WaterBear: Information-Theoretic Asynchronous BFT Made Practical

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

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

Boxin Zhao
Tsinghua University
zhaoboxin@mail.tsinghua.edu.cn

ABSTRACT

This paper refutes the conventional wisdom that information-theoretic BFT is impractical. We design and implement WaterBear, the first practical information-theoretic asynchronous Byzantine fault-tolerant (BFT) protocol. We also present a more efficient, quantum-secure asynchronous BFT protocol, WaterBear-QS, which, compared to WaterBear, additionally uses a collision-resistant hash function.

We show that WaterBear and WaterBear-QS are efficient under both failure-free and failure scenarios, achieving comparable performance to the state-of-the-art asynchronous BFT protocols. In particular, our failure case evaluation is thus far the most comprehensive evaluation for asynchronous BFT settings.

KEYWORDS

asynchronous BFT, blockchain, information-theoretic BFT, quantum security

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 [10, 25, 75] and is used in various ways in hybrid blockchains. It is also well-known that BFT and Byzantine agreement (BA) are equivalent (possibility- and impossibility-wise), and both are fundamental building blocks for secure multi-party computation (MPC) achieving fairness and guaranteed output delivery [15, 16, 31, 57, 61]. This paper designs and implements the first practical information-theoretic (unconditionally secure) asynchronous BFT protocol, WaterBear BFT (or simply WaterBear), resolving a long-standing open problem in fault-tolerant distributed computing and cryptography. We also present a highly efficient BFT protocol without public-key cryptography, using only authenticated channels and hash functions.

Information-theoretic vs. computational security. Depending on the capacities of the adversary, any cryptographic or security protocols can be in one of the following two models:

- *Computational security*, where the adversary is restricted to probabilistic polynomial-time (PPT).
- *Information-theoretic (IT) security*, where the adversary is unbounded.

Computationally secure protocols assume the hardness of some intractability problems (e.g., RSA, Diffie-Hellman). These mathematical problems may, in the future, be proven to computationally easy, or broken or weakened due to newly developed cryptanalysis techniques or some technological breakthrough (e.g., quantum computer). In contrast, information-theoretic security provides everlasting security without relying on any unproven intractability assumptions: information-theoretic protocols are not only quantum secure but future-proof. Moreover, it is shown that information-theoretic protocols are natural candidates for building protocols secure under concurrent composition [34, 36, 56].

PKC vs. no PKC vs. quantum security. It is a major topic in fault-tolerant distributed computing to design various Byzantine-resilient protocols that rely on no public key cryptography (PKC). For instance, the first known practical partially synchronous BFT without PKC is PBFT [30]. Many other Byzantine-resilient protocols in various settings (e.g., secure causal atomic broadcast, atomic register) are known [42, 44, 53]. It is, however, an open problem if one could design practical PKC-free BFT in completely asynchronous environments. Existing asynchronous BFT protocols indeed assume various primitives from public-key cryptography (e.g., common coin, (threshold) signatures).

There are indeed many reasons why one may favor symmetric cryptography. First, symmetric cryptographic primitives are based on basic and well-studied primitives (blockciphers and hashes). Second, PKC is generally many orders of magnitude slower than symmetric cryptography.

Besides, classic symmetric cryptography is believed to be quantum-resistant. In contrast, public-key cryptography based on certain mathematical problems only (e.g., some lattice problems) are quantum-resistant, but we do not know, yet, how to build the efficient building blocks (e.g., threshold signatures, threshold common coins) needed for existing asynchronous BFT protocols based on these mathematical problems. Up to now, there is no quantum secure asynchronous BFT implemented.

Note that information-theoretic security implies quantum security; the reverse does not hold, for one could use non-quantum technologies to attack systems.

1.1 The Challenges of Building IT Asynchronous BFT

We divide asynchronous BFT protocols *ever implemented* into several categories: 1) the BKR paradigm of Ben-Or, Kelmer, and Rabin [16], including HoneyBadgerBFT[64], BEAT [45], and EPIC [59], 2) the CKPS paradigm of Cachin, Kusawe, Petzold, and Shoup [21], including SINTRA [23] and Dumbo [52], 3) the CNV paradigm of Correia, Neves, and Veríssimo [37], with RITAS as an implementation [65], 4) the DKSS paradigm of Danezis, Kokoris Kogias, Sonnino, and Spiegelman [55], implemented in Tusk [40], and 5) the recent DZ paradigm of Duan and Zhang [46], including PACE [46]. While the frameworks mark significant milestones in developing practical asynchronous BFT protocols, the instantiations derived from these frameworks rely critically on various cryptographic tools and are not information-theoretic or quantum-secure. In fact, it is difficult to modify these instantiations to make them information-theoretically secure or quantum secure because these constructions face a common hurdle, and each paradigm additionally has its own issues for the goal, as argued in the following.

A common hurdle—information-theoretic common coin and asynchronous Byzantine agreement (ABA). While the BFT protocols implemented share rather different structures, they all rely on common coin and/or ABA protocols as a central building block. Unfortunately, information-theoretic common coin and ABA protocols with unbounded security are rather inefficient. The classic local coin based ABA protocols of Ben-Or [14] and Bracha [18, 19] need an exponential expected running time. While later ABA constructions using asynchronous verifiable secret sharing (AVSS) are much more efficient, these protocols are not yet practical. First, information-theoretic AVSS is notoriously difficult to build. Even in the statistical setting (allowing errors), such a primitive is overly complex. For instance, to build AVSS, the approach of Canetti and Rabin [28] needs to begin with an information checking protocol (resembling signatures but working in the information-theoretic setting), then asynchronous recoverable sharing, then asynchronous weak secret sharing, and finally AVSS. The improved approach of Patra, Choudhury, and Rangan [69], while being simpler than that of Canetti and Rabin, remains complex, following the route of information-checking protocol, then asynchronous weak commitment, and then AVSS. Second, the transformation from AVSS to ABA in the information-theoretic setting is equally expensive, requiring running n^2 AVSS instances to generate a *single* weak coin that is common for correct replicas with a constant probability only [28] (where n is the total number of replicas). One can then use the weak common coins to build dedicated ABA protocols [28, 68] that are less efficient than state-of-the-art ABA protocols from perfect common coins.

Additional hurdles for each framework. The BKR framework requires running n independent ABA instances for each epoch. This would add a factor of n to the communication and message complexity of the AVSS approach. Namely, one would use at least n^3 expensive AVSS instances for a single epoch. The same argument applies to the recent DZ framework, as the DZ framework also requires running n instances of reproposable ABA (RABA), which shares the same message and communication complexity as ABA.

The CKPS paradigm relies on a reduction from BFT to multi-valued validated Byzantine agreement (MVBA). The definition of MVBA explicitly considers a computationally bounded adversary. Indeed, existing MVBA constructions require the transferability of (threshold) signatures. Hence, the framework does not directly lead to IT BFT protocols. For the same reason, the AMS MVBA of Abraham, Malkhi, and Spiegelman [7], NWH MVBA [6], Dumbo-MVBA [62], and Gao et al.'s MVBA [50] also consider computational security only and rely heavily on (threshold) signatures (and some other cryptographic tools).

Besides the common coin hurdle, the implementation for DAG-Rider, Tusk, extensively uses signatures and hash functions for RBC (Byzantine reliable broadcast). Theoretically, DAG-Rider could use Bracha's RBC for communication, but according to our evaluation, RBC is *the* major performance bottleneck for protocols using RBC. (For one thing, according to our evaluation, the throughput of using all 100 bytes transactions is about half the throughput for 250 bytes. Also, jumping ahead, even if our IT secure system uses just 1 parallel RBC, the instantiation using Bracha's RBC is way less efficient than the one using bandwidth-efficient RBC [24].) Considering DAG-Rider may need to run on average 7 parallel RBC instances sequentially, the approach in the specific model does not (seem to) lead to efficient instantiations either.

The CNV atomic broadcast protocol terminates using up to f non-validated multi-valued Byzantine agreement (MBA) instances. Even if being instantiated using a constant expected time ABA protocol, it has $O(n)$ expected running time. (In contrast, other efficient asynchronous BFT protocols implemented have either $O(1)$ or $O(\log n)$ expected running time.) Moreover, in CNV, each MBA instance relies on a local coin based ABA protocol [19] that terminates in an expected exponential number of rounds. Due to the large expected running time and the intrinsic inefficiency of the local coin based ABA protocol, as shown in RITAS (the instantiation of the CNV protocol), the CNV protocol does not scale to more than 10 replicas.

1.2 Our Contributions

This paper develops the asynchronous BFT framework of Duan and Zhang (the DZ paradigm) in the information-theoretic setting. The DZ paradigm uses Byzantine reliable broadcast (RBC) and reproposable ABA (RABA) (a variant of ABA) as building blocks and removes the two subphase bottleneck of the BKR paradigm, allowing RABA instances to run in a fully parallelizable manner. The concrete instantiation of the DZ paradigm, called PACE [46], uses a common coin based RABA protocol built on the CKS threshold PRF scheme [22], achieving computational security and static security only. In this work, we choose to work directly on local coin based ABA protocols and overcome their inherent performance and scalability limitations.

WaterBear ABA and WaterBear RABA. To our knowledge, up to now, only two local coin based ABA protocols have been proposed, namely Ben-Or's ABA [14] assuming $n > 5f$ and Bracha's ABA [18] assuming optimal resilience. Bracha's ABA has been the most efficient local coin based ABA protocol for nearly three decades. We propose a novel local coin based ABA—WaterBear ABA has 1.8x speedup in latency compared to Bracha's ABA. In particular, WaterBear ABA has 5 steps per round, while Bracha's ABA uses 9 steps. The improvement is significant, as local coin based ABA protocols may take an expected exponential number of rounds to terminate.

