WaterBear: Practical Asynchronous BFT Matching Security Guarantees of Partially Synchronous BFT

Haibin Zhang
bchainzhang@aliyun.com
Beijing Institute of Technology

Sisi Duan*
duansisi@tsinghua.edu.cn
Tsinghua University, Zhongguancun Laboratory

Boxin Zhao*
zhaobx@zgclab.edu.cn
Zhongguancun Laboratory

Liehuang Zhu*
liehuangz@bit.edu.cn
Beijing Institute of Technology

Abstract

Asynchronous Byzantine fault-tolerant (BFT) protocols assuming no timing assumptions are inherently more robust than their partially synchronous counterparts, but typically have much weaker security guarantees.

We design and implement WaterBear, a family of new and efficient asynchronous BFT protocols matching all security guarantees of partially synchronous protocols. To achieve the goal, we have developed the local coin (flipping a coin locally and independently at each replica) based BFT approach—one long deemed as being inefficient—and designed more efficient asynchronous binary agreement (ABA) protocols and their reproposable ABA (RABA) versions from local coins. Our techniques on ABA and RABA are of independent interests and also allow us to build more efficient ABA protocols from common coins (distributively generating the same random coins for all replicas), helping improve various other protocols such as distributed key generation and BFT assuming trusted setup.

We implemented in total five BFT protocols in a new golang library, including four WaterBear protocols and BEAT. Via extensive evaluation, we show that our protocols are efficient under both failure-free and failure scenarios, achieving at least comparable or superior performance to BEAT with much weaker security guarantees. Specifically, the most efficient WaterBear protocol consistently outperforms BEAT in terms of all metrics. For instance, when the number of replicas is 16, the latency of our protocol is about 1/8 of that of BEAT and the throughput of our protocol is 1.23x that of BEAT.

Our work pushes the boundaries of asynchronous BFT, showing the strongest security levels that we know of and high performance can co-exist.

1 Introduction

Byzantine fault-tolerant state machine replication (BFT), a technique traditionally used to build mission-critical systems, has nowadays been the standard model for permissioned blockchains [9, 18, 35, 36, 60, 63, 64] and is used in various ways in hybrid blockchains. Due to their inherent robustness against performance and DoS attacks, asynchronous BFT protocols—relying on no timing assumptions—have been receiving significant attention [11]. While one line of works focuses on performance [28, 44, 45, 53], some other works aim at improving their "security." For instance, BEAT [33], PACE [66], and FIN [34] eliminated the less-established pairing assumption in these protocols; EPIC and HALE aimed at providing adaptive security [49, 69]; DAG-Rider strived to achieve quantum safety (though not quantum liveness) [47]; recent works studied how to avoid trusted setup [4, 30, 48, 67].

Table 1 summarizes the security levels that can be achieved for asynchronous BFT protocols implemented. The situation is in sharp contrast to their partially synchronous BFT counterparts (relying on timing assumptions): for example, the classic PBFT protocol [22]—based on authenticated channels only—easily achieves all the properties listed in the table. It is thus our goal to design and implement *practical* asynchronous BFT protocols achieving all these properties in Table 1—the same security guarantees as in partially synchronous BFT.

1.1 Background on Security Guarantees

Authenticated channels only vs. no PKC vs. quantum security. When designing practical fault-tolerant and cryptographic protocols, it is vital to use weak assumptions to reduce the attack surface and thus achieve strong guarantees. Arguably the "minimal" (and most frequently used) assumption is the point-to-point authenticated channel. A slightly stronger assumption is to use symmetric cryptography only—no public-key cryptography (PKC). Symmetric cryptography primitives, such as message authentication codes (MACs) and hash functions—with appropriately chosen parameters—are believed to defend against quantum adversaries that may leverage quantum mechanical phenomena to solve the problem intractable for conventional computers (see NIST reports [23]).

In the partial synchrony setting, the classic PBFT protocol [22] is the first one relying on no PKC—authenticated channels and hash functions only. (The protocol in Castro's PhD thesis [21] and Cachin's formulation for PBFT [15] as-

*Corresponding author

	authenticated channel only	no pkc	quantum secure	no trusted setup	adaptive security	high WAN throughput
SINTRA [16]						
RITAS [54]		√	√	√	√	
HoneyBadger [53]; BEAT [33]						√
Dumbo family [40,44,45]						√
EPIC [49]					√	√
Tusk [28]; Bullshark [42]						√
PACE [66]						√
SodsBC [31]		√	√			√
WaterBear-QS (this work)		√	√	√	√	√
WaterBear (this work)	√	√	√	√	√	√

Table 1: Comparison of efficient asynchronous BFT systems.

sume authenticated channels and hash functions too.) They can be modified to assume authenticated channels only, as commented in PBFT "It is also possible to modify the algorithm (PBFT) not to use a cryptographic hash function by replacing the hash of a message by the value of the message." Recently, Stern and Abraham proposed a variant of HotStuff relying on authenticated channels only and achieving improved message complexity [62]. All these partially synchronous BFT protocols achieve quantum security.

In contrast, we do not know how to build asynchronous BFT with quantum security, asynchronous BFT without PKC, or asynchronous BFT with authenticated channels only.

Adaptive security vs. static security. Depending on how the adversary decides to corrupt replicas, there are two types of corruptions: static corruptions and adaptive corruptions. A static adversary is restricted to choose its set of corrupted replicas at the start of the protocol. An adaptive adversary can choose its set of corrupted replicas at any moment during the execution of the protocol, based on the information it has accumulated. There is a strong separation result that statically secure protocols are not necessarily adaptively secure [19,27]. Classic partially synchronous protocols such as PBFT naturally achieve adaptive security, while all asynchronous BFT protocols implemented (except RITAS [54] and EPIC [49]) achieve static security and even for RITAS and EPIC, they achieve adaptive security with significant performance penalty.

1.2 Why Matching Security Guarantees of Partially Synchronous BFT Hard?

Unlike partially synchronous BFT protocols, asynchronous BFT protocols must be randomized to achieve liveness (due to the celebrated FLP impossibility result [39]). Existing asynchronous BFT protocols rely critically on cryptographic common coin protocols (a distributed object that generates the same random coins to all replicas) which are inefficient if no trusted setup is assumed (see Sec. 2 for detailed discussion). Even if relaxing to quantum security, we currently lack efficient common coin instantiations from quantum secure primitives (e.g., lattices). Note while SodsBC is a quantum-secure BFT protocol, it directly relies on trusted setup to generate the common coins and thus bypasses the core problem of generating common coins efficiently [31].

Meanwhile, achieving stronger security in asynchronous BFT typically comes with a higher cost. For instance, adding adaptive security in [49] makes the original BFT system much slower. We must guarantee that ensuring these properties altogether would not incur too much overhead.

1.3 Our Approach

Reducing the problem to local coin RABA then to ABA. Instead of using common coins, we revisit the local coin based BFT approach that has been long viewed as being inefficient. In the local coin based approach, replicas need to independently and locally flip coins and existing protocols terminate in exponential expected rounds and fail to scale [25,54].

We thus take a detour and develop the PACE BFT framework [66] using authenticated channels only and achieving quantum security. (Recall existing instantiations in PACE use trusted setup and threshold cryptography, achieving static security only.) PACE devises a variant of asynchronous binary Byzantine agreement (ABA) called reproposable ABA (i.e., RABA). Compared to ABA, RABA has additional properties, allowing all RABA instances to run in parallel and hence improving system throughput. It is also shown that some ABA protocols can be efficiently converted to RABA protocols. Crucially, RABA in PACE enables a fast path for consensus. We observe that while PACE was designed with common coin based RABA, the protocol, even with local coin based RABA, can—on average—terminate within a single RABA round with high probability. As most RABA instances will terminate in one round, the exponential rounds in local coin based ABA is no longer a major efficiency obstacle.[1] Now the per-round complexity of RABA protocols becomes critical. Our strategy is to *reduce asynchronous BFT to RABA with local coins and then to ABA with local coins* and then *improve the per-round complexity of them*.

Improving the per-round complexity of (R)ABA. As reported in almost all asynchronous BFT systems [33,45,66], ABA is their major performance bottleneck. It is shown that

[1] Jumping ahead, we experimentally demonstrate through latency-breakdown and robustness tests to validate the claim.

protocol	reference implementation	RBC	RABA	building blocks
WaterBear	WaterBear-C	Bracha's RBC [14]	Cubic-RABA (this paper)	MAC and hash
	WaterBear-Q	Bracha's RBC [14]	Quadratic-RABA (this paper)	MAC and hash
WaterBear-QS	WaterBear-QS-C	CT RBC [17]	Cubic-RABA (this paper)	MAC
	WaterBear-QS-Q	CT RBC [17]	Quadratic-RABA (this paper)	MAC

Table 2: WaterBear-QS and WaterBear instantiations. As in PACE, both WaterBear-QS and WaterBear have a fast path allowing them to terminate in $O(\log n)$ time. As shown in PACE, the probability of triggering fast paths is high. WaterBear-C and WaterBear-QS-C have $O(n^4)$ messages on average due to the usage of Cubic-RABA, while WaterBear-Q and WaterBear-QS-Q have $O(n^3)$ messages on average due to the usage of Quadratic-RABA—matching those of HoneyBadger, BEAT, and PACE.

ABA (local coins)	messages/round	steps/round
Bracha's ABA [14]	n^3	9 to 12
Cubic-ABA (this work)	n^3	5 to 7
Quadratic-ABA (this work)	n^2	4 or 5

Table 3: Local coin based ABA protocols with optimal resilience. We consider the messages and steps in each round. Messages/round and steps/round denote number of messages and steps among all replicas per round.

the concrete steps *per round* of ABA protocols are vital to the performance of asynchronous BFT: even a single step improvement in ABA, the resulting BFT protocol could be improved by, say, 2x [66].

To our knowledge, only two local coin based ABA protocols have been proposed: Ben-Or's ABA [11] assuming $n > 5f$, and Bracha's ABA [13] with $n > 3f$ (the most efficient protocol for nearly three decades). Bracha's ABA, unfortunately, has a large number of steps (12 steps) and $O(n^3)$ messages per round [13]. (The situation is in sharp contrast to ABA assuming common coins which has 3 steps and $O(n^2)$ messages per round.) Our main technical contributions are indeed efficient local coin based ABA and RABA protocols with improved message complexity and reduced number of steps.

1.4 Our Contributions

1.4.1 Technical Contributions

Efficient local coin based ABA. Table 3 shows two novel local coin based ABA protocols that we introduce in the paper: Cubic-ABA and Quadratic-ABA. Cubic-ABA is easy to understand and implement, and can be viewed as an optimized version of Bracha's ABA. Cubic-ABA has 7 steps per round in the worst case, while Bracha's ABA uses 12 steps, almost doubling the number of steps of Cubic-ABA. In contrast, Quadratic-ABA adopts a novel design, having 4 or 5 steps per round only and being the first local coin based ABA with $O(n^2)$ messages per round. In particular, Quadratic-ABA admits a fast (coin-free) allowing the protocol to terminate in a single step in the optimistic mode.

Tackling a subtle liveness issue for RABA. We go on to design Cubic-RABA and Quadratic-RABA based on Cubic-ABA and Quadratic-ABA, respectively. Unlike prior transformations following a generic approach in [66], we identify and tackle a subtle liveness problem when transforming Quadratic-ABA to Quadratic-RABA. The issue that we identify demonstrates the subtlety of transforming ABA to RABA, and once again underlines the importance of a full proof when designing Byzantine-resilient protocols.

ABA from weak common coins and perfect common coins. The techniques we introduce for Quadratic-ABA are of independent interests, allowing us to obtain CC-ABA that works for both weak common coins and perfect common coins. Here weak common coins mean all correct replicas output 0 and 1, both with probability $1/d$, where d is a constant and $d \geq 2$. If $d = 2$, weak common coins become perfect common coins. In both cases, CC-ABA compares favorably with existing protocols. CC-ABA with weak coins can be used to improve the distributed key generation protocol [4] and VABA protocols [41, 51], while CC-ABA with perfect coins can be used to improve various BFT protocols such as PACE and Dumbo [45], and the recent distributed key generation protocol requiring the good-case-coin-free property [30].

1.4.2 Practical Contributions

The WaterBear family of BFT protocols. Table 2 summarizes the characteristics of WaterBear protocols. We use Cubic-RABA to build WaterBear-C and Quadratic-RABA to build WaterBear-Q. WaterBear has *all* desirable properties a BFT protocol one could think of, being optimally resilient, using authenticated channels only, achieving quantum security and adaptive security, and not relying on trusted setup—matching the security guarantees of the classic PBFT protocol. We comment that WaterBear does not attain information-theoretic (IT) security, as we use HMAC that is not IT-secure.

We also build WaterBear-QS using authenticated channels and hash functions and achieving quantum security for both safety and liveness properties. Similar to WaterBear, WaterBear-QS family also consists of two protocols: WaterBear-QS-C and WaterBear-QS-Q, quantum secure versions of WaterBear-C and WaterBear-Q, respectively.

A new BFT platform. Starting from HoneyBadger, existing asynchronous BFT protocols, including BEAT, Dumbo, and EPIC, use the HoneyBadger programming framework using Python 2.7 (end of life and end of support on January 1, 2020). We instead build a new platform using Golang that is more modular and developer-friendly than existing ones. Our platform currently supports WaterBear-C, WaterBear-Q, WaterBear-QS-C, WaterBear-QS-Q, and BEAT (one of the most efficient open-source asynchronous BFT) [1,33]. Due to its modularity, the library only contains about 11,000 LOC.

Large-scale experiments and robustness evaluation. With a 61-instance deployment on Amazon EC2, we show our protocols offer comparable performance as the state-of-the-art asynchronous BFT protocols, while achieving much stronger security. We also design and evaluate various failure and attack scenarios, showing all our protocols are highly robust during failures and attacks. Specifically, one of our protocols, WaterBear-QS-Q, consistently outperforms BEAT (with much weaker security guarantees); for instance, when $n = 16$, the latency of WaterBear-QS-Q is about 1/8 that of BEAT and the throughput of WaterBear-QS-Q is 1.23x that of BEAT. The peak throughput of WaterBear-QS-Q (as n grows larger) is about 1.47x that of BEAT.

1.5 Paper Organization

In what follows, we first discuss related work (Sec. 2) and describe the system model and definitions (Sec. 3). Then we present Cubic-ABA and Quadratic-ABA from local coins (Sec. 4), their RABA counterparts—Cubic-RABA and Quadratic-RABA (Sec. 5), and WaterBear BFT protocols (Sec. 6). Last, we present our evaluation results (Sec. 7) before concluding the paper (Sec. 8).

1.6 The Proceeding Version

The paper is the full paper of our proceeding version [68] appearing at Usenix Security 2023. The full paper contains many new theoretical and experimental results, including ABA protocols from perfect common coins and weak common coins.

2 Related Work

ADKG. A line of recent works studied how to eliminate trusted setup by using asynchronous distributed key generation (ADKG) [4, 30, 48, 67]. Even if using ADKG in existing asynchronous BFT protocols, they would neither achieve quantum security nor security with no PKC.

Adaptive vs. static security for BFT. Most asynchronous BFT protocols implemented, including SINTRA, HoneyBadgerBFT, BEAT, and Dumbo, defend against static adversary only. These protocols rely critically on efficient but statically secure threshold cryptography. EPIC is an asynchronous BFT that uses adaptively secure threshold pseudorandom function (PRF) to achieve adaptive security but is not as efficient as its statically secure counterparts. RITAS [54] contains an adaptively secure BFT protocol, but due to inefficient local coin based ABA, it is less efficient than other protocols in large-size networks. The situation for asynchronous environments is in sharp contrast to that of partially synchronous protocols, most of which attain adaptive security [8, 22, 24, 32, 43, 46, 61].

Quantum safety (but no quantum liveness). A BFT protocol is quantum secure if its safety is quantum resistant (quantum safety) and its liveness is quantum resistant (quantum liveness) [47]. DAG-Rider [47] achieves quantum safety. Moreover, the BKR protocol and their descendants (e.g., HoneyBadger [53], MiB [50], PACE [66]) achieve quantum safety if using techniques from EPIC [49]. All these protocols, however, do not achieve quantum liveness. Tusk [28] and Bullshark [42] are variants of DAG-Rider; they extensively use signatures and hashes and achieve neither quantum safety nor quantum liveness.

Byzantine agreement and common coins. Byzantine agreement (BA) is a central tool for both distributed computing and cryptography. The condition $n \geq 3f+1$ is both necessary and sufficient for both synchronous and asynchronous BA protocols [58]. The celebrated impossibility result of Fischer, Lynch, and Paterson [39] implies that a randomized BA protocol must have non-terminating executions. ABA protocol may thus be either $(1-\varepsilon)$-terminating, where correct replicas terminate the protocol with an overwhelming probability, or almost-surely terminating, where replicas terminate with probability 1. For both types, we review ABA protocols assuming authenticated channels only. Note there is no need to consider ABA with authenticated channels and hash functions, as the input to ABA is a binary value and we do not need to use hash functions to compress the input.

For almost-surely ABA terminating with probability 1, Ben-Or's ABA requires $n \geq 5f+1$ [11], while Bracha's ABA [13] achieves optimal resilience. The two protocols use local coins and require an exponential expected running time. Feldman and Micali propose a BA protocol having a constant expected running time in synchronous environments and extend it to build a polynomial-time ABA protocol requiring $n \geq 4f+1$ [38]. Abraham, Dolev, and Halpern [3] provide the first almost-surely ABA with polynomial efficiency (concretely, expected $O(n^2)$ time) and optimal resilience. Bangalore, Choudhury, and Patra [10] improve the expected running time of [3] by a factor of n.

For $(1-\varepsilon)$-terminating ABA, Canetti and Rabin [20] build an expected constant-round ABA protocol with optimal resilience. Patra, Choudhury, and Rangan [57] build a more efficient construction in terms of communication complexity.

Both types of ABA protocols follow the classic framework of Feldman and Micali [38] that reduces ABA to asynchronous verifiable secret sharing (AVSS). The framework uses AVSS to build common coins. (The original idea of using common coin for ABA is due to Rabin [59].) Unfortunately, the framework of using AVSS for common coins is prohibitively expensive. For instance, to build AVSS, the approach of Canetti and Rabin [20] needs to begin with an information checking protocol, then asynchronous recoverable sharing, then asynchronous weak secret sharing, and finally AVSS. The improved approach of Patra, Choudhury, and Rangan [57] remains complex, following the route of information-checking protocol, then asynchronous weak commitment, and then AVSS. Moreover, the transformation from AVSS to ABA is equally expensive, requiring running n^2 AVSS instances to generate a *single* (weak) coin. Patra, Choudhury, and Rangan [57] also propose an approach for sharing multiple secrets simultaneously. While such an approach is useful for building more efficient multi-valued BA (MBA), it is unknown if it would yield more efficient ABA protocols. While, for instance, the CNV framework [25] does use MBA, it may run $O(n)$ consecutive MBA instances (which is inefficient).

3 System Model and Definitions

3.1 System and Threat Model

This section describes the system model for distributed protocols in the paper, where f out of n replicas may fail arbitrarily (Byzantine failures). We assume point-to-point authenticated channels between each pair of replicas; some of our protocols additionally assume hash functions. The WaterBear BFT protocols (and subprotocols we use or we invent) have the following properties:

- **Optimal resilience**: The protocols in this work assume $f \leq \lfloor \frac{n-1}{3} \rfloor$, which is optimal. A (Byzantine) *quorum* is a set of $\lceil \frac{n+f+1}{2} \rceil$ replicas. For simplicity, we may assume $n = 3f+1$ and a quorum size of $2f+1$.
- **Asynchronous network**: We consider completely asynchronous systems making no timing assumptions on message processing or transmission delays. In contrast, partially synchronous systems assume that there exist an upper bound on message processing and transmission delays but the bound may be unknown to anyone [37].

  Designing asynchronous systems is challenging, because in asynchronous environments it is impossible to distinguish Byzantine faulty replicas from "slow" replicas. In particular, one cannot use timers or timeout to assist in the design of asynchronous systems.
- **No dealer/trusted setup**: We do not assume the existence of a trusted dealer or trusted setup. Neither do we assume there exists an interactive protocol for any public keys, reference strings, or public parameters.
- **Adaptive corruptions**: We consider adaptive adversary that can choose its set of corrupted replicas at any moment during the execution of the protocol, based on the information it has accumulated thus far (i.e., the messages observed and the states of previously corrupted replicas).

We may associate a protocol instance with a unique identifier *id*, tagging each message in the instance with *id*. If no ambiguity arises, we may omit the identifiers.

3.2 Definitions and Preliminaries

BFT. In a BFT protocol, a replica *a-delivers* (atomically deliver) *transactions*, each *submitted* by some client. The client computes a final response to its submitted transaction from the responses it receives from replicas. We consider the following properties:

- **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 transaction tx before *a-delivering* tx', then no correct replica *a-delivers* a transaction 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.

Below, we first introduce ABA and then its variant—RABA. Then we review the PACE framework using RBC and RABA.

