Dumbo: Faster Asynchronous BFT Protocols*

Bingyong Guo[1,3], Zhenliang Lu[2,3], Qiang Tang[2,3], Jing Xu[1,3], Zhenfeng Zhang[1,3]

[1]Institute of Software, Chinese Academy of Sciences, Email: {guobingyong, xujing, zfzhang}@tca.iscas.ac.cn
[2]Department of Computer Science, New Jersey Institute of Technology, Email: {zl425, qiang}@njit.edu
[3]JDD-NJIT-ISCAS Joint Blockchain Lab

ABSTRACT

HoneyBadgerBFT, proposed by Miller et al. [32] as the first practical asynchronous atomic broadcast protocol, demonstrated impressive performance. The core of HoneyBadgerBFT (HB-BFT) is to achieve batching consensus using asynchronous common subset protocol (ACS) of Ben-Or et al., constituted with n reliable broadcast protocol (RBC) to have each node propose its input, followed by n asynchronous binary agreement protocol (ABA) to make a decision for each proposed value (n is the total number of nodes).

In this paper, we propose two new atomic broadcast protocols (called Dumbo1, Dumbo2) both of which have asymptotically and practically better efficiency. In particular, the ACS of Dumbo1 only runs a small κ (independent of n) instances of ABA, while that of Dumbo2 further reduces it to constant! At the core of our techniques are two major observations: (1) reducing the number of ABA instances significantly improves efficiency; and (2) using multi-valued validated Byzantine agreement (MVBA) which was considered sub-optimal for ACS in [32] in a more careful way could actually lead to a much more efficient ACS.

We implement both Dumbo1, Dumbo2 and deploy them as well as HB-BFT on 100 Amazon EC2 t2.medium instances uniformly distributed throughout 10 different regions across the globe, and run extensive experiments in the same environments. The experimental results show that our protocols achieve multi-fold improvements over HoneyBadgerBFT on both latency and throughput, especially when the system scale becomes moderately large.

1 INTRODUCTION

Byzantine fault tolerant (BFT) protocols enable a set of untrusted peers to reach consensus. As one fundamental research area in distributed computing, the problem has been extensively studied and many variants exist for different application scenarios. One main categorization of the BFT protocols is based on the timing (or network) assumptions. A synchronous BFT protocol assumes all values sent by honest peers will be delivered to the recipients within a certain period of time, which is known to everyone (including the protocol designer). While a partially synchronous BFT protocol relaxes this network requirement by allowing the time bound to be exist but unknown. An asynchronous BFT protocol relies the least on the network assumption that it does not require such a time bound to exist, just that all values will eventually be delivered.

The favored asynchronous BFT. In the earlier years, considering the deployment of BFT protocols mostly in conventional in-house scenarios that the peers are well-connected, research efforts of BFT focused on reducing (cryptographic) computations [5, 37], or increasing the threshold of malicious participants the protocol can tolerate [21], assuming a synchronous network. Nice works on synchronous setting continue to emerge in recent years [1, 11, 30]. Efforts also exist relaxing the network synchrony assumption. One notable example is the classic Practical Byzantine Fault Tolerance (PBFT) protocol [18] that requires partial synchrony. However, removing the synchrony assumption completely in practice becomes more and more desirable for both robustness and efficiency reasons.

Recently, the success of cryptocurrencies and blockchain technology in general brings much broader application scenarios to the BFT protocols, and also demonstrates the possibility of consensuses over wide-area network (WAN). The open Internet environment provides a more adversarial setting that the network latency among the peers could be time-varying. However, the synchronous (or partially synchronous) BFT can only perform in the relatively "private" network with well-connected nodes that guarantees network delivery within certain time bound. Those protocols would fail to make progress and get

*An abridged version of the paper will appear at ACM CCS '20.

stagnated if the timing assumption does not hold. Indeed, it was shown formally in recent work [32] that PBFT cannot make any progress in "intermittently synchronous network", where the adversary only chooses to delay messages at certain time points. The "attack" could similarly be applied to a class of leader-based BFT protocols [4, 5, 10, 19, 20, 38].