More importantly, WaterBear ABA is carefully designed to be readily modified to build an efficient RABA protocol, a core building block in the powerful DZ paradigm. Our RABA protocol, called WaterBear RABA, is as efficient as WaterBear ABA in terms of both steps and rounds. While our RABA protocol may not even terminate by itself, the overall DZ framework guarantees that the RABA protocol terminates in one round with a high probability! Indeed, as long as there are just $f + 1$ replicas, instead of $2f + 1$ replicas, propose 1, then a correct replica that terminates for the instance would decide 1 and the corresponding transactions would be delivered. Both local coin based WaterBear ABA and WaterBear RABA are inherently robust as they are unconditionally secure,

	optimal resilience?	IT secure?	no pkc?	quantum secure?	no trusted setup?	adaptive?	WAN?
SINTRA [23]	√						
RITAS [65]	√		√	√	√	√	
HoneyBadgerBFT [64]	√						√
BEAT [45]	√						√
Dumbo [52]	√						√
EPIC [59]	√					√	√
MiB [60]							√
Tusk [40]	√						√
PACE [46]	√						√
WaterBear-QS (this paper)	√		√	√	√	√	√
WaterBear (this paper)	√	√	√	√	√	√	√

Table 1: Comparison of efficient asynchronous BFT protocols.

which is in sharp contrast to other such protocols using various cryptographic primitives.

WaterBear BFT: unbounded security and beyond. WaterBear develops the DZ paradigm. Besides the critical information-theoretic RABA protocol, WaterBear has two other major differences compared to the prior DZ instantiation. First, we use an information-theoretic RBC, the Bracha's RBC, as the underlying RBC. Second, to achieve unconditional security and adaptive security, we use the technique of EPIC [59] to remove the threshold encryption scheme used in the DZ paradigm. Despite the differences, WaterBear does inherit the performance benefits of the DZ paradigm.

WaterBear assumes authenticated channels only and has *all* desirable properties a BFT protocol we could think of, being optimally resilient, achieving unconditional security and adaptive security, and relying on no trusted setup. We compare WaterBear with efficient asynchronous BFT protocols implemented in Table 1.

WaterBear-QS: no PKC and quantum security. We also present an asynchronous BFT protocol—WaterBear-QS, which does not achieve information-theoretic security but achieves quantum security for both safety and liveness properties. Also, WaterBear and WaterBear-QS the first asynchronous BFT protocols without public-key cryptography. The only difference between WaterBear and WaterBear-QS is that WaterBear-QS additionally uses a collision-resistant hash function.

What motivated us to design WaterBear-QS is a crucial observation from our evaluation that the performance bottleneck for WaterBear is not the ABA phase but the RBC phase. Our experiments show that the size of transactions matters the most for the throughput. For instance, we find the throughput using a 250B transaction is about half of the throughput for 100B, a somewhat surprising finding that has not been reported so far. As the ABA phase does not carry bulk data in both cases, the bottleneck must be the RBC phase. Hence, WaterBear-QS uses the AVID RBC of Cachin and Tessaro that leverages hash function based cross-checksum to reduce the communication complexity of RBC [24]. As the hash function is the only cryptographic tool used, WaterBear-QS is quantum secure. (In contrast, DAG-Rider and PACE only achieve quantum safety but not quantum liveness, as both of them rely on cryptographic common coin protocols.)

A new asynchronous BFT platform. Starting from HoneyBadgerBFT, existing efficient asynchronous BFT protocols, including BEAT, Dumbo, and EPIC, use the HoneyBadgerBFT programming framework using Python. We instead build a new platform using Golang. Our platform implements WaterBear, WaterBear-QS, and BEAT (one of the most efficient open-source asynchronous BFT libraries) [1, 45].

Evaluation in failure-free and failure scenarios. With deployment in 5 continents, we show that WaterBear and WaterBear-QS offer comparable performance to one of the state of the art asynchronous BFT protocols BEAT, which achieves much weaker security (computational security, static security, and trusted dealer needed).

It is believed that asynchronous BFT protocols are more robust than partially synchronous BFT protocols, for these asynchronous protocols do not assume any timing assumptions. It is unclear if these protocols would indeed perform as expected. It is interesting to understand to what extend failures and attacks would have an impact on these asynchronous systems. This paper aims to tackle the problem by carefully designing various failure and attack scenarios. Via extensive experiments, both WaterBear and WaterBear-QS are shown to be highly robust against these failures and attacks.

2 RELATED WORK

Asynchronous vs. partially synchronous BFT. Partially synchronous BFT protocols never violate safety, but they achieve liveness only when the network becomes synchronous [47]. As shown in [11], even for partially synchronous BFT protocols focusing on robustness [9, 33], their performance may reduce 78%-99% in failures or attacks scenarios. Additionally, partially synchronous protocols may experience zero throughput with a network scheduler [64]. In contrast, asynchronous BFT protocols do not rely on any timing assumptions and are intrinsically robust against timing and performance attacks.

Information-theoretic BFT in partially synchronous environments. There exist several partially synchronous BFT protocols that are information-theoretically secure or can be made information-theoretically secure. In particular, PBFT (the journal version) [30], PBFT (described in Castro's PhD thesis) [29], and Cachin's formulation for PBFT [20] assume authenticated channels but use cryptographic hash functions. These protocols are quantum resistant but not information-theoretically secure. They can, however, be modified to achieve information-theoretic security if removing the usage of the hash functions. Recently, Stern and Abraham proposed IT HotStuff, an information-theoretic, partially synchronous BFT protocol that uses $O(1)$ persistent storage and $O(n^2)$ messages, where each message contains a constant number of words [73].

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 [65] contains an adaptively secure asynchronous BFT (atomic broadcast) protocol, but as it relies on inefficient local coin based ABA, it is less efficient than other protocols in WAN or large-size networks. DAG-Rider [55] achieves adaptive security if using adaptively secure common coin protocols.

The situation for asynchronous environments is in sharp contrast to that of partially synchronous BFT protocols, most of which attain adaptive security [9, 30, 33, 43, 51, 54, 72].

WaterBear achieves adaptive security and information-theoretic security and significantly outperforms EPIC that is adaptively secure but not information-theoretically secure.

Quantum safety. A BFT protocol is quantum-secure, if its safety is quantum resistant (quantum safety) and its liveness is quantum resistant (quantum liveness) [55]. DAG-Rider [55] achieves quantum safety, even if when being instantiated using a cryptographic common coin protocol (e.g., [22, 58]). The BKR protocol and their descendants (e.g., HoneyBadger [64], MiB [60], PACE [46]) achieve quantum safety if using techniques from EPIC [59]. All the above-mentioned protocols, however, do not achieve quantum liveness. Tusk [40], an asynchronous BFT protocol that can be viewed as a variant of DAG-Rider, extensively uses signatures and hashes and achieves neither quantum safety nor quantum liveness.

(Information-theoretic) Byzantine agreement. Byzantine agreement (BA) is a central tool for both fault-tolerant distributed computing and cryptography. The condition $n \geq 3f+1$ is both necessary and sufficient for both synchronous and asynchronous BA protocols [70]. The celebrated impossibility result of Fischer, Lynch, and Paterson [49] implies that a randomized ABA protocol must have non-terminating executions. An ABA protocol may be $(1-\epsilon)$-terminating, where correct replicas terminate the protocol with an overwhelming probability, or almost-surely terminating, where correct replicas terminate the protocol with probability one.

For our purpose, we focus on ABA protocols in the information-theoretic setting with a computationally unbounded adversary. For almost-surely ABA, Ben-Or's ABA requires $n \geq 5f+1$ [14], while Bracha's ABA [18] 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 $n \geq 4f+1$ [48]. Abraham, Dolev, and Halpern [5] provide the first almost-surely ABA with polynomial efficiency (concretely, $O(n^2)$ expected running time) and optimal resilience. Bangalore, Choudhury, and Patra [12] improve the expected running time of [5] by a factor of n.

For $(1-\epsilon)$-terminating ABA, Canetti and Rabin [28] build an expected constant-round ABA protocol with optimal resilience. Patra, Choudhury, and Rangan [69] build a more efficient construction in terms of communication complexity.

Both almost-surely ABA and $(1-\epsilon)$-terminating ABA follow the classic framework of Feldman and Micali [48] 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 [71].) Unfortunately, the framework of using AVSS for common coins is prohibitively expensive, as we have argued in the introduction. Patra, Choudhury, and Rangan [69] also propose an approach for sharing multiple secrets simultaneously. While such an approach is useful to build more efficient multi-valued BA (MBA), it is unknown if it would yield more efficient ABA protocols. While, for instance, the CNV asynchronous BFT framework [37] does use MBA, it may run $O(n)$ consecutive MBA instances (which is inefficient).

VSS, AVSS, ACSS, and VSSR. Besides being a core tool for BA, verifiable secret sharing (VSS) [32] is also a core building block in secure multi-party computation (MPC). If an AVSS protocol additionally ensures that all correct replicas, not just $f+1$ replicas, have consistent shares, then the AVSS protocol is called asynchronous complete secret sharing (ACSS, originally called ultimate secret sharing in BKR [16]). ACSS is a direct building block for MPC protocols. Recently, Basu et al. provide an efficient approach to integrating verifiable secret sharing (not necessarily AVSS) with share recovery in BFT protocols to offer privacy guarantees for BFT protocols [13]. Their goal is similar to but different from that of ACSS. The most communication-efficient ACSS protocol with computational security without assuming trusted setup or PKI is due to AlHaddad, Varia, and Zhang [8]. Their construction works both the high threshold of $n-f$ and the conventional threshold of $f+1$.