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

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

RABA. Reproposable ABA (RABA) is a new distributed computing primitive introduced in PACE [66]. In contrast to conventional ABA protocols, where replicas can vote once only, RABA allows replicas to change their votes. Formally, a RABA protocol tagged with a unique identifier *id* is specified by *propose*$(id,\cdot)$, *repropose*$(id,\cdot)$, and *decide*$(id,\cdot)$, with the input domain being $\{0,1\}$. For our purpose, RABA is "biased towards 1." Each replica can propose a vote v at the beginning of the protocol. Each replica can propose a vote only once. A correct replica that proposed 0 is allowed to change its mind and repropose 1. A replica that proposed 1 is not allowed to repropose 0. If a replica repropose 1, it does so at most once. A replica terminates the protocol identified by *id* by generating a decide message. RABA (biased towards 1) satisfies the following properties:

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

Validity is slightly different from those for ABA. They are modified to accommodate the RABA syntax. Integrity is defined to ensure RABA decides once and once only.

Unanimous termination and biased termination are carefully introduced to help achieve RABA termination in certain scenarios. External operations would have to force the protocol to meet these termination conditions.

Biased validity in RABA requires that if $f+1$ correct replicas, not simply all correct replicas, propose 1, then a correct replica that terminates decides 1. The property guarantees the PACE framework has sufficient transactions delivered.

This paper introduces new RABA protocols from local coins.

RBC. In a Byzantine reliable broadcast (RBC) protocol [5–7, 14, 17, 29], a replica p first starts the protocol by executing *r-broadcast* with messages m, and all replicas terminate the protocol by executing *r-deliver* with message m. We consider the following properties:

- **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 the sender is correct, then m was previously *r-broadcast* by the sender.

This paper uses Bracha's broadcast [13] that assumes authenticated channels only and has a bandwidth of $O(n^2|m|)$, and uses CT RBC due to Cachin and Tessaro [17] that additionally uses hash functions (with output length λ) to reduce the bandwidth to $O(n|m| + \lambda n^2 \log n)$.

PACE framework. PACE uses RBC and RABA in a black-box manner to construct efficient asynchronous BFT. The framework allows all RABA instances to run in parallel, removing a well-known bottleneck in the original framework of Ben-Or, Kelmer, and Rabin [12]. Concretely, PACE has a RBC phase and a RABA phase; correct replicas can run the RABA phase in parallel once $n-f$ RBC instances have completed. PACE also provides a fast path for consensus, allowing the protocol to terminate using a single RABA round.

Steps, phases, and rounds. In asynchronous environments, the network delay is unbounded. To measure the latency of asynchronous protocols, we use the standard notion of *asynchronous steps* [20], where a protocol runs in x asynchronous steps if its running time is at most x times the maximum message delay between honest replicas during the execution.

We also use the notion of *phases* for ease of description, where a phase in a protocol consists of a fixed number of steps. When describing some of our protocols, we may divide a protocol into several phases, each of which has several steps.

In this paper, the notion of *rounds* is restricted to ABA protocols: an ABA protocol proceeds in rounds, where an ABA round consists of a fixed number of steps. For instance, local coin ABA protocols terminate in expected exponential rounds, while ABA assuming common coins (including CC-ABA we introduce in this paper) terminates in expected constant rounds. An ABA round may consist of several phases and each phase consists of several steps. In asynchronous ABA systems, replicas proceed in rounds and they might not be always in the same round, but each correct replica eventually terminates every round that it has participated in.

4 ABA from Local Coins

Summary of our results. The state-of-the-art local coin based ABA protocol, Bracha's ABA [13], has $O(n^3)$ messages and 12 steps in each round. We design two new ABA protocols from local coins, Cubic-ABA and Quadratic-ABA, with two goals in mind—being more efficient than Bracha's ABA and being compatible with RABA.

We begin with the simpler one, Cubic-ABA, that achieves the same message complexity as Bracha's ABA but has only 7 steps in each round. Cubic-ABA admits a clean and intuitive proof of correctness. We go on to suggest Quadratic-ABA on top of Cubic-ABA. Compared to Cubic-ABA, Quadratic-ABA reduces the messages from $O(n^3)$ to $O(n^2)$ and reduces the number of steps to 5 in each round. *The improvement is significant, allowing WaterBear to attain the same average message complexity in normal cases as PACE—$O(n^3)$.* Equally important, both ABA protocols can be modified for efficient RABA protocols.

As an important by-product, extending the idea of Quadratic-ABA and assuming the existence of (weak or perfect) common coins, we can present ABA protocols that have expected constant rounds and outperform the state-of-the-art protocols, as shown in Appendix G.

4.1 Cubic-ABA

Overview. Our motivation for Cubic-ABA is to reduce the number of parallel RBCs in Bracha's ABA from three to two. In particular, in each round, Bracha's ABA has three phases, where in each phase, replicas run n parallel RBCs. In total, Bracha's ABA has 12 steps and $O(n^3)$ messages per round. (We recall Bracha's ABA in Appendix A.) In Cubic-ABA, we replace the first two RBC phases with one or two steps of all-to-all broadcast so Cubic-ABA only has 5 to 7 steps.

The protocol. Figure 1 describes the pseudocode of Cubic-ABA and Figure 2 illustrates the workflow. Cubic-ABA uses the *broadcast* primitive (multicasting messages to all replicas) and the *r-broadcast* and *r-deliver* primitives of RBC. The protocol proceeds in rounds, beginning with round 0. Each round r consists of three phases. In the first phase, a replica p_i broadcasts a $\mathsf{pre\text{-}vote}_r(iv_r)$ message, where $\mathsf{pre\text{-}vote}$ is the message type, r is the round when the message was sent, and $iv_r \in \{0,1\}$ is the input value of p_i for round r (ln 07). At ln 08-09, if p_i receives $f+1$ $\mathsf{pre\text{-}vote}_r(v)$ for some $v \in \{0,1\}$ and has not previously broadcast $\mathsf{pre\text{-}vote}_r(v)$, it also broadcasts $\mathsf{pre\text{-}vote}_r(v)$.

At ln 10-14, p_i enters the second phase. If p_i receives $2f+1$ $\mathsf{pre\text{-}vote}_r(v)$, it adds v to its $bset_r$, a set consisting only 0 and 1 (ln 10-11). Letting v be the *first* value added to $bset_r$ for p_i, p_i broadcasts a $\mathsf{main\text{-}vote}_r(v)$ message (ln 12-14).

In the third phase, a correct replica p_i accepts a $\mathsf{main\text{-}vote}_r(v)$ message only if v has already been added lo-

```
01 initialization
02   r ← 0                                                  {round}
03 func propose(v_input)
04   iv_0 ← v_input                             {set input for round 0}
05   start round 0
06 round r
07   broadcast pre-vote_r(iv_r)                              {▷ phase 1}
08   upon receiving pre-vote_r(v) from f+1 replicas
09     if pre-vote_r(v) has not been sent, broadcast pre-vote_r(v)
10   upon receiving pre-vote_r(v) from 2f+1 replicas         {▷ phase 2}
11     bset_r ← bset_r ∪ {v}
12   wait until bset_r ≠ ∅
13     if main-vote_r() has not been sent
14       broadcast main-vote_r(v) where v ∈ bset_r
15   upon receiving n−f main-vote_r() such that for each received
     main-vote_r(b), b ∈ bset_r                              {▷ phase 3}
16     if there are n−f main-vote_r(v)
17       r-broadcast final-vote_r(v)
18     else r-broadcast final-vote_r(∗)
19   upon r-delivering n−f final-vote_r() such that for each
     final-vote_r(v), v ∈ bset_r; for each final-vote_r(∗), bset_r = {0,1}
20     if there are n−f final-vote_r(v)
21       iv_{r+1} ← v, decide v
22     else if there are f+1 final-vote_r(v)
23       iv_{r+1} ← v
24     else
25       iv_{r+1} ← Random()                          {obtain local coin}
26   r ← r+1
```

Figure 1: Cubic-ABA. The code for p_i. $v \in \{0,1\}$.

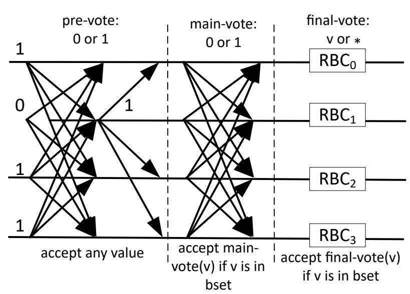


Figure 2: The workflow of Cubic-ABA.

cally to $bset_r$ (ln 15). If p_i has received $n-f$ main-vote$_r(v)$, p_i *r-broadcasts* a final-vote$_r(v)$ message (ln 16-17). Otherwise, p_i *r-broadcasts* final-vote$_r(*)$, where $*$ is a distinguished symbol that is neither 0 nor 1 (ln 18).

A correct p_i accepts a final-vote$_r()$ message, if one of the following two conditions holds (ln 23):

- For a final-vote$_r(v)$ message with $v \in \{0,1\}$, v has been added to $bset_r$ for p_i.
- For a final-vote$_r(*)$ message, $bset_r$ contains both 0 and 1.

Upon *r-delivering* $n-f$ valid final-vote$_r()$ messages, we distinguish three cases:

- Ln 20-21: If p_i *r-delivers* $n-f$ valid final-vote$_r(v)$ for the same $v \in \{0,1\}$, p_i decides v and uses v as iv_{r+1} to enter the next round. Each correct replica that decides in round r continues for one more round (up to the final-vote$_r()$ step) and terminates the protocol.
- Ln 22-23: If p_i *r-delivers* at least $f+1$ valid final-vote$_r(v)$ for some $v \in \{0,1\}$, p_i uses v as input for the next round.
- Ln 24-25: Otherwise, a replica generates a local random coin and uses it as input for the next round.

Analysis. In our approach, the first two phases of Cubic-ABA resemble those of common coin based ABA protocols [26, 52, 55, 56, 66], where we ask replicas to broadcast their values. In particular, the first phase ensures that all correct replicas eventually acknowledge the same set of values $bset_r$; the second phase ensures that no two correct replicas will vote for opposite values in the third phase, though one correct replica may vote for $b \in \{0,1\}$ and one may vote for $*$ (a distinguished vote). Accordingly, we do not have to rely on RBC for the first two phases, as our first two phases already guarantee that correct replicas will not vote for conflicting values for the third phase.

In the third phase, we need to ensure that if a correct replica receives $n-f$ final-vote$_r(v)$ and decide, any correct replica will propose v and eventually decide v. Note for the case where $f+1$ correct replicas vote for v and f correct replicas vote for $*$, we need to guarantee that if a correct replica receives $n-f$ final-vote$_r(v)$, any correct replica will receive at least $f+1$ final-vote$_r(v)$ and therefore vote for v in the following round. Thus, we rely on RBC, ensuring that all correct replicas eventually receive consistent values, even in the presence of Byzantine replicas. As we show in the proof, this is crucial for the agreement property.

Note that our protocol presented is asynchronous and replicas are not always in the same round. If a replica p_i is in round r and receives a message from p_j tagged by $r' > r$ (e.g., pre-vote$_{r'}(v)$), p_i can simply store the message and will process it after p_i enters round r'. As shown in the termination property (Theorem 7), each correct replica will proceed to the next round before it decides a value and all replicas will eventually decide in the same round.

As the number of n parallel RBC instances is 1 instead of 3, the number of steps is reduced from 12 to 7.

4.2 Quadratic-ABA

Overview. In Quadratic-ABA, we replace the only parallel RBC phase used in Cubic-ABA using a novel two-step all-to-all broadcast. The goal is to ensure that at the end of each round, if a correct replica receives $n-f$ matching votes for a value v, any correct replica will receive either $n-f$ votes for v or at least $f+1$ matching v. This will guarantee that all correct replicas will vote for v in the following round. By eliminating parallel RBC instances causing $O(n^3)$ messages and $O(n^3)$ communication, Quadratic-ABA now achieves $O(n^2)$ messages and $O(n^2)$ communication per round.

The protocol. The pseudocode of Quadratic-ABA is shown in Figure 3. The Quadratic-ABA protocol is round-based, starting from round 0. In each round, there are four phases—pre-vote$_r()$, vote$_r()$, main-vote$_r()$, and

```
01 initialization
02    r ← 0                                                    {round}
03 func propose(v_input)
04    iv_0 ← v_input                              {set input for round 0}
05    start round 0
06 round r
07    broadcast pre-vote_r(iv_r)                             {▷ phase 1}
08    upon receiving pre-vote_r(v) from f+1 replicas
09       if pre-vote_r(v) has not been sent, broadcast pre-vote_r(v)
10    upon receiving pre-vote_r(v) from 2f+1 replicas        {▷ phase 2}
11       bset_r ← bset_r ∪ {v}
12    wait until bset_r ≠ ∅
13       if vote_r() has not been sent
14          broadcast vote_r(v) where v ∈ bset_r
15    upon receiving n−f vote_r() such that for each vote_r(v), v ∈
bset_r                                                       {▷ phase 3}
16       if there are n−f vote_r(v)
17          broadcast main-vote_r(v)
18       else broadcast main-vote_r(∗)
19    upon receiving n−f main-vote_r() such that for each
main-vote_r(v), at least f+1 vote_r(v) have been received and for
each main-vote_r(∗), bset_r = {0,1}                          {▷ phase 4}
20       if there are n−f main-vote_r(v)
21          broadcast final-vote_r(v)
22       else broadcast final-vote_r(∗)
23    upon receiving n−f final-vote_r() such that for each
final-vote_r(v), at least f+1 main-vote_r(v) have been received and
for each final-vote_r(∗), bset_r = {0,1}
24       if there are n−f final-vote_r(v)
25          iv_{r+1} ← v, decide v
26       else if there are only final-vote_r(v) and final-vote_r(∗)
27          iv_{r+1} ← v
28       else
29          iv_{r+1} ← Random()                          {obtain local coin}
30    r ← r+1
```

Figure 3: The Quadratic-ABA protocol. The code for p_i.

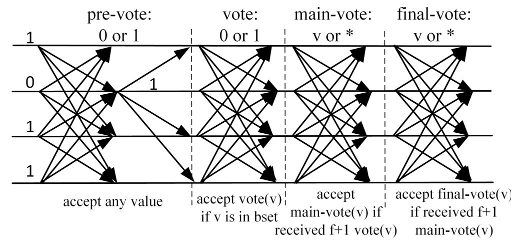


Figure 4: The workflow of Quadratic-ABA.

final-vote$_r$(), as shown in Figure 4. The pre-vote$_r$() and vote$_r$() phases (ln 07-14) are similar to the pre-vote$_r$() and main-vote$_r$() phases in Cubic-ABA. In the first phase, every replica p_i broadcasts a pre-vote$_r(iv_r)$ message, where iv_r is the value p_i votes for in round r (ln 07). After receiving $f+1$ pre-vote$_r(v)$ and p_i has not previously broadcast pre-vote$_r(v)$, p_i also broadcasts pre-vote$_r(v)$ (ln 08-09). At ln 10-11, upon receiving $n-f$ pre-vote$_r(v)$, p_i adds v to $bset_r$. For the first value v added to $bset_r$, p_i broadcasts a vote$_r(v)$ message (ln 12-14).

For each vote$_r(v)$ message, p_i accepts it only if v has been added to $bset_r$. Upon receiving $n-f$ vote$_r$() messages, one of the following two conditions holds.

- Ln 16-17: If p_i receives $n-f$ vote$_r(v)$ messages, it broadcasts a main-vote$_r(v)$ message.
- Ln 18: Otherwise, p_i broadcasts a main-vote$_r(*)$ message.

Every correct replica p_i accepts a main-vote$_r(v)$ message only if p_i has received $f+1$ vote$_r(v)$ messages. Every correct replica accepts a main-vote$_r(*)$ message only if $bset_r = \{0,1\}$. Upon receiving $n-f$ main-vote$_r$() messages, one of the following two conditions holds.

- Ln 20-21: If p_i receives $n-f$ main-vote$_r(v)$ messages, it broadcasts a final-vote$_r(v)$ message.
- Ln 22: Otherwise, p_i broadcasts final-vote$_r(*)$ message.

Every correct replica accepts a final-vote$_r(v)$ message only if it has received $f+1$ main-vote$_r(v)$ messages. Every correct replica accepts a final-vote$_r(*)$ message only if $bset_r = \{0,1\}$. Upon receiving $n-f$ final-vote$_r$() messages, there are three cases:

- Ln 24-25: If p_i receives $n-f$ final-vote$_r(v)$, it decides v and also sets iv_{r+1} as v. It participates in the protocol for one more round and terminates the protocol.
- Ln 26-27: If p_i receives $n-f$ final-vote$_r$() messages that carry only value v and $*$, it uses v for round $r+1$.
- Ln 28-29: Otherwise, p_i generates a local random coin and uses it as input for the next round.

Analysis. Quadratic-ABA guarantees that if a correct replica receives $n-f$ final-vote$_r(v)$, any correct replica will set iv_{r+1} as v (Lemma 15). First, if a correct replica sends main-vote$_r(v)$, no correct replica will send main-vote$_r(\bar{v})$ (Lemma 13). In particular, if a correct replica sends main-vote$_r(v)$, it must have received $n-f$ vote$_r(v)$. If another correct replica sends main-vote$_r(\bar{v})$, at least $n-f$ replicas have sent vote$_r(v)$. Therefore, at least one correct replica has sent both vote$_r(v)$ and vote$_r(\bar{v})$, contradicting the fact that each correct replica only sends a single vote$_r$() message in each round. Furthermore, if a correct replica sends final-vote$_r(v)$, no correct replica will send final-vote$_r(\bar{v})$ or even accept final-vote$_r(\bar{v})$ from other replicas. This is because if a correct replica accepts final-vote$_r(\bar{v})$, at least one correct replica has sent main-vote$_r(\bar{v})$. Meanwhile, if a correct replica accepts final-vote$_r(v)$, at least one correct replica has sent main-vote$_r(v)$, contradicting Lemma 13. Hence, at the end of each round, if a correct replica receives only final-vote$_r(v_1)$ and final-vote$_r(*)$, another correct replica receives only final-vote$_r(v_2)$ and final-vote$_r(*)$, it holds that $v_1 = v_2$. This result is crucial for agreement and termination.

Furthermore, if a correct replica decides v in round r, it must have received $n-f$ final-vote$_r(v)$. Among them, at least $f+1$ correct replicas have sent final-vote$_r(v)$. With $3f+1$ replicas in total, there are at most $2f$ final-vote$_r(\bar{v})$ or final-vote$_r(*)$. Hence, every correct replica receives at least one final-vote$_r(v)$. As no correct replica will accept final-vote$_r(\bar{v})$, every correct replica will only have final-vote$_r(v)$ and final-vote$_r(*)$. Thus, every correct replica either decides in round r, or enters round $r+1$ and sets iv_{r+1} to v. Doing so ensures agreement.

5 RABA from Local Coins

As shown in PACE [66], the PACE framework with RABA significantly outperforms the conventional BKR diagram and enables a fast path for termination. Our goal here is to use local coin based ABA to design RABA without trusted setup. We use Cubic-ABA and Quadratic-ABA to build Cubic-RABA and Quadratic-RABA, respectively. Here, we focus on Quadratic-RABA and present Cubic-RABA in Appendix B.

Subtlety of building Quadratic-RABA. PACE introduced a general approach to converting ABA to RABA [66]. Following their approach, we present Quadratic-RABA in Figure 5. Quadratic-RABA is identical to Quadratic-ABA except for the first round (round 0), where we make the following changes. First, we use a propose() event and a repropose() event (ln 03-07). Upon propose(v), a replica p_i starts round 0 and executes the *broadcast-vote*(v) function. Upon repropose(1) event, regardless of which round a replica is in, p_i still executes the *broadcast-vote*(v) function. The propose() and repropose() events are crucial for *biased termination*. So if a quorum of correct replicas either propose 1 or repropose 1, the protocol will eventually terminate.

Second, in the *broadcast-vote*(v) function, replica p_i broadcasts a pre-vote$_0(v)$ message (ln 09). At ln 10-14, if $v=1$, p_i immediately adds 1 to $bset_0$, and broadcasts vote$_0(1)$, main-vote$_0(1)$, and final-vote$_0(1)$.

Third, the coin value in round 0 is set to 1 (ln 38). The second and the third modifications guarantee both *biased validity* property and a *fast path allowing terminating the protocol in one step only*. Namely, if $f+1$ correct replicas propose 1, no correct replica will receive $2f+1$ final-vote$_0(0)$. As we will show in the proof, every correct replica will either directly decide 1 or set iv_1 as 1, so all correct replicas decide within two rounds. Furthermore, our protocol has a fast path: if all correct replicas propose 1, they will directly send vote$_0(1)$, main-vote$_0(1)$, final-vote$_0(1)$, allowing correct replicas to decide in one step.

