PACE: Fully Parallelizable BFT from Reproposable Byzantine Agreement

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

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

ABSTRACT
The classic asynchronous Byzantine fault tolerance (BFT) framework of Ben-Or, Kemler, and Rabin (BKR) and its descendants rely on reliable broadcast (RBC) and asynchronous binary agreement (ABA). However, BKR does not allow all ABA instances to run in parallel, a well-known performance bottleneck. We propose PACE, a generic framework that removes the bottleneck, allowing fully parallelizable ABA instances. PACE is built on RBC and reproposable ABA (RABA). Different from the conventional ABA, RABA allows a replica to change its mind and vote twice. We show how to efficiently build RABA protocols from existing ABA protocols and a new ABA protocol that we introduce.

We implement six new BFT protocols: three in the BKR framework, and three in the PACE framework. Via a deployment using 91 replicas on Amazon EC2 across five continents, we show that all PACE instantiations, in both failure-free and failure scenarios, significantly outperform their BKR counterparts, and prior BFT protocols such as BEAT and Dumbo, in terms of latency, throughput, latency vs. throughput, and scalability.

KEYWORDS
asynchronous BFT, binary agreement, blockchain, fault tolerance, RABA

1 INTRODUCTION
Byzantine fault tolerance (BFT) is widely viewed as the model for permissioned blockchains. Among BFT protocols, completely asynchronous BFT protocols [4, 12, 16, 34, 39, 43] have received renewed attention because of its intrinsic robustness. Indeed, having long been viewed as a "theoretical" approach, several recent asynchronous BFT systems—such as HoneyBadgerBFT [38], BEAT [29], Dumbo [31], and EPIC [35]—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, Kusaw, Petzold, and Shoup) paradigm [16] including SINTRA [18] 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 [32]); it has an $O(\log n)$ running time. The CKPS framework is only computationally secure and relies on less well-established cryptographic pairing assumptions; 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 $O(n^3)$ pairing-based threshold signatures. Dumbo, however, has 14 steps even in the best-case scenario (where the ABA instance terminates in one round). The protocols following the BKR paradigm terminate in $O(\log n)$ rounds but only 6 steps in the best-case scenario. Our experiment shows that BEAT, for instance, outperforms Dumbo when $n \leq 46$, but is less efficient than Dumbo for larger $n$’s.

This work introduces a new BFT framework called PACE that removes a well-known performance bottleneck in the BKR paradigm, and therefore improves upon the BKR framework and applications along this line of research (e.g., all BKR descendants, asynchronous distributed key generation [27, 33], interactive consistency [11]). We show that our PACE instantiations significantly outperform their BKR counterparts, and existing BFT protocols such as BEAT and Dumbo, in both failure and failure-free scenarios.

The BKR bottleneck. The BKR 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 $p_i$, 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. The BKR paradigm requires that if a replica has not received any 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. The BKR paradigm, however, breaks the parallelism of the “RBC+ABA” structure. 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.

This is a well-known bottleneck for the BKR paradigm repeatedly pointed out by prior works [10, 11] and recently emphasized by HoneyBadgerBFT [38, Section 4.4]. In addition, both Dumbo and BEAT experimentally validate this bottleneck via performance breakdown for the building blocks. The two-subphase pattern not only adds significant latency but causes transaction congestion.

A well-known "naive attempt" that does not work (and was recently recalled in [38]) would be that each replica waits for the first $n - f$ RBCs completed, and then propose 1 for the ABA instances corresponding to those completed and propose 0 for the others. However, RBC instances completed for correct replicas may be different. As ABA ensures the decided value is 1 only if all correct
Table 1: Comparison among asynchronous BFT protocols. "Coin type" includes regular, low threshold \((f+1)\) common coins and high threshold \((n-f)\) common coins. "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 mark "yes" for them for IT security. Quantum safety is defined in DAG-Rider [32]; quantum security = quantum safety + quantum liveness. *New ABA/RABA protocols in this work.

<table>
<thead>
<tr>
<th>protocols</th>
<th>framework</th>
<th>ABA/ RABA</th>
<th>ABA/ RABA used</th>
<th>coin type</th>
<th>IT secure (using IT common coin)?</th>
<th>quantum safety?</th>
<th>cryptographic assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dumbo [31]</td>
<td>CKPS</td>
<td>ABA</td>
<td>Cobalt [37]</td>
<td>regular</td>
<td>√</td>
<td>√</td>
<td>pairing</td>
</tr>
<tr>
<td>HoneyBadgerBFT [38]</td>
<td>BKR</td>
<td>ABA</td>
<td>MMR [41]</td>
<td>regular</td>
<td>√</td>
<td>√</td>
<td>pairing</td>
</tr>
<tr>
<td>BEAT-MMR [29]</td>
<td>BKR</td>
<td>ABA</td>
<td>MMR [41]</td>
<td>regular</td>
<td>√</td>
<td>√</td>
<td>pairing</td>
</tr>
<tr>
<td>BEAT-CrainH (this paper)</td>
<td>BKR</td>
<td>ABA</td>
<td>CrainH [25]</td>
<td>high threshold</td>
<td>√</td>
<td>√</td>
<td>DDH</td>
</tr>
<tr>
<td>BEAT-CrainL (this paper)</td>
<td>BKR</td>
<td>ABA</td>
<td>CrainL [25, 28]</td>
<td>regular</td>
<td>√</td>
<td>√</td>
<td>DDH</td>
</tr>
<tr>
<td>BEAT-Pillar (this paper)</td>
<td>BKR</td>
<td>ABA</td>
<td>Pillar*</td>
<td>regular</td>
<td>√</td>
<td>√</td>
<td>DDH</td>
</tr>
<tr>
<td>PACE-CrainH-R (this paper)</td>
<td>PACE</td>
<td>RABA</td>
<td>CrainH-R*</td>
<td>high threshold</td>
<td>√</td>
<td>√</td>
<td>DDH</td>
</tr>
<tr>
<td>PACE-CrainL-R (this paper)</td>
<td>PACE</td>
<td>RABA</td>
<td>CrainL-R*</td>
<td>regular</td>
<td>√</td>
<td>√</td>
<td>DDH</td>
</tr>
<tr>
<td>PACE-Pisa (this paper)</td>
<td>PACE</td>
<td>RABA</td>
<td>Pisa*</td>
<td>regular</td>
<td>√</td>
<td>√</td>
<td>DDH</td>
</tr>
</tbody>
</table>

1.1 Our Contributions

RABA. We suggest reproposable ABA (RABA), a new ABA primitive allowing replicas to change votes if needed. We formally describe the properties of RABA. We view RABA as a first-class distributed computing primitive.

PACE: A new framework for BFT, parallel ABA, interactive consistency, etc. We design a new framework for asynchronous BFT that use RBC and RABA in a black box manner. The framework leads to the first fully parallelizable asynchronous BFT protocol, allowing all RABA instances to run in a strictly concurrent way. The improvement can both increase the throughput and reduce the latency, effectively removing the BKR two-subphase bottleneck. PACE can also improve interactive consistency [11], the asynchronous common subset (ACS) framework and applications using ACS (e.g., asynchronous distributed key generation [27, 33]).

ABA and RABA protocols. We provide generic strategies that transform efficient ABA protocols to RABA protocols. All our RABA protocols have a fast path for RABA protocols to terminate. In particular, we demonstrate the transformation for the classic CKS ABA [17], Cobalt ABA [37], and the two protocols from Crain’s work—CrainH (using high threshold common coins) and CrainL (using low threshold common coins) [25], and an ABA protocol that we introduce in the paper (called Pillar). For CrainL, we use the restated version with the good-case-coin-free property [28]. For Pillar, it is a new ABA protocol that has the same steps as CrainH but uses more efficient low threshold common coins. The resulting RABA protocols are called CKS-R, Cobalt-R, CrainH-R, CrainL-R, and Pisa, respectively. These ABA and RABA protocols provide interesting efficiency trade-offs.

Practical contributions. As shown in Table 1, we implement three ABA protocols (CrainH, CrainL, Pillar) and three RABA protocols (CrainH-R, CrainL-R, Pillar). Correspondingly, we implement six new BFT protocols: three in the BKR framework (BEAT-CrainH, BEAT-CrainL, BEAT-Pillar), and three in the PACE framework (PACE-CrainH-R, PACE-CrainL-R, PACE-Pisa). We extensively evaluate the performance for these six new protocols and compare them.

replicas unanimously propose 1, the transactions delivered may be empty.

PACE explained. In our work, we propose a new framework, PACE, that makes the above "naive attempt" work and removes the two-subphase bottleneck. We achieve this by replacing ABA with reproposable ABA (RABA), a new binary agreement primitive.

Our key observation is that in the naive attempt, if \(n \geq f\) correct replicas deliver some proposals in \(n - f\) RBC instances, there must exist at least \(f + 1\) correct replicas that will propose 1 for at least \(f + 1\) ABA instances. For these specific ABA instances, we want to guarantee that all these ABA instances will indeed decide 1. The property is called biased validity. Meanwhile, we need to make ABA protocols syntactically reproposable, in the sense that a replica who previously voted 0 (maybe immaturely) may change its mind to vote 1 (if the replica later delivers the corresponding RBC instance). The property (called biased termination) ensures that all RABA instances can terminate.

We offer generic strategies that can convert various efficient ABA protocols to efficient RABA protocols. Interestingly, the "biased" features of RABA provide a "fast" path for RABA to decide, further reducing latency and improving throughput.
with existing protocols, including BEAT-MMR, BEAT-Cobalt, and Dumbo. Via an EC2 deployment using 91 replicas from five continents, we demonstrate that all PACE instantiations, in both failure-free and failure scenarios, dramatically outperform their BKR counterparts, and prior BFT protocols such as BEAT and Dumbo, in terms of all metrics (latency, throughput, latency vs. throughput, and scalability). Let us highlight some evaluation results:

- All PACE instantiations significantly outperform any of the BRK instantiations. Even the slowest PACE instantiation is far more efficient than all the other protocols.
- PACE-Pisa and PACE-CrainsL-R are consistently more efficient than PACE-CrainsH-R. PACE-Pisa slightly outperforms PACE-CrainsL-R, except that when the system is of medium size, the two protocols offer interesting trade-offs. So below we use PACE-Pisa for some concrete comparison for simplicity.
- For \( f = 1 \), PACE-Pisa achieves 1.77x the throughput of BEAT-Cobalt and 5.1x the throughput of Dumbo. For \( f = 30 \), PACE-Pisa achieves 3.6x the throughput of BEAT-Cobalt and 1.66x the throughput of Dumbo.
- PACE-Pisa offers an impressive latency metric (when there is no contention) in both LAN and WAN environments. In WANs, the latency of PACE-Pisa is between 1/5 and 1/2 of that of the BKR protocols and between 1/3 and 1/2 of that of Dumbo. In LANs, Dumbo has 9x the latency of PACE-Pisa for \( f = 1 \).
- All PACE protocols are highly robust against various crash and Byzantine failure scenarios. Each PACE protocol outpaces its BKR counterpart in all failure scenarios. Remarkably, among all failure scenarios, the slowest PACE instantiation remains more efficient than the fastest BEAT protocol in its failure-free scenario.

All protocols introduced in the paper rely on well-studied, standard, and pairing-free cryptographic assumptions (CDH or DDH for elliptic curves), achieving standard 128-bit security.

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 \( f \leq \left\lfloor \frac{n-1}{3} \right\rfloor \), 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 \( \left\lfloor \frac{n + f + 1}{2} \right\rfloor \) 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.