Practical ABA protocols using common coins. Directly assuming the existence of common coins and authenticated channels, a number of expected constant-round ABA protocols have been proposed. The ABA protocol by Mostefaoui, Moumen, and Raynal (MMR) [66] terminates in 2 rounds on average, where each round has 2 or 3 steps. MMR ABA, however, has a liveness issue reported in [2, 74]. Namely, a network scheduler can force correct replicas to enter the next round of consensus with inconsistent values, causing the protocol not to terminate. While the authors present a protocol that can address the issue in their journal version [67] (9-13 steps in each round), Cobalt ABA [63] fixes the problem in a less expensive way, i.e., by modifying MMR ABA and having one additional step in *each* round. Duan and Zhang recently propose Pillar, a new common coin based ABA protocol that is live and as efficient as MMR ABA [46]. Crain [38] also proposes an ABA protocol with the same efficiency but assumes a high-threshold common coin protocol that is more difficult to build.

HoneyBadgerBFT [64] and BEAT [45] use MMR ABA in their proceeding versions. EPIC [59] and Dumbo [52] use Cobalt ABA. Recently, HoneyBadgerBFT and BEAT update their open-source implementations [1, 3] using Cobalt ABA. PACE uses Pilliar as the underlying ABA. HoneyBadgerBFT and Dumbo use Boldyreva's theshold signature (pairing based) [17] to realize the underlying common coin protocol. BEAT and PACE use the pairing-free threshold PRF scheme of Cachin, Kursawe, and Shoup [22]. EPIC uses an adaptively secure threshold signature of Libert, Joye, and Yung [58] for common coins.

These practical ABA protocols can be made IT secure if following the paradigm of Feldman and Micali [48] and Canetti and Rabin [28] (using AVSS). As we have argued, such ABA protocols are not efficient.

The CKS ABA protocol of Cachin, Kursawe, and Shoup [22] has 3 steps in the first round and 2 steps for the following rounds. CKS ABA additionally assumes the use of (threshold) signatures, an assumption that cannot be used in the information-theoretic setting.

Information-theoretic and universally composable MPC. As one of the most significant results in the area of cryptography and distributed computing, Ben-Or, Goldwasser, and Wigderson [15] and Chaum, Crépeau, and Damgård [31] provide generic feasibility results for perfect (information-theoretic and error-free) MPC with adaptive security. Kushilevitz, Lindell, and Rabin establish a framework for the security of protocols in the information-theoretic setting under concurrent composition [56]. Cohen, Coretti, Garay, and Zikas [34] provide a theoretic foundation for BA and MPC with probabilistic termination in the UC framework [26]. Asynchronous protocols in the UC framework are not guaranteed to (eventually) terminate because the UC adversary can delay the computation indefinitely. In light of this, Coretti, Garay, Hirt, and Zikas generalize the UC framework to the asynchronous fault-tolerant computing setting [36].

3 SYSTEM MODEL AND DEFINITIONS

3.1 System and Threat Model

This section describes the system model for our distributed computing protocols, where f out of n replicas may fail arbitrarily (Byzantine failures). The protocols we consider 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 [47]. The protocols in the paper achieve security merely under the assumption that messages transmitted among correct replicas are eventually delivered.
- **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.
- **Unbounded adversary**: Depending on the capacities of the adversary, a protocol may achieve *computational security*, where the adversary is bounded and restricted to probabilistic polynomial-time (PPT), or achieve *information-theoretic (IT) security*, where the adversary is unbounded. We have argued IT security is preferable compared to computational security, but constructing IT secure protocols is more challenging.
- **Adaptive corruptions**: Depending on how the adversary decides to corrupt parties, there are two types of corruptions: static corruptions and adaptive corruptions. In the static corruption model, the adversary is restricted to choose its set of corrupted replicas at the start of the protocol and cannot change this set later on. An adaptive adversary can choose its set of corrupted replicas at any moment during the execution of the protocol, based on the information it accumulated thus far (i.e., the messages observed and the states of previously corrupted replicas). There is a strong separation result that statically secure protocols are not necessarily adaptively secure [27, 39].

For our protocols, we may associate each protocol instance with a unique identifier id, tagging each messages in the instance with id. If no ambiguity arises, we may simply omit the identifiers.

3.2 Definitions

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

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

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 by Duan and Zhang [46]. 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 reproposes 1, it does so at most once. A replica terminates the protocol identified by *id* by generating a decide message.

RABA (biased toward 1) satisfies the following properties:

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

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 $f+1$ replicas, not simply all correct replicas, propose 1, then a correct replica that terminates decides 1. The property guarantees the DZ framework to have sufficient transactions delivered.

RBC. A Byzantine reliable broadcast (RBC) protocol [8, 19, 24, 41] is specified by *r-broadcast* and *r-deliver* such that the following properties hold:

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

Bracha's broadcast [18] has a bandwidth of $O(n^2|m|)$ and is IT secure, and AVID RBC due to Cachin and Tessaro [24] uses hash functions (with output length λ) to reduce the bandwidth to $O(n|m| + \lambda n^2 \log n)$. Other RBC instantiations, such as the recent one by Das, Xiang, and Ren [41], leverage hash functions and authenticated channels and can be used as the RBC for BFT with quantum safety.

4 WATERBEAR ABA FROM LOCAL COINS

We present WaterBear ABA, a new local coin based ABA protocol deviating markedly from previous ones. Compared to the state-of-the-art protocol, Bracha's ABA, that has 9 steps in each round [18], WaterBear ABA has only 5 steps—almost a 2x speedup. More importantly, WaterBear ABA is carefully designed such that it can be modified for an efficient RABA protocol.

Figure 1 describes the pseudocode of WaterBear ABA. WaterBear

```
initialization
  $r \leftarrow 0$                                                  {round}
func propose($v_{input}$)
  $iv_0 \leftarrow v_{input}$                                      {set input for round 0}
  start round 0
round $r$
  broadcast pre-vote$_r(iv_r)$                                     {▷ phase 1}
  upon receiving pre-vote$_r(v)$ from $f+1$ replicas
    if pre-vote$_r(v)$ has not been sent, broadcast pre-vote$_r(v)$
  upon receiving pre-vote$_r(v)$ from $2f+1$ replicas
    $bset_r \leftarrow bset_r \cup \{v\}$
  wait until $bset_r \neq \emptyset$                               {▷ phase 2}
    if main-vote$_r()$ has not been sent, broadcast main-vote$_r(v)$ where $v \in bset_r$
  upon receiving $n-f$ main-vote$_r()$ such that for each received main-vote$_r(b)$, $b \in bset_r$
    if there are $n-f$ main-vote$_r(v)$
      r-broadcast final-vote$_r(v)$                                {▷ phase 3}
    else r-broadcast final-vote$_r(*)$
  upon r-delivering $n-f$ final-vote$_r()$ such that for each final-vote$_r(v)$, $v \in bset_r$; for each final-vote$_r(*)$, $\{0,1\} \subseteq bset_r$
    if there there are $n-f$ final-vote$_r(v)$
      decide $v$
    else if there are $f+1$ final-vote$_r(v)$
      $iv_{r+1} \leftarrow v$
    else
      $c \leftarrow Random()$                                      {obtain local coin}
      $iv_{r+1} \leftarrow c$
  $r \leftarrow r+1$
```

Figure 1: WaterBear ABA. $v \in \{0, 1\}$**.**

ABA uses the *broadcast* primitive of best-effort broadcast and the *r-broadcast* and *r-deliver* primitives of RBC. The protocol proceeds in rounds, beginning with round 0. Each round consists of three phases. In the first phase, a replica p_i broadcasts pre-vote$_r(iv_r)$, where $iv_r \in \{0,1\}$ is the input value of p_i for round r. If p_i receives $f+1$ pre-vote$_r(v)$ for some $v \in \{0,1\}$ and has not previously broadcast pre-vote$_r(v)$, it also broadcasts pre-vote$_r(v)$. If p_i receives $2f+1$ pre-vote$_r(v)$, it adds v to its $bset_r$, a set consisting only 0 and/or 1. Let v be the first value added to $bset_r$ for p_i. p_i enters the second phase by broadcasting a main-vote$_r(v)$ message.

A correct replica p_i accepts a main-vote$_r(v)$ message only if v has already been added locally to $bset_r$. If p_i receives $n-f$ main-vote$_r()$ message from different replicas, it enters the third phase. In the third phase, if p_i has received $n-f$ main-vote$_r(v)$, p_i *r-broadcasts* a final-vote$_r(v)$ message. Otherwise, p_i *r-broadcasts* final-vote$_r(*)$, where $*$ is a distinguished symbol denoting that $* \notin \{0,1\}$.

We say a final-vote$_r()$ message is valid for p_i if one of the following conditions hold:

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

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

- 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), a step needed for all such ABA protocols.

- If p_i *r-delivers* only $f+1$ valid final-vote$_r(v)$ for some $v \in \{0,1\}$, p_i uses v as input to enter the next round.
- Otherwise, a replica generates a local random coin and uses it as input for the next round.

Analysis. Our motivation for designing a new ABA from local coins is to reduce the cost of running multiple RBCs in Bracha's ABA. We recall Bracha's ABA in Appendix A. Bracha's ABA has three phases. In each phase, every replica disperses its value via a RBC instance. In total, there are n parallel RBC instances in each of the three phases. We reduce the number of RBC instances by 2/3, motivated by the design of signature-free common coin based ABA protocols [38, 46, 63, 66, 67]. In the first two phases, we let replicas send their values by simply best-effort broadcasting them to all replicas. The first phase ensures that all correct replicas can acknowledge the same set of values. 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 $*$ (denoting a vote that is neither 0 nor 1). In the third phase, we still rely on RBC, ensuring that all correct replicas eventually receive consistent values, even in the presence Byzantine replicas. In this way, the number of n parallel RBC instances is 1 instead of 3, and the number of steps is reduced from 9 to 5.

Equally important, WaterBear ABA can be readily modified to be a RABA protocol, as shown in the following section.