The above modifications largely follow the generic transformation. We find that these modifications are sufficient for a secure Cubic-ABA, just as all known ABA protocols that can be transformed into their secure RABA counterparts (shown in [66]). Unexpectedly, we find for Quadratic-ABA, however, there is a subtle liveness issue for round 0. Suppose f correct replicas propose 1 and $f+1$ correct replicas propose 0. The f replicas directly broadcast vote$_0(1)$, main-vote$_0(1)$, and final-vote$_0(1)$. Even if the $f+1$ correct replicas that proposed 0 may later repropose 1, they may have already sent vote$_0(0)$, main-vote$_0(0)$, and final-vote$_0(0)$. In this case, no correct replica will accept final-vote$_0(1)$, as they do not receive $f+1$ main-vote$_0(1)$. The issue is, in essence, caused by the fact that each correct replica accepts a main-vote$_r(v)$ message only if it has previously received $f+1$ vote$_r(v)$ messages, and each correct replica accepts a final-vote$_r(v)$ message only if it has previously received $f+1$ main-vote$_r(v)$ messages. We visualize the example in Appendix C.

To resolve the above issue, we introduce another change to round 0 of the protocol. In particular, we relax the conditions for round 0: for each main-vote$_r(v)$ (ln 28) and final-vote$_r(v)$

```
01 initialization
02   r ← 0                                                        {round}
03 func propose(v)
04   broadcast-vote(v)
05   start round 0
06 func repropose(v)
07   broadcast-vote(v)
08 func broadcast-vote(v)
09   if pre-vote_0(v) has not been sent, broadcast pre-vote_0(v)
10   if v = 1
11     bset_0 ← bset_0 ∪ {1}
12     if vote_0() has not been sent, broadcast vote_0(1)
13     if main-vote_0() has not been sent, broadcast main-vote_0(1)
14     if final-vote_0() has not been sent, broadcast final-vote_0(1)
15 round r
16   if r > 0, broadcast pre-vote_r(iv_r)
17   upon receiving pre-vote_r(v) from f+1 replicas
18     if pre-vote_r(v) has not been sent, broadcast pre-vote_r(v)
19   upon receiving pre-vote_r(v) from 2f+1 replicas
20     bset_r ← bset_r ∪ {v}
21   wait until bset_r ≠ ∅
22     if vote_r() has not been sent
23       broadcast vote_r(v) where v ∈ bset_r
24     upon receiving n−f vote_r() such that for each received
   vote_r(b), b ∈ bset_r
25     if there are n−f vote_r(v)
26       broadcast main-vote_r(v)
27     else broadcast main-vote_r(∗)
28     upon receiving n−f main-vote_r() such that for each
   main-vote_r(v): 1) if r = 0, v ∈ bset_r, 2) if r > 0, at least f+1
   vote_r(v) have been received; for each main-vote_r(∗), bset_r = {0,1}
29     if there are n−f main-vote_r(v)
30       broadcast final-vote_r(v)
31     else broadcast final-vote_r(∗)
32     upon receiving n−f final-vote_r() such that for each
   final-vote_r(v), 1) if r = 0, v ∈ bset_r, 2) if r > 0, at least f+1
   main-vote_r(v) have been received; for each final-vote_r(∗), bset_r =
   {0,1}
33     if there are n−f final-vote_r(v)
34       iv_{r+1} ← v, decide v
35     else if there are only final-vote_r(v) and final-vote_r(∗)
36       iv_{r+1} ← v
37     else
38       if r = 0, iv_{r+1} ← 1                  {coin in the first round is 1}
39       else iv_{r+1} ← Random()                          {obtain local coin}
40   r ← r+1
```

Figure 5: The Quadratic-RABA protocol. The code for p_i.

```
upon selecting m_i for p_i using the technique of EPIC
    r-broadcast([e,i],m_i) for RBC_i
upon r-deliver([e,j],m_j) for RBC_j
    if RABA_j has not been started
        propose([e,j],1) for RABA_j
    else
        repropose([e,j],1) for RABA_j
upon delivery of n−f RBC instances
    for RABA instances that have not been started
        propose([e,j],0)
upon decide([e,j],v) for any value v for all RABA instances
    let S be set of indexes for RABA instances that decide 1
    wait until r-deliver([e,j],m_j) for all RABA_j where j ∈ S
        a-deliver(∪_{j∈S}{m_j})
```

Figure 6: The WaterBear family. The code for replica p_i in epoch e. WaterBear uses the EPIC technique to select transactions.

(ln 32), a correct replica accepts it as long as $v \in bset_0$. With this modification, the set of $f+1$ correct replicas that proposed 0 will repropose 1, so every correct replica will eventually add 1 in $bset_0$. Hence, every correct replica will eventually accept $\text{main-vote}_0(1)$ and $\text{final-vote}_0(1)$. Our result underlines the subtlety of constructing RABA from ABA and the importance of a full proof for a new protocol (proof in Appendix I).

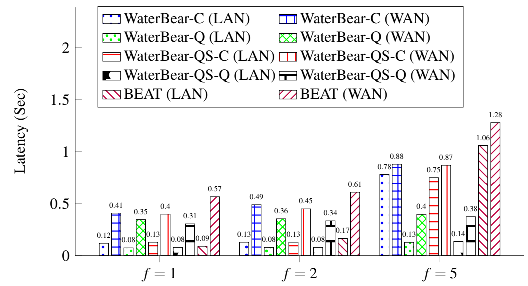
6 The WaterBear Family

This section describes our asynchronous BFT protocols—WaterBear (WaterBear-C and WaterBear-Q), and WaterBear-QS (WaterBear-QS-C, and WaterBear-QS-Q). All the protocols are quantum secure, and WaterBear-C and WaterBear-Q rely on authenticated channels only.

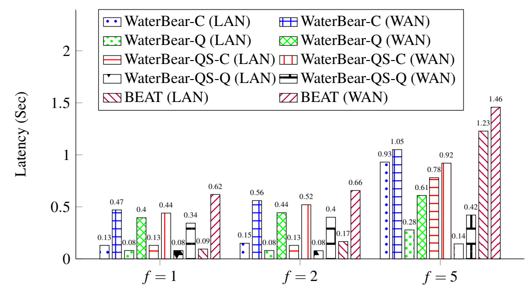
6.1 The WaterBear Protocols

WaterBear follows the PACE paradigm but uses the trick in EPIC [49] to avoid the usage of threshold encryption (needed for achieving adaptive security). In particular, WaterBear uses *r-broadcast* and *r-deliver* primitives of Bracha's broadcast, and *propose*, *repropose* and *decide* primitives of WaterBear RABA. Figure 6 depicts the pseudocode of WaterBear. Earlier asynchronous BFT protocols such as HoneyBadger and BEAT use threshold encryption to allow parallel, random transaction selection while achieving censorship resilience (liveness); however, no efficient adaptively secure threshold encryption was known. EPIC thus proposes a new crypto-free transaction selection strategy: replicas select random transactions *in plaintext* for most epochs, and to achieve liveness, they periodically switch to the first-in, first-out (FIFO) selection, where replicas maintain a log of transactions according to the order that transactions are received and select as input the first transaction group in the buffer. In WaterBear, we use this trick to avoid threshold encryption.

Following the PACE paradigm, for each epoch e, WaterBear consists of n parallel RBC instances and n parallel RABA instances. In the RBC phase, each replica p_i *r-broadcasts* a proposal m_i for RBC_i. If p_i *r-delivers* a proposal from RBC_j, it proposes 1 for RABA_j. Upon delivery of $n-f$ RBC instances, instead of waiting for $n-f$ RABA instances to terminate, p_i 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. We let S be the set of indexes where RABA_j decides 1. When all RABA instances terminate and all RBC_i ($i \in S$) instances are delivered, p_i *a-delivers* $\cup_{j\in S}\{m_j\}$. The security of WaterBear directly follows from that of the PACE paradigm. As we propose two RABA protocols Cubic-RABA and Quadratic-RABA. We use Cubic-RABA to build WaterBear-C and Quadratic-RABA to build WaterBear-Q.



(a) Latency for $b=1$.



(b) Latency for $b=100$.

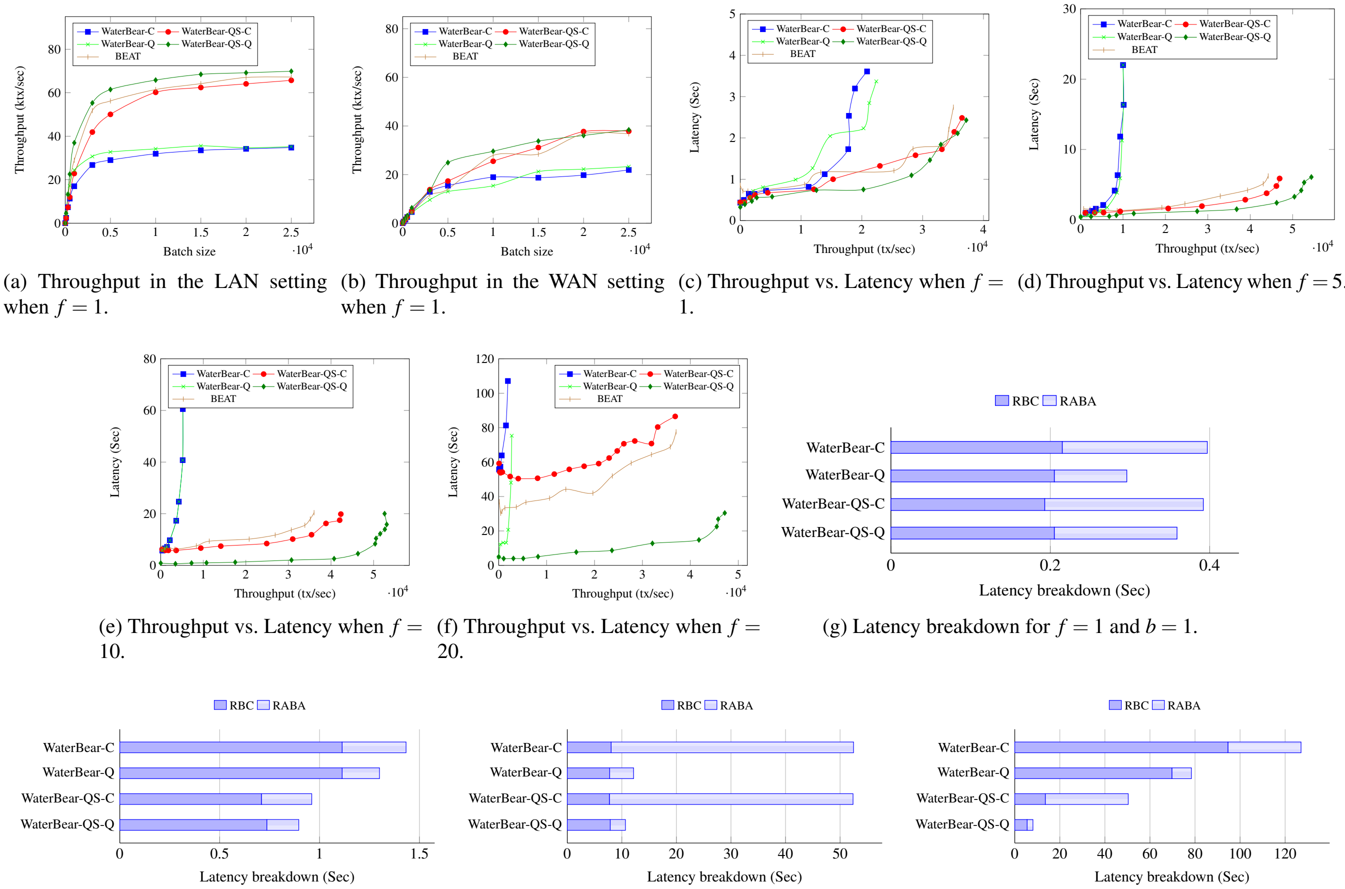
Figure 7: Latency of the protocols.

6.2 The WaterBear-QS Protocols

We now describe WaterBear-QS, also consisting of two asynchronous BFT protocols. We use Cubic-RABA to build WaterBear-QS-C and Quadratic-RABA to build WaterBear-QS-Q. The difference between WaterBear and WaterBear-QS is that WaterBear-QS additionally uses hash functions (used in CT RBC [17]) to reduce the communication complexity of RBC. Jumping ahead, we show the modification leads to a dramatic performance improvement.

7 Implementation and Evaluation

Implementation. We implemented WaterBear-C, WaterBear-Q, WaterBear-QS-C, and WaterBear-QS-Q in a new Golang library. For comparison, we choose to implement BEAT[2] [33]



(a) Throughput in the LAN setting when $f = 1$.

(b) Throughput in the WAN setting when $f = 1$.

(c) Throughput vs. Latency when $f = 1$.

(d) Throughput vs. Latency when $f = 5$.

(e) Throughput vs. Latency when $f = 10$.

(f) Throughput vs. Latency when $f = 20$.

(g) Latency breakdown for $f = 1$ and $b = 1$.

(h) Latency breakdown for $f = 1$ and $b = 5000$.

(i) Latency breakdown for $f = 20$ and $b = 1$.

(j) Latency breakdown for $f = 20$ and $b = 5000$.

Figure 8: Throughput vs. latency and latency breakdown on m5.xlarge instances for $f = 1$ to $f = 20$.

in our library. We implemented a new version of BEAT, replacing MMR ABA with Cobalt-ABA, as Cobalt ABA addressed the liveness issue of MMR. Our implementation has been made publicly available[3] and involves more than 11,000 LOC for the protocol implementations and about 1,000 LOC for evaluation.

All the protocols use authenticated channels and WaterBear-QS additionally uses hash functions. We use HMAC to realize authenticated channels. We use SHA256 as the hash function. We use gRPC as the communication library.

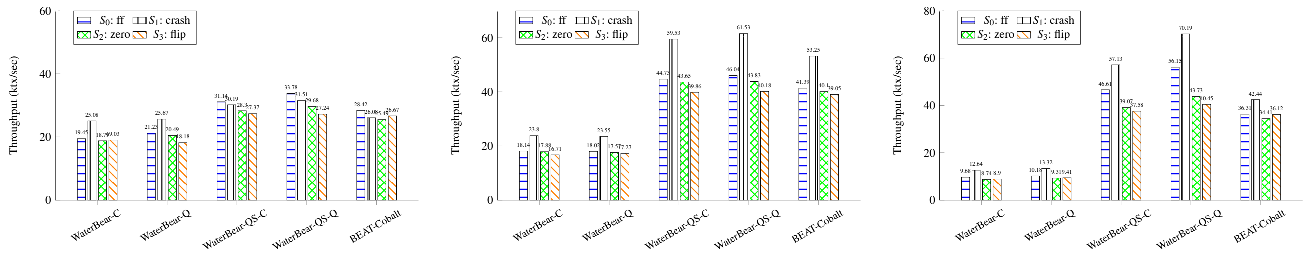
All our implemented asynchronous protocols use RBC in their RBC phases, and WaterBear-C and WaterBear-QS-C additionally use RBC in the ABA phase. For WaterBear-C and WaterBear-Q, we use Bracha's broadcast in the RBC phase. For WaterBear-QS-C and WaterBear-QS-Q, we use CT RBC [17] (using erasure coding and hash functions) in the RBC phase. In ABA phases of WaterBear-C and WaterBear-QS-C, we directly use Bracha's broadcast because there is no bulk data (and no need to use erasure coding). To implement CT RBC, we use a Golang Reed-Solomon code library [2].

[2]https://github.com/fififish/beat

[3]https://github.com/fififish/waterbear

There are several reasons we chose BEAT as the baseline protocol. First, BEAT is one of the most efficient open-source asynchronous BFT implementations. As shown in PACE [66], BEAT is more efficient than Dumbo [45] for $n \leq 46$. Second, all WaterBear protocols achieve adaptive security, and EPIC is the only known adaptively secure asynchronous BFT protocol implemented. It is shown that BEAT significantly outperforms EPIC in both LAN and WAN settings [49]. Hence, if we show the performance difference between BEAT and our protocols, we can argue which is the most efficient adaptively secure asynchronous BFT protocol among EPIC and WaterBear protocols. We do not attempt to compare our protocols with other BFT protocols in Table 1, as those protocols neither achieve adaptive nor quantum security, relying on PKC and trusted setup. *Indeed, our goal is not to claim WaterBear protocols are the most efficient asynchronous BFT protocols, but we aim at refuting the conventional wisdom that asynchronous BFT protocols cannot match the security guarantees of partially synchronous protocols while preserving performance.*

Overview of evaluation. We evaluate the performance of our protocols on Amazon EC2 utilizing up to 61 virtual machines (VMs). We consider both LAN and WAN settings. In the LAN setting, the replicas are run in the same region



(a) Throughput for $f = 1$.

(b) Throughput for $f = 2$.

(c) Throughput for $f = 5$.

Figure 9: Performance of the protocols in failure scenarios.

of EC2—US Virginia. In the WAN setting, the replicas are evenly distributed in four different regions: us-west-2 (Oregon, US), us-east-2 (Ohio, US), ap-southeast-1 (Singapore), and eu-west-1 (Ireland). The lowest one-way latency (resp., the highest latency) is 24.5 ms for Ohio-Oregon (resp., 90 ms for Ireland-Singapore). We use both *t2.medium* and *m5.xlarge* instances for our evaluation. The *t2.medium* type has two virtual CPUs and 4GB memory and the *m5.xlarge* has four virtual CPUs and 16GB memory. Unless otherwise mentioned, we use *m5.xlarge* instances by default. We conduct the experiments under different network sizes and contention levels (batch size). We use f to denote the network size; in each experiment, we use $3f+1$ replicas in total. We let b denote the contention level. In particular, each replica proposes b transactions in each epoch. For each experiment, we vary the batch size b from 1 to 25,000. For each experiment, we run the tests 10 times and compute the average (for both throughput and latency). We evaluate the performance of the protocols with two different transaction sizes—100 bytes by default and also 250 bytes.

We assess the protocols under failure-free and failure scenarios. While our failure-case evaluation is not the first such evaluation for asynchronous BFT protocols, the testbed we built aims to be comprehensive, encompassing realistic failure and attack scenarios we can envision. We roughly summarize our main results in the following:

- The WaterBear protocols using Quadratic-RABA (WaterBear-QS-Q and WaterBear-Q) are much more efficient than the protocols using Cubic-RABA (WaterBear-QS-C and WaterBear-C), as Quadratic-RABA has $O(n^2)$ messages and much fewer steps than Cubic-RABA (with $O(n^3)$ messages). The result justifies the importance of designing Quadratic-RABA and Quadratic-ABA.
- The quantum secure WaterBear protocols (WaterBear-QS-C and WaterBear-QS-Q) drastically outperform their counterparts assuming authenticated channels only (WaterBear-C and WaterBear-Q), as the RBC used for WaterBear-QS-C and WaterBear-QS-Q is more bandwidth-efficient than that for WaterBear-C and WaterBear-Q. The finding highlights the cost of achieving security with authenticated channels only from quantum security for our protocols.
- Regarding latency, all WaterBear protocols have lower latency (under no contention) than BEAT. Regarding throughput, all our protocols, except WaterBear-QS-Q, share similar performance as BEAT.
- WaterBear-QS-Q consistently and significantly outpaces BEAT. For instance, when $n = 16$, WaterBear-QS-Q has about 1/8 the latency that of BEAT and 1.23x the throughput of BEAT. As n grows larger, the peak throughput of WaterBear-QS-Q is about 1.47x that of BEAT.
- All four protocols we propose are highly robust against various crash and Byzantine failures, just as BEAT.

7.1 Performance in Failure-Free Cases

Latency. We report the latency of the asynchronous protocols in both LAN and WAN settings for $f = 1, 2,$ and 5 in Figure 7 with for $b = 1$ and 100. All WaterBear protocols consistently achieve lower latency than BEAT in both LAN and WAN environments, mainly because our protocols have a coin-free fast path. Among the protocols, WaterBear-QS-Q has consistently lower latency than all other protocols, as WaterBear-QS-Q has the lowest communication complexity among the WaterBear protocols. As f increases, the latency difference between WaterBear-QS-Q and other protocols becomes more visible. For instance, when $f = 5$ in the WAN setting, BEAT achieves 3.47x latency of that for WaterBear-QS-Q; in the LAN setting, the latency for BEAT is 8.78x of that for WaterBear-QS-Q.

Let us explain the latency results in more detail. First, take WaterBear-QS-Q in WANs for an example. In the optimistic mode (with no failures and synchronous networks), it takes four steps (3 for RBC and 1 for RABA due to the fast path in Quadratic-RABA) to terminate. When $f = 1$, the latency reported for WaterBear-QS-Q is 310 ms, which is consistent with the ping latency mentioned above (24.5 ms$\sim$90 ms). In contrast, WaterBear-QS-C in its optimistic mode has 6 steps, which justifies its latency of 570 ms.

Throughput and scalability. We report throughput and throughput vs. latency of all our implemented protocols in Figure 8 by varying the network size f from 1 to 20.

Our results show that the throughput of WaterBear-C and WaterBear-Q are consistently lower than the other protocols. As WaterBear-C (resp. WaterBear-Q) and WaterBear-QS-C (resp. WaterBear-QS-Q) differ in RBC only, RBC is clearly one performance bottleneck. The result highlights the overhead of achieving security with authenticated channels only.

