, of course, they cannot repropose 0 according to our syntax) before termination; 2) all correct replicas propose 0 and never repropose 1 before termination.

Unanimous termination is weaker than the conventional termination property: it guarantees termination only when all correct replicas propose the same input. We levy restriction on *correct* replicas only.

Validity and unanimous termination can be combined into a single property:

- If all correct replicas *propose* v and never *repropose* $\bar{v}$, then any correct replica *decides* v.

The agreement property of RABA is identical to that of ABA and VBA.

Biased validity in RABA requires that if $f + 1$ replicas, instead of all correct replicas, propose 1, then a correct replica that terminates decides 1. The property echoes that of VBA. We stress, however, RABA is not in the context of "validated" BA (VBA). This is precisely our goal, as validated BA requires the usage of expensive threshold signatures or a vector of n signatures as proofs which we strive to avoid. Nevertheless, this also means that when using RABA, we no longer have the powerful "validation" technique (and we need to be creative when using RABA).

Our biased termination property ensures that the protocol will eventually terminate as long as all correct replicas either propose 1 (and never repropose 0) or initially propose 0 but change their minds to repropose 1. Q_1 can be an empty set. Q_2 cannot be an empty set because otherwise biased termination is implied by unanimous termination.

Unanimous termination and biased termination complement each other. Remarkably, RABA does not have the usual termination property. (Correspondingly, our RABA protocol may indeed never terminate.) To use RABA in our favor, one must use a high-level protocol to control the inputs of RABA, allowing, when necessary, replicas to change their votes to attain termination eventually.

Allowing a replica to change its mind and enabling a non-validated version of biased validity/termination properties are two central ideas underlying RABA.

Admittedly, RABA may look somewhat complex. Why not use two ABA instances instead, one ABA for the "propose" phase and the other ABA for the "repropose" phase? One possible anomaly for this idea is that one replica might decide twice. Indeed, it is possible that while some replica is participating in the second ABA, the first ABA has terminated; it is also possible that some replicas terminate for the first ABA, while some other replicas terminate for the second ABA. According to our RABA definition, this is not allowed because the "integrity"– no correct replica decides twice– is introduced to govern both the propose operation and the repropose operation in a single RABA instance explicitly associated with a session identifier *sid*.

The overall formalization of RABA allows hiding—as much as *we could*—subtle protocol implementation details, exposing a clean API that can be neatly fit into a simple and novel asynchronous BFT framework (to be described shortly).

7.2 Pisa: Efficient RABA Construction from Pillar

We present a RABA protocol, Pisa, built on top of Pillar. As illustrated in Figure 4, we modify the round $r = 0$ protocol of Pillar, while the code for the round $r > 0$ remains unchanged. In particular, we make the following major changes in round 0. First, if a correct replica p_i proposes 1, it immediately adds 1 to bin_values_r, If p_i has not previously broadcast any $\text{aux}_r()$ messages, it broadcasts $\text{aux}_r(1, 1)$. Second, if p_i proposes 0 and receives $f + 1$ $\text{bval}_r(1, \perp)$, in addition to broadcasting $\text{bval}_r(1, \perp)$, it immediately adds 1 to bin_values_r. If p_i has not sent any $\text{aux}_r()$ message, it also broadcasts $\text{aux}_r(1, 1)$. Third, each replica p_i that proposes 0 can repropose 1. If the $repropose()$ event is triggered, regardless of which round p_i is in, it broadcasts a $\text{bval}_0(1, \perp)$ message if it has not done so. Last, the value of common coin is set to 1 in round 0. This is to ensure that if a replica receives $n - f$ $\text{aux}_r(1, 1)$, it will directly terminate the protocol. For each replica that receives both $\text{aux}_r(1, 1)$

```
upon event $propose(sid, v_i)$
  if $r = 0$, $est_0 \leftarrow v_i$, $maj_0 \leftarrow \bot$
upon event $repropose(id, 1)$
  broadcast $\text{bval}_0(1, \bot)$                                   {1st phase}
round 0
  $bin\_values_r \leftarrow \emptyset$
  broadcast $\text{bval}_r(est_r, maj_r)$
  if $est_r = 1$                                                      {biased toward 1}
    $bin\_values_r \leftarrow \{1\}$
    if $\text{aux}_r()$ has not been sent, broadcast $\text{aux}_r(1, 1)$
  upon receiving $\text{bval}_r(*, *)$ from $j$
    if there are $f + 1$ $\text{bval}_r(b, *)$
      if $\text{bval}_r(b, maj_r)$ has not been sent
        broadcast $\text{bval}_r(b, maj_r)$
      if $b = 1$                              {enter the 2nd phase and biased toward 1}
        $bin\_values_r \leftarrow bin\_values_r \cup \{b\}$
        if $\text{aux}_r()$ has not been sent, broadcast $\text{aux}_r(1, 1)$
    if there are $2f + 1$ $\text{bval}_r(b, *)$                          {2nd phase}
      $bin\_values_r \leftarrow bin\_values_r \cup \{b\}$
      if $\text{aux}_r()$ has not been sent, broadcast $\text{aux}_r(b, b)$
  upon receiving $n-f$ $\text{aux}_r(*, *)$ where the sets of values carried by these messages
are $vals_r$ and $avals_r$; $vals_r$ and $avals_r$ are both subsets of $bin\_values_r$.
    if $\vee_2(vals_r, b) \geq \lceil \frac{n+f+1}{2} \rceil$
      $est_{r+1} \leftarrow b$
      $maj_{r+1} \leftarrow b$
      if $b = 1$, $decide(sid, b)$
    else
      $est_{r+1} \leftarrow 1$
      $maj_{r+1} \leftarrow 1$
    $r \leftarrow r + 1$, continue to the next round
```

Figure 4: The Pisa protocol for round 0 only at p_i.

and $\text{aux}_r(0, 0)$, the replica enters the next round using $est_{r+1} = 1$ and $maj_{r+1} = 1$.

7.3 Intuition and Analysis

The key to correctness is that in round 0, a replica p_i puts 1 in bin_values_r and directly broadcasts $\text{aux}_r(1, 1)$ if its initial input is 1. If p_i proposes 0, receives $f + 1$ $\text{bval}_r(1, \bot)$ messages, and has not sent any $\text{aux}_r()$ message, it also adds 1 to bin_values_r and broadcasts $\text{aux}_r(1, 1)$. This ensures that if some replicas propose 1, all correct replicas tend to accept 1. If more than $f + 1$ correct replicas have 1 as their input in round 0, no correct replicas can receive $2f + 1$ $\text{aux}_r(0, 0)$ and use 0 as input for the next round. In other words, all correct replicas will have 1 as the input for the next round, achieving the biased validity property.

Such an approach does not guarantee the conventional termination property of ABA. Consider a scenario with four replicas in Figure 5 (we use $\text{aux}_r(v)$ in the figure, as each replica broadcasts $\text{aux}_r(v, v)$ in round 0). Replica p_0 proposes 1 while p_1 and p_2 propose 0. The faulty replica p_3 can send $\text{bval}_r(0, \bot)$ to p_2 and p_3 and make them send $\text{aux}_r(0, 0)$. While p_0 can receive $n - f$ $\text{bval}_r(0, \bot)$ messages, put 0 to its bin_values_r, and accept $\text{aux}_r(0, 0)$, it has already sent $\text{aux}_r(1, 1)$. Since p_2 and p_3 can not receive enough $\text{bval}_r(1, \bot)$ messages, they will not put 1 to their bin_values_r. If p_3 does not send any $\text{aux}_r()$ messages, p_1 and p_2 are unable to move to the next round or terminate the protocol.

Pisa can achieve biased termination if a reproposal is allowed. In particular, a replica is allowed to repropose 1 if it previously proposed 0. In this particular example, if correct replicas p_1 and p_2 repropose, they will send $\text{bval}_0(1, \bot)$. Since p_1 and p_2 are still in round 0, they are able to collect $n - f$ $\text{bval}_0(1, \bot)$ and put 1 to bin_values_r. Hence, all replicas will eventually move to the next round. As we use 1 as the common coin for round 0, correct replicas will eventually decide 1. Note that prior to the reproposal of p_1 and p_2, the faulty replica p_3 may have sent $\text{aux}_r(0, 0)$ to p_1 and p_2. In this case, both p_1 and p_2 receive 3 $\text{aux}_r(0, 0)$ messages and use 0 as input for round 1. Therefore, even if p_1 and p_2 repropose later, replicas already move to a new round such that the $\text{bval}_0(1, \bot)$ messages will not be accepted. That is, replicas might not necessarily decide 1 if fewer than $f + 1$ correct replicas propose 1. We prove the correctness of Pisa in the Appendix.

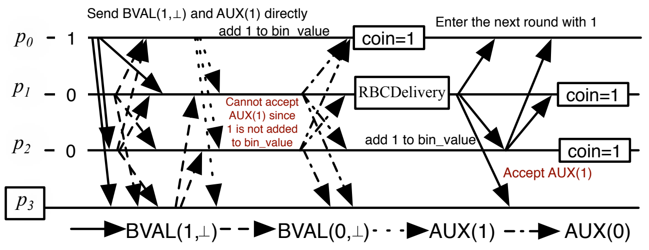


Figure 5: Example of biased termination of Pisa.

Note that Pisa can terminate with one step (round 0) in the best case. This is because if all correct replicas propose 1, they directly add 1 to bin_values_0, and broadcasts $\text{bval}_0(1, \bot)$ and $\text{aux}_0(1, 1)$ simultaneously. As the common coin value is 1 in round 0, all correct replicas decide in one step. (Jumping ahead, the property explains why PACE has one less step compared to ACE in the best case.)

8 A NOVEL ASYNCHRONOUS BFT FRAMEWORK AND THE PACE BFT PROTOCOL

8.1 The "Two-Subphase" Bottleneck for BKR

While the BKR approach ensures that transactions from at least $n - f$ replicas are delivered, it must experience two ABA subphases, the first subphase waiting for $n - f$ ABA instances to terminate and the second subphase dealing with the remaining f ABA instances. Note that an ABA instance terminates on average in 2 rounds, but the running time of $n - f$ ABA instances is $O(\log n)$ in expectation. If a single ABA instance in the first subphase is "unlucky," *all* ABA instances for the second ABA subphase have to wait.

This is a well-known bottleneck for the BKR paradigm (recently emphasized by HoneyBadgerBFT [49, Section 4.4]). Both Dumbo and BEAT experimentally validate the bottleneck via performance breakdown for the building blocks. One attempt to avoid the bottleneck would be that each replica waits for the first $n - f$ RBCs to complete, and then propose 1 for the ABA instances corresponding to those completed and propose 0 for all the others. However, RBC instances completed for correct replicas may be different. As ABA ensures the decided value is 1 if all correct replicas unanimously propose 1, the transactions delivered may be empty. Some works also study how to reduce the time complexity for n parallel ABA instances to a constant expected number of rounds (in some other contexts). Ben-Or's solution tolerates $f = O(\sqrt[4]{n})$ failures only [9]; Ben-Or and El-Yaniv provide a constant expected time protocol [11] which would, unfortunately, yield a prohibitively expensive BFT protocol with $O(n^4)$ message and communication complexity.

```
init
  $e \leftarrow 0$                                              {epoch number}
upon selecting $m_i$ from the buffer of $p_i$
  r-broadcast($[e, i], m_i$) for RBC$_i$
upon r-deliver($[e, j], m_j$) for RBC$_j$
  if RABA$_i$ has not been started
    propose($[e, j], 1$) for RABA$_j$
  else
    repropose($[e, j], 1$) for RABA$_j$
upon delivery of $n - f$ RBC instances
  for RABA instances that have not been started
    propose($[e, j], 0$)
upon decide($[e, j], v$) for any value $v$ for all RABA instances
  let $S$ be set of indexes for RABA instances that decide 1
  wait until r-deliver($[e, j], m_j$) for all RABA$_j$ such that $j \in S$
    a-deliver($\cup_{j \in S}\{m_j\}$) in some deterministic order
    $e \leftarrow e + 1$
```

Figure 6: A new asynchronous BFT framework. The code for replica p_i.

8.2 A New BFT Framework

Our asynchronous BFT framework uses *r-broadcast* and *r-deliver* primitives of RBC, and *propose*, *repropose* and *decide* primitives of RABA to overcome the two-subphase bottleneck. Figure 6 describes the pseudocode of our framework. For each epoch, the framework consists of n parallel RBC instances and n parallel RABA instances. In the RBC phase, each replica p_i r-broadcasts a proposal m_i for RBC$_i$. If p_i r-delivers a proposal from RBC$_j$, it proposes 1 for RABA$_j$. Upon delivery of $n - f$ RBC instances, instead of waiting for $n - f$ RABA instances to terminate, p_i immediately proposes 0 for all RABA instances that have not been started. If p_i later delivers a proposal from some RBC$_j$, it has proposed 0 for RABA$_j$, and has not terminated RABA$_j$, it reproposes 1 for RABA$_j$. Let S be the set of indexes that RABA$_j$ decides 1. If RABA$_j$ decides 1, the proposal from p_j is included in the final delivered set. After all RABA instances terminate and all RBC$_i$ ($i \in S$) instances are delivered, p_i a-delivers ($\cup_{j \in S}\{m_j\}$).