This paper studies BFT protocols. The protocols that we introduce in the paper inherit all features that BKR and its descendants have: quantum safety and information-theoretic (IT) security (assuming IT common coin). All these protocols rely on the standard and well-studied Computational Diffie-Hellman (CDH) or Decisional Diffie-Hellman (DDH) assumptions. In contrast, protocols derived from the CKPS paradigm [16] (e.g., Dumbo) achieve computational security only and use (stronger) pairing assumptions. Our implementations tolerate static corruption, where the adversary needs to choose the set of corrupted replicas before the execution of the protocol. But our protocols can easily achieve adaptive security, if using adaptively secure common coin protocols [9, 35].

We use BFT and (Byzantine) atomic broadcast interchangeably. Syntactically, in BFT, a replica \( a \)-delivers (atomically deliver) transaction \( tx \), then every correct replica \( a \)-delivers \( tx \).

- **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 \( tx \).

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 \( o \), then any correct replica that terminates decides \( o \).
- **Agreement**: If a correct replica decides \( o \), then any correct replica that terminates decides \( o \).
- **Termination**: Every correct replica eventually decides some value.
- **Integrity**: No correct replica decides twice.

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 [19] that achieves \( O(t n |m| + \lambda n^2 \log n) \) communication, where \( \lambda \) is a security parameter. In both BKR and PACE frameworks, the communication is dominated by \( n \) RBC protocols and is \( O(L n^2 + \lambda n^3 \log n) \) (\( L \) is the input size).

Throughout the paper, we also use best-effort broadcast, or simply broadcast, where a sender multicasts a message to all replicas.

Common coins and thresholds. The common coin protocol outputs a binary value at each correct replica [42]. These protocols can be divided into protocols with the regular (low) thresholds (i.e., \( f + 1 \) and protocols with the high thresholds (i.e., \( 2f + 1 \)). Protocols from both regular coins [37, 40] and high threshold coins [17, 24, 25] 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 [6, 27]. Second, even assuming
ABA protocols proceed in rounds, where each round has a fixed number of steps. Asynchronous BFT (MMR ABA) is the first constant-round ABA that relies on authenticated channels only. In each round, MMR ABA has 2 or 3 steps. The ABA can work for both weak coins and perfect coins. Cobalt ABA [37] provides an alternative solution, having 3 or 4 steps in each round. Recent BFT implementations including EPIC [35] and Dumbo [31] use Cobalt ABA.

In this section, we first review existing, practical ABA protocols relying on authenticated channels, focusing on MMR ABA [40] and its descendants. We then present a new ABA protocol assuming authenticated channels. We also briefly review CKS ABA [17] and Crain-Sig20 ABA [24] that use threshold signatures. All of these ABA protocols described in the section are candidate protocols for our PACE framework and can in fact be efficiently converted to RABA protocols.

3 Candidate ABA Protocols for PACE: Existing ABA Protocols

In this section, we first review existing, practical ABA protocols relying on authenticated channels, focusing on MMR ABA [40] and its descendants. We then present a new ABA protocol assuming authenticated channels. We also briefly review CKS ABA [17] and Crain-Sig20 ABA [24] that use threshold signatures. All of these ABA protocols described in the section are candidate protocols for our PACE framework and can in fact be efficiently converted to RABA protocols.

3.1 Existing ABA Protocols

ABA protocols proceed in rounds, where each round has a fixed number of steps.

The ABA protocol by Mostefaoui, Moumen, and Raynal (MMR) [40] (MMR ABA) is the first constant-round ABA that relies on authenticated channels only. In each round, MMR ABA has 2 or 3 steps (without counting the step for common coin). Asynchronous BFT protocols, such as HoneyBadgerBFT [38] and BEAT [29], in their proceeding versions, utilize MMR ABA.

When describing MMR ABA and other protocols, we follow prior notations like, e.g., bvalr(a), where bvalr() is a message pattern with a round r, and a is the message transmitted. In MMR ABA, each round r has two phases: a dispersal phase and an agreement phase. In the dispersal phase, every replica broadcasts its value est− via a bvalr(est−) message. Upon receiving f + 1 bvalr(v), a replica also broadcasts a bvalr(v) if it has not done so. Upon receiving n − f auxr(v), a replica adds v to a local set bin_values. Each replica broadcasts an auxr(v) message for the first value added to bin_values and then enters the agreement phase. Upon receiving n − f auxr(v) messages, replicas generate a common coin sv by querying the common coin protocol coinv. If a replica receives n − f auxr(b) and b = sr, the replica decides b and sets estr+1 to b. Otherwise, the replica sets estr+1 to sr. Each replica then enters round r + 1. After each replica decides, there are two standard approaches to terminating the protocol. One approach is that each replica continues to participate in the protocol until reaching round r′, where sr′ equals the value the replica decides. The other approach is that each replica broadcasts a message to all replicas and each replica terminates the protocol upon receiving n − f such messages. Throughout the paper, we neglect the details of terminating the protocol. The pseudocode of MMR is presented in Appendix A.

MMR ABA, however, has a liveness issue [1, 41, 45]. 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 protocol cannot be guaranteed by any cryptographic common coin protocols. A malicious network scheduler can force correct replicas to always enter the next round with inconsistent values (i.e., some replicas set estr+1 to the common coin value sr after receiving both aux0(0) and aux1(1) and some replicas set estr+1 to sr after receiving n − f auxr(s)).

The journal version of MMR [41, 2nd algorithm] fixed the issue but has 9 to 13 steps for each round. The idea is to repeat the dispersal-agreement pattern four times such that the value each replica decides does not have to be compared with the common coin.

The ABA can work for both weak coins and perfect coins. Cobalt ABA [37] provides an alternative solution, having 3 or 4 steps in each round. Recent BFT implementations including EPIC [35] and Dumbo [31] use Cobalt ABA.

In the same work, Crain [25, 1st algorithm] (denoted as CrainL) optimized the MMR journal protocol by having 5 to 7 steps in each round. The core idea is that the dispersal-agreement pattern only has to be repeated twice instead of four times. As in MMR journal, CrainL work for weak coins and perfect coins. A recent work pointed out that CrainL can have a "good-case-coin-free" property which leads to a fast path to decide values [28]. We use the restated version of CrainL from [28].

In the same work, Crain [25, 2nd algorithm] (denoted as CrainH) also fixed the issue of MMR ABA assuming authenticated channels and has 2 or 3 steps in each round. However, CrainH uses a high threshold perfect common coin protocol which, as we have argued in Sec. 2, is less efficient, and may rely on a stronger assumption.

For all these ABA protocols, if using perfect coins, the expected number of rounds for replicas to reach a state such that a decision can be made is 2. Then replicas continue to execute the protocol until they decide. For MMR and its descendants (Cobalt, CrainH, Pillar), as replicas have to compare its value with the common coin before they decide, another 2 rounds are expected and the total expected number of rounds to terminate is 4. For CKS, the MMR journal protocol, and CrainL, as replicas do not have to compare their values with the coin, replicas are expected to decide in another one round and the expected number of rounds for these protocols to terminate is 3.

Table 2 summarizes some representative ABA protocols terminating in an expected constant number of rounds. We divide these protocols into two categories: ABA assuming authenticated channels only and ABA requiring the transferability of (threshold) signatures (including CKS ABA and Crain-Sig20 [24]). Jumping ahead, in Sec. 4 and Appendix G, we show how to convert Pillar, CKS, CrainH, CrainL, and Cobalt to RABA.

3.2 Pillar
Table 2: Comparison of ABA protocols using common coins.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>signature?</th>
<th>steps/round</th>
<th>rounds</th>
<th>coin type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXMR [40] (Insecure)</td>
<td>yes</td>
<td>2 or 3</td>
<td>high threshold</td>
<td></td>
</tr>
<tr>
<td>MMR [17]</td>
<td>no</td>
<td>0 to 13</td>
<td>regular (weak) coin</td>
<td></td>
</tr>
<tr>
<td>Cnolk [37]</td>
<td>yes</td>
<td>0 or 4</td>
<td>regular</td>
<td></td>
</tr>
<tr>
<td>Crain-Sig20 [24, 28]</td>
<td>yes</td>
<td>1 or 2</td>
<td>high threshold</td>
<td></td>
</tr>
<tr>
<td>CrainL [25, 1st alg]</td>
<td>no</td>
<td>5 to 7</td>
<td>regular (weak) coin</td>
<td></td>
</tr>
<tr>
<td>CrainH [25, 2nd alg]</td>
<td>no</td>
<td>2 or 3</td>
<td>high threshold</td>
<td></td>
</tr>
</tbody>
</table>

Piller (this work) | yes | 2 or 3 | regular |

![Table 2: Comparison of ABA protocols using common coins.](image)

Figure 2: The Piller protocol. The code for replica \( p_i \). Broadcast in the code best-effort broadcast.

Piller is designed to address the liveness issue of MMR without introducing more steps or using less efficient high threshold common coins. As in MMR ABA and its descendants, each round in Piller consists of a dispersal phase and an agreement phase. The core idea of Piller is that instead of having each replica include one value in each message, each replica includes messages that carry auxiliary values. The first value is the "main" value for which each replica votes. The second value carries auxiliary value. The auxiliary value guarantees that correct replicas will only vote for the same value in the same round greater than 0.

Table 2: Comparison of ABA protocols using common coins.

Steps/round denotes number of steps per round (common coin steps not counted). Rounds denote the expected number of steps/round and rounds. The total number of steps is a product of steps/round and rounds. *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.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>signature?</th>
<th>steps/round</th>
<th>rounds</th>
<th>coin type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXMR [40] (Insecure)</td>
<td>yes</td>
<td>2 or 3</td>
<td>high threshold</td>
<td></td>
</tr>
<tr>
<td>MMR [17]</td>
<td>no</td>
<td>0 to 13</td>
<td>regular (weak) coin</td>
<td></td>
</tr>
<tr>
<td>Cnolk [37]</td>
<td>yes</td>
<td>0 or 4</td>
<td>regular</td>
<td></td>
</tr>
<tr>
<td>Crain-Sig20 [24, 28]</td>
<td>yes</td>
<td>1 or 2</td>
<td>high threshold</td>
<td></td>
</tr>
<tr>
<td>CrainL [25, 1st alg]</td>
<td>no</td>
<td>5 to 7</td>
<td>regular (weak) coin</td>
<td></td>
</tr>
<tr>
<td>CrainH [25, 2nd alg]</td>
<td>no</td>
<td>2 or 3</td>
<td>high threshold</td>
<td></td>
</tr>
</tbody>
</table>

Piller (this work) | yes | 2 or 3 | regular |

![Table 2: Comparison of ABA protocols using common coins.](image)

Figure 2: The Piller protocol. The code for replica \( p_i \). Broadcast in the code best-effort broadcast.

Piller is designed to address the liveness issue of MMR without introducing more steps or using less efficient high threshold common coins. As in MMR ABA and its descendants, each round in Piller consists of a dispersal phase and an agreement phase. The core idea of Piller is that instead of having each replica include one value in each message, each replica includes messages that carry auxiliary values. The first value is the "main" value for which each replica votes. The second value carries auxiliary value. The auxiliary value guarantees that correct replicas will only vote for the same value in the same round greater than 0.

by each correct replica from being manipulated by an adversary. We introduce some notation to better present the protocol.