We assess the throughput of all the protocols for $f = 1$ in WAN as depicted in Figure 8b: the peak throughput of WaterBear-QS-Q is slightly higher in most experiments. We also conduct a separate experiment in LANs, as shown in Figure 8a. Unlike the results in WANs, the throughput of BEAT in LANs is marginally higher than WaterBear-QS-C, and the throughput of WaterBear-QS-Q is marginally higher than BEAT: the peak throughput of BEAT is 2.3% higher than WaterBear-QS-C, and the peak throughput of WaterBear-QS-Q is 3.9% higher than BEAT. The peak throughput of WaterBear-QS-C is 65.7 ktx/sec in LANs and 37.8 ktx/sec in WANs, and the peak throughput of WaterBear-QS-Q is 69.9 ktx/sec in LANs and 38.4 ktx/sec in WANs.

When f increases, in general, WaterBear-QS-Q and WaterBear-QS-C outpace all the other protocols. The peak throughput of WaterBear-QS-C is higher than BEAT when $f = 5$ and $f = 10$ but lower when $f = 20$ only. Meanwhile, WaterBear-QS-Q is consistently more efficient than all other asynchronous protocols (higher throughput and lower latency). For instance, when $f = 10$, the peak throughput of WaterBear-QS-Q is 47.4% higher than BEAT. The reason is WaterBear-QS-Q uses a more communication-efficient RABA protocol.

We report the latency breakdown of the WaterBear protocols for $b = 1$ and 5000 in Figure 8g-8j. These experiments justify the design of our most efficient protocol—WaterBear-QS-Q (with CT RBC that is communication-efficient and Quadratic-RABA with asymptotically reduced per-round message complexity and concretely reduced number of steps). Indeed, as f increases, RABA dominates the latency and Quadratic-RABA performs indeed much better than Cubic-RABA; when b increases, RBC becomes the latency bottleneck and CT RBC outperforms Bracha's RBC significantly.

Additional evaluation results. We show in Appendix D additional evaluation results, including the performance comparison with HotStuff [65], the performance on different types of VMs, the performance with different transaction sizes, and the memory and the CPU usage for the protocols implemented.

7.2 Performance under Failures

To assess the protocol performance under failures and attacks, we carefully design various experiments as follows. We focus on the five asynchronous protocols.

- S_0**: (failure-free)** All replicas are correct. S_0 is the baseline scenario used to compare with failure scenarios.
- S_1**: (crash)** f replicas crash by not participating in the protocols.
- S_2**: (Byzantine; keep voting 0)** We control all f faulty replicas to keep voting 0 in each step of (R)ABA. For all protocols, doing so would intuitively make fewer (R)ABA instances to decide 1 and would likely decrease the throughput of the protocols. We aim to observe the throughput reduction compared to failure-free scenarios.
- S_3**: (Byzantine; flipping the (R)ABA input)** We let f replicas exhibit Byzantine behavior in the (R)ABA phase. The strategy is to vote for a flipped value in (R)ABA. In other words, in each (R)ABA step, each Byzantine replica inputs $\bar{b}$ when it should have input b. Doing so could potentially force each (R)ABA instance to experience more steps to terminate for all five protocols. For WaterBear and WaterBear-QS, the strategy would, at first glance, likely be more fruitful. For both protocols, a RABA instance may terminate in round 0, thanks to the biased validity property of RABA. The flipping strategy illustrated above may make them not decide in round 0 and force them to enter the second round of RABA, where the two protocols start to query the local coins.

We assess the performance for $f = 1$ (Figure 9a), $f = 2$ (Figure 9b), and $f = 5$ (Figure 9c).

Performance under crash failures (S_1**).** The throughput of all the five protocols implemented under crash failures is higher than that in the failure-free case, except for $f = 1$, where all protocols share similar performance between the two scenarios. Our result echoes those of previous works. The reason is that under crash failures, the network bandwidth consumption is much lower (about 33% lower) than in the failure-free case. Note that when $f = 1$, the network bandwidth consumption is not as dominating as in other cases; hence, the performance difference among the protocols for $f = 1$ is less visible.

Performance under Byzantine failures (S_2 **and** S_3**).** The performance of all the protocols under Byzantine failures is slightly lower than that in the failure-free scenario and the crash failure scenario. WaterBear-QS-C and WaterBear-QS-Q suffer from slightly higher performance degradation under Byzantine failures compared to BEAT. The higher performance degradation is due to the use of local coins. As replicas start to use local coins in round $r > 0$, the RABA protocol may decide in more rounds. In all cases, WaterBear-QS-C and WaterBear-QS-Q remain more efficient than BEAT.

The difference between S_2 and S_3 is that faulty replicas broadcast 0 in S_2 but broadcast the flipped value in S_3. For BEAT, the performance in S_3 is higher for $f = 1$ and $f = 5$ but lower for $f = 2$; the difference in all the cases is not significant though. In contrast, for WaterBear-QS-C and WaterBear-QS-Q, the performance in S_3 is consistently lower than S_2, showing that the flipping strategy in S_3 works slightly better than that in S_2.

8 Conclusion

This paper designs and implements a family of practical asynchronous BFT protocols matching the security guarantees of their partially synchronous counterparts. Our experiments demonstrate that our protocols are efficient in both failure and failure-free scenarios. In particular, one of our protocols, WaterBear-QC-Q, consistently outperforms the state-of-the-art asynchronous protocols with much weaker security guarantees. We also build in different settings more efficient

ABA and RABA protocols that can be used to improve various high-level Byzantine-resilient protocols. Our work, for the first time, shows that the strongest security models and high performance can co-exist for asynchronous BFT.

Acknowledgment

This work was supported in part by the National Key R&D Program of China under 2022YFB2701700 and 2022YFB2701500, the National Natural Science Foundation of China under 62272043 and 92267203, Beijing Natural Science Foundation under M23015, Major Program of Shandong Provincial Natural Science Foundation for the Fundamental Research under ZR2022ZD03, and the Piloting Fundamental Research Program for the Integration of Scientific Research, Education and Industry of Qilu University of Technology (Shandong Academy of Sciences) 2022XD001.

References

[1] BEAT library. https://github.com/fififish/beat, 2021.

[2] Reed-Solomon library. https://github.com/klauspost/reedsolomon, 2021.

[3] Ittai Abraham, Danny Dolev, and Joseph Y. Halpern. An almost-surely terminating polynomial protocol for asynchronous byzantine agreement with optimal resilience. PODC, 2008.

[4] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *PODC*, 2021.

[5] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In *PODC*, pages 399–417, 2022.

[6] Nicolas Alhaddad, Sisi Duan, Mayank Varia, and Haibin Zhang. Succinct erasure coding proof systems. *Cryptology ePrint Archive*, 2021.

[7] Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. High-threshold avss with optimal communication complexity. In *FC*, 2021.

[8] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. Prime: Byzantine replication under attack. *TDSC*, 8(4):564–577, 2011.

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

[10] Laasya Bangalore, Ashish Choudhury, and Arpita Patra. The power of shunning: Efficient asynchronous byzantine agreement revisited. *J. ACM*, 2020.

[11] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *PODC*, pages 27–30, 1983.

[12] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience. In *PODC*, pages 183–192. ACM, 1994.

[13] Gabriel Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *PODC*, pages 154–162. ACM, 1984.

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

[15] Christian Cachin. Yet another visit to paxos. IBM Research Report RZ 3754, 2010.

[16] Christian Cachin and Jonathan A Poritz. Secure intrusion-tolerant replication on the internet. In *DSN*, pages 167–176. IEEE, 2002.

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

[18] Christian Cachin and Marko Vukolic. Blockchain consensus protocols in the wild. In *DISC*, 2017.

[19] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, 1996.

[20] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *STOC*, volume 93, pages 42–51. Citeseer, 1993.

[21] Miguel Castro. Practical byzantine fault tolerance. PhD thesis, 2001.

[22] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *TOCS*, 20(4):398–461, 2002.

[23] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray A Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*, volume 12. US Department of Commerce, National Institute of Standards and Technology, 2016.

[24] Allen Clement, Edmund L Wong, Lorenzo Alvisi, Michael Dahlin, and Mirco Marchetti. Making byzantine fault tolerant systems tolerate byzantine faults. In *NSDI*, volume 9, pages 153–168, 2009.

[25] Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. *Comput. J.*, 49(1):82–96, 2006.

[26] Tyler Crain. Two more algorithms for randomized signature-free asynchronous binary byzantine consensus with t<n/3 and o(n^2) messages and O(1) round expected termination. *CoRR*, abs/2002.08765, 2020.

[27] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT*, 1999.

[28] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: A dag-based mempool and efficient bft consensus. EuroSys '22.

[29] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In *CCS*, pages 2705–2721, 2021.

[30] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. *IEEE Symposium on Security and Privacy*, 2022.

[31] Shlomi Dolev and Ziyu Wang. Sodsbc: Stream of distributed secrets for quantum-safe blockchain. In *2020 IEEE International Conference on Blockchain (Blockchain)*, pages 247–256. IEEE, 2020.

[32] Sisi Duan, Hein Meling, Sean Peisert, and Haibin Zhang. BChain: Byzantine replication with high throughput and embedded reconfiguration. In *OPODIS*, pages 91–106, 2014.

[33] Sisi Duan, Michael K Reiter, and Haibin Zhang. BEAT: Asynchronous BFT made practical. In *CCS*, pages 2028–2041. ACM, 2018.

[34] Sisi Duan, Xin Wang, and Haibin Zhang. Fin: Practical signature-free asynchronous common subset in constant time. ACM CCS, 2023.

[35] Sisi Duan and Haibin Zhang. Foundations of dynamic bft. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1317–1334. IEEE, 2022.

[36] Sisi Duan and Haibin Zhang. Recent progress on bft in the era of blockchains. *National Science Review*, 9(10):nwac132, 2022.

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

[38] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *STOC*, 1988.

[39] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. Technical report, Massachusetts Inst of Tech Cambridge lab for Computer Science, 1982.

[40] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency. *arXiv preprint arXiv:2209.00750*, 2022.

[41] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Efficient asynchronous byzantine agreement without private setups. *ICDCS*, 2022.

[42] Neil Giridharan, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Bullshark: Dag bft protocols made practical. *ACM CCS*, 2022.

[43] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. *ACM Transactions on Computer Systems*, 32(4):12:1–12:45, 2015.

[44] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Speeding dumbo: Pushing asynchronous bft closer to practice. *NDSS*, 2022.

[45] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous bft protocols. In *CCS*, 2020.

[46] James Hendricks, Shafeeq Sinnamohideen, Gregory R Ganger, and Michael K Reiter. Zzyzx: Scalable fault tolerance through byzantine locking. In *DSN*, pages 363–372. IEEE, 2010.

[47] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *PODC*, 2021.

[48] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. CCS, 2020.

[49] Chao Liu, Sisi Duan, and Haibin Zhang. Epic: Efficient asynchronous bft with adaptive security. In *DSN*, 2020.

[50] Chao Liu, Sisi Duan, and Haibin Zhang. MiB: Asynchronous BFT with more replicas. *CoRR*, abs/2108.04488, 2021.

[51] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *PODC*, pages 129–138, 2020.

[52] Ethan MacBrough. Cobalt: Bft governance in open networks. *arXiv preprint arXiv:1802.07240*, 2018.

[53] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *CCS*, pages 31–42. ACM, 2016.

[54] Henrique Moniz, Nuno Ferreria Neves, Miguel Correia, and Paulo Verissimo. Ritas: Services for randomized intrusion tolerance. *TDSC*, 8(1):122–136, 2008.

[55] Achour Mostefaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous Byzantine consensus with $t \leq n/3$ and $o(n^2)$ messages. In *PODC*, pages 2–9. ACM, 2014.

[56] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with t < n/3, o(n2) messages, and O(1) expected time. *J. ACM*, 62(4):31:1–31:21, 2015.

[57] A. Patra, A. Choudhury, and C.P. Rangan. Asynchronous byzantine agreement with optimal resilience. *Distrib. Comput.*, 27:111–146, 2014.

[58] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *JACM*, 27(2):228–234, April 1980.

[59] Michael O Rabin. Randomized byzantine generals. In *SFCS*, pages 403–409. IEEE, 1983.

[60] Alan T Sherman, Farid Javani, Haibin Zhang, and Enis Golaszewski. On the origins and variations of blockchain technologies. *IEEE Security & Privacy*, 17(1):72–77, 2019.

[61] João Sousa, Eduardo Alchieri, and Alysson Bessani. State machine replication for the masses with bft-smart. In *DSN*, pages 355–362, 2014.

[62] Gilad Stern and Ittai Abraham. Information theoretic hotstuff. In *OPODIS*, 2018.

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

[64] Xin Wang, Sisi Duan, James Clavin, and Haibin Zhang. Bft in blockchains: From protocols to use cases. *ACM Computing Surveys (CSUR)*, 54(10s):1–37, 2022.

[65] Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi, and Mike Dahlin. Separating agreement from execution for Byzantine fault tolerant services. *SOSP*, 37(5):253–267, 2003.

[66] Haibin Zhang and Sisi Duan. PACE: fully parallelizable bft from reproposable byzantine agreement. In *CCS 2022*.

[67] Haibin Zhang, Sisi Duan, Chao Liu, Boxin Zhao, Xuanji Meng, Shengli Liu, Yong Yu, Fangguo Zhang, and Liehuang Zhu. Practical asynchronous distributed key generation: Improved efficiency, weaker assumption, and standard model. *DSN*, 2023.

[68] Haibin Zhang, Sisi Duan, Boxin Zhao, and Liehuang Zhu. Waterbear: Practical asynchronous bft matching security guarantees of partially synchronous bft. Usenix Security, 2023.

[69] Haibin Zhang, Chao Liu, and Sisi Duan. How to achieve adaptive security for asynchronous bft? *Journal of Parallel and Distributed Computing*, 169:252–268, 2022.

A Bracha's ABA

We present Bracha's ABA [13]. The pseudocode is shown in Figure 10. Bracha's ABA has three phases. In each phase, each replica broadcasts its value via a RBC instance, i.e., there are n parallel RBC instances in each of the three phases. As the underlying RBC has $O(n^2)$ messages and 4 steps, Bracha's ABA has $O(n^3)$ messages and 12 steps in each round.

In Bracha's ABA, every replica maintains a set *vset* containing *valid* values. In each phase, every replica only accepts messages that carry valid values. The valid values *vset* must be congruent with the values each replica receives from the previous phase (or the last phase of the previous round). In the first phase of round 0, both 0 and 1 are considered valid. In the second and third phases, a value is added to *vset* only if the replica receives the value from enough replicas.

In the first phase, every replica p_i *r-broadcasts* a $\textsf{pre-vote}_r(iv_r)$ message (ln 08), where iv_r is the input value of p_i for round r.

In the second phase, p_i waits for $n-f$ $\textsf{pre-vote}_r()$ messages such that for each $\textsf{pre-vote}_r(v)$, $v \in vset$. There are two cases:

- Ln 10-13: If p_i has received $n-f$ $\textsf{pre-vote}_r(v)$ for some $v \in \{0,1\}$, p_i decides v and sets both *vset* and iv_{r+1} as v. Replica p_i continues for one more round and terminates the protocol (up to either ln 10 or ln 25 before p_i decides some value again).
- Ln 14-15: Otherwise, p_i sets v as the majority value in the set of $\textsf{pre-vote}_r()$ messages it receives. The set *vset* is not changed, i.e., $vset = \{0,1\}$.

In both cases, p_i *r-broadcasts* a $\textsf{main-vote}_r(v)$ message (ln 16).

```
01 Initialization
02    r ← 0                                                   {round}
03 func propose(v)
04    iv_0 ← v
05    vset ← {0,1}          {valid binary values that will be accepted}
06    start round 0
07 round r
08    r-broadcast pre-vote_r(iv_r)                            {▷ phase 1}
09    upon r-delivering n − f pre-vote_r() such that for each
pre-vote_r(v), v ∈ vset                                       {▷ phase 2}
10       if there are n − f pre-vote_r(v)
11          decide v
12          iv_{r+1} ← v
13          vset ← {v}
14       else
15          v ← majority value in the set of pre-vote_r() messages
16       r-broadcast main-vote_r(v)
17    upon r-delivering n − f main-vote_r() such that for each
main-vote_r(v), v ∈ vset                                      {▷ phase 3}
18       if there are at least n/2 main-vote_r(v)
19          vset ← {v}
20       else
21          v ← {⊥}
22          vset ← {0,1}
23       r-broadcast final-vote_r(v)
24    upon r-delivering n − f final-vote_r() such that for each
final-vote_r(v), v ∈ vset; for each final-vote_r(∗), vset = {0,1}
25       if there are at least 2f + 1 final-vote_r(v)
26          decide v
27          iv_{r+1} ← v
28          vset ← {v}
29       else if there are f + 1 final-vote_r(v)
30          iv_{r+1} ← v
31          vset ← {0,1}
32       else
33          iv_{r+1} ← Random()                       {obtain local coin}
34          vset ← {0,1}
35    r ← r + 1
```

Figure 10: The Bracha's ABA protocol [13]. The code for p_i.

```
01 initialization
02    r ← 0                                                   {round}
03 func propose(v)
04    broadcast-vote(v)
05    start round 0
06 func repropose(v)
07    broadcast-vote(v)
08 func broadcast-vote(v)
09    if pre-vote_0(v) has not been sent, broadcast pre-vote_0(v)
10    if v = 1
11       bset_0 ← bset_0 ∪ {1}
12       if main-vote_0() has not been sent, broadcast main-vote_0(1)
13       if final-vote_0() has not been sent, r-broadcast final-vote_0(1)
14 round r
15    if r > 0, broadcast pre-vote_r(iv_r)
16    upon receiving pre-vote_r(v) from f + 1 replicas
17       if pre-vote_r(v) has not been sent, broadcast pre-vote_r(v)
18    upon receiving pre-vote_r(v) from 2f + 1 nodes
19       bset_r ← bset_r ∪ {v}
20    wait until bset_r ≠ ∅
21       if main-vote_r() has not been sent
22          broadcast main-vote_r(v) where v ∈ bset_r
23    upon receiving n − f main-vote_r() such that 1) final-vote_r()
has not been sent; 2) for each received main-vote_r(b), b ∈ bset_r
24       if there are n − f main-vote_r(v)
25          r-broadcast final-vote_r(v)
26       else r-broadcast final-vote_r(∗)
27    upon r-delivering n − f final-vote_r() such that for each
final-vote_r(v), v ∈ bset_r; for each final-vote_r(∗), bset_r = {0,1}
28       if there are n − f final-vote_r(v)
29          decide v
30       else if there are f + 1 final-vote_r(v)
31          iv_{r+1} ← v
32       else
33          if r = 0, iv_{r+1} ← 1
34          else iv_{r+1} ← Random()
35    r ← r + 1
```

Figure 11: Cubic-RABA. The code for p_i.

In the third phase, every replica p_i waits for $n-f$ valid main-vote$_r$() messages (ln 17). There are two cases:

- Ln 18-19: If p_i receives at least $n/2$ main-vote$_r(v)$, it sets *vset* as $\{v\}$.
- Ln 20-22: Otherwise, p_i sets v as $*$ and *vset* as $\{0,1\}$.

In both cases, p_i *r-broadcasts* a final-vote$_r(v)$ message (ln 23). Then every replica waits for $n-f$ valid final-vote$_r$() messages (ln 24). Note that final-vote$_r(*)$ is considered valid only if $vset = \{0,1\}$. There are three cases:

- Ln 25-28: If p_i receives at least $2f+1$ final-vote$_r(v)$, it decides v and sets iv_{r+1} as v. Replica p_i continues for one more round (up to either ln 10 or ln 25) and terminates the protocol.
- Ln 29-31: If p_i receives at least $f+1$ final-vote$_r(v)$, it sets iv_{r+1} as v and *vset* as $\{v\}$.
- Ln 32-34: Otherwise, p_i uses the local coin value as iv_{r+1} and *vset* as $\{0,1\}$, i.e., p_i accepts both 0 and 1 in the first phase of the following round.

B Cubic-RABA

The pseudocode of Cubic-RABA protocol is shown in Figure 11. Cubic-RABA is identical to Cubic-ABA, except for round 0 (the first round). We have made the following changes for round 0. First, both propose() and repropose() events are allowed. Upon the propose(v) event (ln 03), a replica p_i executes the *broadcast-vote*(v) function and starts round 0. Upon the repropose(v) function (ln 06), p_i executes *broadcast-vote*(v). Note that upon a repropose() event, p_i must have already started the protocol and may even proceed to a round greater than 0. In this case, regardless of which round the replica is in, it executes the *broadcast-vote*(v) function and broadcasts a $\mathsf{pre\text{-}vote}_0(v)$ message.

Second, in the *broadcast-vote*(v) function (ln 08-13), p_i broadcasts a $\mathsf{pre\text{-}vote}_0(v)$ message. If $v = 1$, p_i adds 1 to $bset_0$ (ln 11). If p_i has not previously broadcast $\mathsf{main\text{-}vote}_0()$, it broadcasts $\mathsf{main\text{-}vote}_0(1)$ (ln 12). If p_i has not *r-broadcast* $\mathsf{final\text{-}vote}_0()$, it *r-broadcasts* $\mathsf{final\text{-}vote}_0(1)$ (ln 13).