RABA does not itself attain termination. We use RBC carefully to "control" the API of RABA and force RABA to meet the unanimous termination condition or the biased termination condition. To see this, we distinguish several cases:

- *Case 1: All correct replicas propose 1 for some RABA.* According to unanimous termination, the RABA instance eventually terminates with output 1.
- *Case 2: All correct replicas propose 0.* We further distinguish two cases:
  - *Case 2-1*: If they never repropose 1, the RABA instance eventually terminates due to unanimous termination.
  - *Case 2-2*: If some replicas repropose 1, then these replicas must have r-delivered the corresponding messages. Due to the agreement property of RBC, all correct replicas will deliver the messages and repropose 1. The protocol will terminate according to biased termination.
- *Case 3: Some correct replicas propose 0 and some other correct replicas propose 1.* Similar to Case 2-2, due to agreement of RBC, correct replicas will eventually repropose 1, and the RABA instance will terminate.

Thanks to the biased validity property, we can bound the number of transactions delivered for each epoch, conditioned on protocol termination. In particular, we prove that in the worst case (with a network scheduler), transactions from at least $f + 1$ replicas will be delivered, while in the normal case, transactions from at least $\lceil \frac{n+f+1}{2} \rceil$ replicas will be delivered.

While our paradigm does not improve the worst-case running time of BKR, it does improve the concrete time complexity as shown in Appendix A. More importantly, it avoids the "two-subphase" bottleneck for BKR. Instead of waiting for at least $n - f$ ABA instances to terminate, each replica can now trigger all RABA instances once $n - f$ RBC instances are delivered. This improvement, not just triggering the ABA phase earlier, allows all RABA instances to run in a fully parallel manner, essentially avoiding transactions congestion.

8.3 PACE: An Efficient BFT Instantiation

We instantiate our new framework using Pisa as the underlying RABA protocol. The resulting protocol is called PACE.

The correctness of our framework directly implies the correctness of PACE. It may still be helpful to examine an execution example. We discuss the same example in Figure 5. The biased termination condition for Pisa is met conditioned on RBC delivery in our framework. The agreement property of RBC ensures that if a correct replica r-delivers a request m, all correct replicas will eventually r-deliver m. Namely, if p_0 proposes 1, both p_1 and p_2 will eventually repropose 1. Thus, PACE terminates due to biased termination. We show the proof of PACE in Appendix E.

8.4 Running Time Comparison

To show the benefit of our framework itself, we let Dumbo-Pillar be Dumbo using our more efficient ABA construction introduced in this paper (Pillar). We compare the new protocols introduced in the paper, ACE and PACE, with HoneyBadgerBFT, BEAT, Dumbo, and Dumbo-Pillar in Table 1 and in a detailed manner in Appendix A. Remarkably, if examined concretely instead of just asymptotically, we find CKPS, Dumbo, and Dumbo-Pillar have much more steps and expensive pairing operations for best-case and average-case scenarios than our protocols for any practically large n's. Even when $n = 91$, PACE needs 19 steps on average, while Dumbo, Dumbo-Pillar, and BEAT have 32, 26, and 44 steps, respectively; PACE is pairing-free, while Dumbo-Pillar requires 8.54×10^5 expensive pairings operations. In the best case, PACE has just 4 steps, while Dumbo-Pillar has 14 steps. Dumbo-Pillar has fewer expected steps than PACE only when $n > 1000$. Even compared with DAG-Rider (21 steps), PACE would have fewer steps when $n < 150$.

9 IMPLEMENTATION AND EVALUATION

Implementation. We first implement the two ABA protocols introduced in this paper—Pillar and Pisa. We then implement ACE and PACE using Pillar and Pisa, respectively. We compare them with BEAT-MMR (insecure) and BEAT-Cobalt. The four BFT protocols are summarized in Table 3.

Overview. We deploy the protocols on Amazon EC2 utilizing up to 91 t2.medium VMs. Each VM has two vCPUs and 4GB memory. We evaluate both LAN and WAN settings, where the LAN VMs are launched in the same data center (DC), and the WAN VMs are

	BEAT-MMR (insecure)	BEAT-Cobalt	ACE	PACE
ABA/RABA	MMR	Cobalt	Pillar	Pisa

Table 3: Asynchronous BFT protocols.

evenly distributed in five continents. We evaluate the protocols using different number of replicas (i.e., network sizes) and batch sizes (i.e., contention levels). We use the number of the faulty replicas, f, to denote the network size. All transactions are of size 250 bytes.

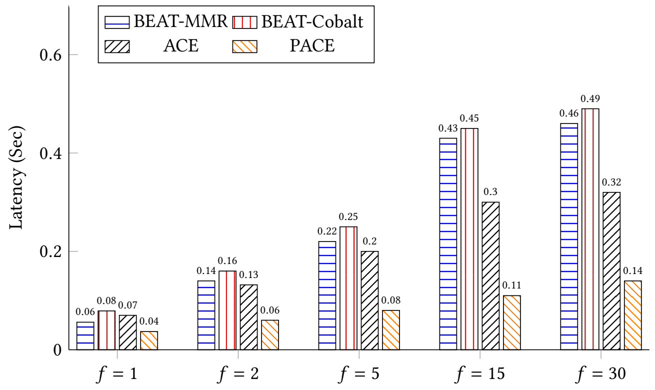


Figure 7: Latency of the protocols under no contention where the replicas are located in the same DC.

Latency. We evaluate the latency of the four BFT protocols under no contention, where each replica proposes only one batch of transactions. We report the latency for $f = 1$, $f = 2$, $f = 5$, $f = 15$, and $f = 30$ in WAN. As illustrated in Figure 7, the latency of BEAT-Cobalt is consistently the highest among the four protocols since Cobalt ABA has one more step in each round. Meanwhile, the latency of PACE is much lower than others because Pisa terminates faster than other ABA protocols, and PACE has only one subphase.

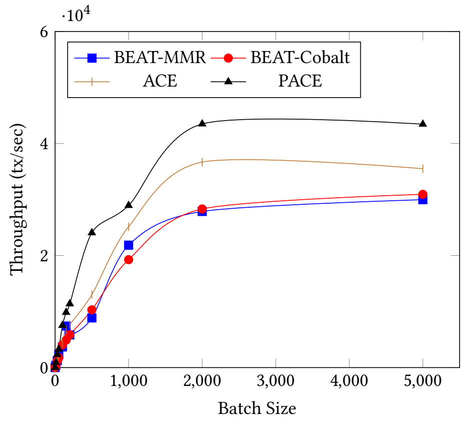


Figure 8: Throughput for $f = 1$ where the replicas are located in the same DC.

Throughput. In each epoch, each replica proposes $|B|$ transactions. We simply let $|B|$ be the batch size of transactions. Hence, all replicas propose in total $n|B|$ transactions for an epoch. We evaluate the throughput and latency vs. throughput as B increases. We report

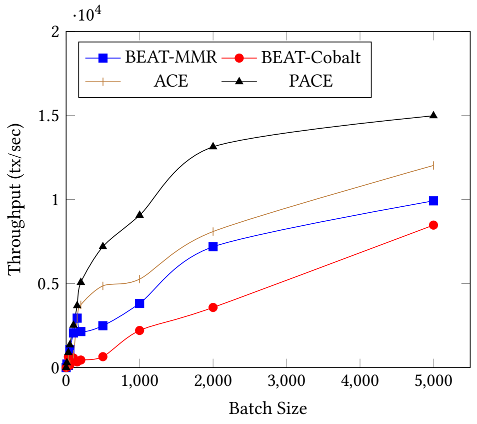


Figure 9: Throughput for $f = 1$ where the replicas are from 4 different DCs.

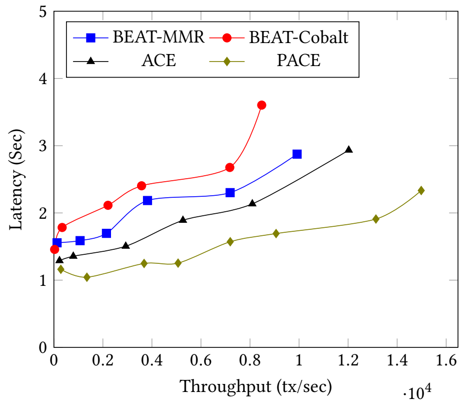


Figure 10: Latency vs. throughput in the WAN setting with $f = 1$.

the throughput of the four BFT protocols for $f = 1$ in both LAN (Figure 8) and WAN settings (Figure 9) and report the throughput vs. latency in Figure 10.

We observe that PACE consistently and significantly outperforms the other protocols. This is due to the faster (biased) termination for Pisa and the fully parallel feature of PACE. The throughput of PACE is around 43,000 tx/sec in the LAN setting and 14,000 tx/sec in the WAN setting. PACE is 40% and 77% higher throughput than BEAT-Cobalt in the LAN and WAN settings, respectively.

ACE has 14% and 42% higher throughput than BEAT-Cobalt in the LAN and WAN environments, respectively. This is also expected, as our new ABA protocol, Pillar performs better than Cobalt ABA.

In blockchain applications, a typical block size is about 2MB, roughly matching a batch size $|B|$=2,000 in our evaluation for $n = 4$ replicas. Looking at this setting, ACE and PACE achieve 113% and 227% higher throughput than BEAT-Cobalt for WANs, respectively.

BEAT-Cobalt has similar performance as BEAT-MMR in the LAN setting. However, BEAT-MMR has higher throughput than BEAT-Cobalt in WANs, as Cobalt has one more step in each round, and the network latency becomes more relevant in WANs.

Scalability. We evaluate the throughput of PACE, ACE, and BEAT-Cobalt by varying f from 2 to 30 in WANs in Figure 11. We observe

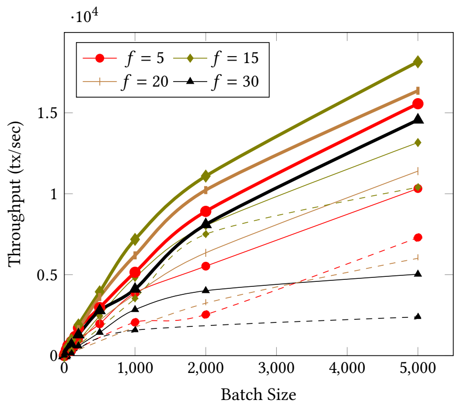


Figure 11: Scalability where f varies from 2 to 30. Thick lines denote the throughput of PACE, thin lines denote the throughput of ACE, and dashed lines denote the throughput of BEAT-Cobalt.

	BKR	NAIVE (insecure)	PACE
$f = 1 (n = 4)$	3.00	3.00	3.00
$f = 2 (n = 7)$	5.54	5.45	5.66
$f = 5 (n = 16)$	11.18	10.81	12.36
$f = 15 (n = 46)$	31.24	28.12	33.62
$f = 20 (n = 61)$	41.10	38.12	46.37
$f = 30 (n = 91)$	61.00	54.75	65.25

Table 4: The number of proposals that are delivered in ABA protocols for different frameworks. The number n is the total number of replicas and also the maximum number of proposals that could be delivered.

that for all network sizes, PACE significantly outperforms ACE in all experiments, while ACE has consistently better throughput than BEAT-Cobalt.

We find that as the network size f increases, the throughput for these protocols first increases and then decreases. This echoes the results for HoneyBadgerBFT and EPIC. When f grows, the number of transactions proposed concurrently grows accordingly. Nevertheless, when f further grows, the protocol itself becomes the bottleneck. All these protocols reach their peaks when $f = 15$. In the largest experiment ($n = 91$), the throughput of PACE is around 14,558 tx/sec, around 6.1x the throughput of BEAT-Cobalt. Meanwhile, ACE has about 2x the throughput of BEAT-Cobalt.

It is worth mentioning that the peak throughput in our scalability experiments can be higher if using larger batches. Indeed, prior works such as BEAT, EPIC, and Dumbo evaluated at least 10^4 and larger batch sizes (in our notation). We comment that evaluating a batch size larger than 5,000 for a network with a large f may be misleading. In all recent asynchronous BFT evaluations, to report the best possible throughput, each replica needs to propose disjoint proposals. For $f = 30$ and $|B|$ =5,000, if each replica proposes a batch of around 1.2 MB transactions, all $n = 91$ replicas would propose over 100 MB transactions in total, where $n - f = 61$ batches are expected to include non-overlapping transactions. This would require each replica to have a huge buffer of pending transactions.