5 THE WATERBEAR RABA PROTOCOL

We present a WaterBear RABA protocol based on WaterBear ABA. The RABA primitive, as shown in DZ [46], may make the protocol terminate faster than conventional ABA. In the context of asynchronous BFT, the DZ framework with RABA significantly outperforms the conventional BKR diagram. We aim to design a RABA protocol without trusted setup by using local coins.

The pseudocode of WaterBear RABA protocol is shown in Figure 2. WaterBear RABA is identical to WaterBear 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, a replica p_i executes the *broadcast-vote*(v) function and starts round 0. Upon the *repropose*(v) function, 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 still executes the *broadcast-vote*(v) event and broadcasts a pre-vote$_0(v)$ message. Another interesting fact is that since we make WaterBear RABA biased toward 1, each correct replica only calls *repropose*(1) if it previously has called the *propose*(0) function.

Second, in the *broadcast-vote*(v) function, a replica broadcasts a pre-vote$_0(v)$ message. If $v = 1$, the replica adds 1 to $bset_0$. If the replica has not previously broadcast main-vote$_0$(1), it broadcasts main-vote$_0$(1). If the replicas has not previously *r-broadcast* final-vote$_0$(1), it *r-broadcasts* final-vote$_0$(1). Furthermore, in round 0, if a replica receives $f+1$ pre-vote$_0$(1) and has not broadcast main-vote$_0$() or final-vote$_0$(), it also directly broadcasts main-vote$_0$(1) and *r-broadcasts* final-vote$_0$(1).

```
initialization
  $r \leftarrow 0$                                                    {▷ round}
func $propose(v)$
  $broadcast\text{-}vote(v)$
  start round 0
func $repropose(v)$
  $broadcast\text{-}vote(v)$
func $broadcast\text{-}vote(v)$                                      {▷ phase 1}
  if pre-vote$_0(v)$ has not been sent, broadcast pre-vote$_0(v)$
  if $v = 1$
    $bset_0 \leftarrow bset_0 \cup \{1\}$
    if main-vote$_0()$ has not been sent, broadcast main-vote$_0(1)$
    if final-vote$_0()$ has not been sent, r-broadcast final-vote$_0(1)$
round r
  if $r > 0$, broadcast pre-vote$_r(iv_r)$
  upon receiving pre-vote$_r(v)$ from $f+1$ replicas
    if pre-vote$_r(v)$ has not been sent, broadcast pre-vote$_r(v)$
    if $r = 0$ and $v = 1$
      $bset_0 \leftarrow bset_0 \cup \{1\}$
      if main-vote$_0()$ has not been sent, broadcast main-vote$_0(1)$
      if final-vote$_0()$ has not been sent, r-broadcast final-vote$_0(1)$
  upon receiving pre-vote$_r(v)$ from $2f+1$ nodes
    $bset_r \leftarrow bset_r \cup \{v\}$
  wait until $bset_r \neq \emptyset$                                  {▷ phase 2}
    if main-vote$_r()$ has not been sent, broadcast main-vote$_r(v)$ where $v \in bset_r$
  upon receiving $n-f$ main-vote$_r()$ such that 1) final-vote$_r()$ has not been sent; 2) for each received main-vote$_r(b)$, $b \in bset_r$
    if there are $n-f$ main-vote$_r(v)$
      r-broadcast final-vote$_r(v)$                                   {▷ phase 3}
    else r-broadcast final-vote$_r(*)$
  upon r-delivering $n-f$ final-vote$_r()$ such that for each final-vote$_r(v)$, $v \in bset_r$; for each final-vote$_r(*)$, $\{0,1\} \subseteq bset_r$
    if there are $n-f$ final-vote$_r(v)$
      decide $v$
    else if there are $f+1$ final-vote$_r(v)$
      $iv_{r+1} \leftarrow v$
    else
      if $r = 0$, $iv_{r+1} \leftarrow 1$
      else $iv_{r+1} \leftarrow Random()$
  $r \leftarrow r+1$
```

Figure 2: WaterBear RABA. $v \in \{0,1\}$**.**

Finally, the coin value for round 0 is set to 1. In round $r \geq 1$, WaterBear RABA is identical to WaterBear ABA.

Analysis. The proof of WaterBear RABA is shown in Appendix C. We show that the three changes we made on top of WaterBear ABA can turn WaterBear ABA into a RABA protocol. The first change is made for the *biased termination* property. In particular, it ensures 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 the third changes are made for the biased validity property. If $f+1$ correct replicas propose 1, they will directly add 1 to $bset_0$, broadcast pre-vote$_0$(1), main-vote$_0$(1), and *r-broadcast* final-vote$_0$(1). In other words, no correct replica is able to receive $n-f$ main-vote$_0$(0) to *r-broadcast* final-vote$_0$(0). Furthermore, no correct replica can receive $n-f$ final-vote$_0$(0) or even $f+1$ final-vote$_0$(0). Even in the extreme case that a correct

```
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$_j$ such that $j \in S$
        a-deliver($\cup_{j \in S}\{m_j\}$) according to some deterministic order
```

Figure 3: The WaterBear protocol. The code for replica p_i in epoch e. WaterBear uses Bracha's broadcast as the underlying the RBC and WaterBear RABA as the underlying RABA. WaterBear uses the technique of EPIC to select transactions for each replica.

replica uses the local coin to enter the next round, the coin value is also 1. Therefore, WaterBear RABA achieves biased validity. Other properties of WaterBear RABA simply follow from WaterBear ABA, as we only modify round 0 of the protocol.

Note a RABA primitive may on average make replica decide 1 *faster* than conventional ABA. It may also reduce the number of steps in the optimal case. In particular, if all replicas propose 1, they will add 1 to $bset_0$, and directly broadcast $\text{final-vote}_0(1)$, $\text{main-vote}_0(1)$, and *r-broadcast* $\text{final-vote}_0(1)$, all simultaneously. After replicas *r-deliver* $n - f$ $\text{final-vote}_0(1)$, they can decide. Hence, in the optimal case, replicas decide in only three steps using RBC.

6 WATERBEAR AND WATERBEAR-QS

This section describes our asynchronous BFT protocols—WaterBear and WaterBear-QS. Both protocols are quantum secure and WaterBear is additionally information-theoretically secure.

6.1 The WaterBear Protocol

WaterBear follows the DZ paradigm but uses the trick in EPIC to avoid he 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 3 depicts the pseudocode of WaterBear. In terms of transaction selection strategy, we follow EPIC and ask replicas to select random transactions *in plaintext* for most epochs and periodically switch to the FIFO selection, where replicas maintain a log of transactions according to the order transactions are received and replicas select the first group of transactions in the buffer as input. As shown in EPIC, the approach shares similar performance as the random selection approach used in HoneyBadgerBFT and BEAT. Following the DZ 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 DZ paradigm.

6.2 The WaterBear-QS Protocol

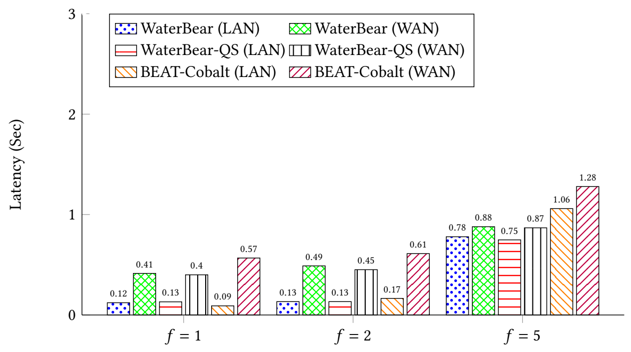
We now present WaterBear-QS, an asynchronous BFT protocol that does not achieve information-theoretic security but achieves quantum security for both safety and liveness properties. Prior to our work, no such BFT protocol has been implemented. DAG-Rider and PACE only achieve quantum safety but not quantum liveness. The difference between WaterBear and WaterBear-QS is that WaterBear-QS leverages hash function based AVID RBC of Cachin and Tessaro [24] to reduce the communication complexity of RBC. Jumping ahead, we show the modification leads to a dramatic performance improvement compared to WaterBear.

7 IMPLEMENTATION AND EVALUATION

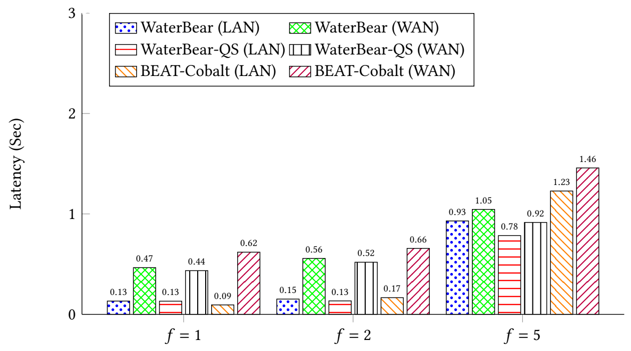
Implementation. We implement WaterBear and WaterBear-QS in Golang. Both protocols use authenticated channels, and WaterBear-QS additionally uses a hash function. We use HMAC to realize the authenticated channel in WaterBear and WaterBear-QS. Note HMAC is used just for an instantiation of the authenticated channel: one would need to implement the channel for a practical system. We use SHA256 for the hash functions. We use gRPC as the underlying communication library. To implement AVID RBC, we use a Golang Reed-Solomon code library [4].

Both WaterBear and WaterBear-QS use RBC in both the RBC and ABA phases. For WaterBear, we use Bracha's broadcast (which is information-theoretically secure) for both phases. For WaterBear-QS, we use AVID RBC of Cachin and Tessaro [24] (using erasure coding and hash functions) in the RBC phase. In ABA (the third phase), we directly use Bracha's broadcast, because in the ABA phase of WaterBear-QS, there is no bulk data.