Finally, the coin value for round 0 is set to 1 (ln 33). In round $r \geq 1$, Cubic-RABA is identical to Cubic-ABA.

Analysis. The proof of Cubic-RABA is shown in Appendix H. We show that the changes we have made on top of Cubic-ABA can transform Cubic-ABA into a RABA protocol. The first change can ensure the *biased termination* property. In particular, it guarantees that if a quorum of correct replicas either directly propose 1 or propose 0 and later on repropose 1, the protocol will terminate. The second and third changes ensure the biased validity property. If $f + 1$ correct replicas propose 1, they will directly add 1 to $bset_0$, broadcast $\mathsf{pre\text{-}vote}_0(1)$, $\mathsf{main\text{-}vote}_0(1)$, and *r-broadcast* $\mathsf{final\text{-}vote}_0(1)$. Namely, no correct replica can receive $n - f$ $\mathsf{main\text{-}vote}_0(0)$ or *r-broadcast* $\mathsf{final\text{-}vote}_0(0)$. Furthermore, no correct replica can receive $n - f$ $\mathsf{final\text{-}vote}_0(0)$ or $f + 1$ $\mathsf{final\text{-}vote}_0(0)$. Furthermore, for the case where a correct replica uses the local coin to enter the next round, the coin value is also 1. Accordingly, Cubic-RABA achieves biased validity. Other properties of Cubic-RABA follow from Cubic-ABA, as we only modify round 0 of the protocol.

C Liveness Challenge of Building Quadratic-RABA

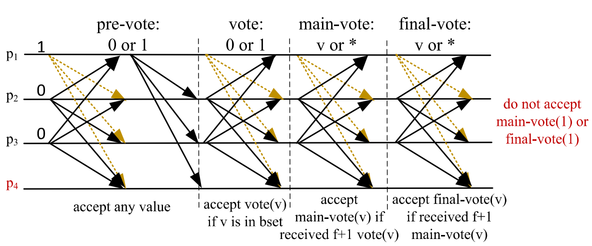


Figure 12: Subtle challenge when converting Quadratic-ABA to Quadratic-RABA. Votes for 1 are marked in dashed lines.

We illustrate an example in Figure 12 to show the subtle liveness challenge when converting Quadratic-ABA to Quadratic-RABA. We illustrate a scenario to show that, if we only apply the three modifications mentioned in Sec. 5, the protocol may not terminate. In this example, we have four replicas—p_1 to p_4, and p_4 simply crashes. In round 0, the input of p_1 is 1, so it broadcasts the $\mathsf{pre\text{-}vote}_0(1)$, $\mathsf{vote}_0(1)$, $\mathsf{main\text{-}vote}_0(1)$, and $\mathsf{final\text{-}vote}_0(1)$ messages simultaneously. Meanwhile, p_2 and p_3 propose 0, so they send $\mathsf{pre\text{-}vote}_0(0)$ to all replicas. After p_1 receives $f + 1$ $\mathsf{pre\text{-}vote}_0(0)$, it will also send a $\mathsf{pre\text{-}vote}_0(0)$ message to all replicas. However, it will not send $\mathsf{vote}_0(0)$, $\mathsf{main\text{-}vote}_0(0)$, or $\mathsf{final\text{-}vote}_0(0)$ messages, as it only sends any of these messages once. As p_2 and p_3 receive $n - f$ $\mathsf{pre\text{-}vote}_0(0)$ messages, they will send $\mathsf{vote}_0(0)$, $\mathsf{main\text{-}vote}_0(0)$, and $\mathsf{final\text{-}vote}_0(0)$. Hence, every replica receives two $\mathsf{final\text{-}vote}_0(0)$ messages and one $\mathsf{final\text{-}vote}_0(1)$. For p_2 or p_3 to proceed to round 1, it needs to *accept* $n - f$ $\mathsf{final\text{-}vote}_0()$ messages. According to our Quadratic-ABA specification, each replica accepts a $\mathsf{final\text{-}vote}_0(v)$ message only if it has received $f + 1$ $\mathsf{main\text{-}vote}_0(v)$ and $bset_r = \{0, 1\}$. Clearly, p_2 and p_3 accept $\mathsf{final\text{-}vote}_0(0)$ but will not accept $\mathsf{final\text{-}vote}_0(1)$, because they fail to receive $f + 1$ $\mathsf{main\text{-}vote}_0(1)$ messages (they might have $bset_r = \{0, 1\}$ if at least one of them reproposes). Therefore, the protocol may not terminate in this case.

Region	Ohio	Oregon	Singapore	Ireland
Ohio	-	24.5	55	39
Oregon	24.5	-	81	59
Singapore	55	81	-	90
Ireland	39	59	90	-

Table 4: One-way latency (ms) between any two regions on Amazon EC2. The regions we use to evaluate the protocols are Ohio (us-east-2), Oregon (us-west-2), Singapore (ap-southeast-1), and Ireland (eu-west-1).

D Setup Detail and Additional Evaluation Results

D.1 Detailed Evaluation Setup

We show in Table 4 one-way latency between any two regions on Amazon EC2. As shown in the table, the lowest one-way latency (resp., the highest latency) is 24.5 ms for Ohio-Oregon (resp., 90 ms for Ireland-Singapore).

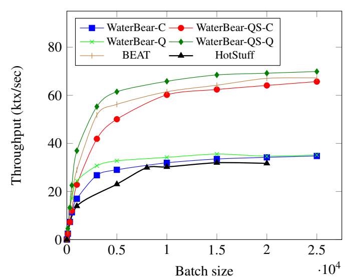
Let us explain why our latency result makes sense in such a setting. WaterBear-QC-Q, for example, has a RBC phase and a RABA phase. The RBC phase has 3 steps; the RABA phase in the optimistic mode just takes 1 step, as Quadratic RABA has a (coin-free) fast path that allows terminating in 1 step only. Namely, in the optimistic mode, WaterBear-QC-Q has 4 steps (i.e., about 4x average one-way latency).

The latency we reported for Water-QC-Q is 310 ms when $f = 1$ and $n = 4$, reasonably matching the one-way latency in Table 4: in practice, a replica delivers a transaction once receiving (fastest) $2f + 1$ replicas instead of all $3f + 1$ replicas.

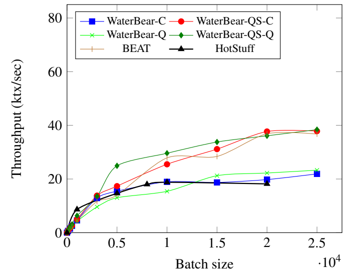
Consider another example for WaterBear-QS-C. WaterBear-QS-C in its optimistic mode has 6 steps. The number of steps for WaterBear-QS-C justifies its latency of 570 ms for $f = 1$ and $n = 4$.

D.2 Additional Evaluation Results

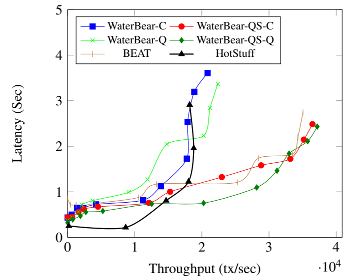
We provide additional evaluation results, including the performance comparison with HotStuff [65], the performance on different types of VMs, the performance with different transaction sizes, and the memory and the CPU usage for the asynchronous BFT protocols implemented.



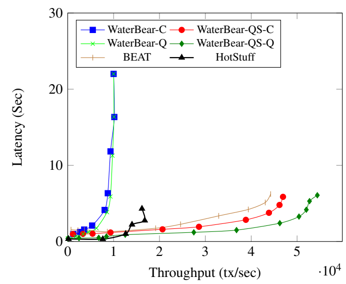
(a) Throughput in the LAN setting when $f = 1$.



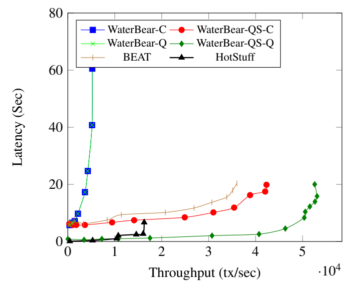
(b) Throughput in the WAN setting when $f = 1$.



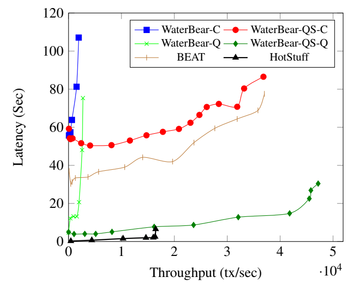
(c) Throughput vs. Latency when $f = 1$.



(d) Throughput vs. Latency when $f = 5$.



(e) Throughput vs. Latency when $f = 10$.



(f) Throughput vs. Latency when $f = 20$.

Figure 13: Throughput vs. latency on m5.xlarge instances for $f = 1$ to $f = 20$.

Performance compared with HotStuff. We compare the performance among WaterBear protocols and HotStuff [65], the state-of-the-art partially synchronous BFT. Different from the asynchronous BFT protocols we study in this work that are leaderless, HotStuff is a leader-based protocol. With the same batch size b, only the leader in HotStuff proposes b transactions, while in contrast, each replica in asynchronous BFT protocols can propose b transactions.

We first demonstrate the throughput result and the throughput vs. latency result in Figure 13. Our results show that our quantum secure asynchronous protocols (WaterBear-QS-C and WaterBear-QS-Q) consistently outperform HotStuff, achieving higher throughput. For instance, when $f = 20$, WaterBear-QS-Q achieves 1.65x throughput of HotStuff. The reason is that WaterBear-QS-Q is leaderless and all replicas can propose transactions.

Meanwhile, compared to the asynchronous protocols, HotStuff achieves much lower latency. For instance, when $f = 20$ and $b = 100$, HotStuff has 1/10 latency of WaterBear-QS-Q. The latency result is expected, because the message complexity of HotStuff is $O(n)$, in contrast to $O(n^3)$ in WaterBear protocols, and also because HotStuff is deterministic and has a fixed number of steps.

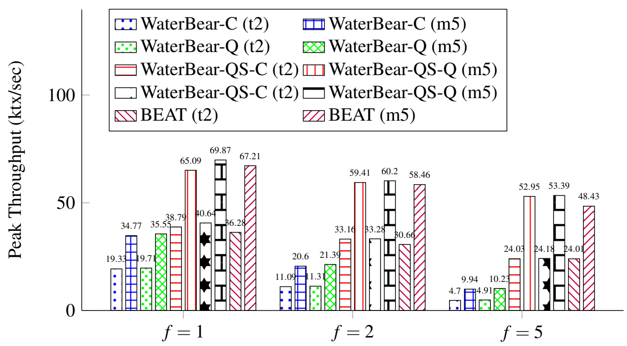
Performance on different types of VMs. Different from prior protocols (HoneyBadger, BEAT, Dumbo, EPIC) that all evaluate the performance on *t2.medium* instances, we evaluate the performance of the protocols using both *t2.medium* (t2 in the figures) and *m5.xlarge* (m5 in the figures) instances. In particular, we evaluate the throughput with $b = 15{,}000$ for $f = 1$, $f = 2$, and $f = 5$. The results are shown in Figure 14a. For all the protocols, the peak throughput on *m5.xlarge* instances is about $2\times$ that on *t2.medium*.

Performance with different transaction sizes. We also report the throughput of the protocols by fixing b to 15,000 but using different sizes of transactions (100 bytes and 250 bytes), the results of which are shown in Figure 14b. For all five protocols, the performance using transaction size of 100 bytes is consistently higher, being at least twice as efficient as that with 250 bytes. The finding highlights the main bottleneck for the protocols for large transaction sizes is RBC.

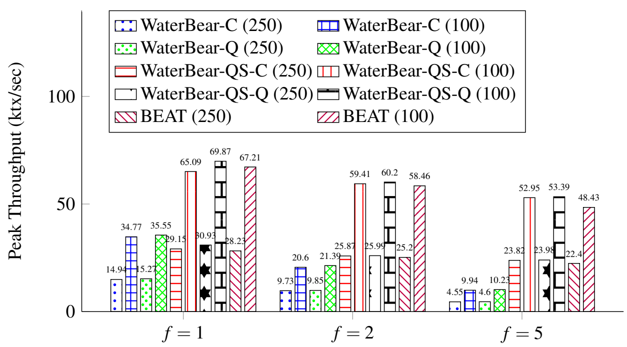
Memory and CPU usage. We present in Figure 15 memory and CPU usage for $n = 16$ and varying batch sizes. The results are obtained by using the *top* Linux monitoring tool. For the memory usage, all protocols consume higher memory when b increases. This is expected, since replicas need to process more transactions as the batch size grows. Meanwhile, WaterBear-C and WaterBear-Q consistently consume slightly higher memory than the other protocols, because the RBC for both protocols require more bandwidth. For the CPU usage, WaterBear-QS-C and WaterBear-QS-Q have lower CPU usage than other protocols, as these two protocols are PKC-free and use symmetric cryptography only.

E Proof of Cubic-ABA

We show that Cubic-ABA achieves validity, agreement, termination, and integrity.

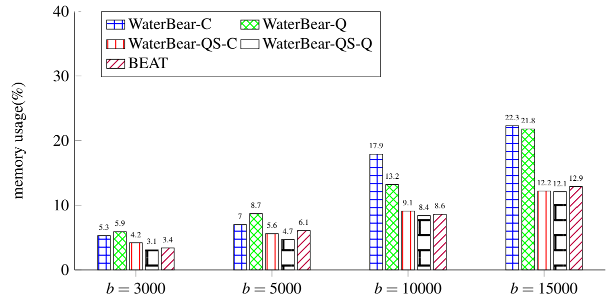


(a) Peak throughput of protocols running on different EC2 instances.

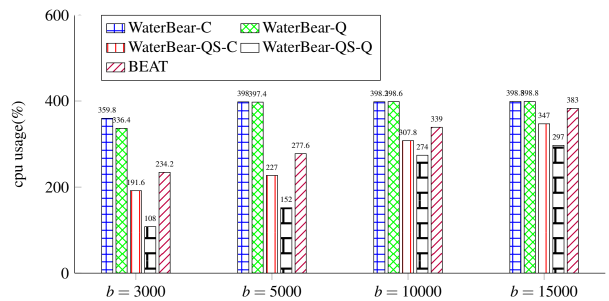


(b) Peak throughput for transaction size of 100 bytes and 250 bytes.

Figure 14: Performance of the protocols for $f = 1,2$, and 5.



(a) Memory.



(b) CPU.

Figure 15: Memory and CPU usage.

Lemma 1. *If all correct replicas propose $iv_r = v$ in round r, then any correct replica that enters round $r+1$ sets iv_{r+1} as v.*

Proof. If all correct replicas propose v in round r, every correct replica broadcasts $\mathsf{pre\text{-}vote}_r(v)$. No correct replica will forward $\mathsf{pre\text{-}vote}_r(\bar{v})$, as there are no more than $f+1$ $\mathsf{pre\text{-}vote}_r(\bar{v})$ messages. Hence, no correct replica will add $\bar{v}$ to $bset_r$. Furthermore, all correct replicas will eventually send $\mathsf{main\text{-}vote}_r(v)$ and *r-broadcast* $\mathsf{final\text{-}vote}_r(v)$. No correct replica accepts $\mathsf{final\text{-}vote}_r(\bar{v})$ or $\mathsf{final\text{-}vote}_r(*)$, since they only have v in their $bset_r$. Hence, any correct replica that enters round $r+1$ sets iv_{r+1} as v. ∎

Note that the lemma above holds for the case where a correct replica decides v in round r.

Lemma 2. *If all correct replicas propose v in round r, then for any $r' > r$, any correct replica that enters round r' sets $iv_{r'}$ as v.*

Proof. The proof is by induction on the round number. The base case holds for r according to Lemma 1. For the induction step, we show that the lemma holds for round $r'+1$. In other words, if all correct replicas propose $iv_{r'} = v$ in round r', then in round $r'+1$, any correct replica sets $iv_{r'+1}$ as v.

In round r', as no correct replica sends $\mathsf{pre\text{-}vote}_{r'}(\bar{v})$, no correct replica can receive $f+1$ $\mathsf{pre\text{-}vote}_{r'}(\bar{v})$ messages. In other words, no correct replica will forward $\mathsf{pre\text{-}vote}_{r'}(\bar{v})$. Meanwhile, no correct replica will accept $\mathsf{final\text{-}vote}_{r'}(\bar{v})$ or $\mathsf{final\text{-}vote}_{r'}(*)$ since correct replicas only have v in their $bset_{r'}$. Furthermore, every correct replica will *r-broadcast* $\mathsf{final\text{-}vote}_{r'}(v)$. Therefore, any correct replica that enters round $r'+1$ sets $iv_{r'+1}$ as v. ∎

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

Proof. We assume that a correct replica p_i terminates and decides $\bar{v}$ and prove the correctness by contradiction.

If p_i terminates and decides $\bar{v}$ in round 0, it will enter round 1 with $iv_1 = \bar{v}$. This is a contradiction with Lemma 1, as if all correct replicas propose v, any correct replica that enters round 1 sets iv_1 as v. If p_i terminates and decides $\bar{v}$ in round $r > 0$, it *r-delivers* $n-f$ $\mathsf{final\text{-}vote}_r(\bar{v})$. Similarly, it has put $\bar{v}$ in its $bset_r$. Therefore, at least one correct replicas has set $iv_r = \bar{v}$ and broadcast $\mathsf{pre\text{-}vote}_r(\bar{v})$. This is a contradiction with Lemma 2 since any correct replica that enters round r sets iv_r as v. This completes the proof of the theorem. ∎

Lemma 4. *If a correct replica p_i decides v in round r, any correct replica that enters round $r+1$ sets iv_{r+1} as v.*

Proof. If p_i decides v in round r, it *r-delivers* $n-f$ $\mathsf{final\text{-}vote}_r(v)$. In other words, at least $f+1$ correct replicas *r-broadcast* $\mathsf{final\text{-}vote}_r(v)$. We assume that a correct replica p_k enters round $r+1$ using value $iv_{r+1} = \bar{v}$ and prove the lemma by contradiction. If p_k sets iv_{r+1} as $\bar{v}$, there are three conditions: A) p_k *r-delivers* at least $n-f$ $\mathsf{final\text{-}vote}_r(\bar{v})$; B) p_k *r-*

delivers $f+1$ final-vote$_r(\bar{v})$; C) none of the conditions holds. In other words, p_k has received fewer than $f+1$ final-vote$_r(v)$ and fewer than $f+1$ final-vote$_r(\bar{v})$. We now show that none of the three conditions is possible.

Condition A): Replica p_k *r-delivers* $n-f$ final-vote$_r(\bar{v})$. We already know that at least $n-f$ replicas *r-broadcast* final-vote$_r(v)$. Therefore, at least one correct replica *r-broadcasts* both final-vote$_r(v)$ and final-vote$_r(\bar{v})$, a contradiction.

Condition B): Replica p_k *r-delivers* $f+1$ final-vote$_r(\bar{v})$. We already know that p_i *r-delivers* $n-f$ final-vote$_r(v)$. Therefore, at least one replica (correct or Byzantine) *r-broadcast* both final-vote$_r(\bar{v})$ and final-vote$_r(v)$ such that p_k *r-delivers* final-vote$_r(\bar{v})$ and p_i *r-delivers* final-vote$_r(v)$. This is a violation of the agreement property or RBC.

Condition C): Replica p_k *r-delivers* $n-f$ final-vote$_r()$ messages (let the set of replicas be S_1). Among the messages from S_1, fewer than $f+1$ are final-vote$_r(\bar{v})$ and fewer than $f+1$ are final-vote$_r(v)$. Other messages can only be final-vote$_r(*)$. We already know that p_i *r-delivers* $n-f$ final-vote$_r(v)$ (let the set of replicas be S_2). S_1 and S_2 have at least $n-2f \geq f+1$ replicas in common. Therefore, at least one replica *r-broadcasts* both v and $*$ (or $\bar{v}$) such that p_i *r-delivers* final-vote$_r(v)$ and p_k has *r-delivers* final-vote$_r(*)$ (or final-vote$_r(\bar{v})$), a violation of agreement property of RBC. ■

Theorem 5 (Agreement)**.** *If a correct replica decides v, then any correct replica that terminates decides v.*

Proof. We assume that a correct replica p_i decides v and another correct replica p_j decides $\bar{v}$ and prove the theorem by contradiction. There are two cases: 1) p_i and p_j decide in the same round r; 2) p_i and p_j decide in different rounds.

We first prove case 1). If replica p_i decides v in round r, it *r-delivers* $n-f$ final-vote$_r(v)$. If p_j decides $\bar{v}$, it *r-delivers* $n-f$ final-vote$_r(\bar{v})$. The two sets of $n-f$ replicas have at least $f+1$ replicas in common. Among the $f+1$ replicas, at least one is correct. Therefore, at least one correct replica must have *r-broadcast* both final-vote$_r(v)$ and final-vote$_r(\bar{v})$, a contradiction.

We now prove case 2) by assuming that p_i decides value v in round r and p_j decides $\bar{v}$ in round r' where $r' > r$.