Comparing with Dumbo. We do not directly compare our systems with Dumbo or Dumbo-Pillar, for a variety of reasons: 1) ACE and PACE belong to the BKR paradigm that achieves information-theoretic (IT) security if assuming an IT secure common coin protocol and achieves quantum safety as defined in DAG-Rider [40], while Dumbo follows the CKPS paradigm that is only computationally secure (a weaker security model). 2) Dumbo is a patented and proprietary system and does not have an open-source implementation. 3) Dumbo, too, can benefit from this paper by using our improved ABA. 4) We analyze the concrete number of steps and cryptographic operations of Dumbo-Pillar, HoneyBadgerBFT, BEAT, ACE, and PACE. According to our concrete analysis in Appendix A, Dumbo-Pillar (even using our two-round ABA), would have fewer steps than ours only when n is prohibitively large ($n > 1000$). 5) Even if ACE or PACE may be slower than Dumbo when n is large enough, Dumbo cannot be used in any of the following that the BKR paradigm or its variants can handle only: MPC without trusted setup; asynchronous distributed key generation (e.g., [33, 41]), etc.

We, however, roughly compare Dumbo and PACE based on published data in terms of throughput and scalability. Note HoneyBadgerBFT, BEAT, Dumbo, ACE, and PACE use the same VM type (t2.medium) and use the same message size (250 B). We first observe that it is evident that PACE has significantly higher throughput than Dumbo when n is small. Even for the largest setting of $f = 30$, the throughput of PACE is still about 3x that of Dumbo for $B =$ 5,000.

The number of proposals delivered in different BFT frameworks. We devise an experiment to compare the "acceptance rate" between the BKR framework and our framework. By acceptance rate, we mean the average number of proposals delivered for *each epoch*, if replicas propose disjoint sets of transactions. More specifically, we need to analyze the number of ABA or RABA instances that decide 1. Just for performance comparison, we implement an incorrect framework directly running all instances in parallel: after delivering $n - f$ RBC instances, a replica immediately starts the ABA instances that have not been started by proposing 0. We call it NAIVE, because in failure cases, its throughput can be 0 [11, 49].

We summarize in Table 4 the number of ABA instances deciding 1, corresponding to the number of proposals delivered in failure-free scenarios for $f = 1$ in the WAN setting. We run the experiments 50 times for each network size and report the average number for all experiments. For almost all cases, the number of ABA instances that terminate with 1 in the BKR framework is close to $n - f$. In contrast, the number of ABA instances that terminate with 1 in NAIVE is visibly lower. This is because replicas do not wait for $n - f$ ABA instances to terminate with 1 before starting other ABA instances. Therefore, the number of delivered batches can be much lower than $n - f$. For our new framework, replicas tend to deliver 1 in ABA, and the number of ABA instances that terminate with 1 is shown to be slightly higher than that in BKR. Roughly, while our framework reduces the ABA phase latency (from two subphases to one), the "acceptance rate" (efficiency) of our framework is also higher.

10 CONCLUSION

We present Pillar that is by far the most efficient ABA protocol assuming regular common coins and authenticated channels only. Pillar is nearly optimal, requiring just 2 or 3 steps per round. We propose the notion of reproposable ABA (RABA) and use it to solve a long-standing problem of running ABA concurrently, leading to a fully parallelizable BFT framework outperforming prior ones in terms of *concrete* time complexity. We provide efficient instantiations of RABA and the new BFT framework. We show our BFT protocol outperforms existing protocols in terms of all metrics for practically large systems. All our instantiations rely on the well-established CDH assumption and achieve standard 128-bit security.

ACKNOWLEDGMENT

We thank Yang Yu for writing a python program for the $Y(n)$ function in Figure 12. We thank Xiao Sui and Xiaoyun Wang for insightful comments.

REFERENCES

[1] 2019. Bug in ABA protocol's use of Common Coin. https://github.com/amiller/HoneyBadgerBFT/issues/59. (2019).

[2] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically Optimal Validated Asynchronous Byzantine Agreement. In *Proceedings of the Symposium on Principles of Distributed Computing*. ACM, 337–346.

[3] Nicolas Alhaddad, Sisi Duan, Mayank Varia, and Haibin Zhang. 2021. Succinct Erasure Coding Proof Systems. *Cryptology ePrint Archive* (2021).

[4] Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. High-Threshold AVSS with Optimal Communication Complexity. In *FC 2021*.

[5] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. 2011. Prime: Byzantine replication under attack. *IEEE Transactions on Dependable and Secure Computing* 8, 4 (2011), 564–577.

[6] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger fabric: A distributed operating system for permissioned blockchains. *EuroSys*.

[7] P. Aublin, S. B. Mokhtar, and V. Quéma. 2013. RBFT: Redundant Byzantine Fault Tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*. 297–306. https://doi.org/10.1109/ICDCS.2013.53

[8] M. Joye B. Libert and M. Yung. 2016. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. In *Theoretical Computer Science*.

[9] Michael Ben-Or. 1985. Fast Asynchronous Byzantine Agreement (Extended Abstract) *(PODC '85)*.

[10] Michael Ben-Or, Ran Canetti, and Oded Goldreich. 1993. Asynchronous Secure Computation *(STOC)*.

[11] Michael Ben-Or and Ran El-Yaniv. 2003. Resilient-Optimal Interactive Consistency in Constant Time. *Distrib. Comput.* (2003).

[12] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. 1994. Asynchronous secure computations with optimal resilience. In *Proceedings of the 13th annual symposium on Principles of distributed computing*. ACM, 183–192.

[13] Piotr Berman and Juan A Garay. 1993. Randomized distributed agreement revisited. In *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*. IEEE, 412–419.

[14] Alexandra Boldyreva. 2002. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Public Key Cryptography — PKC 2003*.

[15] Gabriel Bracha. 1984. An asynchronous [(n-1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*. ACM, 154–162.

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

[17] Christian Cachin, Daniel Collins, Tyler Crain, and Vincent Gramoli. 2019. Byzantine Fault Tolerant Vector Consensus with Anonymous Proposals. *arXiv preprint arXiv:1902.10010* (2019).

[18] Christian Cachin, Rachid Guerraoui, and Lus Rodrigues. 2011. *Introduction to Reliable and Secure Distributed Programming* (2nd ed.).

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

[20] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology* 18, 3 (2005), 219–246.

[21] Christian Cachin and Jonathan A. Poritz. 2002. Secure INtrusion-Tolerant Replication on the Internet. In *Proceedings International Conference on Dependable Systems and Networks*. 167–176.

[22] Christian Cachin and Stefano Tessaro. 2005. Asynchronous verifiable information dispersal. In *SRDS*. IEEE, 191–201.

[23] Christian Cachin and Marko Vukolic. 2017. Blockchain Consensus Protocols in the Wild. In *DISC*.

[24] Ran Canetti and Tal Rabin. 1993. Fast asynchronous Byzantine agreement with optimal resilience. In *STOC*, Vol. 93. Citeseer, 42–51.

[25] Tushar Deepak Chandra and Sam Toueg. 1996. Unreliable Failure Detectors for Reliable Distributed Systems. *J. ACM* 43, 2 (1996).

[26] Ashish Choudhury and Arpita Patra. 2017. An Efficient Framework for Unconditionally Secure Multiparty Computation. *IEEE Transactions on Information Theory* 63, 1 (2017), 428–468.

[27] Allen Clement, Edmund L Wong, Lorenzo Alvisi, Michael Dahlin, and Mirco Marchetti. 2009. Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults.. In *NSDI*, Vol. 9. 153–168.

[28] Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. 2004. How to tolerate half less one Byzantine nodes in practical distributed systems. In *SRDS*. IEEE, 174–183.

[29] Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. 2006. From Consensus to Atomic Broadcast: Time-Free Byzantine-Resistant Protocols without Signatures. *Comput. J.* 49, 1 (2006), 82–96.

[30] Tyler Crain. 2020. A Simple and Efficient Asynchronous Randomized Binary Byzantine Consensus Algorithm. *CoRR* abs/2002.04393 (2020). arXiv:2002.04393 https://arxiv.org/abs/2002.04393

[31] Tyler Crain. 2020. Two More Algorithms for Randomized Signature-Free Asynchronous Binary Byzantine Consensus with t<n/3 and O(n^2) Messages and O(1) Round Expected Termination. *CoRR* abs/2002.08765 (2020). arXiv:2002.08765 https://arxiv.org/abs/2002.08765

[32] George Danezis, Eleftherios Kokoris Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus *(arxiv.org/abs/2105.11827)*.

[33] Sourav Das, Zhuolun Xiang, and Ling Ren. 2021. Asynchronous data dissemination and its applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2705–2721.

[34] Sisi Duan, Michael K Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2028–2041.

[35] Sisi Duan, Haibin Zhang, and Boxin Zhao. 2022. WaterBear: Information-Theoretic Asynchronous BFT Made Practical. *IACR Cryptol. ePrint Arch.* 2022 (2022).

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

[37] Roy Friedman, Achour Mostefaoui, and Michel Raynal. 2005. Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems. *IEEE Transactions on Dependable and Secure Computing* 2, 1 (2005), 46–56.

[38] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. 2021. Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback. *arXiv preprint arXiv:2106.10362* (2021).

[39] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster Asynchronous BFT Protocols.. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*.

[40] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All You Need is DAG. *(PODC '21)*.

[41] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. 2020. Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures. *(CCS)*.

[42] Klaus Kursawe and Victor Shoup. 2005. Optimistic Asynchronous Atomic Broadcast. In *ICALP*. 204–215.

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

[44] Chao Liu, Sisi Duan, and Haibin Zhang. 2020. EPIC: Efficient Asynchronous BFT with Adaptive Security.. In *DSN*.

[45] Chao Liu, Sisi Duan, and Haibin Zhang. 2021. MiB: Asynchronous BFT with More Replicas. (2021). arXiv:2108.04488

[46] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. 2019. HoneyBadgerMPC and AsynchroMix: Practical Asynchronous MPC and Its Application to Anonymous Communication. In *CCS*.

[47] Yuan Lu, Zhenliang Lu, and Qiang Tang. 2021. Bolt-Dumbo Transformer: Asynchronous Consensus As Fast As Pipelined BFT. *arXiv preprint arXiv:2103.09425* (2021).

[48] Ethan MacBrough. 2018. Cobalt: BFT governance in open networks. *arXiv preprint arXiv:1802.07240* (2018).

[49] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proceedings of the SIGSAC Conference on*

Computer and Communications Security*. ACM, 31–42.

[50] Henrique Moniz, Nuno Ferreria Neves, Miguel Correia, and Paulo Verissimo. 2008. RITAS: Services for randomized intrusion tolerance. *IEEE transactions on dependable and secure computing* 8, 1 (2008), 122–136.

[51] Achour Mostefaoui, Hamouma Moumen, and Michel Raynal. 2014. Signature-free asynchronous byzantine consensus with t< n/3 and o (n^2) messages. In *PODC*. ACM, 2–9.

[52] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. 2015. Signature-Free Asynchronous Binary Byzantine Consensus with t < n/3, O(n2) Messages, and O(1) Expected Time. *J. ACM* 62, 4 (2015), 31:1–31:21.

[53] Michael O Rabin. 1983. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*. IEEE, 403–409.

[54] HariGovind V. Ramasamy and Christian Cachin. 2006. Parsimonious Asynchronous Byzantine-fault-tolerant Atomic Broadcast. In *Proceedings of the 9th International Conference on Principles of Distributed Systems (OPODIS)*. Springer-Verlag, Berlin, Heidelberg, 88–102. https://doi.org/10.1007/11795490_9

[55] Victor Shoup. 2000. Practical Threshold Signatures. In *Advances in Cryptology — EUROCRYPT 2000*.

[56] Yee Jiun Song and Robbert van Renesse. 2008. Bosco: One-step byzantine asynchronous consensus. In *International Symposium on Distributed Computing*. Springer, 438–450.

[57] Joao Sousa, Alysson Bessani, and Marko Vukolic. 2018. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In *2018 48th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 51–58.

[58] TK Srikanth and Sam Toueg. 1987. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing* 2, 2 (1987), 80–94.

[59] Pierre Tholoniat and Vincent Gramoli. 2019. Formal verification of blockchain Byzantine fault tolerance. In *FRIDA*.

[60] Sam Toueg. 1984. Randomized byzantine agreements. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*. ACM, 163–178.

[61] Marko Vukolić. 2015. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *International workshop on open problems in network security*. Springer, 112–125.

[62] Marko Vukolić. 2017. Rethinking permissioned blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM, 3–7.