- For \( b \in \{0, 1\} \), \( \Delta = 7b = 1 - b \).
- \( \ast \) in a message may represent \( b, \bot \), or \( \bot \), where \( b \in \{0, 1\} \) and \( \bot \notin \{0, 1\} \). For instance, \( aux_\ast(b, b) \) may represent \( aux_\ast(b, b) \), \( aux_\ast(b, \bot) \), or \( aux_\ast(\bot, b) \), \( bval_\ast(b, b) \) may represent \( bval_\ast(b, b) \), \( bval_\ast(b, \bot) \), or \( bval_\ast(\bot, b) \). We may simply omit \( \ast \) when there is no ambiguity; for example, we may write \( aux_\ast() \) to denote \( aux_\ast(\ast, \ast) \).
- Let \( avail_b \) is a vector (multiset) consisting of 0, 1, and \( \bot \). \( V_i(\text{vals}) = b \) if the only value in \( \text{vals} \) is \( b \) for \( b \in \{0, 1, \bot\} \).
- \( V_i(\text{vals}, b) = t \) if \( \text{vals} \) includes \( b \) only or both \( b \) and \( \bot \), where \( b \in \{0, 1\} \), and the number of \( b \)'s in \( \text{vals} \) is \( t \).
- \( \text{majority}(\text{vals}) = b \) for \( b \in \{0, 1\} \), if \( b \) is a simple majority in \( \text{vals} \), i.e., the number of \( b \)'s is no less than \( \lfloor (|\text{vals}| + 1)/2 \rfloor \).
- \( \text{majority}(\text{vals}) = \bot \) otherwise.

As illustrated in Figure 2, the Piller protocol has two message types: \( bval(t) \) and \( aux_\ast() \), corresponding to the dispersal phase and the agreement phase, respectively.

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 \( maj_r \in \{0, 1, \bot\} \). In the first phase (ln 05-17), \( p_i \) first broadcasts a \( bval(est_r, maj_r) \) message. A correct replica does not change its \( maj_r \) within a round. If \( p_i \) receives more than \( f + 1 \) \( bval(\ast, 

In the second phase (ln 18-34), replica \( p_i \) only accepts an \( aux_\ast(v_1, v_2) \) message if both \( v_1 \) and \( v_2 \) are added to \( \text{vals} \). \( V_i(\text{vals}) = 1 \) if both \( v_1 \) and \( v_2 \) are added to \( \text{vals} \). \( V_i(\text{vals}) = \bot \) otherwise (ln 15-17). A correct replica only sends one \( aux_\ast() \) message in each round.

In the second phase (ln 18-34), replica \( p_i \) only accepts an \( aux_\ast(v_1, v_2) \) message if both \( v_1 \) and \( v_2 \) are added to \( \text{vals} \). \( V_i(\text{vals}) = 1 \) if both \( v_1 \) and \( v_2 \) are added to \( \text{vals} \). \( V_i(\text{vals}) = \bot \) otherwise (ln 15-17). A correct replica only sends one \( aux_\ast() \) message in each round.

In the second phase (ln 18-34), replica \( p_i \) only accepts an \( aux_\ast(v_1, v_2) \) message if both \( v_1 \) and \( v_2 \) are added to \( \text{vals} \). \( V_i(\text{vals}) = 1 \) if both \( v_1 \) and \( v_2 \) are added to \( \text{vals} \). \( V_i(\text{vals}) = \bot \) otherwise (ln 15-17). A correct replica only sends one \( aux_\ast() \) message in each round.
We introduce a new distributed computing primitive, reproposable biased works such as MMR, Cobalt, and CrainH. Note in round \( r \), after that, as replicas have to compare their value with the common process that do not converge to the same value at the end of each round, the expected number of rounds is 4.

The property is called biased termination. The above two properties are two core properties for RABA. Other properties are either the same as ABA or require some tweaks due to the syntactic difference between ABA and RABA.

RABA syntax (API). Syntactically, a RABA protocol tagged with a unique identifier \( \text{sid} \) is specified by \( \text{propose}(\text{sid}, \cdot) \), \( \text{repropose}(\text{sid}, \cdot) \), and \( \text{decide}(\text{sid}, \cdot) \), where the input domain is \( \{0, 1\} \). For our purpose, RABA is “biased towards 1.” (Namely, “1” is the favorable value.) It is up to the high-level protocol to decide when to trigger the \( \text{propose} \) and \( \text{repropose} \) functions for each correct replica. Below are the function constraints enforced by correct replicas:

- \( \text{propose}(\text{sid}, \cdot) \): Each replica can propose a value \( v \) at the beginning of the protocol \( \text{sid} \). Each replica can propose a value only once.
- \( \text{repropose}(\text{sid}, \cdot) \): A 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. If a replica reproposes 1, it does so at most once.
- \( \text{decide}(\text{sid}, \cdot) \): A replica terminates the protocol identified by \( \text{sid} \) by generating a decide message.

Definitions of security. 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 \( \text{decides} \ \bar{v} \).
- **Unanimous termination**: If all correct replicas propose \( v \) and never repropose \( \bar{v} \), then all correct replicas eventually terminate.
- **Agreement**: If a correct replica \( \text{decides} \ \bar{v} \), then any correct replica that terminates \( \text{decides} \ \bar{v} \).
- **Biased validity**: If \( f + 1 \) correct replicas \( \text{propose} \ 1 \), then any correct replica that terminates \( \text{decides} \ 1 \).
- **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 \( Q_1 \neq \emptyset \) 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. 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 \( \text{decides} \ v \).

The agreement property of RABA is identical to that of ABA.
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. While the property echoes that of VABA [16], RABA is not in the context of “validated” ABA (VABA). This is precisely our goal, as VABA requires the usage of expensive (threshold) signatures 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 (and we need to be creative when using RABA).

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. Note we do not further restrict the APIs of RABA, as we find RABA in 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).

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

One might wonder: 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 $\text{id}$.

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—PACE.

4.2 Pisa: Efficient RABA from Pillar

We present a RABA protocol, Pisa, built on top of Pillar. As illustrated in Figure 3, we modify the round $r = 0$ of Pillar, while the code for round $r > 0$ remains unchanged. In particular, we make the following major changes in round 0. First, each replica $p_i$ that proposes 0 can repropose 1. At ln 03-04, if the repropose() event is triggered, regardless of which round $p_i$ is in, it broadcasts a $bval_0(1, \bot)$ message if it has not done so. Second, if a correct replica $p_i$ proposes 1, it immediately adds 1 to $bin_values_i$ (ln 08-09). If $p_i$ has not previously broadcast an aux$_i$() message, it broadcasts aux$_i(1, 1)$ (ln 10). Third, if $p_i$ proposes 0 and receives $f + 1$ $bval_i(1, \bot)$ (ln 12-17), in addition to broadcasting $bval_i(1, \bot)$ (ln 14), it immediately adds 1 to $bin_values_i$ (ln 16). If $p_i$ has not sent any aux$_i$() message, it also broadcasts aux$_i(1, 1)$ (ln 17). Last, ln 24 sets the value of

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{pisa.png}
\caption{The Pisa protocol for round 0 only at $p_i$.}
\end{figure}

common coin to 1 in round 0. Doing so ensures that if a replica receives $n - f$ aux$_i(1, 1)$, it will directly terminate the protocol. For each replica that receives both aux$_i(1, 1)$ and aux$_i(0, 0)$ (ln 26), the replica enters the next round using $est_{r+1} = 1$ and $maj_{r+1} = 1$.

Intuition and analysis. Our motivation for Pisa is that upon proposing 1, replicas directly broadcast both $bval_i(1, \bot)$ and aux$_i(1, 1)$ all at once, knowing that all correct replicas will either propose 1 or repropose 1. In other words, the protocol will eventually terminate and replicas can decide in only one step in the optimal case.

In round 0, a replica $p_i$ puts 1 in $bin_values_i$ and directly broadcasts aux$_i(1, 1)$ if its initial input is 1. This ensures that if more than $f + 1$ correct replicas have 1 as their input in round 0, no correct replicas can receive $2f + 1$ aux$_i(0, 0)$ and use 0 as input for the next round. As the common coin is set to 1 in round 0, 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 4 (we use aux$_i(v)$ in the figure, as each replica broadcasts aux$_i(v, v)$ in round 0). Replica $p_0$ proposes 1 while $p_1$ and $p_2$ propose 0. The faulty replica $p_3$ can send $bval_i(0, \bot)$ to $p_2$ and $p_3$ and make them send aux$_i(0, 0)$. While $p_0$ can receive $n - f$ bval$_i(0, \bot)$ messages, put 0 to its $bin_values_i$, and accept aux$_i(0, 0)$, it has already sent aux$_i(1, 1)$. Since $p_2$ and $p_3$ cannot receive enough bval$_i(1, \bot)$ messages, they will not put 1 to their $bin_values_i$. If $p_3$ does not send any aux$_i()$ messages, $p_1$ and $p_2$ are unable to move to the next round or terminate the protocol.

Pisa achieves biased termination if a repropose 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 $bval_0(1, \bot)$. Since $p_1$ and $p_2$ are still in round 0, they are able to collect $n - f$ bval$_0(1, \bot)$ and put 1 to $bin_values_i$. Hence, all replicas will eventually move to the next.
We are now ready to describe PACE, our new asynchronous BFT framework. PACE uses \( r\text{-broadcast} \) and \( r\text{-deliver} \) primitives of RBC, and \( \text{propose} \), \( \text{repropose} \) and \( \text{decide} \) primitives of RABA to overcome the two-subphase bottleneck. Figure 5 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 \( p_i \) \( r\text{-broadcasts} \) a proposal \( m_i \) for RBC\(i\). If \( p_i \) \( r\text{-delivers} \) 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 \( (\cup_{j \in S}(m_j)) \) in some deterministic order.

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 \( r\text{-delivered} \) 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. Indeed, according to the biased termination property, a RABA instance RABA\(i\) will decide 1, if there are \( f + 1 \) or more correct replicas proposing 1. We observe that a correct replica will propose 1 for at least \( 2f + 1 \) RABA instances (which is ensured by RBC). All correct replicas will input 1 for \( (2f + 1)/(2f + 1) \) inputs for all RABA instances. As there are at most \( (3f + 1)/(2f + 1) \) inputs for correct replicas, the total number of the 0 input from all correct replicas for all RABA instances is upper bounded by \( (3f + 1)/(2f + 1) \times (2f + 1) = 2f + f \). Hence, the number of RABA instances that decide 0 is at most \( \frac{2f + f}{3f + 1} > \frac{2f + f}{2f + 1} = 2f \). That is, the number of RABA instances that decide 1 is at least \( f + 1 \). An example is shown in Figure 6 below.
We analyze the worst-case scenario of BKR and PACE. We let $n$ wait for $t$.

An adversary could delay all the $2f + 1$ instances. Each column represents a RABA instance and the inputs of run the ABA protocols and complete them at the time no later than $2f + 1$.

$satisfies the predicate of biased termination. For simplicity, we let after every correct replica proposes or repososes some value that be the max delay an adversary could inject on any RABA protocol.

$\Delta$ every correct replica proposes some value. For RABA, we let delay an adversary could inject for any ABA protocol instance after replicas to complete any RBC instance. We further let $\Delta$ be the max delay an adversary could inject for any ABA protocol instance after every correct replica proposes some value. For RABA, we let $\Delta$ be the max delay an adversary could inject for any ABA protocol instance after every correct replica proposes or repososes some value that satisfies the predicate of biased termination. For simplicity, we let $\Delta_2 = \Delta_3$, as none of our RABA constructions introduce additional steps on top of ABA.

Figure 7a illustrates the worst-case scenario for the BKR diagram. An adversary could delay all the $2f + 1$ RBC instances (such that the sender is correct for any RBC) as much as possible until time $t + \Delta_1$. The $f$ faulty replicas can also choose not to start their RBC instances. After the $2f + 1$ RBC instances complete, replicas will run the ABA protocols and complete them at the time no later than $t + \Delta_1 + \Delta_2$. Since the $f$ RBC instances (such that the sender is faulty) are not even started, replicas will vote for 0 for the corresponding ABA instances. Thus, the entire epoch terminates at the time no later than $t + \Delta_1 + 2\Delta_2$.