According to Lemma 4, any correct replica enters round $r+1$ sets iv_{r+1} as v. Furthermore, according to Lemma 2, for any round $r'' \geq r+1$, any correct replica sets enters round r'' sets $iv_{r''}$ as v. If replica p_j decides value $\bar{v}$ in round r', at least one correct replica has set $iv_{r'}$ as $\bar{v}$ and sent pre-vote$_{r'}(\bar{v})$, a contradiction with Lemma 2. ■

Lemma 6. *Let $v_1 \in \{0,1\}$ and $v_2 \in \{0,1\}$. If a correct replica p_i r-delivers $f+1$* final-vote$_r(v_1)$ *and enters round $r+1$, another correct replica p_j r-delivers $f+1$* final-vote$_r(v_2)$ *and enters round $r+1$, then it holds that $v_1 = v_2$.*

Proof. If p_i *r-delivers* $f+1$ final-vote$_r(v_1)$, at least one correct replica *r-broadcasts* final-vote$_r(v_1)$. According to the protocol, the correct replica has received $n-f$ main-vote$_r(v_1)$. Therefore, for any other correct replicas, among the $n-f$ main-vote$_r()$ messages, at least one must be main-vote$_r(v_1)$. They either receive $n-f$ main-vote$_r(v_1)$ and *r-broadcast* final-vote$_r(v_1)$, or receive both main-vote$_r(v_1)$ and main-vote$_r(\bar{v}_1)$ and *r-broadcast* final-vote$_r(*)$. No correct replica will *r-broadcast* final-vote$_r(\bar{v}_1)$. For replica p_j, if it *r-delivers* $f+1$ final-vote$_r(v_2)$, at least one correct replica *r-broadcasts* final-vote$_r(v_2)$. Therefore, it must hold that $v_1 = v_2$. ■

Theorem 7. (Termination)**.** *Every correct replica eventually decides some value.*

Proof. The proof consists of two parts. First, in each round r, correct replicas will enter the next round. Second, the value iv_r used by any correct replica cannot be manipulated by the adversary.

We first show that in round r, correct replicas will enter the next round. In each round, every replica sets iv_r as either 0 or 1 in Cubic-ABA. Accordingly, at least $f+1$ correct replicas have the same $iv_r = v$. Therefore, all correct replicas will eventually receive $2f+1$ pre-vote$_r(v)$ for some v and send main-vote$_r()$ message. Correct replicas will have at least v in their $bset_r$ and *r-broadcast* either final-vote$_r(v)$ for some v or final-vote$_r(*)$. Similarly, any correct replica will eventually *r-deliver* $n-f$ final-vote$_r()$ messages and enter the next round.

We then show that if a correct replica p_i does not decide in round r, the value $iv_{r+1} = v$ cannot be manipulated by a malicious network scheduler such that correct replicas always enter the next round with inconsistent values. If p_i does not decide in round r, there are two conditions: A) p_i *r-delivers* $f+1$ final-vote$_r(v)$; B) p_i *r-delivers* $n-f$ final-vote$_r()$ messages. In the final-vote$_r()$ messages, fewer than $f+1$ are final-vote$_r(v)$ and fewer than $f+1$ are final-vote$_r(\bar{v})$. For condition B, a correct replica enters the next round with its local coin. The value of the local coin is independent with the value chosen by any correct replica. We now prove that the value v in condition A cannot be manipulated.

According to Lemma 6, if a correct replica receives $f+1$ final-vote$_r(v_1)$ and another correct replica receives $f+1$ final-vote$_r(v_2)$, then it holds that $v_1 = v_2$. If correct replicas use local coins to enter the next round, with a probability of $\frac{1}{2^{n-f}}$, replicas will enter the next round with the same value. The protocol will reach a state where agreement can be reached in 2^{n-f} expected rounds. After that, it takes another round for each replica to terminate, i.e., the protocol terminates in $2^{n-f}+1$ expected rounds. ■

Theorem 8 (Integrity)**.** *No correct replica decides twice.*

Proof. According to the protocol, after a correct replica decides some value, it participates in one more round of the protocol. However, it terminates the protocol after it *r-broadcasts* a final-vote$_r()$ message. Thus, the replica does not decide again in the following round. This completes the proof of the theorem. ■

F Proof of Quadratic-ABA

We show that Quadratic-ABA achieves validity, agreement, termination, and integrity.

Lemma 9. *If all correct replicas propose $iv_r = v$ in round r, then any correct replica that enters round $r+1$ sets iv_{r+1} as v.*

Proof. If all correct replicas propose $iv_r = v$ in round r, every correct replica broadcasts $\mathsf{pre\text{-}vote}_r(v)$. No correct replica will receive more than $f+1$ $\mathsf{pre\text{-}vote}_r(\bar{v})$ messages. Hence, no correct replica will add $\bar{v}$ to $bset_r$. Furthermore, all correct replicas will eventually send $\mathsf{vote}_r(v)$ and receive $n-f$ $\mathsf{vote}_r(v)$. As no correct replica ever has $\bar{v}$ in $bset_r$, all correct replica will not accept $\mathsf{vote}_r(\bar{v})$. Therefore, all correct replicas will send $\mathsf{main\text{-}vote}_r(v)$. No correct replica will accept $\mathsf{main\text{-}vote}_r(\bar{v})$ or $\mathsf{main\text{-}vote}_r(*)$ as $\bar{v} \notin bset_r$ and it cannot receive more than $f+1$ $\mathsf{vote}_r(\bar{v})$. Accordingly, every correct replicas will send $\mathsf{final\text{-}vote}_r(v)$ and receive $n-f$ $\mathsf{final\text{-}vote}_r(v)$. No correct replica accepts $\mathsf{final\text{-}vote}_r(\bar{v})$ as they only have v in their $bset_r$. Hence, any correct replica that enters round $r+1$ sets iv_{r+1} as v. ■

Note that the lemma above holds for the case where a correct replica decides v in round r.

Lemma 10. *If all correct replicas propose $iv_r = v$ in round r, then for any $r' > r$, any correct replica that enters round r' sets $iv_{r'}$ as v.*

Proof. The proof is by induction on the round number. The base case holds for r according to Lemma 9. For the induction step, we show that the lemma holds for round $r'+1$. In other words, if all correct replicas propose $iv_{r'} = v$ in round r', then in round $r'+1$, any correct replica sets $iv_{r'+1}$ as v.

In round r', as no correct replica sends $\mathsf{pre\text{-}vote}_{r'}(\bar{v})$, no correct replica can receive $f+1$ $\mathsf{pre\text{-}vote}_{r'}(\bar{v})$ messages. In other words, no correct replica will put $\bar{v}$ to $bset_{r'}$. Therefore, all correct replicas will send $\mathsf{vote}_{r'}(v)$ and no correct replicas will receive $f+1$ $\mathsf{vote}_{r'}(\bar{v})$. Accordingly, all correct replicas will send $\mathsf{main\text{-}vote}_r(v)$ and will not accept $\mathsf{main\text{-}vote}_{r'}(\bar{v})$. Any correct replica then only sends $\mathsf{final\text{-}vote}_{r'}(v)$. Meanwhile, no correct replica will accept $\mathsf{final\text{-}vote}_{r'}(\bar{v})$ or $\mathsf{final\text{-}vote}_{r'}(*)$ since correct replicas only have v in their $bset_{r'}$ and no correct replica can receive $f+1$ $\mathsf{main\text{-}vote}_{r'}(v)$. Furthermore, every correct replica will receive $n-f$ $\mathsf{final\text{-}vote}_{r'}(v)$. It is now clear that any correct replica that enters round $r'+1$ sets $iv_{r'+1}$ as v. ■

Lemma 11. *If a correct replica p_i sends $\mathsf{final\text{-}vote}_r(v)$, at least one correct replica has proposed $iv_r = \bar{v}$ and broadcast $\mathsf{pre\text{-}vote}_r(\bar{v})$.*

Proof. If p_i sends $\mathsf{final\text{-}vote}_r(v)$, it must have received $n-f$ $\mathsf{main\text{-}vote}_r(\bar{v})$. Among the replicas that sent $\mathsf{main\text{-}vote}_r(\bar{v})$, at least $f+1$ are correct. The correct replicas must have sent $\mathsf{vote}_r(\bar{v})$ and put $\bar{v}$ to $bset_r$. Each replica puts $\bar{v}$ to $bset_r$ only if it receives $n-f$ $\mathsf{pre\text{-}vote}_r(\bar{v})$. Therefore, at least one correct replicas has proposed $iv_r = \bar{v}$ and broadcast $\mathsf{pre\text{-}vote}_r(\bar{v})$. ■

Theorem 12 (Validity)**.** *If all correct replicas propose v, then any correct replica that terminates decides v.*

Proof. We assume that a correct replica p_i terminates and decides $\bar{v}$ and prove the correctness by contradiction.

If p_i terminates and decides $\bar{v}$ in round 0, it will enter round 1 with $iv_1 = \bar{v}$. This is a contradiction with Lemma 9. If p_i terminates and decides $\bar{v}$ in round $r > 0$, it receives $n-f$ $\mathsf{final\text{-}vote}_r(\bar{v})$. Among the replicas that sent $\mathsf{final\text{-}vote}_r(\bar{v})$, at least $f+1$ are correct. According to Lemma 11, at least one correct replica has broadcast $\mathsf{pre\text{-}vote}_r(\bar{v})$. This is a contradiction with Lemma 10 since any correct replica that enters round r sets iv_r as v. ■

Lemma 13. *If a correct replica p_i sends $\mathsf{main\text{-}vote}_r(v)$, any correct replica p_j only sends $\mathsf{main\text{-}vote}_r(v)$ or $\mathsf{main\text{-}vote}_r(*)$.*

Proof. If p_i sends $\mathsf{main\text{-}vote}_r(v)$, it has received $n-f$ $\mathsf{vote}_r(v)$. We assume that p_j sends $\mathsf{main\text{-}vote}_r(\bar{v})$ and prove the lemma by contradiction. If p_j sends $\mathsf{main\text{-}vote}_r(\bar{v})$, it has received $n-f$ $\mathsf{vote}_r(\bar{v})$. According to the protocol, every correct replica only sends $\mathsf{vote}_r()$ message once and each replica only sends either $\mathsf{vote}_r(v)$ or $\mathsf{vote}_r(\bar{v})$. Therefore, at least one correct replica has sent $\mathsf{vote}_r(v)$ to p_i and sent $\mathsf{vote}_r(\bar{v})$ to p_j, a contradiction. ■

Lemma 14. *If a correct replica p_i sends $\mathsf{final\text{-}vote}_r(v)$, any correct replica p_j only sends $\mathsf{final\text{-}vote}_r(v)$ or $\mathsf{final\text{-}vote}_r(*)$.*

Proof. If p_i sends $\mathsf{final\text{-}vote}_r(v)$, it has received $n-f$ $\mathsf{main\text{-}vote}_r(v)$. We assume that p_j sends $\mathsf{final\text{-}vote}_r(\bar{v})$ and prove the lemma by contradiction. If p_j sends $\mathsf{final\text{-}vote}_r(\bar{v})$, it has received $n-f$ $\mathsf{main\text{-}vote}_r(\bar{v})$. According to the protocol, every correct replica only sends $\mathsf{main\text{-}vote}_r()$ message once. Therefore, at least one correct replica has sent $\mathsf{main\text{-}vote}_r(v)$ to p_i and sent $\mathsf{main\text{-}vote}_r(\bar{v})$ to p_j, a contradiction. ■

Lemma 15. *If a correct replica p_i decides v in round r, any correct replica that enters round $r+1$ sets iv_{r+1} as v.*

Proof. If p_i decides v in round r, it receives $n-f$ $\mathsf{final\text{-}vote}_r(v)$. In other words, at least $f+1$ correct replicas have broadcast $\mathsf{final\text{-}vote}_r(v)$. We assume that a correct replica p_k enters round $r+1$ sets $iv_{r+1} = \bar{v}$ and prove the lemma by contradiction. If p_k sets iv_{r+1} as $\bar{v}$, there are three conditions: A) p_k receives at least $n-f$ $\mathsf{final\text{-}vote}_r(\bar{v})$; B) p_k only receives $\mathsf{final\text{-}vote}_r(\bar{v})$ and $\mathsf{final\text{-}vote}_r(*)$; C) none of the above holds. In other words, p_k receives only $\mathsf{final\text{-}vote}_r(*)$

or receives both $\mathsf{final\text{-}vote}_r(v)$ and $\mathsf{final\text{-}vote}_r(\bar{v})$. We now show that none of the three conditions is possible.

Condition A): Replica p_k receives $n-f$ $\mathsf{final\text{-}vote}_r(\bar{v})$. We already know that at least $n-f$ replicas have sent $\mathsf{final\text{-}vote}_r(v)$ as p_i receives $n-f$ $\mathsf{final\text{-}vote}_r(v)$. Therefore, at least one correct replica has sent both $\mathsf{final\text{-}vote}_r(v)$ and $\mathsf{final\text{-}vote}_r(\bar{v})$, a contradiction.

Condition B): Replica p_k receives $n-f$ $\mathsf{final\text{-}vote}_r(*)$ and $\mathsf{final\text{-}vote}_r(\bar{v})$ and has not received $\mathsf{final\text{-}vote}_r(v)$. We already know that p_i receives $n-f$ $\mathsf{final\text{-}vote}_r(v)$. Therefore, at least one correct replica has sent $\mathsf{final\text{-}vote}_r(v)$ to p_i and either $\mathsf{final\text{-}vote}_r(*)$ or $\mathsf{final\text{-}vote}_r(\bar{v})$ to p_k, a contradiction.

Condition C): Replica p_k receives only $\mathsf{final\text{-}vote}_r(*)$ or receives both $\mathsf{final\text{-}vote}_r(v)$ and $\mathsf{final\text{-}vote}_r(\bar{v})$. We know that p_i receives $n-f$ $\mathsf{final\text{-}vote}_r(v)$. Therefore, at least $f+1$ correct replicas have sent $\mathsf{final\text{-}vote}_r(v)$. If p_k receives $n-f$ $\mathsf{final\text{-}vote}_r()$ messages, at least one of them must be $\mathsf{final\text{-}vote}_r(v)$. In this case, if p_k enters round $r+1$ with iv_{r+1} as $\bar{v}$, p_k must have received at least one $\mathsf{final\text{-}vote}_r(\bar{v})$, as if p_k only receives $\mathsf{final\text{-}vote}_r(v)$ and $\mathsf{final\text{-}vote}_r(*)$, it will set iv_{r+1} as $\bar{v}$. If p_k accepts $\mathsf{final\text{-}vote}_r(v)$, it has received $f+1$ $\mathsf{main\text{-}vote}_r(v)$, among which at least one is sent by a correct replica. If p_k accepts $\mathsf{final\text{-}vote}_r(\bar{v})$, it has received $f+1$ $\mathsf{main\text{-}vote}_r(\bar{v})$, among which at least one is sent by a correct replica. This is a contradiction with Lemma 13. ■

Theorem 16 (Agreement)**.** *If a correct replica decides v, then any correct replica that terminates decides v.*

Proof. We assume that a correct replica p_i decides v and another correct replica p_j decides $\bar{v}$ and prove the theorem by contradiction. There are two cases: 1) p_i and p_j decide in the same round r; 2) p_i and p_j decide in different rounds.

We first prove case 1). If replica p_i decides v in round r, it receives $n-f$ $\mathsf{final\text{-}vote}_r(v)$. If p_j decides $\bar{v}$, it receives $n-f$ $\mathsf{final\text{-}vote}_r(\bar{v})$. The two sets of $n-f$ replicas have at least $f+1$ replicas in common. Among the $f+1$ replicas, at least one is correct. Therefore, at least one correct replica must have sent both $\mathsf{final\text{-}vote}_r(v)$ and $\mathsf{final\text{-}vote}_r(\bar{v})$, a contradiction.

We now prove case 2) by assuming that p_i decides value v in round r and p_j decides $\bar{v}$ in round r' where $r' > r$.

According to Lemma 15, if p_i decides v, any correct replica enters round $r+1$ sets iv_{r+1} as v. Furthermore, according to Lemma 10, for any round $r'' \geq r+1$, any correct replica that enters round r'' sets $iv_{r''}$ as v. If replica p_j decides value $\bar{v}$ in round r', it has received $n-f$ $\mathsf{final\text{-}vote}_r(v)$ so at least $f+1$ correct replicas have sent $\mathsf{final\text{-}vote}_r(v)$. According to Lemma 11, at least one correct replica has set $iv_{r'}$ as $\bar{v}$ and sent $\mathsf{pre\text{-}vote}_{r'}(\bar{v})$, a contradiction with Lemma 10. ■

Lemma 17. *If a correct replica p_i sends $\mathsf{vote}_r(v)$ for $v \in \{0,1\}$, any correct replica eventually accepts $\mathsf{vote}_r(v)$.*

Proof. If p_i sends $\mathsf{vote}_r(v)$ message, it has received $n-f$ $\mathsf{pre\text{-}vote}_r(v)$, among which at least $f+1$ are sent by correct replicas. Accordingly to the protocol, any correct replica that has not sent $\mathsf{pre\text{-}vote}_r(v)$ will also send $\mathsf{pre\text{-}vote}_r(v)$ upon receiving $f+1$ $\mathsf{pre\text{-}vote}_r(v)$. Therefore, every correct replica eventually sends $\mathsf{pre\text{-}vote}_r(v)$, receives $n-f$ $\mathsf{pre\text{-}vote}_r(v)$, and then adds v to $bset_r$. Hence, every correct replica eventually accepts $\mathsf{vote}_r(v)$. ■

Lemma 18. *If a correct replica p_i broadcasts a $\mathsf{main\text{-}vote}_r(v)$ or a $\mathsf{main\text{-}vote}_r(*)$ message given that $v \in \{0,1\}$, any correct replica accepts the $\mathsf{main\text{-}vote}_r()$ message from p_i.*

Proof. If p_i sends a $\mathsf{main\text{-}vote}_r(v)$ message, it has received and accepted $n-f$ $\mathsf{vote}_r(v)$, among which at least $f+1$ are sent by correct replicas. Accordingly to Lemma 17, every correct replica eventually accepts $\mathsf{vote}_r(v)$. After each correct replica receives $f+1$ $\mathsf{vote}_r(v)$, it accepts $\mathsf{main\text{-}vote}_r(v)$.

If p_i sends a $\mathsf{main\text{-}vote}_r(*)$ message, it must have received and accepted both $\mathsf{vote}_r(v)$ and $\mathsf{vote}_r(\bar{v})$, or it has received at least one $\mathsf{vote}_r(*)$. In any of the cases, p_i has put both 0 and 1 to $bset_r$. If p_i puts v to $bset_r$, it has received $2f+1$ $\mathsf{pre\text{-}vote}_r(v)$, among which at least $f+1$ are sent by correct replicas. Then any correct replica eventually receives $f+1$ $\mathsf{pre\text{-}vote}_r(v)$ and sends $\mathsf{pre\text{-}vote}_r(v)$. Every correct replica eventually receives $n-f$ $\mathsf{pre\text{-}vote}_r(v)$ and adds v to $bset_r$. Therefore, every correct replica eventually accepts $\mathsf{main\text{-}vote}_r(*)$. ■

Lemma 19. *If a correct replica p_i broadcasts a $\mathsf{final\text{-}vote}_r(v)$ or a $\mathsf{final\text{-}vote}_r(*)$ message given that $v \in \{0,1\}$, any correct replica accepts the $\mathsf{final\text{-}vote}_r()$ message.*

Proof. The lemma can be proved similarly as in Lemma 18. ■

Lemma 20. *Let $v_1 \in \{0,1\}$ and $v_2 \in \{0,1\}$. If a correct replica p_i receives only $n-f$ $\mathsf{final\text{-}vote}_r(*)$ and $\mathsf{final\text{-}vote}_r(v_1)$ messages, another correct replica p_j only receives $n-f$ $\mathsf{final\text{-}vote}_r(v_2)$ and $\mathsf{final\text{-}vote}_r(*)$ messages, $v_1 = v_2$.*

Proof. If p_i accepts $\mathsf{final\text{-}vote}_r(v_1)$, it has previously received $f+1$ $\mathsf{main\text{-}vote}_r(v_1)$, among which at least one is sent by a correct replica. If p_j accepts $\mathsf{final\text{-}vote}_r(v_1)$, it has previously received $f+1$ $\mathsf{main\text{-}vote}_r(v_2)$, among which at least one is sent by a correct replica. According to Lemma 13, it holds that $v_1 = v_2$. ■

Theorem 21 (Termination)**.** *Every correct replica eventually decides some value.*

Proof. The proof consists of two parts. First, in each round r, correct replicas will enter the next round. Second, the value iv_r used by any correct replica cannot be manipulated by the adversary.