Another important reason that asynchronous protocols maybe favorable is due to efficiency, particularly a property called *responsiveness*. A synchronous BFT protocol, when designed, is parameterized by the *assumed* network latency, which is normally chosen to be large so that the actual network latency is indeed smaller thus the synchrony assumption can be ensured. As a consequence, the efficiency of most of the synchronous BFT protocols depends on the assumed network latency. While "responsiveness" instead requires the performance is only related to the *actual* network latency, thus it should not rely on any timing assumption and the protocol makes progress as soon as messages are delivered.

Moreover, it is well-known that asynchronous protocols simplify the engineering efforts substantially when actually building the distribute system, as no time-out mechanism will be needed. While building a system implementing a synchronous protocol, one should design all kinds of ad-hoc, error-prone time-out mechanisms.

The first practical asynchronous BFT [32]. Even though it is preferable or even necessary in many cases when deployed in real-world WAN, most of the previous researches on asynchronous BFT are theoretical in nature until the first practical protocol HoneyBadgerBFT was proposed in [32]. Previous asynchronous BFT protocols normally are inefficient, e.g., having a high (per message) communication complexity (up to $O(n^2)$ or even $O(n^3)$ if there are n peers) [2, 9, 15, 17, 25, 37]. The performance of these protocols will drop sharply when the system scales up. The elegant work of HoneyBadgerBFT [32], on the other hand, made several critical observations to push asynchronous BFT towards being practical.

The first observation is that an atomic broadcast protocol which is a continuous execution of BFT protocols maintaining an ever-growing log, (or to put it another way, regular BFT protocols can be considered as a one-shot instance of it), could be built very lightly from a weaker variant called asynchronous common subset (ACS) together with a threshold encryption scheme. An ACS protocol only requires peers to agree on a subset of all their inputs and was originally proposed for different purposes [7].

More importantly, it was observed in [32] that the classic ACS protocol from Ben-Or et al. [9] is much more promising for efficiency both asymptotically and practically (than another related protocol called multi-valued validated Byzantine agreement (MVBA) [15], to be further explained soon), when picking the underlying building blocks carefully. The ACS protocol from [9, 32] was built from two sub-protocols: *reliable broadcast* (RBC) and *asynchronous binary agreement* (ABA). The structure is fairly simple: each node invokes an RBC to broadcast its input value, and participates in n instances of the ABA protocol to agree on which subset of inputs to include, see Figure 1.

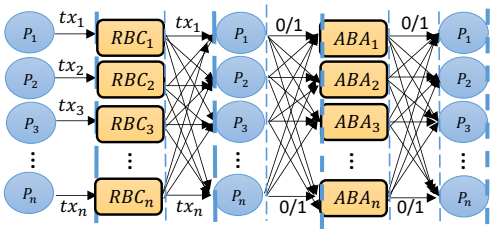

Fig. 1. The structure of ACS in HoneyBadgerBFT [32]

Experimental results show impressive performance of HoneyBadgerBFT. In a nice work BEAT [22], the authors gave an extensive study about most suitable instantiations of the building blocks for HB-BFT (while keeping the protocol structure intact) when considering diverse deployment scenarios. [1] We take a different path and ask the following question:

Can we redesign the ACS protocol to improve both its asymptotic and practical efficiency?

[1]We remark here that most of the techniques of BEAT [22] can be directly applied to our protocols as well, to choose more suitable instantiations of the underlying building blocks such as RBC, and further optimize the performance. Our focus is to show asymptotic and practical improvements due to **protocol redesign**, so we mainly compare a **basic** instantiation of our protocols with HoneyBadgerBFT in experiments. See more comparison in section 1.2.