In contrast, Figure 7b illustrates the situation for PACE. Each correct replica starts the RABA phase after it completes $2f + 1$ RBC instances. In the worst case, an adversary can simply delay some RABA instances (e.g., by triggering Case 3 mentioned for instances such that fewer than $f + 1$ correct replicas propose 1). Every correct replica eventually reproposes 1 according to the agreement property of RBC, i.e., no later than time $t + \Delta_1$, all correct replicas complete all RBC instances and propose or repropose for all RABA instances. Thus, the completion time of PACE is no later than $t + \Delta_1 + \Delta_2$. To conclude, in the worst case, PACE still outpaces BKR.

### 5.2 Efficient Instantiations

We instantiate our new framework using CrainH-R, CrainL-R, and Pisa as the underlying RABA protocols. The resulting protocols are called PACE-CrainH-R, PACE-CrainL-R, and PACE-Pisa, respectively.

Figure 7: PACE vs. BKR. Comparison of the worst-case scenario.

In summary, while our paradigm does not improve the worst-case running time and communication complexity of BKR, it does improve the concrete time complexity. More importantly, it avoids the “two-sub-phase” bottleneck for BKR.

Note that in the worst case, BKR always delivers transactions from $n - f$ replicas in an epoch. This is not a benefit compared to PACE which delivers transactions from $f + 1$ replicas. BKR has to wait for $n - f$ transactions to proceed, causing transaction congestion. Also note that for PACE, in the normal case, transactions from at least $\lceil \frac{n + f + 1}{2} \rceil$ replicas will be delivered. Jumping ahead, according to our experiments, in both failure and failure-free scenarios, the throughput of PACE is higher that of BKR.

In summary, while our paradigm does not improve the worst-case running time and communication complexity of BKR, it does improve the concrete time complexity. More importantly, it avoids the “two-sub-phase” bottleneck for BKR.

Note that in the worst case, BKR always delivers transactions from $n - f$ replicas in an epoch. This is not a benefit compared to PACE which delivers transactions from $f + 1$ replicas. BKR has to wait for $n - f$ transactions to proceed, causing transaction congestion. Also note that for PACE, in the normal case, transactions from at least $\lceil \frac{n + f + 1}{2} \rceil$ replicas will be delivered. Jumping ahead, according to our experiments, in both failure and failure-free scenarios, the throughput of PACE is higher that of BKR.

### 5.1 PACE vs. BKR: Worst-Case Scenario

We analyze the worst-case scenario of BKR and PACE. We let $t$ be the time when an epoch begins, i.e., when any RBC is started. We let $\Delta_1$ be the max delay an adversary could inject for all correct replicas to complete any RBC instance. We further let $\Delta_2$ be the max delay an adversary could inject for any ABA protocol instance after every correct replica proposes some value. For RABA, we let $\Delta_3$ be the max delay an adversary could inject on any ABA protocol after every correct replica proposes or reposposes some value that satisfies the predicate of biased termination. For simplicity, we let $\Delta_2 = \Delta_3$, as none of our RABA constructions introduce additional steps on top of ABA.

Figure 7a illustrates the worst-case scenario for the BKR diagram. An adversary could delay all the $2f + 1$ RBC instances (such that the sender is correct for any RBC) as much as possible until time $t + \Delta_1$. The $f$ faulty replicas can also choose not to start their RBC instances. After the $2f + 1$ RBC instances complete, replicas will run the ABA protocols and complete them at the time no later than $t + \Delta_1 + \Delta_2$.

Since the $f$ RBC instances (such that the sender is faulty) are not even started, replicas will vote for 0 for the corresponding ABA instances. Thus, the entire epoch terminates at the time no later than $t + \Delta_1 + 2\Delta_2$.

In contrast, Figure 7b illustrates the situation for PACE. Each correct replica starts the RABA phase after it completes $2f + 1$ RBC instances. In the worst case, an adversary can simply delay some RABA instances (e.g., by triggering Case 3 mentioned for instances such that fewer than $f + 1$ correct replicas propose 1). Every correct replica eventually reproposes 1 according to the agreement property of RBC, i.e., no later than time $t + \Delta_1$, all correct replicas complete all RBC instances and propose or repropose for all RABA instances. Thus, the completion time of PACE is no later than $t + \Delta_1 + \Delta_2$. To conclude, in the worst case, PACE still outpaces BKR.

### 5.2 Efficient Instantiations

We instantiate our new framework using CrainH-R, CrainL-R, and Pisa as the underlying RABA protocols. The resulting protocols are called PACE-CrainH-R, PACE-CrainL-R, and PACE-Pisa, respectively.

Note that the correctness of our framework directly implies the correctness of all these protocols. It may still be helpful to examine an execution example. Here we discuss the same example in Figure 4. 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-Pisa terminates due to biased termination.

### 6 IMPLEMENTATION AND EVALUATION

**Implementation.** We first implement six ABA protocols—Pillar, Pisa, CrainH, CrainH-R, CrainL, and CrainL-R. We use threshold PRF of Cachin, Kursawe, and Shoup [17] for common coins. For Pillar, Pisa, CrainL and CrainL-R, we use the $f + 1$ threshold. For CrainH and CrainH-R, we use the $2f + 1$ threshold. We use Pillar, CrainH, and CrainL in the BEAT library [2] and implement BEAT-Pillar, BEAT-CrainH, and BEAT-CrainL. We also instantiate the PACE framework using Pisa, CrainH-R, and CrainL-R, yielding PACE-Pisa, PACE-CrainH-R, and PACE-CrainL-R. We compare the six new protocols with BEAT (both BEAT-MMR and BEAT-Cobalt)
and Dumbo [3]. Therefore, in total, we evaluate the performance of nine BFT protocols.

**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 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. The LAN evaluation is limited to the case for \( f = 1 \), because our EC2 account in general cannot launch enough VMs in a single DC.

![Figure 8: Latency of the protocols under no contention where the replicas are located in WANs. (This and subsequent figures are best viewed in color.)](image)

**Latency.** We first evaluate the latency of the protocols under no contention in WANs, where each replica proposes only a batch of one transaction. We report the latency for \( f = 1, f = 5, f = 15 \), and \( f = 30 \). As illustrated in Figure 8, the latency of BEAT-Cobalt is consistently the highest among the protocols, mostly because Cobalt ABA has one more step in each round. Meanwhile, the latency of the PACE protocols (PACE-Pisa, PACE-CrainH-R, PACE-CrainL-R) is much lower than the others, because RABA terminates faster than ABA, and PACE has only one subphase. In contrast, the latency of Dumbo is in most of the cases slightly higher than the BKR protocols. The latency of PACE-Pisa is between 1/3 to 1/2 of the latency of Dumbo. We also evaluate the latency in the LAN for \( f = 1 \) (not shown in the figure): the latency of PACE-Pisa is 0.04s while the latency of Dumbo is 0.36s.

**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 \( nB \) transactions for an epoch. We evaluate the throughput and latency vs. throughput of \( B \) increases. We report the throughput of the BFT protocols for \( f = 1 \) in both LAN (Figure 9a) and WAN settings (Figure 9b) and report the throughput vs. latency in Figure 10a.

![Figure 9: Throughput and latency vs. throughput as \( nB \) increases.](image)

Protocols in the PACE framework (PACE-Pisa, PACE-CrainH-R, PACE-CrainL-R) outperform all other protocols. For instance, PACE-Pisa achieves 40% and 77% higher throughput than BEAT-Cobalt in the LAN and WAN settings, respectively. PACE-Pisa achieves 5.1x the throughput of Dumbo. This is due to the faster (biased) termination for RABA and the fully parallel feature of PACE-Pisa.

Furthermore, every PACE protocol outperforms its counterpart under the BKR framework (BEAT-Pillar, BEAT-CrainH, BEAT-CrainL).

The throughput of PACE-Pisa, PACE-CrainH-R, and PACE-CrainL-R are 24.5%, 171%, and 15.6% higher than BEAT-Pillar, BEAT-CrainH, and BEAT-CrainL, respectively.

Among the three PACE protocols, PACE-Pisa achieves the best performance. PACE-CrainH-R achieves lower performance than PACE-Pisa, mainly due to the fact that replicas have to collect \( 2f + 1 \) threshold PRF shares to generate common coins. PACE-CrainL-R achieves the lowest performance among the three, because each round of CrainL consists of more steps.

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, BEAT-Pillar and PACE-Pisa achieve 176% and 287% higher throughput than BEAT-Cobalt for WANs, respectively.

**Scalability.** We evaluate the throughput of the protocols by varying \( f \) from 2 to 30 in WAN. We report the throughput in Figure 9 and latency vs. throughput in Figure 10.

All the protocols under PACE framework outperform their counterparts under the BKR framework. Even the slowest PACE instantiation is more efficient than all other protocols.

For the three BEAT protocols (BEAT-Pillar, BEAT-CrainH, and BEAT-CrainL), BEAT-CrainL achieves the highest performance in most cases (except for \( f = 1 \)). This may be due to the fact that CrainL has the good-case-coin-free property and has a fewer expected number of rounds than CrainH and Pisa (3 vs. 4).

Different from the results we obtain for \( f = 1 \), as \( f \) grows, the performance difference among PACE-Pisa, PACE-CrainH-R, and PACE-CrainL-R become comparatively smaller. This may be due to the fact all these protocols have a fast and biased path (and it does not matter if the underlying ABA has a fast path). PACE-Pisa and PACE-CrainL-R perform better than PACE-CrainH-R. PACE-Pisa is also more efficient than PACE-CrainL-R, except for the case when \( n \) is medium-sized, where the two protocols offer interesting trade-offs.

For all the protocols, as the network size \( f \) increases, the throughput for these protocols first increases and then decreases. This echoes the results from prior works. When \( f \) grows, the number of transactions proposed concurrently grows accordingly. Nevertheless, when \( f \) further grows, the protocol itself becomes the bottleneck.

In the largest experiment (\( n = 91 \)), the throughput of PACE-Pisa is around 15,994 tx/sec, around 3.6x the throughput of BEAT-Cobalt and 1.66x the throughput of Dumbo. (Recall for \( f = 1 \), PACE-Pisa achieves 1.77x the throughput of BEAT-Cobalt and 5.1x the throughput of Dumbo.)

Dumbo has lower throughput than BEAT-Cobalt for \( f \leq 15 \) and has roughly the same throughput as BEAT-Pillar for larger \( f \)s. BEAT-CrainH slightly outperforms Dumbo when \( f \leq 15 \) and Dumbo slightly outperforms BEAT-CrainL for \( f = 20 \) and \( f = 30 \).

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 \( 10^3 \) 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
We analyze the number of ABA or RABA instances that works.

Table 3: The number of proposals that are delivered in ABA instances that have not been started by proposing running all instances in parallel (the “naive attempt”): after deciding 1 to compare PACE and BKR frameworks. Just for performance comparison, we implement an incorrect framework directly running all instances in parallel (the “naive attempt”): 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 here, and recall that in failure cases, its throughput can be 0 [11, 38].

We summarize in Table 3 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 efficiency of our framework is also higher.

**Performance under failures.** We evaluate the performance of the protocols under failures by fixing \( f = 5 \) and \( B = 5,000 \). We evaluate three failure scenarios: 1) \( S_1 \)-crash, where we let \( f \) replicas simply crash; 2) \( S_2 \)-zero, where each faulty replica always broadcasts 0 in every step of ABA/RABA; 3) \( S_3 \)-flip, where each replica replica...
always broadcasts a flipped value in every step of ABA/RABA. We compare the results with \( S_0 \)-ff, the failure-free scenario.