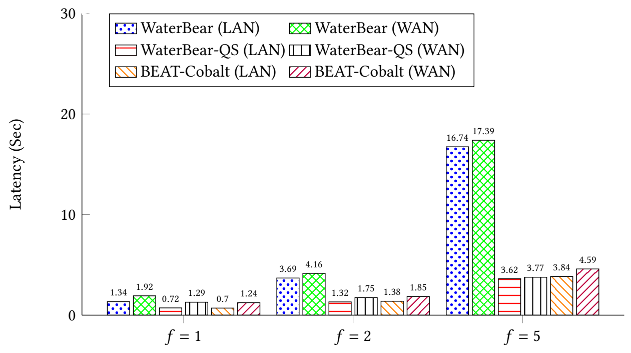
For comparison, we choose to implement BEAT-Cobalt in our Golang library. BEAT [1, 45] was originally implemented in Python 2.7 using MMR ABA [66]. We implement BEAT-Cobalt, replacing MMR by Cobalt-ABA, as Cobalt ABA addressed the liveness issue of MMR. There are several reasons we chose BEAT-Cobalt as the baseline protocol. First, BEAT-Cobalt is the most efficient open-source asynchronous BFT implementation. Other protocols are either not open-sourced yet (e.g., Dumbo [52]) or have a model outside the scope of the conventional BFT setting (e.g., Tusk [40]). In particular, Tusk utilizes additional worker nodes in addition to replicas. Our protocols, however, can be adapted with the strategies proposed by Tusk to achieve better performance, as our protocols, too, use RBC for message transmission. Second, EPIC is the only known adaptively secure asynchronous BFT protocol implemented. The authors of EPIC have shown that BEAT-Cobalt significantly outperforms EPIC in both LAN and WAN settings. Hence, once we demonstrate the performance difference among BEAT-Cobalt, WaterBear, and WaterBear-QS, we can reasonably argue which is



(a) Latency for $b = 1$.



(b) Latency for $b = 100$.



(c) Latency for $b = 15{,}000$.

Figure 4: Latency of the protocols.

the most efficient adaptively secure asynchronous BFT protocol among EPIC, WaterBear, and WaterBear-QS. Of course, EPIC neither achieves quantum security nor information-theoretic security, and EPIC requires trusted setup. Third, jumping ahead, we find WaterBear-QS and BEAT-Cobalt share similar performance.

Here are some other reasons why we chose not to (and do not need to) compare with Dumbo and Tusk. Neither protocols achieve any properties our protocols can achieve: no adaptive security, relying on PKC, no quantum resistance, and assuming trusted setup.

The WaterBear library involves more than 10,000 LOC for the protocol implementations and about 1,000 LOC for evaluation.

Overview of evaluation. We evaluate the performance of our protocols on Amazon EC2 utilizing up to 61 virtual machines (VMs) from different regions in five continents. 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 deploy our protocols in both "LAN" and "WAN" settings. In the LAN setting, the replicas are run in the same region of EC2 (e.g., US Virginia), but these replicas may be located in different physical datacenters. In the WAN setting, the replicas are evenly distributed across different continents.

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 10 epochs and report the average performance (for both throughput and latency). We use two different transaction sizes. We evaluate the performance of the protocols for transactions with 100 bytes by default and evaluate that using 250 bytes per transaction.

We assess the performance of the protocols under the failure-free scenario 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:

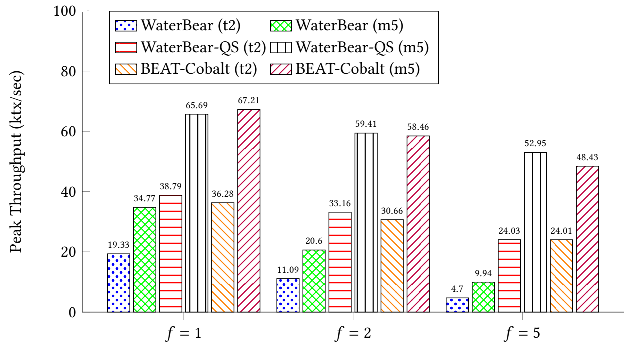
- WaterBear-QS is about as efficient as BEAT-Cobalt. The two protocols, however, offer interesting trade-offs for different n's.
- We confirm that the bandwidth of RBC is one of the major bottlenecks for the BKR framework and DZ framework. WaterBear-QS and BEAT-Cobalt (using bandwidth-efficient AVID RBC) are about twice as efficient as WaterBear (using bandwidth-expensive Bracha's broadcast). Moreover, for all the three protocols, the throughput with a transaction size of 100 bytes is more than 2x the throughput with a transaction size of 250 bytes. To put it differently, WaterBear can be easily made more efficient if a more efficient information-theoretic RBC exists.
- Both WaterBear-QS and WaterBear are highly robust against various crash and Byzantine failures, just as BEAT-Cobalt.

7.1 Performance in Failure-Free Cases

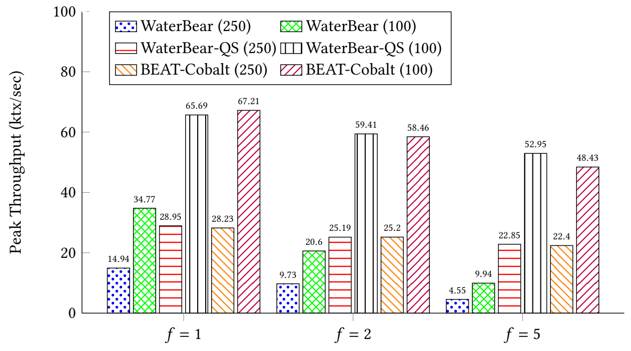
Latency. We report the latency of the protocols in both LAN and WAN settings for $f = 1, 2$, and 5 in Figure 4c. In all these experiments, we let $b = 15{,}000$.

WaterBear and BEAT-Cobalt share similar performance. When $f = 1$, the latency for both WaterBear-QS and BEAT-Cobalt are nearly half of that for WaterBear in both LAN and WAN settings. As f increases, the difference in latency between WaterBear-QS/BEAT-Cobalt and WaterBear also increases. When $f = 5$, the latency of WaterBear is almost four times of that for WaterBear-QS and BEAT-Cobalt. This is expected, since as f increases, the network bandwidth consumption for WaterBear is significantly higher than the other protocols.

The latency of WaterBear-QS, compared to BEAT-Cobalt, is gen-



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



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

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

erally lower in both LAN and WAN settings except for $f = 1$, where the latency of WaterBear-QS is slightly higher than BEAT-Cobalt in both LAN and WAN settings. When $f = 5$ in the WAN environment, the latency of WaterBear-QS is 17.9% lower than BEAT-Cobalt. The better performance for WaterBear-QS is mainly because WaterBear-QS utilizes a biased ABA, causing the protocol to terminate faster.

It is not surprising that all protocols achieve higher latency in WANs than in LANs. This is certainly expected, but we find the difference is more visible when f is smaller. For instance, the latency of WaterBear is 43.3% higher for $f = 1$ in WANs compared to that in LAN but is only 3.9% higher for $f = 5$.

Throughput and scalability. We evaluate the throughput and scalability for WaterBear, WaterBear-QS, and BEAT-Cobalt by varying the network size f from 1 to 20. Unless otherwise specified, all experiments are conducted in the WAN setting running on *m5.xlarge* type instances. We also report throughput vs. latency in Figure 6.

First of all, similar to the results for latency, the throughput of WaterBear is consistently lower than the other two protocols. As WaterBear and WaterBear-QS differ in RBC only, RBC is the clear performance bottleneck.

We examine the throughput of all three protocols for $f = 1$. For $f = 1$, as depicted in Figure 6b, the performance of WaterBear-QS and BEAT-Cobalt in WANs are very close, though the peak throughput of WaterBear-QS is slightly higher. We also conduct a separate experiment in the LAN setting and evaluate the throughput, as shown in Figure 6a. In both experiments, both WaterBear-QS and BEAT-Cobalt outperform WaterBear. Unlike the results in WANs, the throughput of BEAT-Cobalt in LANs is marginally higher than WaterBear-QS: the peak throughput of BEAT-Cobalt is 2.3% higher than WaterBear-QS. The peak throughput of WaterBear-QS is 65.6 ktx/sec in LANs and 52.8 ktx/sec in WANs.

When f increases, the performance trend is slightly different from the case for $f = 1$. In particular, when $f = 5$ and $f = 10$, the throughput of WaterBear-QS is consistently higher than BEAT-Cobalt. When $f = 20$, the peak throughput of the two protocols is again very close. The latency of WaterBear-QS, however, is consistently higher than BEAT-Cobalt.

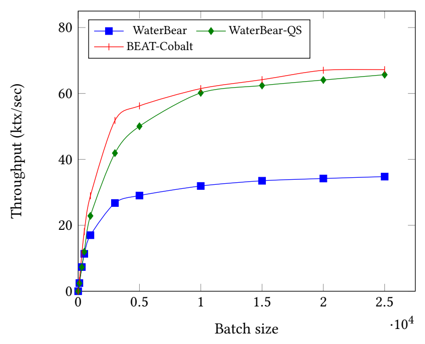
Performance on different types of VMs. Different from prior protocols (HoneyBadgerBFT, BEAT, Dumbo, EPIC) that all evaluate the performance on *t2.medium* instances, we evaluate the performance of the protocols using both *t2.medium* (t2 in the figures) and *m5.xlarge* (m5 in the figures) instances.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

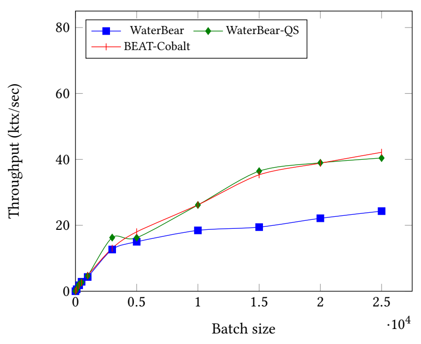
7.2 Performance under Failures

To assess the protocol performance under failures and attacks, we carefully design various experiments as follows. The attack scenarios are by no means exhaustive, but they are the best strategies we envision could impact the system performance.