1.1 Our contributions

We design two new ACS protocols, both of which improve the running time asymptotically and practically. Our experimental results demonstrate a multi-fold improvements over HoneyBadgerBFT [32] when they are run back to back in the same environment on Amazon AWS. More interestingly, our two main observations (1. the number of ABA instances should be reduced; 2. MVBA would be more efficient if used carefully for ACS) that lead to our two protocols would be of independent interests. Let us elaborate in more detail below.

Dumbo1: a faster asynchronous BFT. We first go over the structure of the ACS protocol used in HB-BFT in slightly more details: first each peer broadcasts its input via an RBC instance; whenever a peer receives value from peer P_i, it sets its input for the i-th ABA instance to be 1 and starts the ABA protocol. Once an honest peer has got 1 from $n - f$ ABA instances, it will input 0 to all the remaining ABA instances which have not input yet and move on.

Identifying the major bottleneck. Due to the famous FLP impossibility [23], an ABA must be a randomized protocol. This brings in the following drawback: though the expected number of "rounds" of each ABA protocol is constant, the expected number of rounds of running n concurrent ABA sessions could be significant, i.e., at least $O(\log n)$ [8] More seriously, those ABA instances do not really execute in a fully concurrent fashion: as (1) not all instances start at the same time, some of the instances may start later as inputs of (the previous RBC) haven't been delivered; (2) normal node also has an efficiency degradation facing large scale concurrent execution (not enough CPU cores etc). When n gets larger, and the network is unstable, there would likely be some ABA instances that terminate very slowly. The slowest ABA instance determines the running time of the ACS of HoneyBadgerBFT.

To see the practical impact of ABA protocols on the performance, we carry out experiments of HB-BFT and do statistics about the average running time between RBC and ABA. As shown in Figure 2, it is clear that for HB-BFT, the cost of ABA is dominating [2]. The pattern becomes even more significant when the scale of the system grows. This simple observation inspires us to reduce the number of ABA instances needed in the ACS protocol.

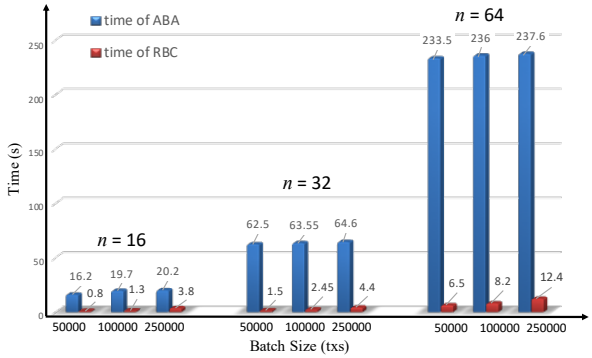

Fig. 2. Time costs of RBC and ABA in HoneyBadgerBFT, we get the running time of RBC by starting a timer when protocol starts and ending the timer when nodes get the output of RBC, averaging among multiple nodes and instances. The time of ABA is taking the maximum among all ABA instances a node needs to run.

Reducing # of ABA *instances.* We redesign the structure of ACS, and propose Dumbo1-ACS. Different with the HoneyBadgerBFT (and also the BEAT protocols), Dumbo1-ACS only needs to run κ instead of all the n ABA instances, and achieves $O(\log \kappa)$ running time, where κ is a security parameter independent of n. Other complexity metrics remain the same.

In a simplified view, the first phase remains unchanged: every node broadcasts its input through an RBC instance. Then, imagine that if we have one honest node to take the role as the leader, then it can first finish $n - f$ RBC instances and then informs all other nodes to output the deliveries of these RBC instances. To get such an honest node, we can select a small number κ of nodes as the "leaders" such that at least *one of them* is honest with an overwhelming probability.

[2]We ignored some "unnoticeable" time cost of local computations such as threshold encryption/decryption, picking an input from the buffer etc as they do not change the ratio much.