A CONCRETE RUNNING TIME COMPARISON

We provide a concrete running time comparison among HoneyBadgerBFT, BEAT, Dumbo, Dumbo-Pillar, ACE, and PACE.

A.1 Concrete Expected Time for our BFT Framework

We first examine the concrete expected number of rounds for our new BFT framework and for PACE. Let $X_1, X_2, \cdots, X_n$ be independent random variables and for $i \in [1..n]$ and $k \geq 1$, $\Pr[X_i = k] = 2^{-k}$. Let $X^{(n)} = \max\{X_1, \cdots, X_n\}$ and $Y(n) = E(X^{(n)})$. We have

$$Y(n) = E(X^{(n)}) = \sum_{k=1}^{+\infty} k[(1-2^{-k})^n - (1-2^{-(k-1)})^n]. \quad (1)$$

It is easy to show the running time of our new framework $Y(n) = O(\log n)$; a computer-aided simulation on Equation 1 shows $Y(n)$ is bounded between $\log_2(n) + 1$ and $\log_2(n) + 2$ for practically large n's, as shown in Figure 12.

For the BKR paradigm, the expected number of ABA rounds needed is $Y(2f + 1) + Y(f)$. The first item and the second item correspond to the first subphase and the second subphase, respectively. The value is apparently larger than $Y(n) = Y(3f + 1)$.

A.2 Running Time Comparison

To perform a fair comparison, Moreover, we let Dumbo-Pillar incorporate the more efficient ABA construction introduced in this paper (namely, Pillar). We compare HoneyBadgerBFT (using Cobalt), BEAT-Cobalt (using Cobalt), Dumbo-Pillar (using Pillar), DAG-Rider, ACE (using Pillar), and PACE (using Pisa) in terms of concrete running time complexity—the number of steps and the number of pairing operations—for both best-case and average-case scenarios. Both the number of steps and pairing operations are vital to the concrete running time: the former dominates the network latency and the latter dominates the computational latency at replicas. Recall a pairing operation is about 10x slower than elliptic curve cryptography (for the same level of security). Best cases represent scenarios where all ABA instances terminate in one round. We stress that best cases are not unusual, e.g., when networks are synchronous or there is no contention. Average cases represent the usual expected number of steps and pairing operations. Evaluation for both scenarios are needed to understand the running time of protocols.

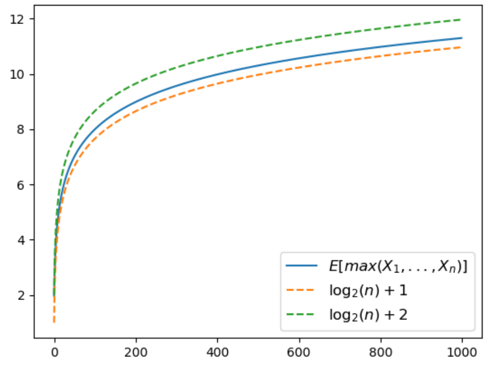


Figure 12: With a simple python program, we find Y is lower bounded by $\log_2(n) + 1$ and upper bounded by $\log_2(n) + 2$ (for $n \leq 1,000$).

PACE has significantly fewer steps than HoneyBadger and BEAT. For instance, when $n = 31$, PACE has 15.6 steps, while HoneyBadger and BEAT have 34.5 steps. When $n = 91$, PACE has 18.7 steps, but HoneyBadger and BEAT have 43.6 steps. This shows that our improvement is significant.

We find Dumbo-Pillar has (much) more steps and pairing operations, for both best-case and average-case scenarios, than our protocols for any practically large n's. Namely, Dumbo and Dumbo-Pillar have a large hidden factor in time complexity.

A computer-aided calculation on Equation 1 shows Dumbo-Pillar has fewer rounds than PACE only when n is significantly (prohibitively) large—$n > 1000$. It does not seem that the case for this large n is interesting, because when n is about this large, the asynchronous BFT protocols that we know have (unacceptably) low throughput. Similarly, DAG-Rider (21 steps) has fewer rounds than PACE when n is larger than 150.

In terms of paring operations, PACE is pairing-free, while Dumbo-Pillar requires $n^3 + 12n^2$ pairing operations. When $n = 31$, 91 and 121, the number of pairing operations in Dumbo-Pillar is 2.72×10^5, 8.53×10^5, and 1.95×10^6, respectively.

Dumbo has higher latency than HoneyBadgerBFT and BEAT when n is small or reasonably large and in the LAN setting, while we have shown that PACE has much better latency than BEAT in these settings. Thus, PACE outperforms Dumbo and Dumbo-Pillar in terms of latency in these cases. If using a pairing type achieving 128 bit security, say, the popular BN curve, we find one pairing operation takes about 4.3 ms in the same EC2 machine that both Dumbo and PACE use. The time for pairing operations alone, without counting client-to-server, server-to-server, and server-to-client communication latency, or other cryptographic operations, has already been far larger than the entire running time of BEAT, and so larger than ACE or PACE.

			concrete time complexity	$n = 7$ $f = 2$	$n = 16$ $f = 5$	$n = 31$ $f = 10$	$n = 61$ $f = 20$	$n = 91$ $f = 30$	$n = 121$ $f = 40$	$n = 151$ $f = 50$	$n = 301$ $f = 100$
HoneyBadgerBFT (Cobalt ABA)	#steps	best	6	6	6	6	6	6	6	6	6
		average	$3 + 3[Y(2f + 1) + Y(f)]$	22.38	28.95	34.45	40.19	43.61	46.06	47.96	53.91
	#paring	best	$n^3 + 2n^2$	441	4,608	31,713	2.34×10^5	7.7×10^5	1.8×10^6	3.49×10^6	2.75×10^7
		average	$n^3[Y(2f + 1) + Y(f)] + 2n^2$	2,264	35,686	$3.13 * 10^5$	2.81×10^6	1.02×10^7	2.54×10^7	5.16×10^7	4.64×10^8
BEAT (Cobalt ABA)	#steps	best	6	6	6	6	6	6	6	6	6
		average	$3 + 3[Y(2f + 1) + Y(f)]$	22.38	28.95	34.45	40.19	43.61	46.06	47.96	53.91
	#paring	best	0	0	0	0	0	0	0	0	0
		average	0	0	0	0	0	0	0	0	0
Dumbo-Pillar (Pillar)	#steps	best	14	14	14	14	14	14	14	14	14
		average	26	26	26	26	26	26	26	26	26
	#paring	best	$n^3 + 8n^2$	735	6,144	37,479	2.56×10^5	8.2×10^5	1.89×10^6	3.63×10^6	2.8×10^7
		average	$n^3 + 12n^2$	931	7,168	41,323	2.72×10^5	8.53×10^5	1.95×10^6	3.72×10^6	2.84×10^7
ACE (Pillar)	#steps	best	5	5	5	5	5	5	5	5	5
		average	$3 + 2[Y(2f + 1) + Y(f)]$	15.92	20.30	23.96	27.79	30.07	31.70	32.97	36.94
	#paring	best	0	0	0	0	0	0	0	0	0
		average	0	0	0	0	0	0	0	0	0
PACE (Pisa)	#steps	best	4	4	4	4	4	4	4	4	4
		average	$3 + 2Y(n)$	11.48	13.75	15.62	17.55	18.69	19.51	20.15	22.13
	#paring	best	0	0	0	0	0	0	0	0	0
		average	0	0	0	0	0	0	0	0	0

Table 5: Concrete running time comparison. "Best case" for ABA represents scenarios where all ABA instances terminate in one round; best cases are not unusual (e.g., when networks are synchronous or there is no contention). Note the best case for ABA is two (for Pillar) or three steps (for Pillar), while the best case for our RABA Pisa is 1 step. "Average case" represents the usual expected number of steps and pairing operations. The function $Y(\cdot)$ is evaluated according to Equation 1. When counting the number of steps and pairing operations, Cobalt ABA has 3 steps per round and Pillar and Pisa has 2 steps per round.

B REVIEW OF MMR AND COBALT ABA

The pseudocode of MMR is given in Figure 13. First, each replica broadcasts $\mathsf{bval}_r(est_r)$ where est_r is the input of the round (in round 0, est_r is the ABA vote). If a replica receives $f+1$ $\mathsf{bval}_r(v)$ and has not broadcast v, it broadcasts $\mathsf{bval}_r(v)$. Upon receiving $n-f$ $\mathsf{bval}_r(v)$, a replica adds v to bin_values_r. Then each replica sends an $\mathsf{aux}_r()$ message for the first value added to bin_values_r. Upon receiving $n-f$ $\mathsf{aux}_r()$ messages such that the set of values carried by these message, $vals$, is a subset of bin_values_r, the replica compares the value(s) with the common coin. (By default, $\perp$ is an element of bin_values_r.) If there is only one value in $vals$ messages and the value is the same as the coin, the replica decides it. Otherwise, the replica enters the next round and uses the common coin as est_r.

MMR ABA uses authenticated channels only and has 2 or 3 steps per round. In practice, there is a liveness issue discovered independently in [1, 59]. As illustrated in Figure 14, an adversary can make correct replicas always move to the next round with inconsistent values, and the protocol will never terminate.

MacBrough proposed a solution in the Cobalt protocol [48] which has one additional $\mathsf{conf}_r()$ step in each round, also as shown in Figure 13.

```
upon event propose($v_i$)
  if $r = 0$, $est_0 \leftarrow v_i$
round $r$
  $bin\_values_r \leftarrow \emptyset$
  broadcast $\mathsf{bval}_r(est_r)$
  upon receiving $\mathsf{bval}_r(v)$ from $f+1$ replicas
    if $\mathsf{bval}_r(v)$ has not been sent, broadcast $\mathsf{bval}_r(v)$
  upon receiving $\mathsf{bval}_r(v)$ from $n-f$ replicas
    $bin\_values_r \leftarrow bin\_values_r \cup \{v\}$
  wait until $bin\_values_r \neq \emptyset$
    broadcast $\mathsf{aux}_r(v)$ where $v \in bin\_values_r$
  upon receiving $n-f$ $\mathsf{aux}_r()$ such that the set of values carried by these messages,
  $vals$, is a subset of $bin\_values_r$
    broadcast $\mathsf{conf}_r(vals)$
  upon receiving $n-f$ $\mathsf{conf}_r()$ such that the set of values carried by these messages,
  $vals$, is a subset of $bin\_values_r$
    $s \leftarrow \mathsf{coin}_r$
    if $vals = \{b\}$
      $est_{r+1} \leftarrow b$
      if $b = s$, decide($b$)
    else $est_{r+1} \leftarrow s$
  $r \leftarrow r+1$
```

Figure 13: MMR ABA and Cobalt ABA. Cobalt ABA has the boxed code, while MMR ABA does not have it.

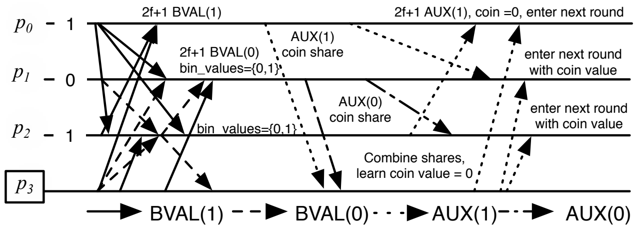


Figure 14: The liveness issue of MMR. A faulty replica p_3 first sends $\mathsf{bval}_r(1)$ to p_0. Since p_0 receives $\mathsf{bval}_r(1)$ from p_2 and itself, p_0 will send $\mathsf{aux}_r(1)$ and its threshold signature shares for the common coin. p_3 generates a share and combines the $f+1=2$ shares to obtain the common coin value (e.g., 0). p_3 then makes p_2 send $\mathsf{aux}_r(1)$ by letting p_2 receive 3 $\mathsf{bval}_r(1)$. Also, p_3 sends $\mathsf{aux}_r(1)$ to p_0, making p_0 receive 3 $\mathsf{aux}_r(1)$ (a value different from the common coin) and use 1 as the input for the next round. For p_1 and p_2, since they have added both 0 and 1 in their bin_values_r and receive both $\mathsf{aux}_r(0)$ and $\mathsf{aux}_r(1)$, they will enter the next round with the common coin value 0. In this way, the protocol may never terminate.

C PROOF OF CORRECTNESS FOR PILLAR

In this section, we prove the correctness of Pillar.