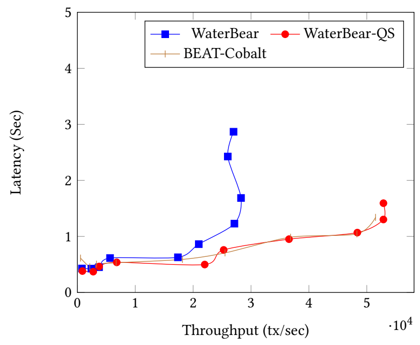
- S_0**: (failure-free)** In this scenario, all replicas are correct. S_0 is the baseline scenario we use to compare with failure scenarios.
- S_1**: (crash)** In this scenario, we let f replicas crash by not participating in the protocols.
- S_2**: (Byzantine; keep voting 0)** In this scenario, we control all f faulty replicas to keep voting for 0 in each step of (R)ABA. For all the three protocols, doing so would intuitively make fewer ABA and RABA instances decide 1 and would likely decrease the throughput of the protocols. We would like to observe the throughput reduction in this scenario for the protocols compared to the failure-free scenario.
- S_3**: (Byzantine; flipping the (R)ABA input)** In this scenario, 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 three 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 to decide in round 0 and



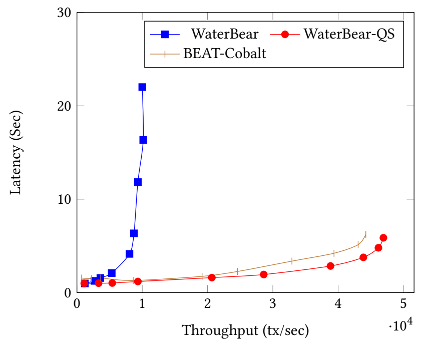
(a) Throughput in the LAN setting when $f = 1$.



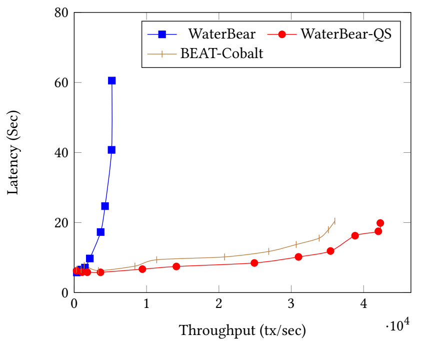
(b) Throughput in the WAN setting when $f = 1$.



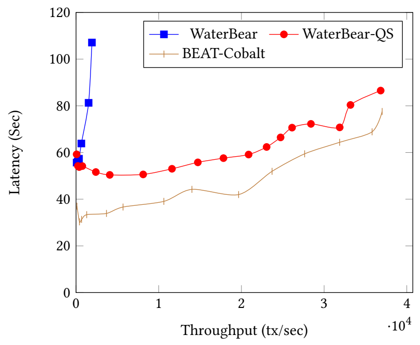
(c) Throughput vs Latency when $f = 1$.



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



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



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

Figure 6: Throughput vs latency on m5.xlarge instances for $f = 1$ to $f = 20$.

force them to enter the second round of RABA, where the two protocols start to run our local coin protocol. Jumping ahead, our experiment actually shows that the flipping strategy S_3 works slightly better than S_2, but in general is not destructive.

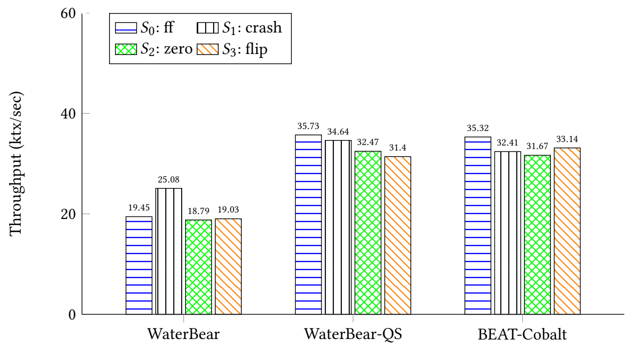
Note that we do not attempt to attack the RBC phase for all these protocols, because RBC is highly robust during failures and attacks (see, e.g., [35]).

We assess the failure-case performance for $f = 1$ (Figure 7a), $f = 2$ (Figure 7b), and $f = 5$ (Figure 7c).

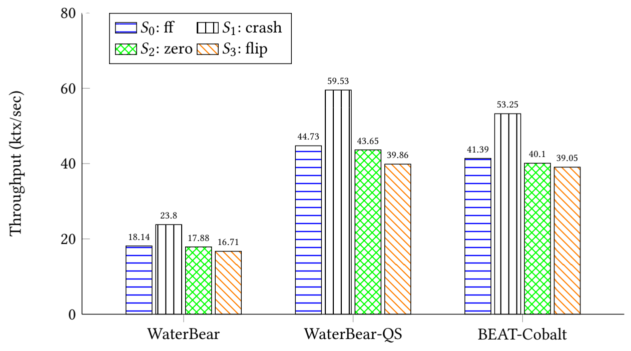
Performance under crash failures (S_1). When $f = 1$, WaterBear achieves higher throughput under crash failures compared to that in the failure-free case. For WaterBear-QS and BEAT-Cobalt, the performance is slightly lower under crash failures than those in the crash scenarios. The throughput of WaterBear-QS is 3.0% lower in the crash failure scenario, and BEAT-Cobalt is 8.2% lower in the crash failure scenario. For all other cases, all three protocols achieve higher performance under crash failures compared to that in the failure-free case. For instance, when $f = 5$, the throughput of WaterBear-QS is 22.6% higher, and the throughput of BEAT-Cobalt is 16.9% higher in the crash failure scenario. This is mainly because under crash failures, the network bandwidth consumption is much lower (about 25% lower) than in the failure-free case. When $f = 1$, as the network bandwidth consumption does not dominate the overhead, the performance of WaterBear-QS and BEAT-Cobalt under crash failures is similar to that in failure-free cases. When $f = 2$ and $f = 5$, the performance improvement under crash failures for WaterBear-QS is higher than that of BEAT-Cobalt. We believe this is partly because WaterBear-QS involves multiple RBC instances (in both the RBC phase and the ABA phase) and uses more bandwidth than BEAT-Cobalt.

Performance under Byzantine failures. The performance of all the protocols is slightly lower under Byzantine failures compared to the failure-free scenario and crash failure scenario. The performance degradation of WaterBear-QS and BEAT-Cobalt may appear higher than that of WaterBear. This is actually because the performance of WaterBear is much lower than the other two protocols. For WaterBear-QS, the throughput of is 2.4%-19.3% lower under Byzantine failures compared to failure-free scenario. In comparison, the throughput of BEAT-Cobalt is 0.5%-10.3% under Byzantine failures. The higher performance degradation for WaterBear-QS 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.

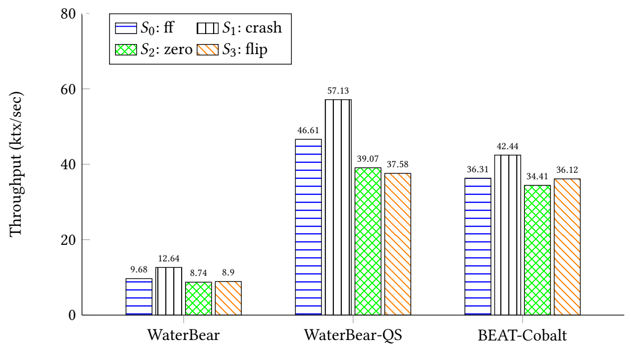
S_2 vs. S_3. The difference between S_2 and S_3 is that faulty replicas broadcast 0 in S_2 but broadcast the flipped value in S_3. For BEAT-Cobalt, the performance in S_3 is higher for $f = 1$ and $f = 5$ but lower for $f = 2$; the difference in all the cases is not significant. In contrast, for WaterBear-QS, the performance is consistently lower in S_3 than S_2. This is expected: first, Cobalt ABA uses common coin and has an expected constant rounds, thereby being less sensitive to the attack in S_3. Second, as WaterBear ABA uses local coins, making



(a) Throughput under different failure scenarios in WAN for $f = 1$ and $b =$ 15,000.



(b) Throughput under different failure scenarios in WAN for $f = 2$ and $b =$ 15,000.



(c) Throughput under different failure scenarios in WAN for $f = 5$ and $b =$ 15,000.

Figure 7: Performance of the protocols in failure scenarios.

replicas receive flipped values may force replicas to receive both 0 and 1 such that the local coin value will be used; when WaterBear ABA uses local coins, it ends with more rounds.

8 CONCLUSION

We provide two novel asynchronous BFT protocols—WaterBear and WaterBear-QS. WaterBear is information-theoretic, assuming the existence of authenticated channels. WaterBear-QS is quantum secure and only uses hash functions and authenticated channels. Via extensive evaluation, we show both WaterBear and WaterBear-QS are efficient under both failure-free and failure scenarios.

ACKNOWLEDGMENT

We thank Yang Yu and Xiao Sui for helpful comments.