Further care is needed as now two honest nodes may receive different values from different selected "leaders". Next, we should enable honest nodes to decide which of the κ selected nodes to believe. It actually becomes similar to HB-BFT that we can invoke ABA instances to confirm whose nomination of subset to include. Once some ABA instances output 1, the corresponding messages can be identified and output. Importantly, now the peers just need to agree on the κ (which could be much smaller than n) nodes. See pictorial illustration in Fig. 3.

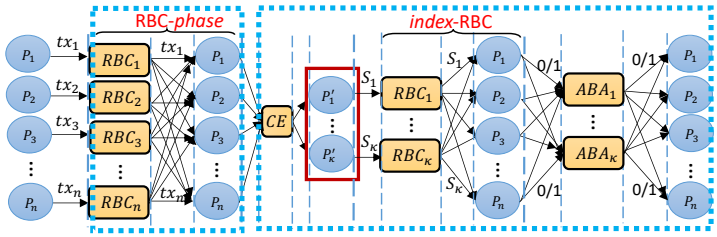

Fig. 3. The structure of Dumbo1-ACS

We would like to stress that the changes in the remaining parts are kept at minimal so that the reduction of ABA instances also yields significant practical improvements. It looks like we have a handful extra RBC instances than HB-BFT; however, we make the input of each peer in those extra (index)-RBC instances to be a tiny index-set (S_i instead of the actual data loads). An honest player inputs 1 for the i-th ABA if he indeed receives all messages corresponding to S_i. Moreover, the added coin-tossing protocol to select κ nodes is just a sub-routine of ABA protocol. So those added overhead would be unnoticeable compared to the cost of eliminated ABA instances.

To see why it works: when one honest node determines to output the values corresponding to S_i, it must be the case that the i-th ABA instance outputs a bit 1. The property of ABA ensures that (1) all other honest nodes will also output 1 and (2) at least one honest node inputs 1 for this ABA instance. The latter means at least one honest node indeed receives all the input values $\{v_j\}$ corresponding to the index set S_i. Thus following the security of RBC, all other honest nodes will also receive those values eventually. While condition (1) ensures all honest nodes will actually output the same subset of values.

Dumbo2: an even faster asynchronous BFT. Dumbo1 now runs only κ concurrent ABA instances, we now ask a more ambitious question: can we push it all the way to constant?

Pushing # of ABA *to minimum.* HoneybadgerBFT requires n executions of ABA instances, due to the fact that each ABA instance determines only for input from one peer. Dumbo1 can reduce it as now the "committee" members are prepared with a vector of values. But still, Dumbo1 needs to run κ instances: the procedure after the RBC phase is very similar to the structure of HB-BFT, that picks a common subset containing the index-sets $\{S_i\}$ as elements. Since each node will invoke/enter the i-th ABA instance once it receives S_i from the i-th committee and all values corresponding to the S_i. This causes a challenge that different nodes may enter different ABA instances, there is no "global coordinator" for those instances, thus the only viable way is to concurrently run all of them.

In principle, we still "waste" $\kappa - 1$ ABA instances. This inspires us to find a way to correctly identify only one input vector, thus leads us to re-examine the applicability of multi-value validated Byzantine agreement (MVBA), which outputs one of the inputs of n peers as long as the input satisfies some pre-defined predicate. MVBA was considered impractical for building ACS in [32]. The reason was that existing constructions suffer from a high communication complexity, i.e., the MVBA protocol in [15] has communication complexity $O(n^2|m| + \lambda n^2 + n^3)$ in expectation, where $|m|$ represents the size of MVBA's input values. In many cases, $|m| > \lambda n \log n$, thus the dominating term in the per message communication of its direct construction of ACS [15] is $O(n^2|m|) = \Omega(n^3)$ (Although [15] did not explicitly mention ACS, their atomic broadcast already contains the construction from MVBA to ACS, and the complexity remains even for the recently improved MVBA [2]) and makes the MVBA protocol impractical for building ACS.