We first show that in round r, correct replicas will enter the next round. In each round, every replica sets iv_r to either 0 or 1 in Quadratic-ABA. Accordingly, at least $f+1$ correct replicas have the same $iv_r = v$. All correct replicas will eventually receive $2f+1$ pre-vote$_r(v)$ for some v and send a vote$_r()$ message. Correct replicas will send either vote$_r(0)$ or vote$_r(1)$ and receive at least $n-f$ main-vote$_r()$ messages. For any correct replica, if it sends vote$_r(v)$ for $v \in \{0,1\}$, any correct replica will eventually accept vote$_r(v)$, according to Lemma 17. All correct replicas will then send either main-vote$_r(v)$ for $v \in \{0,1\}$ or main-vote$_r(*)$. According to Lemma 18, every correct replica eventually accepts any main-vote$_r()$ message sent by a correct replica. Then every correct replica either sends final-vote$_r(v)$ or final-vote$_r(*)$. According to Lemma 19, every correct replica accepts any final-vote$_r()$ message from a correct replica. Therefore, any correct replica will eventually receive $n-f$ final-vote$_r()$ messages and enter the next round.

We then show that if a correct replica p_i does not decide in round r, the value $iv_{r+1} = v$ cannot be manipulated by a malicious network scheduler such that correct replicas always enter the next round with inconsistent values. If p_i does not decide in round r, there are two conditions: A) p_i receives $n-f$ final-vote$_r(v)$ and final-vote$_r(*)$; B) p_i receives both final-vote$_r(v)$ and final-vote$_r(\bar{v})$ messages, or receives $n-f$ final-vote$_r(*)$. For condition B, a correct replica enters the next round with its local coin. The value of the local coin is independent with the value chosen by any correct replica. We now prove that the value v in condition A cannot be manipulated.

According to Lemma 20, if p_i receives $n-f$ final-vote$_r(v_1)$ and final-vote$_r(*)$ and p_j receives $n-f$ final-vote$_r(v_1)$ and final-vote$_r(*)$, $v_1 = v_2$. In other words, the value v used by any correct replica cannot be manipulated by the network scheduler.

If correct replicas use local coins to enter the next round, with a probability of $\frac{1}{2^{n-f}}$, replicas will enter the next round with the same value. Replicas will reach a state where agreement can be reached in 2^{n-f} expected rounds and execute the protocol for another round before terminating the protocol. Therefore, the protocol will terminate in $2^{n-f}+1$ expected rounds. ■

Theorem 22 (Integrity). *No correct replica decides twice.*

Proof. According to the protocol, after a correct replica decides some value, it participates in one more round of the protocol. However, it terminates the protocol after it receives a final-vote$_r()$ message. Hence, the replica does not decide again in the following round. ■

G CC-ABA

Both Cubic-ABA and Quadratic-ABA can be transformed to ABA from weak common coins [20, 56] and perfect common coins. Here by weak common coins, we mean that all correct replicas output 0 with probability $1/d$ and output 1 with probability $1/d$ where d is a constant and $d \geq 2$, and the probability that correct replicas obtain different values is $(d-2)/d$. By perfect common coins, we mean that all correct replicas always output the same random coin. Note perfect coins are a special case of weak coins (by setting $d = 2$).

As Quadratic-ABA is more efficient, we here focus on Quadratic-ABA. Our main result is that by replacing local coins of Quadratic-ABA with weak (or perfect) common coins, we immediately obtain CC-ABA terminating in $O(1)$ time. As shown in Figure 16, we only need to replace one line of code in Figure 3 to obtain CC-ABA.

```
replace ln 29 in Figure 3 using the following line
29      iv_{r+1} ← coin_r
{coin_r can be either a weak coin or a perfect common coin}
```

Figure 16: The CC-ABA protocol from weak coin or common coin.

CC-ABA reduces the expected number of steps of prior constructions, as shown in Table 5 and Table 6. Note that ABA is the major bottleneck in asynchronous BFT protocols as reported in [33, 44, 45]. The improvement is significant and has practical implications, as the recent work of PACE has shown that even a single step improvement can lead to a drastic performance improvement (for instance, easily 2x) in BFT protocols [66].

ABA (weak common coins)	steps/round	rounds
MMR15 [56, 2nd alg]	9 to 13	$d+1$
Crain [26, 1st alg]	5 to 7	$d+1$
CC-ABA (this work)	4 or 5	$d+1$

Table 5: ABA protocols using weak common coins. Rounds denote the expected number of rounds. The total number of steps is a product of steps/round and rounds.

ABA (common coins)	steps/round	rounds	good-case-coin-free
MMR15 [56, 2nd alg]	9 to 13	3	yes
Cobalt [52]	3 or 4	4	no
Crain [26, 1st alg]	5 to 7	3	yes
Crain [26, 2nd alg]	2 or 3†	4	no
Pillar [66]	2 or 3	4	no
CC-ABA (this work)	4 or 5	3	yes

Table 6: ABA protocols using perfect common coins. †The second algorithm of Crain relies high threshold common coins and is less efficient than Pillar. Compared to Pillar, CC-ABA has the good-case-coin-free property that is vital for the asynchronous distributed key generation protocol [30].

G.1 Proof of CC-ABA

We prove the correctness of CC-ABA that simply replaces the local coins of Quadratic-ABA with weak common coins or perfect common coins. We comment that our proof presented in the subsection applies to both cases.

Theorem 23. *CC-ABA achieves validity, agreement, and integrity.*

Proof. The three properties follow that of Quadratic-ABA. ■

Lemma 24. *If a correct replica receives and accepts both* $\textsf{final-vote}_r(v_1)$ *and* $\textsf{final-vote}_r(v_2)$ *such that* $v_1, v_2 \in \{0,1\}$, $v_1 = v_2$.

Proof. If a correct replica accepts $\textsf{final-vote}_r(v_1)$, it has previously received at least $f+1$ $\textsf{main-vote}_r(v_1)$. If the replica accepts $\textsf{final-vote}_r(v_2)$, it has previously received at least $f+1$ $\textsf{main-vote}_r(v_2)$. Therefore, at least one correct replica has sent $\textsf{main-vote}_r(v_1)$ and at least one correct replica has sent $\textsf{main-vote}_r(v_2)$. According to Lemma 13, if a correct replicas sends $\textsf{main-vote}_r(v_1)$, any correct replicas will only send $\textsf{main-vote}_r(v_1)$ or $\textsf{main-vote}_r(*)$. Therefore, we conclude that $v_1 = v_2$. ■

Theorem 25 (Termination)**.** *Every correct replica eventually decides some value.*

Proof. The proof consists of two parts. First, in each round r, correct replicas will enter the next round. Second, the value iv_r used by any correct replica cannot be manipulated by the adversary.

We first show that in round r, correct replicas will enter the next round. In each round, every replica sets iv_r as either 0 or 1. Thus, at least $f+1$ correct replicas have the same $iv_r = v$. All correct replicas will eventually receive $2f+1$ $\textsf{pre-vote}_r(v)$ for some v and send $\textsf{vote}_r()$ message. Correct replicas will send either $\textsf{vote}_r(0)$ or $\textsf{vote}_r(1)$ and receive at least $n-f$ $\textsf{main-vote}_r()$ messages. For any correct replica, according to Lemma 17, if it sends $\textsf{vote}_r(v)$ such that $v \in \{0,1\}$, any correct replica will eventually accept $\textsf{vote}_r(v)$. All correct replicas will then send either $\textsf{main-vote}_r(v)$ ($v \in \{0,1\}$) or $\textsf{main-vote}_r(*)$. According to Lemma 18, every correct replica eventually accepts any $\textsf{main-vote}_r()$ message sent by a correct replica. Then every correct replica either sends $\textsf{final-vote}_r(v)$ or $\textsf{final-vote}_r(*)$. Due to Lemma 19, every correct replica accepts any $\textsf{final-vote}_r()$ message from a correct replica. Therefore, any correct replica will eventually receives $n-f$ $\textsf{final-vote}_r()$ messages and enter the next round.

We then show that if a correct replica p_i does not decide in round r, the value $iv_{r+1} = v$ cannot be manipulated by a malicious network scheduler such that correct replicas always enter the next round with inconsistent values. If p_i does not decide in round r, there are two conditions: A) p_i receives $n-f$ $\textsf{final-vote}_r()$ messages with only $\textsf{final-vote}_r(v)$ and $\textsf{final-vote}_r(*)$; B) p_i receives both $\textsf{final-vote}_r(v)$ and $\textsf{final-vote}_r(\bar{v})$ messages, or receives $n-f$ $\textsf{final-vote}_r(*)$.

If condition A applies to at least two correct replicas, then according to Lemma 20, if p_i receives $n-f$ $\textsf{final-vote}_r(v_1)$ and $\textsf{final-vote}_r(*)$ and p_j receives $n-f$ $\textsf{final-vote}_r(v_1)$ and $\textsf{final-vote}_r(*)$, $v_1 = v_2$. In other words, the value v used by any correct replica cannot be manipulated by an adversary.

If condition B applies to at least two correct replicas, the correct replicas enter the next round with the weak common coin. With a probability of $2/d$, all correct replicas will have the same iv_{r+1} value. This value cannot be manipulated by an adversary.

We now show that if condition A applies to a correct replica p_i and condition B applies to a correct replica p_j, the values cannot be manipulated by an adversary. If p_j sets iv_{r+1} as the weak common coin value, it has either received $n-f$ $\textsf{final-vote}_r(*)$ or both $\textsf{final-vote}_r(v)$ and $\textsf{final-vote}_r(\bar{v})$. According to Lemma 24, the latter case is impossible. Therefore, p_j receives $n-f$ $\textsf{final-vote}_r(*)$. Hence, at least $f+1$ correct replicas have sent $\textsf{final-vote}_r(*)$. From Lemma 13, the correct replicas have previously sent either $\textsf{main-vote}_r(v)$ or $\textsf{main-vote}_r(*)$ for some $v \in \{0,1\}$. No correct replica will send $\textsf{main-vote}_r(\bar{v})$. If condition A applies to p_i and p_i sets iv_{r+1} as v_1 ($v_1 \in \{0,1\}$), p_i has received at least $f+1$ $\textsf{main-vote}_r(v_1)$. Since at least one correct replica has sent $\textsf{main-vote}_r(v_1)$ and no correct replica will send $\textsf{main-vote}_r(\bar{v})$, this value v_1 can only be v. Namely, the value iv_{r+1} cannot be manipulated by an adversary.

CC-ABA uses weak or perfect common coins. If correct replicas begin the protocol with different input values, replicas will reach a state where decisions can be made in expected $1 - \Sigma_{r=1}^{\infty} \frac{r}{d}(1-\frac{1}{d})^{r-1} = d$ rounds. After that, it takes another round for replicas to terminate the protocol. Thus, the expected number of rounds for CC-ABA using weak common coins is $d+1$. For the special case that uses perfect common coins, the expected number of rounds is 3. ■

H Proof of Cubic-RABA

We now show that Cubic-RABA achieves validity, unanimous termination, agreement, biased validity, biased termination, and integrity.

Lemma 26. *If all correct replicas propose v in round* 0 *and never repropose $\bar{v}$, then any correct replica enters the round 1 sets iv_1 as v.*

Proof. In round 0, all replicas send $\textsf{pre-vote}_0(v)$. No correct replica will receive $f+1$ $\textsf{pre-vote}_0(\bar{v})$ and send $\textsf{pre-vote}_0(\bar{v})$. Similarly, all correct replicas will send $\textsf{main-vote}_0(v)$ and will never accept $\textsf{main-vote}_0(\bar{v})$. All correct replicas will *r-broadcast* $\textsf{final-vote}_0(v)$ and will never accept $\textsf{final-vote}_0(\bar{v})$. Therefore, any correct replica that enters round 1 sets iv_1 as v. ■

Theorem 27 (Validity)**.** *If all correct replicas propose v and never repropose $\bar{v}$, then any correct replica that terminates decides v.*

Proof. We assume that a correct replica p_i terminates and decides $\bar{v}$ and prove the correctness by contradiction. If p_i

terminates and decides $\bar{v}$ in round 0, correctness follows from Lemma 26. We now prove the case where p_i decides in round $r > 0$.

Since Cubic-RABA follows Cubic-ABA starting from round 1, Lemma 2 holds for $r > 0$. If p_i terminates and decides $\bar{v}$ in round $r > 0$, it *r-delivers* $n-f$ $\text{final-vote}_r(\bar{v})$. Additionally, p_i has added $\bar{v}$ to its $bset_r$. Therefore, at least one correct replica has set iv_r as $\bar{v}$ and broadcast $\text{pre-vote}_r(\bar{v})$. This is a contradiction with Lemma 2 since any correct replica that enters round r sets iv_r as v. This completes the proof of the theorem. ■

Theorem 28 (Unanimous termination)**.** *If all correct replicas propose v and never repropose $\bar{v}$, then all correct replicas eventually terminate.*

Proof. If all correct replicas propose v and never repropose $\bar{v}$, all correct replicas only send $\text{pre-vote}_0(v)$. No correct replica will add $\bar{v}$ to $bset_0$. Furthermore, no correct replica will accept $\text{main-vote}_0(\bar{v})$ or $\text{final-vote}_0(\bar{v})$. Eventually all correct replicas will receive $2f+1$ $\text{pre-vote}_0(v)$, add v to $bset_0$, and broadcast $\text{main-vote}_0(v)$. Similarly, all correct replicas will eventually receive $n-f$ $\text{main-vote}_0(v)$ and *r-broadcast* $\text{final-vote}_0(v)$. All correct replicas will *r-deliver* $n-f$ $\text{final-vote}_0(v)$. In other words, all correct replicas will terminate and decide v. ■

Lemma 29. *If p_i decides v in round 0, any correct replica that enters round 1 sets iv_1 as v.*

Proof. If p_i decides v in round 1, it *r-delivers* $n-f$ $\text{final-vote}_0(v)$, among which at least $f+1$ replicas are correct. We assume that a correct replica p_k enters round 1 with $iv_1 = \bar{v}$ and prove the correctness by contradiction. If p_k enters round $r+1$ and sets iv_1 as $\bar{v}$, there are three conditions: A) p_k *r-delivers* at least $n-f$ $\text{final-vote}_r(\bar{v})$; B) p_k *r-delivers* $f+1$ $\text{final-vote}_0(\bar{v})$; C) p_k has not received more than $f+1$ $\text{final-vote}_0(v)$ and p_k has not received more than $f+1$ $\text{final-vote}_0(\bar{v})$. We now show that none of the three conditions is possible.

Condition A): Replica p_i *r-delivers* $n-f$ $\text{final-vote}_0(\bar{v})$. We already know that at least $f+1$ corect replicas *r-broadcast* $\text{final-vote}_0(v)$. Therefore, at least one correct replica *r-broadcasts* both $\text{final-vote}_0(v)$ and $\text{final-vote}_0(\bar{v})$, a contradiction.

Condition B): Replica p_k *r-delivers* $f+1$ $\text{final-vote}_0(\bar{v})$. We already know that p_i *r-delivers* $n-f$ $\text{final-vote}_0(v)$. Therefore, at least one replica (correct or Byzantine) *r-broadcasts* both $\text{final-vote}_0(\bar{v})$ and $\text{final-vote}_0(v)$ such that p_k *r-delivers* $\text{final-vote}_0(\bar{v})$ and p_i *r-delivers* $\text{final-vote}_0(v)$, a violation of the agreement property of RBC.

Condition C): Replica p_k *r-delivers* $n-f$ $\text{final-vote}_0()$ messages (let the set of replicas be S_1). In the messages, fewer than $f+1$ are $\text{final-vote}_0(\bar{v})$ and fewer than $f+1$ are $\text{final-vote}_0(v)$. Other messages must be $\text{final-vote}_0(*)$. We already know that p_i *r-delivers* $n-f$ $\text{final-vote}_0(v)$ (let the set of replicas be S_2). S_1 and S_2 have at least $2n-2f-n = n-2f \geq f+1$ replicas in common. In other words, at least one replica *r-broadcasts* a $\text{final-vote}_0()$ message such that p_i *r-delivers* $\text{final-vote}_0(v)$ and p_k *r-delivers* $\text{final-vote}_0(\bar{v})$ (or $\text{final-vote}_0(*)$), a violation of the agreement property of RBC. ■

Theorem 30 (Agreement)**.** *If a correct replica decides v, then any correct replica that terminates decides v.*

Proof. We assume that a correct replica p_i decides v and a correct replica p_j decides $\bar{v}$ and prove the theorem by contradiction. Since Cubic-RABA follows Cubic-ABA starting from round $r > 0$, if both p_i and p_j decide in round $r > 0$, correctness follows from the agreement property of Cubic-ABA. We now show the correctness in the following cases: 1) both p_i and p_j decide in round 0; 2) p_i decides in round 0 and p_j decides in round $r > 0$.
Case 1): If p_i decides v, it *r-delivers* $n-f$ $\text{final-vote}_0(v)$. If p_j decides $\bar{v}$, it *r-delivers* $n-f$ $\text{final-vote}_0(\bar{v})$. The two quorum of replicas have at least $n-2f$ replicas in common. Among the $n-2f$ replicas, at least one is correct since $n-2f \geq f+1$. Therefore, at least one correct replica *r-broadcasts* both $\text{final-vote}_0(v)$ and $\text{final-vote}_0(\bar{v})$, a contradiction since each replica only *r-broadcasts* a $\text{final-vote}_r()$ message once in each round.
Case 2): If p_j decides $\bar{v}$ in round $r = 1$, it has received at least $2f+1$ $\text{pre-vote}_1(\bar{v})$, where at least one correct replica has sent $\text{pre-vote}_1(\bar{v})$, a contradiction with Lemma 29. Starting from round 1, Cubic-RABA follows Cubic-ABA so that Lemma 2 holds. If p_j decides $\bar{v}$ in round $r > 1$, at least one correct replica must have sent $\text{pre-vote}_r(\bar{v})$, a contradiction with Lemma 2 since any correct replica sets iv_r as v.

This completes the proof of the theorem. ■

Lemma 31. *If $f+1$ correct replicas propose 1 in round 0, every replica either directly decides 1 in round 0 or/and enters round 1 with $iv_1 = 1$.*

Proof. If a correct replica p_i enters round 1, there are three conditions: A) p_i *r-delivers* $n-f$ $\text{final-vote}_0(v)$ with the same v; B) p_i *r-delivers* at least $f+1$ $\text{final-vote}_0(v)$ for some v; C) none of condition A or B holds. We show that $v=1$ for all three conditions and replicas will set iv_1 as $v=1$.

For condition A, we already know that at least $f+1$ correct replicas have broadcast $\text{final-vote}_0(1)$. Therefore, p_i must have received $n-f$ $\text{final-vote}_0(1)$. This is because if p_i receives $n-f$ $\text{final-vote}_0(0)$, at least one correct replica *r-broadcasts* both $\text{final-vote}_0(1)$ and $\text{final-vote}_0(0)$, a contradiction. In other words, p_i decides 1.

For condition B, we assume p_i *r-delivers* $f+1$ $\text{final-vote}_0(0)$ and prove the correctness by contradiction. If p_i *r-delivers* $f+1$ $\text{final-vote}_0(0)$, at least one correct replica *r-broadcasts* $\text{final-vote}_0(0)$. If the correct replica *r-broadcasts* $\text{final-vote}_0(0)$, the replica must have received $n-f$ $\text{main-vote}_0(0)$. We already know that at least $f+1$

correct replicas have sent $\mathsf{main\text{-}vote}_0(1)$. Any correct replica broadcasts $\mathsf{main\text{-}vote}_0()$ message once. In other words, at least one correct replica has broadcast both $\mathsf{main\text{-}vote}_0(0)$ and $\mathsf{main\text{-}vote}_0(1)$, a contradiction. Therefore, in this condition, p_i must have *r-delivered* $f+1$ $\mathsf{final\text{-}vote}_0(1)$. It is now clear that any correct replica uses $iv_1 = 1$ to enter round 1.

For condition C, any correct replica will use 1 as iv_1 since the local coin value is set as 1 in round 0. This completes the proof of the lemma. ■

Theorem 32 (Biased validity). *If $f+1$ correct replicas propose 1, then any correct replica that terminates decides 1.*

Proof. We assume that a correct replica p_i decides 0 and prove the correctness by contradiction. If p_i decides in round 0, correctness follows from Lemma 31. If p_i decides 0 in round $r > 0$, at least one correct replica has set iv_r as 0 and broadcast $\mathsf{pre\text{-}vote}_r(0)$. Since Cubic-RABA follows Cubic-ABA starting from round 1, Lemma 2 holds. Therefore, the claim that at least one correct replica has set iv_r as 0 is a contradiction with Lemma 2. This completes the proof of the theorem. ■

Theorem 33 (Biased termination). *Let Q be the set of correct replicas. Let Q_1 be the set of correct replicas that propose 1 and never repropose 0. Let Q_2 be correct replicas that propose 0 and later repropose 1. If $Q_2 \neq \emptyset$ and $Q = Q_1 \cup Q_2$, then each correct replica eventually terminates.*

Proof. The proof consists of two parts. First, every replica correct eventually enters the next round. Second, if a correct replica enters the next round with input v, v cannot be manipulated by the adversary.