The results are summarized in Figure 11. For all protocols, the performance under crash failures are similar to that in the failure-free scenario. When Byzantine failures occur, their performance degrade. We find that for most protocols under the BKR framework, the number of ABA instances that decide 1 is slightly larger than \( n - f \), while the number of ABA instances that decide 1 under the PACE framework is between 8 to 12 (slightly lower than \( n - f \)).

But this does not hurt the performance of PACE protocols: the performance degradation for PACE protocols, by percentage, is no larger than that for BEAT protocols. This is because PACE protocols do not have the two-subphase bottleneck and this is consistent with our analysis in Sec. 5.1.

We find that for all PACE protocols, the performance under all failure scenarios is still better than their individual counterparts in the BKR framework. Even more remarkably, among all failure scenarios, the slowest PACE instantiation (PACE-CrainH-R with \( S_2 \)-zero) outpaces the most efficient BEAT instantiation in its failure-free scenario (BEAT-Pillar with \( S_0 \)-ff).

7 CONCLUSION

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 (PACE) outperforming prior ones. We provide efficient instantiations of RABA and PACE. We show that all PACE instantiations outperform existing protocols, both in terms of the number of ABA instances that decide 1 and their overall performance in failure-free scenarios.

Figure 11: Performance of the protocols in failure scenarios in WANs for \( f = 5 \) and \( B = 5,000 \). We use B- and P- to denote BEAT- and PACE- respectively to save space in this figure.
ACKNOWLEDGMENT
We thank Chao Liu, Yuan Lu, Xiao Sui, Yue Huang, Ren Zhang, Yang Yu, Liehuang Zhu, and Xiaoyuan Wang for their help and comments. We are indebted to the CCS 2022 reviewers (the blockchain track) for their insightful and constructive comments that significantly helped improve the paper.

REFERENCES

Optimal Validated Asynchronous Byzantine Agreement. In Proceedings of the
replication under attack. IEEE Transactions on Dependable and Secure Computing
distributed non-interactive adaptively-secure threshold signatures with short
shares. In Theoretical Computer Science.
putations with optimal resilience. In Proceedings of the 13th annual symposium on
In Proceedings of the third annual ACM symposium on Principles of distributed
computing. ACM, 154–162.
tine Fault Tolerant Vector Consensus with Anonymous Proposals. arXiv preprint
Reliable and Secure Distributed Programming (2nd ed.).
cure and efficient asynchronous broadcast protocols. In Annual International
Introduction to dependable and secure computing 4, 4 (2011), 154–162.
SIGSAC Conference on Computer and Communications Security. ACM.
rate Fault Tolerant Vector Consensus with Anonymous Proposals.
with Adaptive Security. In DSN.
2008. RITAS: Services for randomized intrusion tolerance. IEEE transactions on
dependable and secure computing 8, 1 (2008), 122–136.
[26] Achour Mostefai, Hamouma Moumen, and Michel Raynal. 2014. Signature-
free asynchronous byzantine consensus with t< n/3 and o (t2) messages. In
PODC. ACM, 2–9.
[27] Achour Mostefai, Hamouma Moumen, and Michel Raynal. 2015. Signature-
Free Asynchronous Binary Byzantine Consensus with t < n/3, O(t2) Messages, and
[29] Harsh Govind V Ramasamy and Christian Cachin. 2005. PArsimonymous Asyn-
Byzantine fault tolerance. In FRIDA.

A REVIEW OF MMR AND COBALT ABA

The pseudocode of MMR is given in Figure 12. First, each replica broadcasts bvval, (est,) where est is the input of the round (in round 0, est is the ABA vote). If a replica receives f+1 bvval, (v) and has not broadcast v, it broadcasts bvval, v. Upon receiving n − f bval, v, a replica adds v to bin_values. Then each replica sends an aux(, ) message for the first value added to bin_values. Upon receiving
n − f aux(, ) messages such that the set of values carried by these message, vals, is a subset of bin_values, the replica compares the value(s) with the common coin. 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,. Cobalt ABA [37] has one additional conf(, ) step in each round, also as shown in Figure 12.

B PSEUDOCODE OF THE BKR PROTOCOL

We show the pseudocode of the BKR paradigm in Figure 14 that uses the r-broadcast and r-deliver primitives of RBC, and propose and decide primitives of ABA.
we tag each message in the instances with

includes a RBC phase, including

then makes

+1 broadcast

aux

if

upon receiving

RBC phase, the replica must abstain from proposing

wait until

broadcast

aux

for the common coin.

upon receiving

aux

such that the set of values carried by these messages,

broadcast

conf

+1

upon receiving

BVAL(0)

aux

coin share

AUX(0)

2f+1

AUX(1)

coin share

AUX(0)

2f+1

AUX(1), coin =0, enter next round

enter next round

with coin value

enter next round

with coin value

Figure 12: MMR ABA and Cobalt ABA. Cobalt ABA has the boxed code, while MMR ABA does not have it.

Figure 13: The liveness issue of MMR. A faulty replica

first sends

valr(1) to

Since

will send

and its threshold signature shares for the common coin.

generates a share and combines the

+1 = 2 shares to obtain the common coin value (e.g., 0). Then

makes

send

aux

by letting

receive

valr(1).

Also,

sends

aux

to

making

receive

aux

(a value different from the common coin) and use 1 as the input for the next round. For

and

since they have added both

and

in their

, and receive both

and

, they will enter the next round with the common coin value

0. In this way, the protocol may never terminate.

BKR proceeds in epochs initialized as

Each epoch

includes a RBC phase, including

parallel RBC instances

for

, and an ABA phase, including

ABA instances

for

, where

is triggered by replica

to r-broadcast a proposal

(a batch of transactions) selected from its buffer, and

is triggered by correct replicas to decide if

has been r-delivered. The ABA phase is not fully parallel, requiring that if a replica has not received some proposal from

during the RBC phase, the replica must abstain from proposing

for

until

ABA instances decide 1. To isolate the primitive instances, we tag each message in the instances with

Figure 14: The BKR paradigm. The code for replica

for an epoch

Figure 15: Dumbo. PRBC denotes provable reliable broadcast, a four-step protocol. CBC stands for consistent broadcast and has three steps [16]. Dumbo (built on top of CKPS) has one PRBC phase (4 steps), two CBC phases (2 × 3 steps), one permutation step ("P"), and three sequential distribution (1 step) and ABA (3 for Cobalt ABA) instances. As each Cobalt ABA instance takes on average 3 × 4 or 4 × 4 steps, Dumbo has on average 50 or 62 steps.

C REVIEWING THE CKPS PARADIGM AND ITS DERIVATIVES

The CKPS paradigm reduces asynchronous BFT to (multi-valued) validated asynchronous Byzantine agreement (VABA) [16]. CKPS shows that VABA can be built using (verifiable) consistent broadcast (CBC) and ABA.

SINTRA [18]. 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 [31]. Dumbo includes two asynchronous BFT protocols—Dumbo1 and Dumbo2. Dumbo2 performs consistently better than Dumbo1, so we focus on Dumbo2 only (see Figure 15). Dumbo is motivated by the fact that the VABA construction in CKPS requires a large bandwidth. Instead of directly applying the bulk data to VABA, Dumbo introduces an additional provable RBC (PRBC) phase such that only the PRBC phase carries bulk data, and the VABA phase takes as input data fingerprints only. Hence, Dumbo has four more steps due to the PRBC phase, including three expensive

paired

and an additional

pairing operations, higher latency than CKPS. Our experiment shows BEAT-Pillar outpaces Dumbo until

. Even when

is between

and

, Dumbo has marginally higher throughput than BEAT-Pillar.
D PROOF OF CORRECTNESS FOR PILLAR

In this section, we prove the correctness of Pillar.

**Lemma D.1.** In round $r > 0$, if a correct replica $p_i$ sets $\delta_i(v) = 1$ and another correct replica sets $\delta_j(\bar{v}) = 1$, $v = \bar{v}$.

**Proof.** In round $r > 0$, correct replicas may send $bval_i(\ast, v)$, $bval_i(\ast, \bar{v})$, or $bval_i(\ast, \bot)$ and do not change the $maj_f$ 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 $2f + 1 bval_i(v, v)$ and $bval_i(v, \bot)$ and sets $\delta_i(v) = 1$, at least $f + 1$ correct replicas have sent $bval_i(v, v)$ or $bval_i(v, \bot)$, but will never send $bval_i(v, \bar{v})$ or $bval_i(\bar{v}, \bar{v})$. On the other hand, if $p_j$ sets $\delta_j(\bar{v}) = 1$, it has received $2f + 1 bval_i(\bar{v}, \bar{v})$, at least $f + 1$ of which are sent by correct replicas. Therefore, at least one correct replica has sent both $bval_i(v, v)$ (or $bval_i(v, \bot)$) and $bval_i(\bar{v}, \bar{v})$, which is a contradiction. The case for $v = \bar{s}_{r-1}$ can be proved similarly. If $p_i$ sets $\delta_i(v) = 1$, it receives $2f + 1 bval_i(v, v)$. If $p_j$ sets $\delta_j(\bar{v}) = 1$, it receives $2f + 1 bval_i(\bar{v}, \bar{v})$ or $bval_i(\bar{v}, \bot)$. In other words, at least one correct replica has sent both $bval_i(v, \ast)$ and $bval_i(\ast, v)$ (or $bval_i(\ast, \bot)$), which is impossible.

**Lemma D.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 $bval_i(v, v)$, $bval_i(v, \bar{v})$, or $bval_i(v, \bot)$. Hence, correct replicas will not receive more than $f + 1$ $bval_i(\ast, \ast)$ and no correct replica will put $\bar{v}$ in its $bin\_values$. Therefore, all correct replicas will send $aux_i(v, v)$ or $aux_i(\bot, v)$ and will not accept $aux_i(\ast, \ast)$ or $aux_i(\ast, \bot)$. Each correct replica therefore eventually receives either $2f + 1 aux_i(v, v)$ such that $V_2(v, aux_i, v) \geq 2f + 1$ or $2f + 1 aux_i(\ast, v)$ such that $V_2(v, aux_i, v) \geq 2f + 1$.

If a replica $p_i$ receives $2f + 1 aux_i(v, v)$, it either decides $v$ (for the case $v = s_{r-1}$), or enters the next round with $v$ as $est_{r+1}$ (for the case $v \neq s_{r-1}$). If $p_i$ receives both $aux_i(v, v)$ and $aux_i(\bot, v)$ (or only $aux_i(\bot, v)$), $V_2(\bot, aux_i, v) \geq 2f + 1$ is satisfied. In this case, $p_i$ will either decide $v$ (for the case $v = s_{r-1}$), or enter the next round with $v$ as $est_{r+1}$ (for the case $v \neq s_{r-1}$).

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

**Proof.** Lemma D.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 D.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 $est_0$. As proven in Lemma D.2 and Lemma D.4, any correct replica that terminates decides $v$.

In round $r = 0$, all correct replicas broadcast $bval_i(v, \bot)$. Since all correct replicas have the same input $v$ and there are only $f$ faulty replicas, correct replicas will not receive more than $f + 1 bval_i(\ast, \bot)$ or send $bval_i(\ast, \bot)$. All correct replicas will eventually receive $2f + 1 bval_i(v, \bot)$ and send $aux_i(v, \ast)$. Since no correct replica puts $\bar{v}$ in $bin\_values$, all correct replicas will have only one single value $v$. If a correct replica terminates in round $0$ and $v = s_f$, it decides $v$. If $v \neq s_f$, correct replicas use $v$ as the input $est_0$ for round $1$. According to Lemma D.4, any correct replica that terminates decides $v$.