But the above claim holds only when MVBA is directly invoked with large size inputs. If we give a closer look, we notice that if $|m|$ is small, then the overall communication complexity of MVBA (and also the corresponding ACS [15]) is no bigger or

even substantially smaller than the ACS in HoneyBadgerBFT ! [3] See Table 1 in Sec. 6. And MVBA has the benefit of a constant number of ABA instances [15]. The key challenge is now reduced to how to invoke MVBA with small inputs to construct an ACS which *may still have large inputs*. This reminds us the widely used conventional wisdom of *"Hybrid Encryption"* in the setting of cryptography.

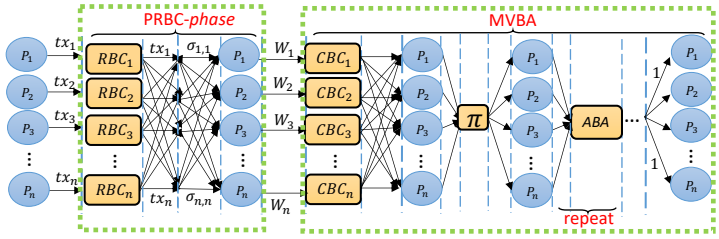

Fig. 4. The structure of Dumbo2-ACS

The right way of applying MVBA. We present an even faster asynchronous BFT protocol via an innovative use of MVBA, we call it Dumbo2. It achieves asymptotically optimal (constant) running time, i.e., Dumbo2 only needs to run (expected) three consecutive instances of ABA, and other complexities remain the same [4].

To work out the details of ACS requires further ideas. Since ACS outputs a subset of inputs, we would first prepare each peer node with a vector of inputs via RBC type of protocols. More importantly, instead of feeding those message vectors into the MVBA protocol, we further prepare each peer with a very *short* "indicator" (the W_i in Fig. 4) and use it as input to join the MVBA protocol. The MVBA protocol will output one such "indicator" which would be used to inform each honest peer to pick the corresponding RBC instances. The tricky part is, in an MVBA protocol, honest peers may output the input (here the short "indicator") from a malicious peer.

We resolve this by designing the "indicator" in a way that any of it serves as a warrant all honest peers would receive the corresponding messages. We formulate a new primitive called *provable reliable broadcast* (PRBC) which augments RBC and further outputs a succinct proof (even by a malicious node) that at least one honest peer has received the input. This can be realized by threshold signing on the RBC index. The ABA within MVBA only needs to be repeated (expected) three times. See Figure 4 for pictorial illustration, where π is a random permutation.

To see why it works: the actual inputs W_i to MVBA includes a indices set and the corresponding proofs. When one honest node outputs W_i, the proof in W_i is valid. This means the messages corresponding to the indices in W_i were all received by enough peers which include at least one honest peer. Then all other honest node will eventually receive those as well.

We remark that though Dumbo2 out-performs Dumbo1 in most of the cases, we choose to keep Dumbo1 for clarity of the presentation: the idea of using each ABA to vote whether to output the *vector* of each "committee" member in Dumbo1, instead of each input as in HB-BFT is simple and intuitive. Such a possibility of more effective voting could be viewed as a stepping stone to motivate the idea of voting to output only one guy's vector, which eventually leads to Dumbo2's idea of using MVBA. Also, since MVBA is still fairly complicated, in some benign cases when f is very small, Dumbo2 might not be better than Dumbo1.

Implementation and experimental evaluations. Besides the asymptotic improvements (see Table 1 in Sec. 6), we implement Dumbo1 and Dumbo2, and also test the performance of the our schemes in the practical WAN environment. We deploy Dumbo1, Dumbo2 and HoneyBadgerBFT on 100 Amazon EC2 t2.medium instances uniformly distributed from 10 different regions across the globe. For a fair comparison, we use the same language and cryptography libraries as [32], and carry out a variety of tests in the same environment. Results show that the efficiency of our schemes indeed outperforms HB-BFT by

[3] Similar phenomenon was also noticed in BEAT [22] that they chose a seemingly more expensive RBC in BEAT1,2, but getting a more efficient protocol for small message.