We first prove that every replica eventually enters the next round. Since Cubic-RABA follows Cubic-ABA starting from round 1, this part follows from termination of Cubic-ABA. We only need to prove that every correct replica eventually enters round 1. For replicas in Q_1, they broadcast $\mathsf{pre\text{-}vote}_0(1)$ and add 1 to $bset_0$. For replicas in Q_2, they broadcast $\mathsf{pre\text{-}vote}_0(0)$ upon the $\mathsf{propose}(0)$ function, broadcast $\mathsf{pre\text{-}vote}_0(1)$ upon the $\mathsf{repropose}(1)$ function, and eventually add 1 to $bset_0$. There are two cases: 1) the size of Q_1 is greater than $f+1$; 2) the size of Q_1 is smaller than $f+1$.

For the first case, at least $f+1$ replicas in Q_1 will directly broadcast $\mathsf{main\text{-}vote}_0(1)$ and *r-broadcast* $\mathsf{final\text{-}vote}_0(1)$. For any correct replica p_i in Q_2, it may send $\mathsf{main\text{-}vote}_0(1)$ or $\mathsf{main\text{-}vote}_0(0)$. There are two sub-cases: none of the correct replicas send $\mathsf{main\text{-}vote}_0(0)$; at least one correct replica has sent $\mathsf{main\text{-}vote}_0(0)$. For the first sub-case, it is clear that every correct replica eventually receives and accepts $n-f$ $\mathsf{main\text{-}vote}_0(1)$, as every correct replica has 1 in its $bset_0$. Similarly, every correct replica will *r-broadcast* $\mathsf{final\text{-}vote}_0(1)$ and accept $n-f$ $\mathsf{final\text{-}vote}_0(1)$. For the second sub-case, if a correct replica p_i sends $\mathsf{main\text{-}vote}_0(0)$, it receives $2f+1$ $\mathsf{pre\text{-}vote}_0(0)$, among which at least $f+1$ are sent by correct replicas. Therefore, every correct replica will eventually receive $f+1$ $\mathsf{pre\text{-}vote}_0(0)$ and broadcast $\mathsf{pre\text{-}vote}_0(0)$. Every replica eventually adds 0 to $bset_0$. Since every correct replica has both 1 and 0 in $bset_0$, every correct replica accepts both $\mathsf{main\text{-}vote}_0(0)$ and $\mathsf{main\text{-}vote}_0(1)$. Similarly, every correct replica accepts both $\mathsf{final\text{-}vote}_0(0)$ and $\mathsf{final\text{-}vote}_0(1)$. In other words, every correct replica eventually enters the next round.

For the second case, replicas in Q_2 will send $\mathsf{pre\text{-}vote}_0(0)$ upon $\mathsf{propose}(0)$. They will send $\mathsf{pre\text{-}vote}_0(1)$ upon $\mathsf{repropose}(1)$ and add 1 to $bset_0$. Since the size of Q_2 is greater than $f+1$ (the size of Q_1 is smaller than $f+1$ and $Q = Q_1 \cup Q_2$), every replica will receive $f+1$ $\mathsf{pre\text{-}vote}_0(0)$, send $\mathsf{pre\text{-}vote}_0(0)$, and add 0 to $bset_0$. Furthermore, every correct replica in Q_2 broadcasts $\mathsf{pre\text{-}vote}_0(1)$ upon $\mathsf{repropose}(1)$. Since the size of Q_2 is greater than $f+1$, it holds that every correct replica eventually adds 1 to $bset_0$. Therefore, every replica will accept $\mathsf{main\text{-}vote}_0(0)$ and $\mathsf{main\text{-}vote}_0(1)$, $\mathsf{final\text{-}vote}_0(0)$, and $\mathsf{final\text{-}vote}_0(1)$. In other words, every correct replica eventually enters the next round.

We now prove the second part that the value iv used by any correct replica cannot be manipulated by the adversary. Since Cubic-RABA follows Cubic-ABA starting from round 1, correctness follows from Lemma 6 and termination of Cubic-ABA. ■

Theorem 34 (Integrity). *No correct replica decides twice.*

Proof. In each round, every replica only sends a $\mathsf{main\text{-}vote}_r()$ message and a $\mathsf{final\text{-}vote}_r()$ message once. Hence, only one value will be decided and integrity thus follows. ■

I Proof of Quadratic-RABA

We now show that Quadratic-RABA achieves validity, unanimous termination, agreement, biased validity, biased termination, and integrity.

Lemma 35. *If all correct replicas propose v in round 0 and never repropose $\bar{v}$, then any correct replica enters the round 1 sets iv_1 as v.*

Proof. If all correct replicas propose v in round 0, every correct replica broadcasts $\mathsf{pre\text{-}vote}_0(v)$. No correct replica will receive more than $f+1$ $\mathsf{pre\text{-}vote}_0(\bar{v})$ messages. Hence, no correct replica will add $\bar{v}$ to $bset_0$. Furthermore, all correct replicas will eventually send $\mathsf{vote}_0(v)$ and receive $n-f$ $\mathsf{vote}_0(v)$. As no correct replica ever has $\bar{v}$ in $bset_0$, no correct replica will accept $\mathsf{vote}_0(\bar{v})$. Therefore, all correct replicas will send $\mathsf{main\text{-}vote}_0(v)$. No correct replica will accept $\mathsf{main\text{-}vote}_0(\bar{v})$ or $\mathsf{main\text{-}vote}_0(*)$ as $\bar{v} \notin bset_0$ and no replicas will never receive more than $f+1$ $\mathsf{vote}_0(\bar{v})$. Accordingly, every correct replicas will send $\mathsf{final\text{-}vote}_0(v)$ and receive $n-f$ $\mathsf{final\text{-}vote}_0(v)$. No correct replica accepts $\mathsf{final\text{-}vote}_r(\bar{v})$ as they only have v in their $bset_r$ and no correct replica can receive more than $f+1$ $\mathsf{final\text{-}vote}_0(\bar{v})$. Hence, any correct replica that enters round $r+1$ sets iv_{r+1} as v. ■

Theorem 36 (Validity). *If all correct replicas propose v and never repropose $\bar{v}$, then any correct replica that terminates decides v.*

Proof. We assume that a correct replica p_i terminates and decides $\bar{v}$ and prove the correctness by contradiction. If p_i terminates and decides $\bar{v}$ in round 0, correctness follows from Lemma 35. We now prove the case where p_i decides in round $r > 0$.

Since Quadratic-RABA follows Quadratic-ABA starting from round 1, Lemma 10 holds for $r > 0$. If p_i terminates and decides $\bar{v}$ in round $r > 0$, it receives $n-f$ $\mathsf{final\text{-}vote}_r(\bar{v})$. Among the replicas that sent $\mathsf{final\text{-}vote}_r(\bar{v})$, at least $f+1$ are correct. According to Lemma 11, at least one correct replica has broadcast $\mathsf{pre\text{-}vote}_r(\bar{v})$. This is a contradiction with Lemma 10 since any correct replica that enters round r sets iv_r as v. This completes the proof of the theorem. ∎

Theorem 37 (Unanimous termination). *If all correct replicas propose v and never repropose $\bar{v}$, then all correct replicas eventually terminate.*

Proof. If all correct replicas propose v and never repropose $\bar{v}$, all correct replicas only send $\mathsf{pre\text{-}vote}_0(v)$. No correct replica will add $\bar{v}$ to $bset_0$. Furthermore, no correct replica will accept $\mathsf{vote}_0(\bar{v})$, $\mathsf{main\text{-}vote}_0(\bar{v})$, or $\mathsf{final\text{-}vote}_0(\bar{v})$. Eventually all correct replicas will receive $n-f$ $\mathsf{pre\text{-}vote}_0(v)$, add v to $bset_0$, and broadcast $\mathsf{vote}_0(v)$. Similarly, all correct replicas will eventually receive $n-f$ $\mathsf{vote}_0(v)$ and broadcast $\mathsf{main\text{-}vote}_0(v)$. All correct replicas will receive $n-f$ $\mathsf{main\text{-}vote}_0(v)$ and broadcast $\mathsf{final\text{-}vote}_0(v)$. In other words, all correct replicas will eventually receive $n-f$ $\mathsf{final\text{-}vote}_0(v)$ and decide v. ∎

Lemma 38. *If p_i decides v in round 0, any correct replica that enters round 1 sets iv_1 as v.*

Proof. If p_i decides v in round 1, it receives $n-f$ $\mathsf{final\text{-}vote}_0(v)$, among which at least $f+1$ are sent by correct replicas. We assume that a correct replica p_k enters round 1 with $iv_1 = \bar{v}$ and prove the correctness by contradiction. If p_k enters round $r+1$ and sets iv_1 as $\bar{v}$, there are three conditions: A) p_k receives at least $n-f$ $\mathsf{final\text{-}vote}_r(\bar{v})$; B) p_k receive only $\mathsf{final\text{-}vote}_0(\bar{v})$ and $\mathsf{final\text{-}vote}_0(*)$; C) none of the above applies. In case C), as p_j will use the common coin value 1 as iv_1, the case is impossible. We now show that none of the first two conditions is possible.

Condition A): Replica p_i receives $n-f$ $\mathsf{final\text{-}vote}_0(\bar{v})$. We already know that at least $f+1$ correct replicas have sent $\mathsf{final\text{-}vote}_0(v)$. Therefore, at least one correct replica sends both $\mathsf{final\text{-}vote}_0(v)$ and $\mathsf{final\text{-}vote}_0(\bar{v})$, a contradiction.

Condition B): Replica p_k receives $\mathsf{final\text{-}vote}_0(\bar{v})$ and $\mathsf{final\text{-}vote}_0(*)$. We already know that p_i receives $n-f$ $\mathsf{final\text{-}vote}_0(v)$. Therefore, at least one replica has sent $\mathsf{final\text{-}vote}_0(v)$ to p_i and a $\mathsf{final\text{-}vote}_0(\bar{v})$ (or $\mathsf{final\text{-}vote}_0(*)$ message) to p_j, a contradiction. ∎

Theorem 39 (Agreement). *If a correct replica decides v, then any correct replica that terminates decides v.*

Proof. We assume that a correct replica p_i decides v and a correct replica p_j decides $\bar{v}$ and prove the theorem by contradiction. Since Quadratic-RABA follows Quadratic-ABA starting from round $r > 0$, if both p_i and p_j decide in round $r > 0$, correctness follows from the agreement property of Quadratic-ABA. We now show the correctness in the following cases: 1) both p_i and p_j decide in round 0; 2) p_i decides in round 0 and p_j decides in round $r > 0$.
Case 1): If p_i decides v, it receives $n-f$ $\mathsf{final\text{-}vote}_0(v)$. If p_j decides $\bar{v}$, it receives $n-f$ $\mathsf{final\text{-}vote}_0(\bar{v})$. The two quorum of replicas have at least $n-2f$ replicas in common. Among the $n-2f$ replicas, at least one is correct since $n-2f \geq f+1$. Therefore, at least one correct replica sends both $\mathsf{final\text{-}vote}_0(v)$ and $\mathsf{final\text{-}vote}_0(\bar{v})$, a contradiction since each replica only sends a $\mathsf{final\text{-}vote}_r()$ message once in each round.
Case 2): If p_j decides $\bar{v}$ in round $r = 1$, it has received at least $n-f$ $\mathsf{pre\text{-}vote}_1(\bar{v})$, where at least one correct replica has sent $\mathsf{pre\text{-}vote}_1(\bar{v})$, a contradiction with Lemma 38. Starting from round 1, Quadratic-RABA follows Quadratic-ABA so that Lemma 10 holds. If p_j decides $\bar{v}$ in round $r > 1$, at least one correct replica must have sent $\mathsf{pre\text{-}vote}_r(\bar{v})$, a contradiction with Lemma 10 since any correct replica sets iv_r as v. ∎

Lemma 40. *If $f+1$ correct replicas propose 1 in round 0, every replica either directly decides 1 in round 0 or/and enters round 1 with $iv_1 = 1$.*

Proof. If a correct replica p_i enters round 1, there are three conditions: A) p_i receives $n-f$ $\mathsf{final\text{-}vote}_0(v)$ with the same v; B) p_i receives at least a $\mathsf{final\text{-}vote}_0(v)$ message for some v; C) none of condition A or B holds. We show that $v = 1$ for all three conditions and replicas will set iv_1 as $v = 1$.

For condition A, we already know that at least $f+1$ correct replicas have broadcast $\mathsf{final\text{-}vote}_0(1)$. If p_i receives $n-f$ $\mathsf{final\text{-}vote}_0(0)$, at least one correct replica has sent both $\mathsf{final\text{-}vote}_0(1)$ and $\mathsf{final\text{-}vote}_0(0)$, a contradiction. In other words, in this condition p_i decides 1.

For condition B, we p_i receives only $\mathsf{final\text{-}vote}_0(0)$ and $\mathsf{final\text{-}vote}_0(*)$. We already know that at least $f+1$ correct replicas have sent $\mathsf{final\text{-}vote}_0(1)$. Therefore, at least one correct replica must have sent both $\mathsf{final\text{-}vote}_0(1)$ and $\mathsf{final\text{-}vote}_0(0)$ (or $\mathsf{final\text{-}vote}_0(*)$), a contradiction.

For condition C, any correct replica will use 1 as input for round 1 since the local coin value is set as 1 in round 0. ∎

Theorem 41 (Biased validity). *If $f+1$ correct replicas propose 1, then any correct replica that terminates decides 1.*

Proof. If p_i decides in round 0, correctness follows from Lemma 40. If p_i decides 0 in round $r > 0$, at least one correct replica has set iv_r as 0 and broadcast $\text{pre-vote}_r(0)$. Since Quadratic-RABA follows Quadratic-ABA starting from round 1, Lemma 10 holds. Therefore, the claim that at least one correct replica has set iv_r as 0 is a contradiction with Lemma 10. This completes the proof of the theorem. ■

Lemma 42. *If $f+1$ correct replicas propose 1 in round 0, every correct replica eventually accepts* $\text{final-vote}_0(1)$.

Proof. If $f+1$ correct replicas propose 1, they will directly broadcast $\text{pre-vote}_0(1)$, $\text{vote}_0(1)$, $\text{main-vote}_0(1)$, and $\text{final-vote}_0(1)$. Every correct replica will eventually receive $f+1$ $\text{pre-vote}_0(1)$. For those correct replicas that have not sent $\text{pre-vote}_0(1)$, they will also broadcast $\text{pre-vote}_0(1)$. Therefore, every correct replica eventually adds 1 to $bset_0$. As $f+1$ correct replicas broadcast $\text{vote}_0(1)$, every correct replica eventually accepts $\text{main-vote}_0(1)$ message. Similarly, as $f+1$ correct replicas broadcast $\text{main-vote}_0(1)$, every correct replica eventually accepts $\text{final-vote}_0(1)$. ■

Lemma 43. *If a correct replica p_i sends $\text{final-vote}_r(0)$ or $\text{final-vote}_r(*)$, every correct replica eventually accepts the $\text{final-vote}_r()$ message sent by p_i.*

Proof. Case 1: If a correct replica p_i sends $\text{final-vote}_r(0)$, it has received $n-f$ $\text{main-vote}_r(0)$, among which at least $f+1$ are sent by correct replicas. Furthermore, the correct replica has put 0 in its $bset_r$ so it receives $n-f$ $\text{pre-vote}_r(0)$. As $f+1$ correct replicas have sent $\text{pre-vote}_r(0)$, every correct replica eventually receives $f+1$ $\text{pre-vote}_r(0)$ and send $\text{pre-vote}_r(0)$. Accordingly, every correct replica puts 0 in $bset_r$. There are two cases: $r=0$ and $r>0$. If $r=0$, every correct replica will accept $\text{main-vote}_r(0)$ and $\text{final-vote}_r(0)$. If $r>0$, every correct replica will accept $\text{vote}_r(0)$ as $0 \in bset_r$. Additionally, since $f+1$ correct replicas have sent $\text{main-vote}_r(0)$, they all have received $n-f$ $\text{vote}_r(0)$, among which at least $f+1$ are sent by correct replicas. As every correct replica receives $f+1$ $\text{vote}_r(0)$, they will accept $\text{main-vote}_r(0)$. As every correct replica eventually receive $f+1$ $\text{main-vote}_r(0)$, every correct replica will accept $\text{final-vote}_r(0)$.

Case 2: If a correct replica sends $\text{final-vote}_r(*)$, its $bset_r$ is $\{0,1\}$, i.e., it has received both $n-f$ $\text{pre-vote}_r(0)$ and $n-f$ $\text{pre-vote}_r(1)$. Following the prior case, every correct replica eventually has $bset_r = \{0,1\}$, so every correct replica accepts $\text{final-vote}_r(*)$. ■

Theorem 44 (Biased termination)**.** *Let Q be the set of correct replicas. Let Q_1 be the set of correct replicas that propose 1 and never repropose 0. Let Q_2 be correct replicas that propose 0 and later repropose 1. If $Q_2 \neq \emptyset$ and $Q = Q_1 \cup Q_2$, then each correct replica eventually terminates.*

Proof. The proof consists of two parts. First, every replica correct eventually enters the next round. Second, if a correct replica enters the next round with input v, v cannot be manipulated by the adversary.

We first prove that every replica eventually enters the next round. Since Quadratic-RABA follows Quadratic-ABA starting from round 1, termination for $r>0$ follows from termination of Cubic-ABA. We only need to prove that every correct replica eventually enters round 1. For replicas in Q_1, they broadcast $\text{pre-vote}_0(1)$ and add 1 to $bset_0$. For replicas in Q_2, they broadcast $\text{pre-vote}_0(0)$ upon the $\text{propose}(0)$ event, broadcast $\text{pre-vote}_0(1)$ upon the $\text{repropose}(1)$ event, and eventually add 1 to $bset_0$. There are two cases: 1) the size of Q_1 is greater than $f+1$; 2) the size of Q_1 is smaller than $f+1$.

For the first case, at least $f+1$ replicas in Q_1 will directly broadcast $\text{vote}_0(1)$, $\text{main-vote}_0(1)$, and $\text{final-vote}_0(1)$. For any correct replica p_i in Q_2, it may send $\text{vote}_0(1)$ or $\text{vote}_0(0)$. There are two sub-cases: none of the correct replicas send $\text{vote}_0(0)$; at least one correct replica has sent $\text{vote}_0(0)$. For the first sub-case, it is straightforward to see that every correct replica eventually receives and accepts $n-f$ $\text{vote}_0(1)$, as every correct replica has 1 in its $bset_0$. Similarly, every correct replica will send $\text{main-vote}_0(1)$ and accept $n-f$ $\text{main-vote}_0(1)$. Similarly, every correct replica will send $\text{final-vote}_0(1)$. According to Lemma 42, every correct replica eventually accepts $\text{final-vote}_0(1)$ so correct replicas will enter the next round. For the second sub-case, if a correct replica p_i sends $\text{vote}_0(0)$, it receives $n-f$ $\text{pre-vote}_0(0)$, among which at least $f+1$ are sent by correct replicas. Therefore, every correct replica will eventually receive $f+1$ $\text{pre-vote}_0(0)$ and broadcast $\text{pre-vote}_0(0)$. Every replica eventually adds 0 to $bset_0$. Since every correct replica has both 1 and 0 in $bset_0$, every correct replica accepts both $\text{vote}_0(0)$ and $\text{vote}_0(1)$. Similarly, every correct replica accepts both $\text{main-vote}_0(0)$ and $\text{main-vote}_0(1)$. In this case, every correct replica will eventually send a $\text{final-vote}_0()$ message. According to Lemma 42, every correct replica eventually accepts $\text{final-vote}_0(1)$. According to Lemma 43, any correct replica accepts the $\text{final-vote}_0()$ message sent by any correct replica. Therefore, every correct replica eventually enters the next round.

For the second case, replicas in Q_2 will send $\text{pre-vote}_0(0)$ upon $\text{propose}(0)$. They will send $\text{pre-vote}_0(1)$ upon $\text{repropose}(1)$ and add 1 to $bset_0$. Since the size of Q_2 is greater than $f+1$ (the size of Q_1 is smaller than $f+1$ and $Q = Q_1 \cup Q_2$), every replica will receive $f+1$ $\text{pre-vote}_0(0)$, send $\text{pre-vote}_0(0)$, and add 0 to $bset_0$. Furthermore, every correct replica in Q_2 broadcasts $\text{pre-vote}_0(1)$ upon $\text{repropose}(1)$. Since the size of Q_2 is greater than $f+1$, every correct replica eventually adds 1 to $bset_0$. According to the protocol, in round 0, every correct replica accepts $\text{main-vote}_0(v)$ and $\text{final-vote}_0(v)$ if v is added to $bset_0$. Therefore, every replica will accept both $\text{vote}_0(0)$ and $\text{vote}_0(1)$, and $\text{main-vote}_0()$ and $\text{final-vote}_0()$ with any value. Accordingly, every correct replica eventually enters the next round.

We now prove that the value iv used by any correct replica cannot be manipulated by the adversary. Since Quadratic-RABA follows Quadratic-ABA starting from round 1, correctness follows from Lemma 20 and termination of Quadratic-ABA. ■

Theorem 45 (Integrity)**.** *No correct replica decides twice.*

Proof. In each round, every replica only sends a $\mathsf{final\text{-}vote}_r()$ message once. Hence, only one value will be decided and integrity thus follows. ■