This completes the proof of the theorem.

**Theorem D.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 D.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 $2f + 1 aux_i(v, v)$ messages and $v = s_r$; $2$) $p_i$ receives both $aux_i(v, v)$ and $aux_i(\bot, v)$. Also, $p_i$ receives at least a quorum of $aux_i(\ast, v)$ messages, $v = s_{r-1}$, and $v = s_r$; $3$) $p_i$ enters the next round with value $est_{r+1} = \bar{v}$, it cannot use the common coin value as input, as $v$ is the common coin. Therefore, one of the following conditions must apply: $A$) $p_j$ receives at least $2f + 1 aux_i(\bar{v}, \bar{v})$; $B$) $p_j$ receives both $aux_i(\bar{v}, \bar{v})$ and $aux_i(\bot, v)$. At least a quorum of the messages are of the form $aux_i(\ast, \ast)$; $C$) $p_j$ receives both $aux_i(\ast, \bar{v})$ and $aux_i(\bot, \ast)$ 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 satisfied.

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

**Case 2:** $p_j$ receives both $aux_i(v, v)$ and $aux_i(\bot, v)$. Also, $p_j$ receives at least a quorum of $aux_i(\ast, v)$ messages, $v = s_{r-1}$, and $v = s_r$. $1$) If condition $A$ is true, $p_j$ receives at least $2f + 1 aux_i(\bar{v}, \bar{v})$, at least $f + 1$ are sent by correct replicas. Also, $p_j$ receives $2f + 1 aux_i(v, v)$ and $2f + 1 aux_i(\bot, v)$. Therefore, at least one correct replica must have sent $aux_i(\bar{v}, \bar{v})$ to $p_j$ and $aux_i(v, v)$ (or $aux_i(\bot, v)$) to $p_j$, which is impossible. Condition $A$ cannot be true. $2$) If condition $B$ is true, $p_j$ receives both $aux_i(\bar{v}, \bar{v})$ and $aux_i(\bot, \bar{v})$, and at least a quorum of the
messages are of the form $aux_r(\ast, \hat{v})$. Since $p_1$ receives a quorum of $aux_r(\ast, v)$ messages, at least one correct replica has sent $aux_r(\ast, v)$ to $p_1$ and $aux_r(\ast, \hat{d})$ to $p_j$. Condition B cannot be true. 3) $p_j$ only receives $aux_r(\bot, \ast)$ and $aux_r(\ast, \ast)$ and $\hat{d} = s_{r-1}$. This cannot be true since $\hat{d} = s_{r-1}$. □

For the two cases, it is clear that if $p_j$ decides in round $r$, it outputs $\hat{d}$. We now show that if $p_j$ terminates in round $r'$ and decides $\hat{d}$, $v = \hat{d}$ where $r' > r$. In round $r'$, one of the following cases must apply: A) $p_j$ receives at least $2f + 1$ $aux_r(\ast, \hat{d})$ or $bval_r(\ast, \ast)$. Also, $\hat{d} = s_{r-1}$ and $\hat{d} = s_{r'}$. If any of the two cases is true, at least one correct replica has sent $bval_r(\ast, \ast)$. Meanwhile, according to Lemma D.7, any correct replica that enters round $r + 1$ uses $v$ as input. Furthermore, according to Lemma D.2, if all correct replicas enter round $r + 1$ and use $v$ as input, any correct replica either decides $v$ or uses $est_{r+2} = v$ and enters round $r + 2$. Therefore, there exists some round $r'''$ where $r \leq r'' < r'$, a correct replica uses $\hat{d}$ as input and broadcasts $bval_r(\ast, \ast)$, i.e., Lemma D.2 is violated, a contradiction.

**Theorem D.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 $bval_r(v, \ast)$ or $bval_r(\hat{d}, \ast)$. Also, there are at least $2f + 1$ correct replicas, among which at least $f + 1$ correct replicas propose the same value. Therefore, if $\hat{d}$ is different from its proposed value, each correct replica $p_i$ will forward $bval_r(\hat{d}, \hat{v}_i)$. Correct replicas will eventually receive $2f + 1$ $bval_r(\ast, v)$ messages for at least one binary value (e.g., $v$) and send $aux_r(v, \ast)$ or $aux_r(\hat{d}, \ast)$. Similarly, all correct replicas eventually receive $2f + 1$ $aux_r(\ast, v)$ messages and proceed to the next round.

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

If $p_j$ enters the next round with $est_{r+1} = v = s_{r-1}$, it must satisfy one of the three conditions: 1) $p_j$ receives at least $2f + 1$ $aux_r(v, v)$ messages; 2) $p_j$ receives both $aux_r(v, v)$ and $aux_r(\hat{d}, \ast)$. At least a quorum of the messages are $aux_r(\ast, v)$ or $aux_r(\hat{d}, \ast)$. Therefore, each correct replica receives at least one $aux_r(v, v)$ message. According to Lemma D.1, correct replicas may send $aux_r(\ast, v)$, $aux_r(\hat{d}, \ast)$, but will not send $aux_r(\hat{d}, \ast)$. No correct replica will set $\delta_r(\hat{d}) = 1$. The value $v$ is determined based on the $maj$ values by the replicas so it cannot be manipulated by the adversary. Therefore, with a probability of $1/2$, $p_j$ decides $v$. Otherwise $p_i$ enters the next round and use $est_{r+1} = v$.

For the second case, $p_j$ decides if $b = s_{r-1}$ and $b = s_{r}$. With a probability of $1/2$, $s_{r-1} = s_{r}$. In this case $p_i$ will not decide. $p_j$ uses the value of $s_r$ as $est_{r+1}$ with a probability of $1/2$.

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

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

**E PROOF OF CORRECTNESS FOR PISA**

In this section, we prove the correctness of Pisa.

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

**Proof.** If all correct replicas propose $v$ and do not repropose $\hat{v}$, all correct replicas will only have $v$ in their $bin\_values_r$. Hence, replicas do not accept an $aux_r(\hat{d}, \hat{v})$ message. Each correct replica will collect $2f + 1$ $aux_r(v, v)$. If $v = 1$, replicas decide in round 0. Otherwise replicas enter round 1. Starting from round 1, Pisa follows Pillar. Therefore, according to Lemma D.2 and Lemma D.4, any correct replica that terminates decides $v$.

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

**Proof.** If all correct replicas propose $v$, they will all send $bval_r(v, \bot)$. All correct replicas eventually put $v$ to $bin\_values_r$. Correct replicas will not send or accept any $aux_r(\hat{d}, \hat{v})$ message and will accept either $aux_r(v, v)$ or $aux_r(\bot, v)$. According to Lemma D.2, all correct replicas will enter the next round with the same $est_{r+1}$ value (including the cases 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 E.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 $v$ 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 E.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 $V_2(vals_r, 0) \geq 2f + 1$ must be true. Hence, excluding the $aux_r(\bot, v)$ messages, the number of $aux_r(0, v)$ messages $p_j$ receives
is greater than $2f + 1$. Among the replicas that sent $2f + 1$ aux$_r(\bar{v}, v)$ and aux$_r(\bar{v}, \bar{v})$ messages, at least one correct replica must have sent both aux$_r(v, v)$ and aux$_r(\bar{v}, \bar{v})$. This is a contradiction, since a correct replica broadcasts aux$_r()$ once in each round. □

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

For the case where $p_1$ decides in round $r > 0$, agreement simply follows that of Pillar, as Pisa is the same as Pillar starting from $r > 0$.

**Theorem E.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 aux$_r(1, 1)$ and will not send aux$_r(0, 0)$. Therefore, all correct replicas will either receive $2f + 1$ aux$_r(1, 1)$ or both aux$_r(1, 1)$ and aux$_r(0, 0)$, but not $2f + 1$ aux$_r(0, 0)$. This is because if a correct replica receives $2f + 1$ aux$_r(0, 0)$, at least $f + 1$ correct replicas must have sent aux$_r(0, 0)$. Therefore, at least one correct replica must have sent both aux$_r(0, 0)$ and aux$_r(1, 1)$, which is impossible. If a correct replica receives $2f + 1$ 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 D.2 and Lemma D.4, all correct replicas that terminate decide 1.

**Theorem E.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 represpose 0. Let $Q_2$ be correct replicas that propose 0 and later represpose 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 $2f + 1$ aux$_r()$ messages.

**Case 1: More than $f + 1$ correct replicas propose 1.** All correct replicas will eventually receive $f + 1$ val$_0(1, \bot)$. According to the protocol, all correct replicas will eventually receive $2f + 1$ val$_0(1, \bot)$, put 1 in bin$_{\text{values}}$, and accept aux$_r(1, 1)$. Correct replicas may send aux$_r(1, 1)$ or aux$_r(0, 0)$. If a correct replica sends aux$_r(0, 0)$, it previously received $2f + 1$ val$_0(0, \bot)$ messages, among which at least $f + 1$ replicas are correct. Therefore, all correct replicas will eventually put 0 in their bin$_{\text{values}}$ and accept 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 $2f + 1$ val$_0(0, \bot)$ and put 0 to bin$_{\text{values}}$. It is, however, not guaranteed that a correct replica will receive $2f + 1$ val$_0(0, \bot)$ and put 1 in bin$_{\text{values}}$. Therefore, some correct replica that only has 0 in bin$_{\text{values}}$ will not accept an aux$_r(1, 1)$ message. The termination is guaranteed by the fact that correct replicas in $Q_2$ represpose 1. In particular, correct replicas will represpose 1, making correct replicas eventually receive a quorum of val$_0(1, \bot)$ messages. Hence, correct replicas will either enter the next round or eventually collect a quorum of val$_0(1, \bot)$ messages. In the latter case, correct replicas will be able to accept both aux$_r(0, 1)$ and 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 val$_0(0, \bot)$ messages and add 0 to bin$_{\text{values}}$. Furthermore, each replica in $Q_2$ may represpose. Therefore, all correct replicas will receive $2f + 1$ val$_0(1, \bot)$ and put 1 to bin$_{\text{values}}$. 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 E.7. (Integrity) No correct replica decides twice.**

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

**F PROOF OF CORRECTNESS FOR OUR BFT FRAMEWORK**

We prove the correctness of our BFT framework. The proof immediately implies the correctness of our BFT instantiations. 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 restrict 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—noll 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 F.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 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 propose 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 correct 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. \( \Box \)

We now prove set agreement.

Theorem F.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\(_j\) 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 F.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 F.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 F.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\(_j\) 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 \( (2f + 1)/(2f + 1) \) inputs 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 \( (3f + 1)/(2f + 1) - (2f + 1)/(2f + 1) = 2f^2 + f \), while in the normal case, transactions from at least \( \frac{nf^3 + 1}{2f} \) replicas will be delivered. 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.

G CONVEXTNG ABA TO RABA

Our strategy that converts ABA to RABA is generic. We now show how to convert a number of representative ABA protocols to RABA protocols, including the classic CKS ABA [17], CrainL ABA [25, 2nd algorithm], CrainL ABA [25, 1st algorithm], and Cobalt ABA [37]. As Cobalt, CrainL, and CrainL follow MMR, the proofs are similar to that for Pisa. Thus, we first focus on CKS [17] and provide a full proof for it. We then show in detail how to convert CrainL to RABA and briefly sketch how to do it for other protocols.

G.1 Converting CKS ABA to RABA

The pseudocode of CKS is shown in Figure 16. While the original CKS paper uses quite different notations, we use the same notations as other protocols presented in this paper. CKS uses a low-threshold \( (n, f + 1) \) threshold signature scheme (denoted \( ts1 \)) and high-threshold \( (n, n - f) \) threshold signature scheme (denoted \( ts2 \)).