[4] There exist theoretical works [8] that can achieve constant expected running time for n concurrent execution of ABA protocol, but at the cost of larger message and communication complexities; As pointed out in [32], if we directly adopt their technique to construct ACS, the message and communication complexities will be $O(n^4)$, which render the ACS infeasible for practice.

multifolds, especially when the system is sufficiently large. For example, when $n = 100$, Dumbo1 has a basic latency that is only 22% and Dumbo2 has only 5% of that of HB-BFT. Moreover, Dumbo1 has a peak throughput 3.5× and Dumbo2 has more than 9× of that of HB-BFT. See more details about more tests in Sec. 7.

To showcase the effectiveness of our observation to reduce the number of ABA instances, we pick the result (the running time recorded of each sub protocol from a random node) from one experiment where $n = 32$ with 10^5 transactions (250 bytes each) as input, see Fig. 5. In the figure, for each protocol each line denotes the execution of a sub-protocol instance, e.g., the two bars in the first line of HB-BFT correspond to the first instances RBC_1 and ABA_1 respectively, the second line corresponds to the second instances $\text{RBC}_2/\text{ABA}_2$, and so on. The *consistent broadcast* (CBC) protocol in Dumbo2 is a part of MVBA, which can be regarded as a simplified version of RBC (see section 8.2 for detailed definition).

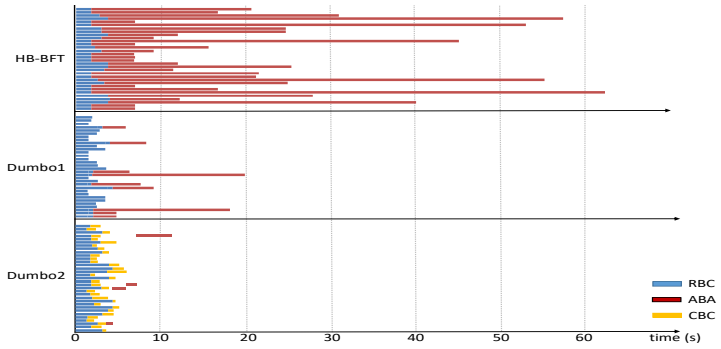

Fig. 5. Running time breakdown of Dumbo1/2 and HoneyBadgerBFT on one random node [5].

1.2 Related work

The consensus problem was firstly introduced by Shostak, Pease and Lamport [27]. As a fundamental problem in distributed computing, it has received extensive attention such that many different variants of the consensus problem have been studied, e.g., [18, 24, 26, 35].

Classic research on asynchronous BFT focused more on understanding the theoretical limits and feasibilities. The famous FLP-impossibility [23] shows that no deterministic consensus protocol can be possible in asynchronous settings as soon as one node may crash. In contrast, Ben-Or [7] and Rabin [36] showed how to circumvent the impossibility via randomization. Those pioneering works inspire many other classic works along the line of asynchronous binary agreement (ABA) [7, 13] which consider input of each node to be just a bit. ABA protocols are known to be an important component towards building a full-fledged BFT or atomic broadcast protocol [2, 15, 22, 25, 32, 37]. We observe (and verified in experiments) that running a large number of ABA instances becomes the bottleneck for efficiency and we strive to minimize the use of it.

HoneyBadgerBFT [32] is the first practical asynchronous atomic broadcast protocol that comes with two major observations: (1) a weaker problem of asynchronous common subset (ACS), originally proposed by Ben-Or et al. [9] can easily be converted to an atomic broadcast without much overhead; (2) the ACS protocol constructed from *reliable broadcast* (RBC) and *asynchronous binary agreement* (ABA) with careful instantiations over-performs the previous thought of constructing it from a more expensive protocol called multi-valued Byzantine agreement (MVBA) which was sometimes used to directly build an atomic broadcast [15].