LEMMA C.1. *In round $r > 0$, if a correct replica p_i sets $\delta_r(v) = 1$ and another correct replica sets $\delta_r(\bar{v}) = 1$, $v = \bar{v}$.*

Proof. In round $r > 0$, correct replicas may send $\mathsf{bval}_r(*, v)$, $\mathsf{bval}_r(*, \bar{v})$, or $\mathsf{bval}_r(*, \perp)$ and do not change the maj_r value in the same round. There are two cases: $v = s_{r-1}$ and $v = \bar{s}_{r-1}$. We first show the case for $v = s_{r-1}$. Assume, towards a contradiction, there exists a replica p_i that receives $2f+1$ $\mathsf{bval}_r(v, v)$ and $\mathsf{bval}_r(v, \perp)$. In our protocol, if p_i receives $2f+1$ $\mathsf{bval}_r(v, v)$ and $\mathsf{bval}_r(v, \perp)$ and sets $\delta_r(v) = 1$, at least $f+1$ correct replicas have sent $\mathsf{bval}_r(v, v)$ or $\mathsf{bval}_r(v, \perp)$, but will never send $\mathsf{bval}_r(v, \bar{v})$ or $\mathsf{bval}_r(\bar{v}, \bar{v})$. On the other hand, if p_j sets $\delta_r(\bar{v}) = 1$, it has received $2f+1$ $\mathsf{bval}_r(\bar{v}, \bar{v})$, at least $f+1$ of which are sent by correct replicas. Therefore, at least one correct replica has sent both $\mathsf{bval}_r(v, v)$ (or $\mathsf{bval}_r(v, \perp)$) and $\mathsf{bval}_r(\bar{v}, \bar{v})$, which is a contradiction. The case for $v = \bar{s}_{r-1}$ can be proved similarly. If p_i sets $\delta_r(v) = 1$, it receives $2f+1$ $\mathsf{bval}_r(v, v)$. If p_j sets $\delta_r(\bar{v}) = 1$, it receives $2f+1$ $\mathsf{bval}_r(\bar{v}, \bar{v})$ or $\mathsf{bval}_r(\bar{v}, \perp)$. In other words, at least one correct replica has sent both $\mathsf{bval}_r(*, v)$ and $\mathsf{bval}_r(*, \bar{v})$ (or $\mathsf{bval}_r(*, \perp)$), which is impossible. □

LEMMA C.2. *In round $r > 0$, if all correct replicas have the same input v, any correct replica either enters round $r+1$ with v, or decides v in round r.*

Proof. In round $r > 0$, correct replicas may send $\mathsf{bval}_r(v, v)$, $\mathsf{bval}_r(v, \bar{v})$, or $\mathsf{bval}_r(v, \perp)$. Hence, correct replicas will not receive more than $f+1$ $\mathsf{bval}_r(\bar{v}, *)$ and no correct replica will put $\bar{v}$ in its bin_values_r. Therefore, all correct replicas will send $\mathsf{aux}_r(v, v)$ or $\mathsf{aux}_r(\perp, v)$ and will not accept $\mathsf{aux}_r(\bar{v}, *)$ or $\mathsf{aux}_r(*, \bar{v})$. Each correct replica therefore eventually receives either $2f+1$ $\mathsf{aux}_r(v, *)$ such that $\mathsf{V}_2(vals_r, v) \geq 2f+1$ or $2f+1$ $\mathsf{aux}_r(*, v)$ such that $\mathsf{V}_2(avals_r, v) \geq 2f+1$.

If a replica p_i receives $2f+1$ $\mathsf{aux}_r(v, v)$, it either decides v (for the case $v = s_r$), or enters the next round with v as est_{r+1} (for the case $v \neq s_r$). If p_i receives both $\mathsf{aux}_r(v, v)$ and $\mathsf{aux}_r(\perp, v)$ (or only $\mathsf{aux}_r(\perp, v)$), $\mathsf{V}_2(avals_r, v) \geq 2f+1$ is satisfied. In this case, p_i will either decide v (for the case $v = s_r = s_{r-1}$), or enter the next round with v as est_{r+1} (for the case $v \neq s_r$ or $v \neq s_{r-1}$). □

THEOREM C.3. ***(Validity)*** *If all correct replicas propose v, then any correct replica that terminates decides v.*

PROOF. The proof follows from the following two lemmas.

LEMMA C.4. *In round $r > 0$, if all correct replicas have the same input v, any correct replica that terminates decide v.*

Proof. Lemma C.2 shows that correct replicas will either enter round $r + 1$ with v or decide v. If correct replicas start round r with the same input v and enter the next round, the input for round $r + 1$ must be v. As the probability that the common coin value equals v is 1/2, the probability that the protocol terminates in round $r + 1$ is 1/2. It is straightforward to see that any replica that terminates decides v. □

Lemma C.5. *If all correct replicas propose v in round 0, any correct replica that terminates decides v.*

Proof. We show that if all correct replicas propose est_0 in round 0, correct replicas either terminate in the current round with est_0 or enter round 1 with the same est_r. As proven in Lemma C.2 and Lemma C.4, any correct replica that terminates decides v.

In round $r = 0$, all correct replicas broadcast $\text{bval}_r(v, \perp)$. Since all correct replicas have the same input v and there are only f faulty replicas, correct replicas will not receive more than $f + 1$ $\text{bval}_r(\bar{v}, \perp)$ or send $\text{bval}_r(\bar{v}, \perp)$. All correct replicas will eventually receive $2f + 1$ $\text{bval}_r(v, \perp)$ and send $\text{aux}_r(v, v)$. Since no correct replica puts $\bar{v}$ in bin_values_r, all correct replicas will have only one single value v. If a correct replica terminates in round 0 and $v = s_r$, it decides v. If $v \neq s_r$, correct replicas use v as the input est_r for round 1. According to Lemma C.4, any correct replica that terminates decides v. □

This completes of the proof of the theorem. ■

Theorem C.6. ***(Agreement)*** *If a correct replica decides v, then any correct replica that terminates decides v.*

Proof. We prove agreement by showing that if p_i decides v in round r, all other correct replicas either decide in the same round or enter the next round with v as est_r. Assume, towards a contradiction, that another correct replica p_j enters the next round with $\bar{v}$.

Lemma C.7. *If p_i terminates in round r and decides v, any correct replica p_j either 1) terminates in round r and decides v; or 2) enters round $r + 1$ with v as est_{r+1}.*

Proof. If p_i decides v in round r, there are three cases: 1) p_i receives at least $2f + 1$ $\text{aux}_r(v, v)$ messages and $v = s_r$; 2) p_i receives both $\text{aux}_r(v, v)$ and $\text{aux}_r(\perp, *)$. Also, p_i receives at least a quorum of $\text{aux}_r(*, v)$ messages, $v = s_{r-1}$, and $v = s_r$; For p_j, if it enters the next round with value $est_{r+1} = \bar{v}$, it cannot use the common coin value as input. This is because v is the common coin. Therefore, one the following conditions must apply: A) p_j receives at least $2f + 1$ $\text{aux}_r(\bar{v}, \bar{v})$; B) p_j receives both $\text{aux}_r(\bar{v}, \bar{v})$ and $\text{aux}_r(\perp, \bar{v})$. At least a quorum of the messages are of the form $\text{aux}_r(*, \bar{v})$; C) p_j receives both $\text{aux}_r(\bar{v}, *)$ and $\text{aux}_r(\perp, *)$ and $\bar{v} = s_{r-1}$. We now distinguish the two cases for p_i and show that none of the three conditions for p_j can be satisfies.

Case 1: p_i receives at least $2 + 1$ $\text{aux}_r(v, v)$. At least $f + 1$ correct replicas have sent $\text{aux}_r(v, v)$. 1) If condition A is true, p_j receives $2f + 1$ $\text{aux}_r(\bar{v}, \bar{v})$, at least $f + 1$ of which are sent by correct replicas. Therefore, at least one correct replica has sent both $\text{aux}_r(v, v)$ to p_i and $\text{aux}_r(\bar{v}, \bar{v})$ to p_j, which is impossible. Condition A cannot be true. 2) If condition B is true, p_j receives a quorum of $\text{aux}_r(*, \bar{v})$ messages, $f + 1$ of which are sent by correct replicas. Therefore, at least one correct replica has sent $\text{aux}_r(v, v)$ to p_i and sent $\text{aux}_r(*, \bar{v})$ to p_j, which is impossible. Condition B cannot be true. 3) If condition C is true, p_j receives only $\text{aux}_r(\bar{v}, *)$ and $\text{aux}_r(\perp, *)$. In other words, at least one correct replica has sent $\text{aux}_r(v, v)$ to p_i and $\text{aux}_r(\bar{v}, *)$ (or $\text{aux}_r(\perp, *)$) to p_j, which is impossible. Condition C cannot be true.

Case 2: p_i receives both $\text{aux}_r(v, v)$ *and* $\text{aux}_r(\perp, *)$. *Also, p_i receives at least a quorum of* $\text{aux}_r(*, v)$ *messages, $v = s_{r-1}$, and $v = s_r$.* 1) If condition A is true, p_j receives at least $2f + 1$ $\text{aux}_r(\bar{v}, \bar{v})$, at least $f + 1$ are sent by correct replicas. Also, p_i receives $2f + 1$ $\text{aux}_r(v, v)$ and $2f + 1$ $\text{aux}_r(\perp, *)$. Therefore, at least one correct replica must have sent $\text{aux}_r(\bar{v}, \bar{v})$ to p_j and $\text{aux}_r(v, v)$ (or $\text{aux}_r(\perp, *)$) to p_i, which is impossible. Condition A cannot be true. 2) If condition B is true, p_j receives both $\text{aux}_r(\bar{v}, \bar{v})$ and $\text{aux}_r(\perp, \bar{v})$, and at least a quorum of the messages are of the form $\text{aux}_r(*, \bar{v})$. Since p_i receives a quorum of $\text{aux}_r(*, v)$ messages, at least one correct replica has sent $\text{aux}_r(*, v)$ to p_i and $\text{aux}_r(*, \bar{v})$ to p_j. Condition B cannot be true. 3) p_j only receives $\text{aux}_r(\perp, *)$ and $\text{aux}_r(\bar{v}, *)$ and $\bar{v} = s_{r-1}$. This cannot be true since $v = s_{r-1}$. □

For the two cases, it is clear that if p_j decides in round r, it outputs v. We now show that if p_j terminates in round r' and decides $\bar{v}$, $v = \bar{v}$ where $r' > r$. In round r', one of the following cases must apply: A) p_j receives at least $2f + 1$ $\text{aux}_{r'}(\bar{v}, \bar{v})$; B) p_j receives both $\text{aux}_{r'}(\bar{v}, \bar{v})$ and $\text{aux}_{r'}(\perp, \bar{v})$. Also, $\bar{v} = s_{r'-1}$ and $\bar{v} = s_{r'}$. If any of the two cases is true, at least one correct replica has sent $\text{bval}_{r'}(\bar{v}, *)$. Meanwhile, according to Lemma C.7, any correct replica that enters round $r + 1$ uses v as input. Furthermore, according to Lemma C.2, if all correct replicas that enter round $r + 1$ either terminates or uses $est_{r+2} = v$ and enter round $r + 2$. Therefore, there exists some round r'' where $r \leq r'' \leq r'$, a correct replica uses $\bar{v}$ as input and broadcasts $\text{bval}_{r''}(\bar{v}, *)$. Therefore, Lemma C.2 is violated, a contradiction. ■

Theorem C.8. ***(Termination)*** *All correct replicas eventually terminate the protocol.*

Proof. The proof is divided in two parts: 1) In each round, each correct replica will eventually proceed to the next round, and 2) a correct replica terminates the protocol with probability 1/2.

We prove the first part. In our protocol, est_r is always a binary value, either 0 or 1. A correct replica may send $\text{bval}_r(v, *)$ or $\text{bval}_r(\bar{v}, *)$. Also, there are at least $2f + 1$ correct replicas, among which at least $f + 1$ correct replicas propose the same value. Therefore, if $\bar{v}$ is different from its proposed value, each correct replica p_i will forward $\text{bval}_r(\bar{v}, maj_r)$. Correct replicas will eventually receive $2f + 1$ $\text{bval}_r()$ messages for at least one binary value (e.g., v), and send $\text{aux}_r(v, *)$ or $\text{aux}_r(\perp, *)$. Similarly, all correct replicas eventually receive $2f + 1$ $\text{aux}_r()$ messages and proceed to the next round.

We now prove the second part. In particular, we prove in round r, if a correct replica p_i enters round $r + 1$ with $est_{r+1} = \bar{v}$, the protocol will terminate with 1/2 probability. (Hence, the $\bar{v}$ cannot be manipulated by the adversary such that $\bar{v}$ is always different from the common coin.)

If p_i enters the next round with $est_{r+1} = v = \bar{s}_r$, it must satisfy one of the three conditions: 1) p_i receives at least $2f + 1$ $\text{aux}_r(v, v)$ messages; 2) p_i receives both $\text{aux}_r(v, v)$ and $\text{aux}_r(\perp, *)$. At least a quorum of the messages are $\text{aux}_r(*, v)$; 3) p_i receives both $\text{aux}_r(v, v)$ and $\text{aux}_r(\perp, *)$ and $v = s_{r-1}$.