REFERENCES

[1] 2021. BEAT library. https://github.com/fififish/beat. (2021).
[2] 2021. Bug in ABA protocol's use of Common Coin. https://github.com/amiller/HoneyBadgerBFT/issues/59. (2021).
[3] 2021. HoneyBadgerBFT library. https://github.com/amiller/HoneyBadgerBFT. (2021).
[4] 2021. Reed-Solomon library. https://github.com/klauspost/reedsolomon. (2021).
[5] Ittai Abraham, Danny Dolev, and Joseph Y. Halpern. 2008. An Almost-Surely Terminating Polynomial Protocol for Asynchronous Byzantine Agreement with Optimal Resilience *(PODC)*.
[6] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching Consensus for Asynchronous Distributed Key Generation. In *PODC*.
[7] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically Optimal Validated Asynchronous Byzantine Agreement. In *PODC*. ACM, 337–346.
[8] Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. 2021. High-Threshold AVSS with Optimal Communication Complexity. In *FC*.
[9] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. 2011. Prime: Byzantine replication under attack. *TDSC* 8, 4 (2011), 564–577.
[10] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger fabric: A distributed operating system for permissioned blockchains. *EuroSys*.
[11] P. Aublin, S. B. Mokhtar, and V. Quéma. 2013. RBFT: Redundant Byzantine Fault Tolerance. In *ICDCS*. 297–306.
[12] Laasya Bangalore, Ashish Choudhury, and Arpita Patra. 2020. The Power of Shunning: Efficient Asynchronous Byzantine Agreement Revisited*. *J. ACM* (2020).
[13] Soumya Basu, Alin Tomescu, Ittai Abraham, Dahlia Malkhi, Michael K. Reiter, and Emin Gün Sirer. 2019. Efficient Verifiable Secret Sharing with Share Recovery in BFT Protocols. In *CCS*.
[14] Michael Ben-Or. 1983. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols (Extended Abstract). In *PODC*. 27–30.
[15] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation *(STOC)*.
[16] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. 1994. Asynchronous secure computations with optimal resilience. In *PODC*. ACM, 183–192.
[17] Alexandra Boldyreva. 2003. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *PKC*.
[18] Gabriel Bracha. 1984. An asynchronous [(n-1)/3]-resilient consensus protocol. In *PODC*. ACM, 154–162.
[19] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. *Information and Computation* 75, 2 (1987), 130–143.
[20] Christian Cachin. 2010. Yet another visit to paxos. = https://cachin.com/cc/papers/pax.pdf *(IBM Research Report RZ 3754)*.
[21] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*. Springer, 524–541.
[22] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology* 18, 3 (2005), 219–246.
[23] Christian Cachin and Jonathan A Poritz. 2002. Secure intrusion-tolerant replication on the Internet. In *DSN*. IEEE, 167–176.
[24] Christian Cachin and Stefano Tessaro. 2005. Asynchronous verifiable information dispersal. In *SRDS*. IEEE, 191–201.
[25] Christian Cachin and Marko Vukolic. 2017. Blockchain Consensus Protocols in the Wild. In *DISC*.
[26] Ran Canetti. 2020. Universally Composable Security. *J. ACM* (2020).
[27] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. 1996. Adaptively Secure Multi-party Computation. In *STOC*.
[28] Ran Canetti and Tal Rabin. 1993. Fast asynchronous Byzantine agreement with optimal resilience. In *STOC*, Vol. 93. Citeseer, 42–51.
[29] Miguel Castro. 2001. Practical byzantine fault tolerance *(PhD thesis)*.
[30] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine fault tolerance and proactive recovery. *TOCS* 20, 4 (2002), 398–461.

[31] David Chaum, Claude Crépeau, and Ivan Damgard. 1988. Multiparty Unconditionally Secure Protocols. In *STOC*.

[32] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. 1985. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *SFCS*.

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

[34] Ran Cohen, Sandro Coretti, Juan Garay, and Vassilis Zikas. 2019. Probabilistic Termination and Composability of Cryptographic Protocols. *J. Cryptol.* (2019), 690–741.

[35] Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, and Athanasios Xygkis. 2020. Online payments by merely broadcasting messages. In *DSN*. IEEE, 26–38.

[36] Sandro Coretti, Juan Garay, Martin Hirt, and Vassilis Zikas. 2016. Constant-Round Asynchronous Multi-Party Computation Based on One-Way Functions. In *ASIACRYPT*.

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

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

[39] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. 1999. Efficient Multiparty Computations Secure Against an Adaptive Adversary. In *EUROCRYPT*.

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

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

[42] Dan Dobre, Ghassan O. Karame, Wenting Li, Matthias Majuntke, Neeraj Suri, and Marko Vukolic. 2019. Proofs of Writing for Robust Storage. *IEEE Trans. Parallel Distributed Syst.* 30, 11 (2019), 2547–2566.

[43] Sisi Duan, Hein Meling, Sean Peisert, and Haibin Zhang. 2014. BChain: Byzantine Replication with High Throughput and Embedded Reconfiguration. In *OPODIS*. 91–106.

[44] Sisi Duan, Michael K Reiter, and Haibin Zhang. Secure causal atomic broadcast, revisited. In *DSN*.

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

[46] Sisi Duan and Haibin Zhang. 2022. PACE: Fully Parallelizable BFT from Reproposable Byzantine Agreement. In *IACR Cryptology ePrint Archive, 2022*.

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

[48] Paul Feldman and Silvio Micali. 1988. Optimal Algorithms for Byzantine Agreement. In *STOC*.

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

[50] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2021. Efficient Asynchronous Byzantine Agreement without Private Setups. *arXiv preprint arXiv:2106.07831* (2021).

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

[52] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster Asynchronous BFT Protocols.. In *CCS*.

[53] James Hendricks, Gregory R. Ganger, and Michael K. Reiter. 2007. Low-overhead byzantine fault-tolerant storage. In *SOSP*.

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

[55] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. 2021. All You Need is DAG.. In *PODC*.

[56] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. 2006. Information-Theoretically Secure Protocols and Security under Composition *(STOC)*.

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

[58] Benoît Libert, Marc Joye, and Moti Yung. 2016. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. *Theoretical Computer Science* 645 (2016), 1–24.

[59] Chao Liu, Sisi Duan, and Haibin Zhang. 2020. EPIC: Efficient asynchronous BFT with adaptive security. In *DSN*.

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

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

[62] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*. 129–138.

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

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

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

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

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

[68] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. 2015. Signature-free asynchronous binary Byzantine consensus with t< n/3, O (n2) messages, and O (1) expected time. *Journal of the ACM (JACM)* 62, 4 (2015), 1–21.

[69] A. Patra, A. Choudhury, and C.P. Rangan. 2014. Asynchronous Byzantine agreement with optimal resilience. *Distrib. Comput.* 27 (2014), 111–146.

[70] M. Pease, R. Shostak, and L. Lamport. 1980. Reaching Agreement in the Presence of Faults. *JACM* 27, 2 (April 1980), 228–234.

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

[72] João Sousa, Eduardo Alchieri, and Alysson Bessani. 2014. State machine replication for the masses with BFT-SMaRt. In *DSN*. 355–362.

[73] Gilad Stern and Ittai Abraham. 2018. Information Theoretic HotStuff. In *OPODIS*.

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

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

A BRACHA'S ABA

We describe Bracha's ABA [18]. The pseudocode is shown in Figure 8. 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. Every replica maintains a set $vset$ containing $valid$ values. In each phase, every replica only accept messages that carry valid values. The valid values $vset$ must be congruent with the values each replica receives from the previous step/round. In the first phase of the round 0, both 0 and 1 are considered valid. In other phases, a value is added to $vset$ only if the replica receives the value from a sufficiently large fraction of replicas.

In the first phase, every replica p_i *r-broadcasts* pre-vote$_r(iv_r)$. In round 0, the iv_0 is set to the value a replica proposes. For round $r > 0$, iv_r is set at the end of the third phase in round $r - 1$. When p_i *r-delivers* $n - f$ pre-vote$_r()$ messages, there are two cases. If p_i *r-delivers* $n - f$ pre-vote$_r(v)$, it delivers v and sets $vset$ to $\{v\}$. Replica p_i still participates in the protocol for one more round but skips the step of deciding again. Otherwise, p_i selects the majority value v it receives from the pre-vote$_r()$ messages and uses the value for the next phase. At the end of the first phase, every replica *r-broadcasts* a main-vote$_r(v)$ message. In the second phase, every replica p_i waits for $n - f$ valid main-vote$_r()$ messages. If p_i receives at least $n/2$ main-vote$_r(v)$, it sets $vset$ to $\{v\}$. Otherwise, it sets v to $\perp$ and $vset$ to $\{0, 1\}$ (i.e., it accepts both 0 and 1 in the next phase). At the end of the second phase, *r-broadcasts* final-vote$_r(v)$

```
Initialization
  $r \leftarrow 0$                                                    {round}
func $propose(v)$
  $iv_0 \leftarrow v$
  $vset \leftarrow \{0,1\}$              {valid binary values that will be accepted}
  start round 0
round $r$
  r-broadcast pre-vote$_r(iv_r)$                                   {first phase}
  upon receiving r-delivering $n-f$ pre-vote$_r()$ such that $vals$ is the set of
  values carried in the messages; for each $v \in vals, v \in vset$
    if $vals = \{v\}$
      deliver $v$, $vset \leftarrow \{v\}$
    else, $v = majority(vals)$
    r-broadcast main-vote$_r(v)$                          {start the second phase}
  upon receiving r-delivering $n-f$ main-vote$_r()$ such that $vals$ is the set
  of values carried in the messages; for each $v \in vals, v \in vset$
    if there are at least $n/2$ $v$ in $vals$, $vset \leftarrow \{v\}$
    else
      $v = \perp$, $vset \leftarrow \{0,1\}$
    r-broadcast final-vote$_r(v)$                          {start the third phase}
  upon receiving r-delivering $n-f$ final-vote$_r()$ such that $vals$ is the set
  of values carried in the messages; for each $v \in vals, v \in vset$
    if there are at least $2f+1$ $v$ in $vals$
      deliver $v$, $iv_{r+1} \leftarrow v$, $vset \leftarrow \{v\}$
    else if there are $f+1$ $v$
      $iv_{r+1} = v$, $vset \leftarrow \{0,1\}$
    else
      $c \leftarrow Random()$                              {obtain a local coin}
      $iv_{r+1} = c$, $vset \leftarrow \{0,1\}$
  $r \leftarrow r+1$
```

Figure 8: The Bracha's ABA protocol [18].

and enters the third phase. In the third phase, every replica waits for $n-f$ final-vote$_r()$ messages with valid values. If p_i receives at least $2f+1$ final-vote$_r(v)$, it delivers v and sets iv_{r+1} to v. In the following rounds, it only accepts v. If p_i receives at least $f+1$ final-vote$_r(v)$, it sets iv_{r+1} to v and will accept both 0 and 1 in the following round. Otherwise, p_i uses the local coin value as iv_{r+1} and accepts both 0 and 1 in the following round.