We use \( ts1.\text{share} \) to represent a threshold signature share generated using \( ts1 \) and \( ts1.\text{sig} \) to represent a threshold signature combined from \( f + 1 \) threshold signature shares. Similarly, we use \( ts2.\text{share} \) to represent the threshold signature share for \( ts2 \) and \( ts2.\text{sig} \) 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 (the first round) and two steps in each round starting from round 1. In round 0, at ln 07, every replica broadcasts a bval\(_r\) (\( est_0, ts1.\text{share} \)) message, where \( est_0 \) is the proposed value and \( ts1.\text{share} \) is a threshold signature share. At ln 08-09, upon receiving \( 2f + 1 \) valid bval\(_r\) (\( message \), a replica selects a local parameter \( a \) as the majority value \( a \) received from bval\(_r\) messages and combines the threshold shares to \( sig \). At ln 14, the replica then sends an aux\(_r\) (\( a, sig, ts2.\text{share} \)) message.
to all replicas, where $t_{2.share}$ is a threshold signature share for the aux$_r()$ messages. Replicas then wait for the aux$_r()$ messages (ln 15). If a replica receives $n - f$ aux$_r()$ messages that contain only one valid value (ln 16), a replica combines the signature shares and obtains sig. The replica then broadcasts a conf$_r(a, sig, t_{2.share})$ messages where $t_{2.share}$ is a threshold signature share for the conf$_r()$ message (ln 18). If aux$_r()$ message contains multiple values, the replica broadcasts a conf$_r(\perp, aux())$ message (combined from ln 17-18), where $\perp$ is a special symbol such that $\perp \notin \{0, 1\}$, and aux$_r()$ messages serve as justification for $\perp$. Finally, a replica waits for $n - f$ valid conf$_r()$ messages (ln 19). If the replica receives $n - f$ conf$_r(b, +, +)$, the replica decides $b$ (ln 20). The replica then queries the common coin (ln 21) and enters the next round (ln 22).

Starting from round $r = 1$, replica do not have to broadcast bval$_r()$ messages any more. Instead, at ln 11, a replica checks whether it has received a valid conf$_{r-1}(b)$ message with a valid threshold signature sig for $t_{2}$.

Figure 16: CKS ABA [17]. The code for replica $p_i$.

upon event propose(sid, $v_i$)
01 $r \leftarrow 0$
02 $est_0 \leftarrow v_i$
03 start round 0
05 round $r$
06 if $r = 0$
07 broadcast bval$_r(est_0, t_{1.share})$
08 upon receiving $2f + 1$ bval$_r()$ with oals
09 $v \leftarrow \text{majority(oals)}$, sig $\leftarrow t_{1.combine}(shares)$
10 else
11 if exists conf$_{r-1}(b, sig)$ s.t. $\text{sig}$ is a valid threshold signature
12 $v \leftarrow b$
13 else $v \leftarrow s_{r-1}$
14 broadcast aux$_r(v, sig, t_{2.share})$
15 upon receiving $n - f$ aux$_r()$ messages where oals is a set of values carried by these messages
16 if oals $= \{b\}$, $v \leftarrow b$, sig $\leftarrow t_{2.combine}(shares)$
17 else $v \leftarrow \perp$, sig $\leftarrow aux_r()$ messages
18 broadcast conf$_r(v, sig, t_{2.share})$
19 upon receiving $n - f$ conf$_r()$ messages where oals is a set of values carried by these messages
20 if oals $= \{b\}$, decide(sid, $b$)
21 $s_r \leftarrow \text{coin}_r$
22 $r \leftarrow r + 1$

upon event propose(sid, $v_i$, $\pi$)
01 $r \leftarrow 0$
02 coin$_0 \leftarrow 1$
03 est$_0 \leftarrow v_i$
04 start round 0
05 round $r$
06 if $r = 0$
07 broadcast bval$_r(est_0, t_{1.share}, \pi)$
08 upon receiving bval$_r()$ with invalid $\pi$
09 discard the message
10 upon receiving $2f + 1$ bval$_r()$ with oals
11 $v \leftarrow \text{majority(oals)}$, sig $\leftarrow t_{1.combine}(shares)$
12 else
13 if exists conf$_{r-1}(b, sig)$ s.t. $\text{sig}$ is a valid threshold signature
14 $v \leftarrow b$
15 else $v \leftarrow s_{r-1}$
16 broadcast aux$_r(v, sig, t_{2.share})$
17 upon receiving $n - f$ aux$_r()$ messages where oals is a set of values carried by these messages
18 if oals $= \{b\}$, $v \leftarrow b$, sig $\leftarrow t_{2.combine}(shares)$
19 else $v \leftarrow \perp$, sig $\leftarrow aux_r()$ messages
20 broadcast conf$_r(v, sig, t_{2.share})$
21 upon receiving $n - f$ conf$_r()$ messages where oals is a set of values carried by these messages
22 if oals $= \{b\}$, decide(sid, $b$)
23 if $r = 0$, $s_r \leftarrow 1$
24 else $s_r \leftarrow \text{coin}_r$
25 $r \leftarrow r + 1$

Figure 17: The VABA construction based on CKS ABA protocol [16].

to be verified before a replica accepts the proposed value. In the specific construction, a vote for 1 has to be associated with $\pi$, a threshold signature generated externally. For a vote for 0, $\pi$ can simply be $\perp$. Namely, a bval$_r()$ message with an invalid $\pi$ will be discarded (ln 08-09). Second, the common coin in round 0 is set to 1 (ln 23).

RABA. We present a construction that converts CKS to a RABA protocol CKS-R, as shown in Figure 18. We make several changes. First, at ln 06-07, upon the $\text{repropose}(sid, 1)$ event, regardless of which round a replica is in, it broadcasts a bval$_{0}(1, t_{2.share})$ message where $t_{2.share}$ is a threshold signature share for the bval$_{0}()$ message. Second, the threshold signature scheme we use for the bval$_{r}()$ messages is a $(n, n - f)$ scheme (ln 10-12). In other words, each replica generates a threshold signature share $t_{2.share}$ in the bval$_{r}()$ step (ln 10). In ln 11-12, each replica collects $n - f$ bval$_{r}()$ messages with matching value to proceed to the next step. Third, the common coin of round 1 is set to 0 (ln 24-25).

Proof of Correctness for CKS-R. 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 $\bar{v}$, all replicas are able to receive $n - f$ bval$_{r}()$ and obtain a valid threshold signature. Each replica only sends aux$_r(1)$ with
Among the messages, at least one includes a valid threshold signature. In other words, at least one correct replica that terminates decides $v$. Similarly, every replica only sends a 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 $v$.

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

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

Second, if $p_j$ decides in round $r'$, it receives $n - f$ conf$_{r'}(v')$. In other words, at least $n - f$ replicas receive $n - f$ aux$_r(v')$ messages. Among the messages, at least one includes a conf$_{r-1}(v')$ with a valid threshold signature. In other words, at least $n - f$ replicas sent conf$_{r-1}(v')$ in round $r' - 1$. Recursively, in round $r$, at least $n - f$ replicas sent conf$_r(v')$ in round $r$. Therefore, a correct replica sends both conf$_r(v)$ and 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 $n - f$ bval$_0(0)$. This is because correct replicas do not repropose 0 if they propose 1. If a replica receives $n - f$ bval$_0(0)$ messages and assuming there are $f$ faulty replicas, at least $n - 2f$ correct replicas have sent bval$_0(0)$. Since $n \geq 3f + 1$, $n - 2f \geq f + 1$. This is a violation that at least $f + 1$ replicas propose 1 since a correct replica will not repropose 0.

Also, each replica may repropose 1. In other words, all correct replicas will eventually receive $n - f$ bval$_0(1)$ messages. In this case, all replicas will receive $n - f$ 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 $n - f$ conf$_0(0)$ messages, which is impossible since it requires $n - f$ 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 $n - f$ conf$_{r-1}(0)$. In other words, at least $n - f$ replicas broadcast aux$_{r-1}(1)$ and obtain a valid threshold signature from round $r - 2$. Recursively, in round 0, at least $n - f$ replicas send conf$_0(0)$. It is straightforward to see that in the first step of round 0, at least $n - f$ replicas sent 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 $Q_1 \neq \emptyset$ and $Q = Q_1 \cup Q_2$, then every correct replica eventually terminate.

**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 $n - f$ bval$_0(0)$ and $n - f$ bval$_0(0)$. Each replica, however, only sends aux$_r(1)$ message once with one value. In the next step, a replica makes a decision regardless of the values received from aux$_r(1)$ messages. Similarly, a replica can proceed after it receives $n - f$ conf$_r(1)$ 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 $n - f$ conf$_r(1)$ messages with a matching value; B) At least one but fewer than $f + 1$ correct replicas collect $n - f$ conf$_r(1)$ messages with a matching value; C) At least $f + 1$ correct replicas collect $n - f$ conf$_r(1)$ messages. In case A, all correct replicas use the common coin value as $v$ and each replica sends aux$_r(0)$. It is impossible that another correct replica sends aux$_r(0)$ with a valid threshold signature since it requires $n - f$ valid conf$_{r-1}(v')$ messages. Therefore, all replicas will receive $n - f$ aux$_r(0)$, send conf$_r(0)$, and decide $v$.

- B) Some replicas may send aux$_r(0)$ while other replicas use the common coin value. The probability that the correct 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.

- C) All correct replicas will receive at least one valid conf$_r(0)$ in round $r + 1$. This is because each replica collects $n - f$ bval$_{r+1}(0)$ messages. Among the replicas that have sent bval$_{r+1}(0)$ messages, at least $n - 2f$ are correct. We also know that at least $f + 1$
replicas sent \( \text{conf}_r(v) \). There are in total \( n - f + 1 \) correct replicas that sent \( \text{bval}_{r+1}(v) \) and \( \text{conf}_r(v) \). Therefore, if at least one correct replica fails to receive a valid \( \text{conf}_r(v) \) message, at least one correct replica collects \( n - f \) \( \text{conf}_r(v) \) messages but does not send a \( \text{bval}_{r+1}(v) \) with a valid signature, a contradiction.

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

G.2 Converting Crain’s ABA Protocols to RABA Protocols

We show how to convert both CrainL and CrainH [25] to RABA. As our measures to convert both protocols are similar, in this section, we focus on CrainH, the one that relies on high threshold common coins.

CrainH and CrainH-R. As illustrated in Figure 19, each round of CrainH involves an optional dispersal phase and an agreement phase. The dispersal phase (in 09, which calls operations in ln 24-31) is similar to the \( \text{bval}_r() \) phase in Pillar and Cobalt, where replicas also disperses a value if it has not previously done so but receives the value from \( f + 1 \) replicas (in 28-29). Upon receiving \( n - f \) messages with the same value \( v \), the replica updates its local parameter \( \text{bin}_ptr_r[v] \) to true (In 09 and in 30-31). In this phase, some correct replicas may disperse their values and some may not, depending on a boolean flag \( \text{support}_coin \). The \( \text{support}_coin \) value is set to false by default in the first round and updated from the last round otherwise. In the agreement phase, replicas either broadcasts an aux_r(1) or an aux_r(0) message and only accepts an aux_r(v) if \( \text{bin}_ptr_r[v] \) is true. After receiving \( n - f \) aux_r() messages, replicas query the common coin protocol coi_r and obtain s_r (in 15-16). There are three cases: 1) If a replica receives \( n - f \) aux_r(b) and \( b = s_r \), it decides b and sets \( \text{support}_coin \) to true (In 17-19); 2) If a replicas receives both aux_r(0) and aux_r(1), it sets \( \text{support}_coin \) to true (In 20-21); 3) Otherwise, the replica sets \( \text{support}_coin \) to false (In 22-23). In all the cases, the replica continues to the next round.