For the first case, at least $f+1$ correct replicas have sent $\mathsf{aux}_r(v,v)$. According to the Lemma C.1, we know that in each round, correct replicas send $\mathsf{aux}_r(v,v)$ for at most one value v. In other words, if at least one correct replica sends $\mathsf{aux}_r(v,v)$, no correct replica sends $\mathsf{aux}_r(\bar{v},\bar{v})$. Hence, each correct replica receives at least one $\mathsf{aux}_r(v,v)$. According to Lemma C.1, correct replicas may send $\mathsf{aux}_r(v,v)$, $\mathsf{aux}_r(\perp,v)$, $\mathsf{aux}_r(\perp,\bar{v})$, but will not send $\mathsf{aux}_r(\bar{v},*)$. No correct replica will set $\delta_r(\bar{v})=1$. The value v is determined based on the $majs$ values by the replicas so it cannot be manipulated by the adversary. Therefore, with a probability of 1/2, p_i decides v. Otherwise p_i enter the next round and use $est_{r+1}=v$.

For the second case, p_i decides if $b=s_{r-1}$ and $b=s_r$. With a probability of 1/2, $s_{r-1}=s_r$.

For the third case, with a probability of 1/2, $s_{r-1}=s_r$. In this case p_i will not decide. p_i uses the value of s_r as est_{r+1} with a probability of 1/2. ■

Theorem C.9. ***(Integrity)*** *No correct replica decides twice.*

Proof. In each round, a replica will only sends $\mathsf{aux}_r()$ message once and accepts only one $\mathsf{aux}_r()$ message from each replica. If a replica p_i decides v in round r, it has received $2f+1$ $\mathsf{aux}_r(v,*)$ messages with the same v, or received $2f+1$ $\mathsf{aux}_r(*,v)$. If p_i decides twice, it must have received $2f+1$ $\mathsf{aux}_r(\bar{v},\bar{v})$, or received $2f+1$ $\mathsf{aux}_r(*,v)$. Neither case is possible. Integrity thus follows. ■

D PROOF OF CORRECTNESS FOR PISA

In this section, we prove the correctness of Pisa.

Theorem D.1. ***(Validity)*** *If all correct replicas propose v and never repropose $\bar{v}$, then any correct replica that terminates decides v.*

Proof. If all correct replicas propose v and do not repropose $\bar{v}$, all correct replicas will only have v in their bin_values_r. Hence, replicas do not accept an $\mathsf{aux}_r(\bar{v},\bar{v})$ message. Each correct replica will collect $2f+1$ $\mathsf{aux}_r(v,v)$. If $v=1$, replicas decide in round 0. Otherwise replicas enter round 1. Starting from round 1, Pisa follows Pillar. Therefore, according to Lemma C.2 and Lemma C.4, any correct replica that terminates decides v. ■

Theorem D.2. ***(Unanimous termination)*** *If all correct replicas propose v and never repropose $\bar{v}$, then all correct replicas eventually terminate.*

Proof. If all correct replicas propose v, they will all send $\mathsf{bval}_r(v,\perp)$. All correct replicas eventually put v to bin_values_r. Correct replicas will not send or accept any $\mathsf{aux}_r(\bar{v},\bar{v})$ message and will accept either $\mathsf{aux}_r(v,v)$ or $\mathsf{aux}_r(\perp,v)$. According to Lemma C.2, all correct replicas will enter the next round with the same est_{r+1} value (including the case where replicas decide). According to the property of common coin, we know that correct replicas decide in each round with probability 1/2. Hence, all correct replicas eventually terminate. ■

Theorem D.3. ***(Agreement)*** *If a correct replica decides v, then any correct replica that terminates decides v.*

Proof. We first show the case where a correct replica p_i decides v in round $r=0$. First of all, a correct replica p_j cannot decide $\bar{v}$ in round 0. This is because a correct replica decides a value only when the value equals the common coin, which is 1 in round 0. We now consider the case where p_i still decides in round 0 and p_j decides in round $r>0$. We prove the following lemma:

Lemma D.4. *If p_i decides in round 0, any correct replica either decides in round 0 or uses 1 as input for round 1.*

Proof. If p_i decides in round 0, it decides $v=1$ (the common coin value in round 0 is 1). Assume, towards a contradiction, that a correct replica p_j enters round 1 with 0 as input. In this case, the function $\mathsf{V}_2(vals_r,0)\geq 2f+1$ must be true. Hence, excluding the $\mathsf{aux}_r(\perp,*)$ messages, the number of $\mathsf{aux}_r(0,0)$ messages p_j receives is greater than $2f+1$. Among the replicas that sent $2f+1$ $\mathsf{aux}_r(v,v)$ and $\mathsf{aux}_r(\bar{v},\bar{v})$ messages, at least one correct replica must have sent both $\mathsf{aux}_r(v,v)$ and $\mathsf{aux}_r(\bar{v},\bar{v})$. This is a contradiction, since a correct replica broadcasts $\mathsf{aux}_r()$ once in each round. □

Starting from round $r=1$, Pisa is the same as Pillar. Therefore, according to Lemma C.2 and Lemma C.4, any correct replica that terminates decides $v=1$.

For the case where p_i decides in round $r>0$, agreement simply follows that of Pillar, as Pisa is the same as Pillar starting from $r>0$. ■

Theorem D.5. ***(Biased validity)*** *If $f+1$ correct replicas propose 1, then any correct replica that terminates decides 1.*

Proof. In round 0, correct replicas will directly send $\mathsf{aux}_r(1,1)$ and will not send $\mathsf{aux}_r(0,0)$. Therefore, all correct replicas will either receive $2f+1$ $\mathsf{aux}_r(1,1)$ or both $\mathsf{aux}_r(1,1)$ and $\mathsf{aux}_r(0,0)$, but not $2f+1$ $\mathsf{aux}_r(0,0)$. This is because if a correct replica receives $2f+1$ $\mathsf{aux}_r(0,0)$, at least $f+1$ correct replicas must have sent $\mathsf{aux}_r(0,0)$. Therefore, at least one correct replica must have sent both $\mathsf{aux}_r(0,0)$ and $\mathsf{aux}_r(1,1)$, which is impossible. If a correct replica receives $2f+1$ $\mathsf{aux}_r(1,1)$, it directly decides. Otherwise it uses the common coin value 1 to enter the next round. Since Pisa is the same as Pillar starting from round 1, according to the Lemma C.2 and Lemma C.4, all correct replicas that terminate decide 1. ■

Theorem D.6. ***(Biased termination)*** *Let Q be the set of correct replicas. Let Q_1 be the set of correct replicas that propose 1 and never repropose 0. Let Q_2 be correct replicas that propose 0 and later repropose 1. If $Q_2\neq\emptyset$ and $Q=Q_1\cup Q_2$, then each correct replica eventually terminates.*

Proof. The proof consists of two parts: round $r=0$ and round $r>0$. We first prove the first case ($r=0$) that a correct replica either decides in round 0 or moves to round 1. Depending on the proposed values of replicas, there are three cases: 1) at least $f+1$ correct replicas propose 1; 2) at least one but fewer than $f+1$ correct replicas propose 1; 3) all correct replicas propose 0. We show that each replica can collect $2f+1$ $\mathsf{aux}_r()$ messages.

Case 1: More than $f+1$ correct replicas propose 1. All correct replicas will eventually receive $f+1$ $\mathsf{bval}_r(1,\perp)$. According to the protocol, all correct replicas will eventually receive $2f+1$ $\mathsf{bval}_r(1,\perp)$, put 1 in bin_values_r, and accept $\mathsf{aux}_r(1,1)$. Correct replicas may send $\mathsf{aux}_r(1,1)$ or $\mathsf{aux}_r(0,0)$. If a correct replica sends $\mathsf{aux}_r(0,0)$, it previously received $2f+1$ $\mathsf{bval}_r(0,\perp)$ messages, among which at least $f+1$ replicas are correct. Therefore, all correct replicas will eventually put 0 in their bin_values_r and accept $\mathsf{aux}_r(0,0)$. All correct replicas can then decide or move to round 1.

Case 2: At least one but fewer than $f+1$ correct replicas propose 1. In this case $|Q_1| < f+1$ and $|Q_2| \geq f+1$. This case implies that at least $f+1$ correct replicas propose 0. In this case, all correct replicas will eventually receive $2f+1$ $\mathsf{bval}_r(0,\bot)$ and put 0 to bin_values_r. It is, however, not guaranteed that a correct replica will receive $2f+1$ $\mathsf{bval}_r(1,\bot)$ and put 1 in bin_values_r. Therefore, some correct replica that only has 0 in bin_values_r will not accept an $\mathsf{aux}_r(1,1)$ message. The termination is guaranteed by the fact that correct replicas in Q_2 repropose 1. In particular, correct replicas will repropose 1, making correct replicas eventually receive a quorum of $\mathsf{bval}_0(1,\bot)$ messages. Hence, correct replicas will either enter the next round or eventually collect a quorum of $\mathsf{bval}_0(1,\bot)$ messages. In the latter case, correct replicas will be able to accept both $\mathsf{aux}_r(0,1)$ and $\mathsf{aux}_r(1,1)$ in round 0. Hence, correct replicas will either terminate in round 0 or move to the next round.

Case 3: All correct replicas have 0 *as their input.* In this case $|Q_1| = 0$. All correct replicas will receive a quorum of $\mathsf{bval}_r(0,\bot)$ messages and add 0 to bin_values_r. Furthermore, each replica in Q_2 may repropose. Therefore, all correct replicas will receive $2f+1$ $\mathsf{bval}_r(1,\bot)$ and put 1 to bin_values_r. It is easy to see that all correct replicas will either decide in round 0 or move to the next round.

For the case where round $r > 0$, since Pisa is the same as Pillar, agreement follows that of Pillar. This completes the proof of the theorem. ■

Theorem D.7. ***(Integrity)*** *No correct replica decides twice.*

Proof. In each round, each replica will only send $\mathsf{aux}_r()$ message once and accept one $\mathsf{aux}_r()$ message from each replica. Hence, only one value will be decided and integrity easily follows. ■

E PROOF OF CORRECTNESS FOR OUR BFT FRAMEWORK (AND PACE)

We prove the correctness of our BFT framework. The proof immediately implies the correctness of our BFT instantiation (PACE). In particular, we will prove the following properties that are equivalent to the definitions of security for BFT:

- **Set agreement**: If any correct replica outputs a set V, then each correct replica outputs V.
- **Efficiency (validity)**: If a correct replica outputs a set V, then V contains a proposal from at least one correct replica.
- **Liveness**: If a proposal is submitted to all correct replicas, then all correct replicas eventually output a set containing some value.

Note that the above definitions generalize and relax prior definitions for systems on asynchronous common subset (ACS) such as HoneyBadgerBFT, BEAT, and Dumbo, where they all consider the following efficiency property:

- **Efficiency (validity) for some prior ACS definitions**: If a correct replica outputs a set V, then V contains proposals from at least $n-2f$ correct replicas.

The original idea of ACS in BKR requires replicas to agree on a common subset but does not levy any restriction on the size of the set V. Our efficiency definition thus echoes that of the original BKR paper, requiring V contains at least one proposal from one correct replica. This is—not at all—a "drawback." First, asking V to contain proposals from $n-2f$ correct replicas is unnecessary; our relaxed definitions are equivalent to the standard definitions for BFT. Second, the efficiency property defined in the previous work may restrict novel or efficient constructions: a system slowly delivering more transactions may not be more efficient than a system delivering fewer transactions but delivering them faster. Third, one can *easily* construct a system that satisfies the efficiency property requiring to output a set containing at least $n-2f$ replicas using a system with our efficiency property.

We begin with the following lemma.

Lemma E.1. *If all correct replicas are activated on some proposals for epoch e, then all correct replicas eventually terminate for epoch e.*

Proof. If all correct replicas are activated on some proposals for some epoch e, they will r-broadcast their proposals. Eventually, all correct replicas will r-deliver at least $2f+1$ RBC instances. Hence, all correct replicas will proposes 0 for all RABA instances that have not been started. We distinguish all possible cases and show for each case all RABA instances will terminate.

We first consider case 1, where all correct replicas propose 1 for a RABA. In this case, according to unanimous termination, the RABA instance eventually terminates.

We now consider case 2, where all correct replicas propose 0. In this case, we further distinguish two sub-cases: 1) If they never repropose 1, the RABA instance eventually terminates due to unanimous termination. 2) If some replicas repropose 1, then these replicas must have r-delivered the corresponding messages. According to the agreement property of RBC, all correct replicas will deliver the messages and repropose 1. The protocol will terminate due to biased termination.

Finally, we consider case 3, where some correct replicas propose 0 and some other correct replicas propose 1. The case is similar to Case 2-2. Due to the agreement property of RBC, correct replicas will eventually repropose 1, and the RABA instance will terminate.