B PROOF OF WATERBEAR ABA

We show that WaterBear ABA achieves validity, agreement, termination, and integrity. For $v \in \{0,1\}$, let $\bar{v}$ be $1-v$.

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

Proof. If all correct replicas propose $iv_r = v$ in round r, every correct replica broadcasts pre-vote$_r(v)$. No correct replica will forward pre-vote$_r(\bar{v})$, as there does not exist more than $f+1$ pre-vote$_r(\bar{v})$ messages. Hence, no correct replica will add $\bar{v}$ to $bset_r$. Furthermore, all correct replicas will eventually send main-vote$_r(v)$ and *r-broadcast* final-vote$_r(v)$. No correct replica accept final-vote$_r(\bar{v})$ or final-vote$_r(*)$ since they only have v in their $bset_r$. Hence, any correct replica that enters round $r+1$ sets $iv_{r+1} = v$. ■

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

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

Proof. The proof is by induction on the round number. The base case holds for r according to Lemma B.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 $iv_{r'+1} = v$.

In round r', as no correct replica sends pre-vote$_{r'}(\bar{v})$. No correct replica can receive $f+1$ pre-vote$_{r'}(\bar{v})$ messages. In other words, no correct replica will forward pre-vote$_{r'}(\bar{v})$. Meanwhile, no correct replica will accept final-vote$_{r'}(\bar{v})$ or final-vote$_{r'}(*)$ since correct replicas only have v in their $bset_{r'}$. Furtheremore, every correct replica will braodcast final-vote$_{r'}(v)$. Therefore, it is straightforward to see that any correct replica that enters round $r'+1$ sets $iv_{r'+1} = v$. ■

Theorem B.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 B.1. If p_i terminates and decides $\bar{v}$ in round $r > 0$, it *r-delivers* $n-f$ final-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 pre-vote$_r(\bar{v})$. This is a contradiction with Lemma B.2 since any correct replica that enters round r sets $iv_r = v$. This completes the proof of the theorem. ■

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

Proof. If p_i decides v in round r, it *r-delivers* $n-f$ final-vote$_r(v)$. In other words, at least $f+1$ correct replicas *r-broadcast* final-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} to $\bar{v}$, there are three conditions: A) p_k *r-delivers* at least $n-f$ final-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-broadcast* 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* a value 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 B.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 B.4, any correct replica enters round $r+1$ sets iv_{r+1} to v. Furthermore, according to Lemma B.2, for any round $r'' \geq r+1$, any correct replica sets enters round r'' sets $iv_{r''}$ to v. If replica p_j decides value $\bar{v}$ in round r', at least one correct replica has set $iv_{r'} = \bar{v}$ and sent pre-vote$_{r'}(\bar{v})$, a contradiction with Lemma B.2. ■

Lemma B.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 $r+1$, $v_1 = v_2$.*

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, 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 is straightforward to see that $v_1 = v_2$. ■

Theorem B.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 to either 0 or 1 in WaterBear 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 c. The c value 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 B.6, it is impossible for a correct replica to receive $f+1$ final-vote$_r(v)$ and another correct replica to receive $f+1$ final-vote$_r(\bar{v})$. 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 terminate in $O(2^{n-f})$ expected rounds. ■

Theorem B.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. In other word, the replica does not decide again in the following round. The theorem is then proved. ■

C PROOF OF WATERBEAR RABA

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

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

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

Theorem C.2 (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 C.1. We now prove the case where p_i decides in round $r > 0$.

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

Theorem C.3 (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 pre-vote$_0(v)$. No correct replica will add $\bar{v}$ to $bset_0$. Furthermore, no correct replica will accept main-vote$_0(\bar{v})$ or final-vote$_0(\bar{v})$. Eventually all correct replicas will receive $2f+1$

pre-vote$_0(v)$, add v to $bset_0$, and broadcast main-vote$_0(v)$. Similarly, all correct replicas will eventually receive $n-f$ main-vote$_0(v)$ and *r-broadcast* final-vote$_0(v)$. All correct replicas will *r-deliver* $n-f$ final-vote$_0(v)$. In other words, all correct replicas will terminate and decide v. ■

Lemma C.4. *If p_i decides v in round 0, any correct replica that enters round 1 sets $iv_1 = v$.*

Proof. If p_i decides v in round 1, it *r-delivers* $n-f$ 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 $iv_1 = \bar{v}$, there are three conditions: A) p_k *r-delivers* at least $n-f$ final-vote$_r(\bar{v})$; B) p_k *r-delivers* $f+1$ final-vote$_0(\bar{v})$; C) p_k has not received more than $f+1$ final-vote$_0(v)$ and p_k has not received more than $f+1$ final-vote$_0(\bar{v})$. We now show that none of the three conditions is possible.

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

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

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

Theorem C.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 a correct replica p_j decides $\bar{v}$ and prove the theorem by contradiction. Since WaterBear RABA follows WaterBear 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 WaterBear 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* $n-f$ final-vote$_0(v)$. If p_j decides $\bar{v}$, it *r-delivers* $n-f$ 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 final-vote$_0(v)$ and final-vote$_0(\bar{v})$, a contradiction since each replica only *r-broadcasts* a 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 $2f+1$ pre-vote$_1(\bar{v})$, where at least one correct replica has sent pre-vote$_1(\bar{v})$, a contradiction with Lemma C.4. Starting from round 1, WaterBear RABA follows WaterBear ABA so that Lemma B.2 holds. If p_j decides $\bar{v}$ in round $r > 1$, at least one correct replica must have sent pre-vote$_r(\bar{v})$, a contradiction with Lemma B.2 since any correct replica sets $iv_r = v$.
This completes the proof of the theorem. ■

Lemma C.6. *If $f+1$ correct replicas propose 1 in round 0, every correct replica eventually accepts* final-vote$_0(1)$.

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

Lemma C.7. *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* $n-f$ final-vote$_0(v)$ with the same v; B) p_i *r-delivers* at least $f+1$ 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 to $v = 1$.

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

For condition B, we assume p_i *r-delivers* $f+1$ final-vote$_0(0)$ and prove the correctness by contradiction. If p_i *r-delivers* $f+1$ final-vote$_0(0)$, at least one correct replica *r-broadcasts* final-vote$_0(0)$. If the correct replica *r-broadcasts* final-vote$_0(0)$, the replica must have received $n-f$ main-vote$_0(0)$. We already know that at least $f+1$ correct replicas have sent main-vote$_0(1)$. Any correct replica broadcasts main-vote$_0()$ message once. In other words, at least one correct replica has broadcast both main-vote$_0(0)$ and main-vote$_0(1)$, a contradiction. Therefore, in this condition, p_i must have *r-deliver* $f+1$ final-vote$_0(1)$. It is then straightforward to see that any correct replica uses $iv_1 = 1$ to enter round 1.

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

Theorem C.8 (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 C.7. If p_i decides 0 in round $r > 0$, at least one correct replica has set $iv_r = 0$ and broadcast pre-vote$_r(0)$. Since WaterBear RABA follows WaterBear ABA starting from round 1, Lemma B.2 holds. Therefore, the claim that at least one correct replica has set $iv_r = 0$ is a contradiction with Lemma B.2. This completes the proof of the theorem. ■

Theorem C.9 (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 WaterBear RABA follows WaterBear ABA starting from round 1, this part follows from termination of WaterBear ABA. We only need to prove that every correct replica eventually moves to round 1. For replicas in Q_1, they broadcast pre-$\text{vote}_0(1)$ and add 1 to $bset_0$. For replicas in Q_2, they broadcast pre-$\text{vote}_0(0)$ upon the $propose(0)$ function, broadcast pre-$\text{vote}_0(1)$ upon the $repropose(1)$ 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 main-$\text{vote}_0(1)$ and *r-broadcast* final-$\text{vote}_0(1)$. For any correct replica p_i in Q_2, it may send main-$\text{vote}_0(1)$ or main-$\text{vote}_0(0)$. There are two sub-cases: none of the correct replicas send main-$\text{vote}_0(0)$; at least one correct replica has sent main-$\text{vote}_0(0)$. For the first sub-case, it is straightforward to see that every correct replica eventually receives and accepts $n - f$ main-$\text{vote}_0(1)$, as every correct replica has 1 in its $bset_0$. Similarly, every correct replica will *r-broadcast* final-$\text{vote}_0(1)$ and accept $n - f$ final-$\text{vote}_0(1)$. For the second sub-case, if a correct replica p_i sends main-$\text{vote}_0(0)$, it receives $2f + 1$ pre-$\text{vote}_0(0)$, among which at least $f + 1$ are sent by correct replicas. Therefore, every correct replica will eventually receive $f+1$ pre-$\text{vote}_0(0)$ and broadcast 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 main-$\text{vote}_0(0)$ and main-$\text{vote}_0(1)$. Similarly, every correct replica accepts both final-$\text{vote}_0(0)$ and final-$\text{vote}_0(1)$. In other words, every correct replica eventually moves to the next round.

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

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

Theorem C.10 (Integrity). *No correct replica decides twice.*

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