Figure 19: CrainH: Crain’s ABA with high threshold common coins ([25, 2nd algorithm]). The code for replica \( r_i \).

<table>
<thead>
<tr>
<th>line</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>\text{upon event} \text{propose}(sid, v_i)</td>
</tr>
<tr>
<td>02</td>
<td>( r \leftarrow 0 )</td>
</tr>
<tr>
<td>03</td>
<td>\text{coin}_0 \leftarrow v_i</td>
</tr>
<tr>
<td>04</td>
<td>\text{support}_coin \leftarrow \text{false}</td>
</tr>
<tr>
<td>05</td>
<td>\text{bin}_ptr[r]_i[0] \leftarrow \text{S_broadcast} EST[1](v_i, \text{false})</td>
</tr>
<tr>
<td>06</td>
<td>start the loop</td>
</tr>
<tr>
<td>07</td>
<td>\text{round } r</td>
</tr>
<tr>
<td>08</td>
<td>( r \leftarrow r + 1 )</td>
</tr>
<tr>
<td>09</td>
<td>\text{bin}_ptr[r]_i[s_r-1] \leftarrow \text{S_broadcast} EST[r](s_r-1, \text{support}_coin)</td>
</tr>
<tr>
<td>10</td>
<td>\text{wait until bin}_ptr[r]_i[0] = \text{true or bin}_ptr[r]_i[1] = \text{true}</td>
</tr>
<tr>
<td>11</td>
<td>if \text{support}_coin, ( w \leftarrow s_r-1 )</td>
</tr>
<tr>
<td>12</td>
<td>else if \text{bin}_ptr[r]_i[0] = \text{true}, ( w \leftarrow 0 )</td>
</tr>
<tr>
<td>13</td>
<td>else if \text{bin}_ptr[r]_i[1] = \text{true}, ( w \leftarrow 1 )</td>
</tr>
<tr>
<td>14</td>
<td>broadcast aux_r(w)</td>
</tr>
<tr>
<td>15</td>
<td>\text{upon receiving } n - f \ aux_r() with ( \text{vals} ) such that for every value ( v \in \text{vals}, \text{bin}_ptr[r]_i[v] = \text{true} )</td>
</tr>
<tr>
<td>16</td>
<td>( s_r \leftarrow \text{coin}_r )</td>
</tr>
<tr>
<td>17</td>
<td>if ( \text{vals} = { b } ) and ( b = s_r )</td>
</tr>
<tr>
<td>18</td>
<td>( v \leftarrow b, \text{support}_coin \leftarrow \text{true} )</td>
</tr>
<tr>
<td>19</td>
<td>\text{decide}(sid, b)</td>
</tr>
<tr>
<td>20</td>
<td>else ( \text{vals} = { 0, 1 } )</td>
</tr>
<tr>
<td>21</td>
<td>\text{support}_coin \leftarrow \text{true}</td>
</tr>
<tr>
<td>22</td>
<td>else</td>
</tr>
<tr>
<td>23</td>
<td>\text{support}_coin \leftarrow \text{false}</td>
</tr>
<tr>
<td>24</td>
<td>\text{operation } S_{\text{Broadcast}} \text{TAG}(v_i, \text{should_broadcast}[i](\text{utility function})</td>
</tr>
<tr>
<td>25</td>
<td>\text{should_broadcast} \leftarrow \text{false}</td>
</tr>
<tr>
<td>26</td>
<td>if \text{should_broadcast} = \text{true, then } \text{broadcast TAG}_S\text{VAL}(v_i)</td>
</tr>
<tr>
<td>27</td>
<td>\text{return } s_\text{value}_r</td>
</tr>
<tr>
<td>28</td>
<td>\text{upon receiving } f + 1 \text{TAG}_S\text{VAL}(v)</td>
</tr>
<tr>
<td>29</td>
<td>\text{broadcast TAG}_S\text{VAL}(v)</td>
</tr>
<tr>
<td>30</td>
<td>\text{upon receiving } 2f + 1 \text{TAG}_S\text{VAL}(v)</td>
</tr>
<tr>
<td>31</td>
<td>\text{should_broadcast} \leftarrow \text{true}</td>
</tr>
</tbody>
</table>

Figure 20: CrainH-R: The RABA construction based on CrainH. The code for replica \( r_i \).

<table>
<thead>
<tr>
<th>line</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>\text{upon event} \text{propose}(sid, v_i)</td>
</tr>
<tr>
<td>02</td>
<td>( r \leftarrow 0 )</td>
</tr>
<tr>
<td>03</td>
<td>\text{coin}_0 \leftarrow v_i</td>
</tr>
<tr>
<td>04</td>
<td>\text{support}_coin \leftarrow \text{false}</td>
</tr>
<tr>
<td>05</td>
<td>\text{bin}_ptr[r]_i[0] \leftarrow \text{S_broadcast} EST[1](v_i, \text{false})</td>
</tr>
<tr>
<td>06</td>
<td>start the loop</td>
</tr>
<tr>
<td>07</td>
<td>\text{upon event} \text{repropose}(sid, v_i)</td>
</tr>
<tr>
<td>08</td>
<td>\text{broadcast EST}[1]\text{S}_{\text{VAL}}(v_i)</td>
</tr>
</tbody>
</table>

[reproposal event]
In the first two cases, since support_coin is set to true, a replica will not disperse the values in the dispersal phase of the next round. In the third case, the replica disperses its value again in the next round.

Two ideas are crucial for the correctness of CrainH. First, a high threshold common coin is used instead of a regular common coin, ensuring that the values chosen by at least \( f + 1 \) correct replicas cannot be manipulated by an adversary. Second, the dispersal phase becomes optional such that a network scheduler cannot make correct replicas change their votes. The optional dispersal phase also makes the protocol enjoy a fast path starting from the second round.

We make the following changes to convert CrainH to a RABA protocol CrainH-R, as shown in Figure 20. Similar to our approach presented in this paper, we achieve this by revising the first round (round 1 in CrainH) only. In particular, upon reproposing 1, regardless of which round a replica is in, it disperses a EST[1], S_VAL(1) message (ln 08). This message is called in the S_broadc ast function for the dispersal phase (ln 29-36). Second, upon proposing 1, a replica immediately sets bin_prt[1] to true (ln 12-13). In other words, the replica will directly broadcast an aux_r(1) message. Finally, the common coin value of the first round is set to 1 (ln 20).

The first change ensures that, if \( f + 1 \) correct replicas propose 1, every correct replica will receive at least one aux_r(1) such that no correct replica will decide 0. The second change ensures that all correct replicas can terminate the protocol if the predicate for biased termination is satisfied. Finally, the third change ensures that if at least \( f + 1 \) correct replicas propose 1, no correct replica will use 0 to enter round 1.

**CrainL and CrainL-R.** We now sketch CrainL and describe how to convert it to a RABA protocol called CrainL-R. CrainL has a SBV_broadc ast function which involves a bval[0](1) phase (dispersal phase) and an aux_r(1) phase (agreement phase). In each round, the SBV_broadc ast function is repeated twice. In the first time the function is called, every replica \( p_i \) inputs one value \( est_i \) and obtains a set \( view_i[r_i, 0] \). The set may include one value \( v \) or both 0 and 1. If \( view_i[r_i, 1] \) includes only one value \( v \) and \( p_i \) inputs the second SBV_broadc ast instance. Otherwise, \( p_i \) inputs \( \perp \). Similarly, at the end of the second SBV_broadc ast instance, each replica obtains a set \( view_i[r_i, 2] \). The set may include one value \( v \), both \( v \) and \( \perp \), or only \( \perp \). In the first case, \( p_i \) decides \( v \). In the second case, \( est_i \) (for the next round) is set as \( v \). In the third case, \( est_i \) is set as the common coin. In CrainL, the common coin protocol is queried right after the second SBV_broadc ast instance completes. In the revised version [28] that has the good-case-coin-free property, the common coin can be queried if \( p_i \) ensures that the first case does not apply.

We can convert CrainL to CrainL-R by making the following changes. First, upon reproposing 1, regardless of which round a replica is in, it calls the first SBV_broadc ast instance and inputs 1. Second, upon proposing 1, each replica can directly sends all the messages for both SBV_broadc ast instances, and sets \( view_i[r_i, 1] \) and \( view_i[r_i, 2] \) to 1. Finally, the common coin value it set to 1 for the first round.

**G.3 Converting Cobalt ABA to RABA**

We can also convert Cobalt ABA [37] to a RABA protocol. We also make three changes. First, upon proposing 1, each replica starts round 0 (the first round) and broadcasts a bval_0(1). Each replica also immediately adds 1 to \( bin\_values_0 \) and broadcasts an aux_0(1) message. Second, upon reproposing 1, regardless of which round a replica is in, each replica broadcasts bval_1(1). The replica adds 1 to \( bin\_values_0 \) if it has not done so already. Furthermore, the replica broadcasts an aux_1(1) if it has not broadcast any aux_1(1) message. Finally, the common coin for round 0 is set to 1. The first change and the third one ensure that no correct replica will ever use 0 as \( est_1 \) if at least \( f + 1 \) correct replicas propose 1. The second change 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 [20] 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 (CKPS) [16]. Correia, Neves, and Verissimo (CV) show that multi-valued consensus (without using signatures) is also equivalent to atomic broadcast [22].

**Parallel ABA.** Some works (directly) 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 \( f = O(\sqrt{n}) \) failures only [10]; Ben-Or and El-Yaniv provide a constant expected time protocol [11]. The protocol, unfortunately, would yield a prohibitively expensive BFT protocol with \( O(n^4) \) message and communication complexity.

**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 [16], atomic broadcast (e.g., [12, 16, 29, 35, 38]), anonymous vector consensus [14], 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 [15] 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 [13]. RITAS [39], for instance, uses local-coin ABA to build asynchronous atomic broadcast using the protocol of Correia, Neves, and Verissimo [23] 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 [30]. In contrast, asynchronous BFT does not rely on any timing assumptions. It is shown that even for partially synchronous BFT protocols focusing on robustness [7, 21], their performance may drop 78%-99% [8] during failures or attacks. Moreover, partially synchronous protocols may achieve zero throughput with an adversarial network scheduler [38].
Asynchronous BFT protocols are intrinsically robust against performance and denial-of-service (DoS) attacks.

**Asynchronous BFT with quantum safety.** DAG-Rider [32] 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 [26] implements a highly efficient asynchronous BFT protocol that significantly outperforms existing protocols, 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 underlying technique, however, seems to work for all asynchronous BFT protocols known, as all such protocols require transmitting bulk data using reliable communication primitives.

**Sub-optimal resilience.** Assuming sub-optimal resilience, MiB [36] implements asynchronous BFT protocols with lower latency and higher throughput based on the BEAT library.

**Mir-BFT.** Mir-BFT [44] allows multiple leaders (replicas) to propose request batches independently, just as in asynchronous BFT protocols, but it works in partially synchronous environments.

**Adaptive security.** While many asynchronous BFT protocols achieve adaptive security, EPIC [35] is the first adaptively secure BFT protocol implemented. In the adaptive security model, the adversary can choose to corrupt replicas at any moment during the execution of the protocol. Prior protocols, such as SINTRA, HoneyBadgerBFT, BEAT, and Dumbo, 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 BEAT 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 [9]. 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.

While one can make protocols in the BKR paradigm adaptively secure as shown in EPIC [35], 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 [9].

**Communication-efficient RBC protocols.** Some recent constructions have improved the RBC protocols asymptotically or concretely [5, 27]. These protocols may benefit practical BFT protocols implemented in the bandwidth usage.