Therefore, the protocol will eventually terminate. □

We now prove set agreement.

Theorem E.2. ***(Agreement)*** *If any correct replica outputs a set V of proposals from replicas, then each correct replica outputs the same set V.*

Proof. We consider an epoch e, where a correct replica p_j a-delivers a set V. According to our protocol, the set V is a set of proposals from different replicas: the i-th element, $V[i]$, may be empty or m_i (a proposal r-broadcast by p_i), depending on if the corresponding RABA instance RABA_i decides 0 or 1, where $i \in [0..n-1]$. We just need to show that each replica p_k will output a set V' such that $V' = V$, i.e., $V[i] = V'[i]$ for $i \in [0..n-1]$.

If p_j outputs a set V, then all RABA instances either decide 0 or 1. According to Lemma E.1, we know all RABA instances must terminate. Due to the agreement property of RABA, these RABA instances decide the same values for p_k. Therefore, all RABA instances for p_k will terminate and decide the same values as p_j. Furthermore, the agreement of RBC instances guarantees that, $V'[i]$ will be r-delivered and $V[i] = V'[i] = m_i$ for all $i \in [0..n-1]$. ■

The above theorem immediately implies the usual agreement definition for BFT. We now prove a theorem implying efficiency and liveness.

Theorem E.3. *For each epoch, a set V containing at least $f+1$ non-empty elements will be output.*

Proof. For simplicity, we assume $n = 3f + 1$. Conditioned on termination for all RABA instances (shown in Lemma E.1), we now bound the number of RABA instances that decide 1, which corresponds to the number of non-empty elements. We mainly use the biased termination property to prove the theorem for this proof.

According to the biased termination property, a RABA instance RABA_i will decide 1, if $f+1$ or more correct replicas propose 1. We now need to bound the number of RABA instances where less than $f+1$ correct replicas propose 1.

A crucial observation is that a correct replica will propose 1 for at least $2f+1$ RABA instances, a fact guaranteed by RBC. All correct replicas will input 1 for $(2f+1)(2f+1)$ for all RABA instances. There are at most $(3f+1)(2f+1)$ inputs for correct replicas. Hence, the total number of the 0 input from all correct replicas for all RABA instances is at most $(3f+1)(2f+1) - (2f+1)(2f+1) = 2f^2 + f$. Thus, the number of RABA instances that decide 0 is bounded by:

$$\frac{2f^2+f}{f+1} < \frac{2f^2+2f}{f+1} = 2f.$$

The number of RABA instances that decide 1 is at least $f+1$. ■

The above theorem implies that our new paradigm will a-deliver at least one proposal from a correct replica. The theorem also implies the liveness of our BFT protocol from the client perspective: a transaction from a correct client will be eventually a-delivered at some epoch.

F SCALABILITY EVALUATION USING INDIVIDUAL FIGURES

Scalability in detail. We show individual scalability figures for different f's for BEAT-Cobalt, ACE, and PACE in Figure 15.

G CONVERTING CONVENTIONAL ABA TO RABA

We discuss an approach that converts a non-RABA protocol to RABA. In particular, we focus on CKS [20], the pseudocode of which is shown in Figure 16. We use consistent message names for the presentation with other protocols presented in this paper, while the original CKS paper uses different notations. We use the term $ts1$ to represent the $(n, f+1, f)$ threshold signature scheme, where the first threshold denotes the combination threshold and the second threshold is the number of faulty replicas. (The threshold signature may be written as $(n, f+1)$ threshold signature for simplicity.)

We use $ts1.share$ to represent a threshold signature share generated by $ts1$ scheme and $ts1.sig$ to represent a threshold signature combined from $f+1$ threshold signature shares. We use $ts2$ to represent the $(n, n-f, f)$ threshold signature scheme, where the first threshold denotes the combination threshold the second one denotes the number of faulty replicas. We use $ts2.share$ to represent the threshold signature share and $ts2.sig$ to present the threshold signature combined from $n-f$ shares. For each message, the threshold signature share is generated for a value v, the round number r, and the ABA instance ID sid.

The CKS protocol consists of three steps in round 0 and two steps in each round starting from round 1. In round 0, every replica broadcasts a $\text{bval}_r(est_0, ts1.share)$ message, where est_0 is the proposed value and share_{f+1} is a threshold signature share. Upon receiving $2f+1$ valid $\text{bval}_r()$ messages, a replica uses the majority value v, combines the shares to obtain the signature. The replica then sends a $\text{aux}_r(v, ts1.sig, ts2.share)$ message to all replicas, where $ts2.share$ is the threshold signature of a $(n, n-f, f)$ threshold signature scheme. The sig is the signature combined from the shares in $\text{bval}_r()$ messages. In round 0, the signature is a $ts1$ signature. In round greater than 0, sig is a $ts2$ signature. If a replica receives $n-f$ $\text{aux}_r()$ messages that contain only one valid value, a replica broadcasts a $\text{conf}_r(v, ts2.sig, ts2.share)$ messages where $ts2.sig$ is a valid threshold signature combined from shares in $\text{aux}_r()$ messages, $ts2.share$ is a valid threshold signature share for v. Finally, a replica waits for $n-f$ valid $\text{conf}_r()$ messages. If there is only one value b from the messages, the replica delivers b. Otherwise, the replica starts the next round. Replicas then generate the common coin after the $\text{conf}_r()$ step. Starting from round 1, replica do not have to broadcast $\text{bval}_r()$ messages. Instead, a replica braodcasts a $\text{aux}_r()$ messages for a value b if there exists a $\text{conf}_{r-1}(b)$ message with a valid threshold signature.

Cachin et al. proposed an approach that converts CKS to VBA, the pseudocode of which is shown in Figure 17. It makes a few changes to make ABA achieve external validity and biased external validity. First, every proposed value is associated with an external proof, which needs to be verified before a replica accepts the proposed value. In other words, a $\text{bval}_r()$ message with an invalid π will be discarded. Second, the common coin in round 0 is set to 1.

We present a construction that converts CKS to a RABA protocol, as shown in Figure 18. We make several changes to make CKS achieve RABA properties. First, the threshold signature scheme we use for the $\text{bval}_r()$ messages is a $(n, n-f, f)$ scheme. In other words, a replica collects $n-f$ $\text{bval}_r()$ messages with matching value to proceed to the next step. Second, the common coin of round 1 is set to 0. Third, upon the $repropose(sid, 1)$ event triggered, a replica broadcasts a $\text{bval}_0(1, ts2.share)$ message.

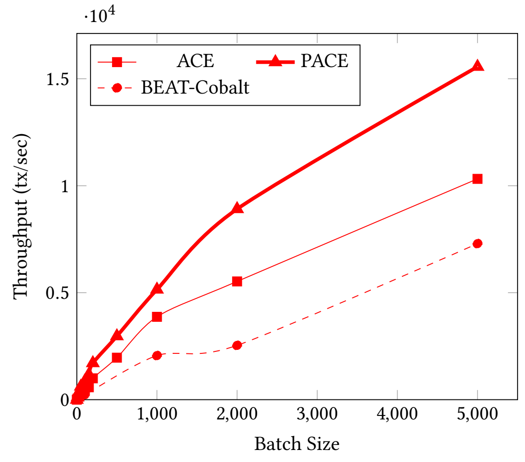
G.1 Proof of Correctness for CKS RABA

We show that our construction in Figure 18 satisfies the RABA security definitions.

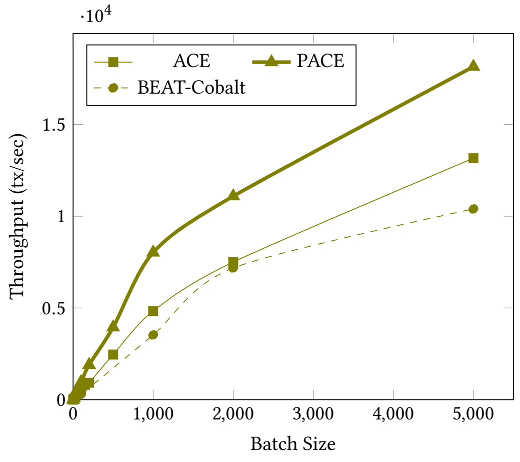
Theorem G.1. ***(Validity)*** *If all correct replicas propose v and never repropose $\bar{v}$, then any correct replica that terminates decides v.*

Proof. Since a correct replica does not repropose a different value v', all replicas are able to receive $n-f$ $\text{bval}_r(v)$ and obtain a valid threshold signature. Each replica only sends $\text{aux}_r(1)$ with a valid signature. Similarly, every replica only sends a $\text{conf}_r(v)$ message and is able to decide v. ■

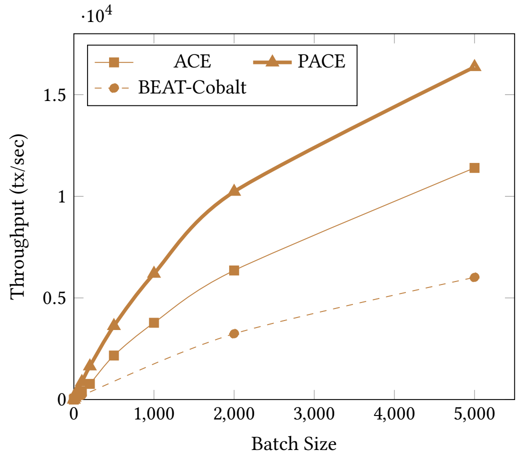
Theorem G.2. ***(Agreement)*** *If a correct replica decides v, then any correct replica that terminates decides v.*



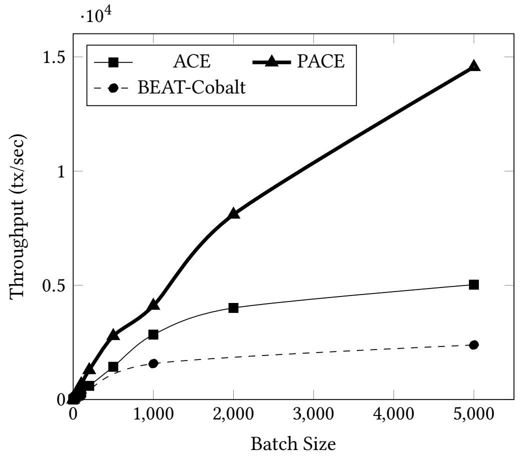
(a) Throughput of ACE, PACE, and BEAT-Cobalt when $f = 5$.



(b) Throughput of ACE, PACE, and BEAT-Cobalt when $f = 15$.



(c) Throughput of ACE, PACE, and BEAT-Cobalt when $f = 20$.



(d) Throughput of ACE, PACE, and BEAT-Cobalt when $f = 30$.

Figure 15: Detailed scalability results as f increases.

Proof. We consider that a correct replica p_i decides v in round r. Replica p_i receives $n - f$ $\text{conf}_r(v)$ messages. We show the case where another correct replica p_j decides v' 1) in round r; 2) in round $r' > r$.

First, if p_j decides v' in round r, it receives $n - f$ $\text{conf}_r(v')$. Among $n - f$ replicas that send $\text{conf}_r(v)$ and $n - f$ replicas that send $\text{conf}_r(v')$, at least one sends both $\text{conf}_r(v)$ and $\text{conf}_r(v')$, a contradiction.

Second, if p_j decides in round r', it receives $n - f$ $\text{conf}_{r'}(v')$. In other words, at least $n-f$ replicas receive $n-f$ $\text{aux}_{r'}(v')$ messages. Atmong the messages, at least one includes a $\text{conf}_{r'-1}(v')$ with a valid threshold signature. In other words, at least $n - f$ replicas sent $\text{conf}_{r'-1}(v')$ in round $r' - 1$. Recursively, in round r, at least $n-f$ replicas sent $\text{conf}_r(v')$ in round r. Therefore, a correct replica sends both $\text{conf}_r(v)$ and $\text{conf}_r(v')$, a contradiction. ■

Theorem G.3. ***(Biased validity)*** *If $f + 1$ correct replicas propose 1, then any correct replica that terminates decides 1.*

Proof. If $f+1$ correct replicas propose 1, none of correct replicas is able to collect $n - f$ $\text{bval}_0(0)$. This is because correct replicas do not repropose 0 if they propose 1. If a replica receives $n-f$ $\text{bval}_0(0)$ messages and assuming there are f faulty replicas, at least $n - 2f$ correct replicas have sent $\text{bval}_0(0)$. Since $n \geq 3f+1$, $n-2f \geq f+1$. This is a violation that at leat $f+1$ replicas propose 1 since a correct replica will not repropose 0.

Also, based on the external condition, each replica may repropose 1. In other words, all correct replicas will eventually receive $n - f$ $\text{bval}_0(1)$ messages. In this case, all replicas will receive $n-f$ $\text{bval}_0(1)$ and proceed to the next step.

Now assume that a correct replica p_j decides 0 in round 0. In this case, p_j receives $n - f$ $\text{conf}_0(0)$ messages, which is impossible since it requires $n - f$ $\text{bval}_0(0)$. We now only need to show the correctness by assuming that p_j decides in round $r > 0$. In this case, p_j receives $n - f$ $\text{conf}_{r-1}(0)$. In other words, at least $n - f$ replicas broadcast $\text{aux}_{r-1}(1)$ and obtain a valid threshold signature from round $r - 2$. Recursively, in round 0, at least $n - f$ replicas send

```
upon event $propose(sid, v_i)$
  $r \leftarrow 0$
  $est_0 \leftarrow v_i$
  start round 0
round $r$
  if $r = 0$
    broadcast $\text{bval}_r(est_0, ts1.share)$
    upon receiving $2f+1$ $\text{bval}_r()$ with $vals$
      $v \leftarrow$ majority($vals$), $sig \leftarrow ts1.combine(shares)$
  else
    if exists $\text{conf}_{r-1}(b)$
      $v \leftarrow b$, $sig \leftarrow$ signature from $\text{conf}_{r-1}(b)$ message
    else $v \leftarrow \text{coin}_{r-1}$
  broadcast $\text{aux}_r(v, sig, ts2.share)$
  upon receiving $n-f$ $\text{aux}_r()$ with $vals$
    if $vals = \{b\}$, $v \leftarrow b$, $sig \leftarrow ts2.combine(shares)$
    else $v \leftarrow \perp$, $sig \leftarrow \text{aux}_r()$ messages
    broadcast $\text{conf}_r(v, ts2.sig, ts2.share)$
  upon receiving $vals$ of $\text{conf}_r()$ from $n-f$ replicas
    $\text{coin}_r \leftarrow Coin()$
    if $vals = \{b\}$, $decide(sid, b)$
    else $r \leftarrow r+1$
```

Figure 16: CKS ABA [20]. The code for replica p_i.

```
upon event $propose(sid, v_i, \pi)$
  $r \leftarrow 0$
  $\text{coin}_0 \leftarrow 1$
  $est_0 \leftarrow v_i$
  start round 0
round $r$
  if $r = 0$
    broadcast $\text{bval}_r(est_0, ts1.share, \pi)$
    upon receiving $\text{bval}_r()$ with invalid $\pi$
      dicard the message
    upon receiving $2f+1$ $\text{bval}_r()$ with $vals$
      $v \leftarrow$ majority($vals$), $sig \leftarrow ts1.combine(shares)$
  else
    if exists $\text{conf}_{r-1}(b)$
      $v \leftarrow b$, $sig \leftarrow$ signature from $\text{conf}_{r-1}(b)$ message
    else $v \leftarrow \text{coin}_{r-1}$
  broadcast $\text{aux}_r(v, sig, ts2.share)$
  upon receiving $n-f$ $\text{aux}_r()$ with $vals$
    if $vals = \{b\}$, $v \leftarrow b$, $sig \leftarrow ts2.combine(shares)$
    else $v \leftarrow \perp$, $sig \leftarrow \text{aux}_r()$ messages
    broadcast $\text{conf}_r(v, sig, ts2.share)$
  upon receiving $vals$ of $\text{conf}_r()$ from $n-f$ replicas
    $\text{coin}_r \leftarrow Coin()$
    if $vals = \{b\}$, $decide(sid, b)$
    else $r \leftarrow r+1$
```

Figure 17: The VBA construction based on CKS ABA protocol [19].

$\text{conf}_0(0)$. It is straightforward to see that in the first step of round 0, at least $n-f$ replicas sent $\text{bval}_0(0)$. As shown previously, this is also impossible. ■

Theorem G.4. ***(Biased termination)*** *Let Q be the set of correct replicas. Let Q_1 be the set of correct replicas that propose 1 and never repropose 0. Let Q_2 be correct replicas that propose 0 and later repropose 1. If $Q_2 \neq \emptyset$ and $Q = Q_1 \cup Q_2$, then every correct replica eventually terminate.*

```
upon event $propose(sid, v_i)$
  $r \leftarrow 0$
  $\text{coin}_0 \leftarrow 1$
  $est_0 \leftarrow v_i$
  start round 0
upon event $repropose(sid, 1)$
  broadcast $\text{bval}_0(1, ts2.share)$
round $r$
  if $r = 0$
    broadcast $\text{bval}_r(est_0, ts2.share)$
    upon receiving $n-f$ $\text{bval}_r(b)$
      $v \leftarrow b$, $sig \leftarrow ts2.combine(shares)$
  else
    if exists $\text{conf}_{r-1}(b)$
      $v \leftarrow b$, $sig \leftarrow$ signature from $\text{conf}_{r-1}(b)$ message
    else $v \leftarrow$ coin
  broadcast $\text{aux}_r(v, ts2.sig, ts2.share)$
  upon receiving $n-f$ $\text{aux}_r()$ with $vals$
    if $vals = \{b\}$, $v \leftarrow b$, $sig \leftarrow ts2.combine(shares)$
    else $v \leftarrow \perp$, $sig \leftarrow \text{aux}_r()$ messages
    broadcast $\text{conf}_r(v, ts2.sig, ts2.share)$
  upon receiving $vals$ of $\text{conf}_r()$ from $n-f$ replicas
    $\text{coin}_r \leftarrow Coin()$
    if $vals = \{b\}$, $decide(sid, b)$
    else $r \leftarrow r+1$
```

Figure 18: The RABA construction based on CKS.

Proof. We distinguish three cases: 1) All replicas propose 1; 2) All replicas propose 0; 3) At least one correct replica proposes 1. We show correctness for the three cases.

1) Since correct replicas do not repropose 0, it is straightforward to see that all replicas will decide 1.

2) All correct replicas may repropose 1. In other words, every replica may receive $n-f$ $\text{bval}_0(0)$ and $n-f$ $\text{bval}_0(0)$. Each replica, however, only sends $\text{aux}_r()$ message once with one value. In the next step, a replica makes a decision regardless of the values received from $\text{aux}_r()$ messages. Similarly, a replica can proceed after it receives $n-f$ $\text{conf}_r()$ messages regardless of the values. Therefore, the protocol can proceed to the next step. There are three sub-cases: A) None of the correct replicas collects $n-f$ $\text{conf}_r()$ messages with a matching value; B) At least one but fewer than $f+1$ correct replicas collect $n-f$ $\text{conf}_r()$ messages with a matching value; C) At least $f+1$ correct replicas collect $n-f$ $\text{conf}_r()$ messages. In case A, all correct replicas use the common coin value as v and each replica sends $\text{aux}_r(v)$. It is impossible that another corect replica sends $\text{aux}_r(v')$ with a valid threshold signature since it requires $n-f$ valid $\text{conf}_{r-1}(v')$ messages. Therefore, all replicas will receive $n-f$ $\text{aux}_r(v)$, send $\text{conf}_r(v)$, and decide v. In case B), some replicas may send $\text{aux}_r(v)$ while other replicas use the common coin value. The possibility that the common coin value is the same as the b value (if any) is 1/2. In other words, all replicas will decide with 1/2 probability. Otherwise, replicas may proceed to the next round. It is then straightforward to see that replicas will

terminate the protocol. In case C), all correct replicas will receive at least one valid $\mathsf{conf}_r(v)$ in round $r+1$. This is because each replica collects $n-f$ $\mathsf{bval}_{r+1}(v)$ messages. Among the replicas that have sent $\mathsf{bval}_{r+1}(v)$ messages, at least $n-2f$ are correct. We also know that at least $f+1$ replicas sent $\mathsf{conf}_r(v)$. There are in total $n-f+1$ correct replicas that sent $\mathsf{bval}_{r+1}(v)$ and $\mathsf{conf}_r(v)$. Therefore, if at least one corect replica fails to receive a valid $\mathsf{conf}_r(v)$ message, at least one correct replica collects $n-f$ $\mathsf{conf}_r(v)$ messages but does not send a $\mathsf{bval}_{r+1}(v)$ with a valid signature, a contradiction.

3) If at least one replica proposes 1, all replicas may receive $n-f$ $\mathsf{bval}_0(0)$ and/or $n-f$ $\mathsf{bval}_0(0)$. It is also possible that a replica cannot collect $n-f$ $\mathsf{bval}_0(0)$ or $n-f$ $\mathsf{bval}_0(1)$ based on the proposed values. In this case, the external condition guarantees that all correct replicas will eventually broadcast $\mathsf{bval}_0(1)$. In other words, replica will eventually proceed to the next step. Similar to case 2), all correct replicas will eventually terminate. ■

G.2 Converting Cobalt ABA to RABA

Similarly, we can convert Cobalt ABA [48] to RABA. We make three modifications. First, upon proposing 1, each replica starts round 0 and broadcasts a $\mathsf{bval}_0(1)$. Each replica also immediately adds 1 to bin_values_0, and broadcasts an $\mathsf{aux}_0(1)$ message. Second, upon reproposing 1, regardless of which round a replica is in, each replica broadcasts $\mathsf{bval}_0(1)$. Furthermore, the replica adds 1 to bin_values_0 if it has not done so already and broadcasts an $\mathsf{aux}_0(1)$ if it has not broadcast any $\mathsf{aux}_r()$ message in round 0. Finally, the common coin for round 0 is set to 1. The first trick and the third trick ensure that no correct replica will ever use 0 as est_1 if at least $f+1$ correct replicas propose 1. The second trick ensures that biased termination can be achieved.

H ADDITIONAL RELATED WORK

Much related work is discussed in the course of the paper. The section discusses additional related work.

Consensus and atomic broadcast. The BKR paradigm implies ABA and atomic broadcast are equivalent in asynchronous environments. Chandra and Toueg [25] demonstrate that multi-valued consensus is equivalent to asynchronous atomic broadcast. They also mention, informally, multi-valued Byzantine agreement and atomic broadcast are equivalent in asynchronous settings, a claim formally proven by Cachin, Kursawe, Petzold, and Shoup [19]. Correia, Neves, and Verissimo (CNV) show that multi-valued consensus (without using signatures) is also equivalent to atomic broadcast [28].

ABA and consensus as a building block. ABA can be used to build many core distributed computing abstractions, such as vector consensus [12], multi-valued Byzantine agreement [19], atomic broadcast (e.g., [12, 19, 34, 44, 49]), anonymous vector consensus [17], and many others. It crash-failure counterpart, consensus, is even more widely used in practice, such as terminating reliable broadcast, dynamic membership, and non-blocking atomic commit (see [18] for a summary of consensus-based systems and references therein).

Local-coin ABA. ABA protocols may rely on local coins. These protocols are information-theoretically secure but terminate in an expected exponential number of rounds [15]. RITAS [50], for instance, uses local-coin ABA to build asynchronous atomic broadcast using the protocol of Correia, Neves, and Veríssimo [29] that does not fall into the category of the BKR paradigm or the CKPS paradigm.

Asynchronous vs. partially synchronous BFT. Partially synchronous BFT systems never violate safety but achieve liveness when the network becomes synchronous [36]. In contrast, asynchronous BFT does not rely on any timing assumptions. It is shown that even for partially synchronous BFT protocols focusing on robustness [5, 27], their performance may drop 78%-99% [7] during failures or attacks. Moreover, partially synchronous protocols may achieve zero throughput with an adversarial network scheduler [49]. Asynchronous BFT protocols are intrinsically robust against performance and denial-of-service (DoS) attacks.

BFT with asynchronous fallback. Bolt-Dumbo [47] and Ditto [38] are two independent and concurrent works developing the idea of Kursawe and Shoup [42]. They aim to provide good performance in the normal case and provide progress during asynchrony.

Asynchronous BFT with quantum safety. DAG-Rider [40] is a recent asynchronous BFT protocol achieving quantum safety but not quantum liveness. DAG-Rider has $O(1)$ running time and achieves $O(n^2 l + \lambda n^3 \log n)$ communication complexity. A more recent work of Das, Xiang, and Lin improves DAG-Rider with a communication of $O(n^2 l + \lambda n^3)$.

Tusk [32] implements a fast asynchronous BFT protocol, but it requires additional workers to help the reliable broadcast phase and thus is outside of the scope of the conventional BFT model we consider in this paper. The technique, however, seem to work for all asynchronous BFT protocols known.

Sub-optimal resilience. Assuming sub-optimal resilience, MiB [45] implements asynchronous BFT protocols with lower latency and higher throughput based on the BEAT library.

Communication-efficient RBC protocols. Some recent constructions have improved the RBC protocols asymptotically or concretely [3, 33]. These protocols may benefit all practical BFT protocols implemented in the bandwidth usage.