A recent practical improvement of HoneyBadgerBFT comes from the nice work of BEAT [22]. In particular, they carefully examine different use cases, and make suggestions about the suitable component to choose to deploy in practice. In more detail, besides BEAT3, BEAT4 are for BFT storage only, they presented BEAT0-2 to meet different goals. The components

[5]For Dumbo2 experiments, we intentionally run ABA more times to "simulate" potential adversarial network scheduler (otherwise there could be only one ABA), while the experiments of HB-BFT, Dumbo1 are done without scheduler intervention (same as before [32]). We also note that in theory, in some very rare cases, it may be possible that some RBC instance gets slow so that users have to wait after ABA instances are finished. We do not observe the case that an RBC takes longer time than the slowest ABA in the experiments. Further optimizations beyond reducing # of ABA instances for the asynchronous atomic broadcast are interesting open problems.

in BEAT1 and BEAT2 are chosen in a delicate way that even though the communication complexity seems to be larger for reasonably large messages, but if the message size is small, they are actually faster, see Table 1 in Sec. 6.

As we briefly mentioned earlier, our methods and BEAT are orthogonal and compatible: their work kept the structure of the HoneyBadgerBFT intact, thus have the same round complexity, but cherry-pick the best instantitaions of the underlying components; we focus on restructuring the ACS protocol, but majority of components are the same. Their techniques from BEAT0-BEAT2 can all be applied to our protocols as well. So for experiments, we focus on comparisons with HB-BFT. Combining all their techniques with ours would be interesting future work.

2 MODELS AND PROBLEM STATEMENT

2.1 System model

We now describe our system model.

Setup. In particular, it involves a designated set of n nodes $\{P_i\}_{i\in[n]}$, we use $[n]$ to denote the integers $\{1, 2, \ldots, n\}$. We consider the identities of these nodes to be public, e.g., certified by a PKI. We denote by (PK_i, SK_i) the public/private key pair of nodes P_i. In addition to the already-established identities, a trusted third-party also runs before the protocol to set up all involved threshold cryptosystems.

Static corruptions. We assume that there are f faulty nodes ($3f + 1 \leqslant n$), and consider these faulty nodes are fully controlled by the adversary [15, 32]. Such adversary model means that before the start of the protocol, the adversary is allowed to choose f nodes to completely corrupt them, then the adversary can gets all the faulty nodes' initial internal states and also can let these nodes arbitrarily misbehave during the execution of the protocol.

Asynchronous network. We consider the underlying communication network consisting of asynchronous fully-meshed authenticated point-to-point (p2p) channels. In this model, between any two nodes, there is an established authenticated p2p channel. However, the adversary can fully control the value delivered over all channels, i.e., the adversary can arbitrarily delay, but the values send between honest nodes will eventually be delivered, which explicitly implies two facts: (i) the adversary can arbitrarily reorder values and (ii) the network will not drop any values from honest nodes.

2.2 Design goals

Atomic broadcast. Our end goal is to design an atomic broadcast protocol among n nodes under the system model above. Formally, an atomic broadcast protocol satisfies the following properties with an overwhelming probability:

- *Agreement.* If one honest node outputs a value v, then every honest node outputs v;
- *Total order.* If two honest nodes output sequences of value $\langle v_0, v_1, \ldots, v_j\rangle$ and $\langle v'_0, v'_1, \ldots, v'_{j'}\rangle$, respectively, then $v_i = v'_i$ for $i \leqslant \min(j, j')$;
- *Censorship resilience.* If a value v is input to $n - f$ honest nodes, then it is eventually output by every honest node.

We require the three properties hold with an overwhelming probability. In short, we will adopt the same model as HoneybadgerBFT [32], i.e., atomic broadcast among n nodes against f static corruptions in an asynchronous network.

Atomic broadcast protocol proceeds in consecutive epochs, after each epoch, a new batch of transactions is output and appended to the committed log (see section 8.1).

Asynchronous common subset (ACS). One nice observation from HoneyBadgerBFT [32] is an efficient and simple conversion to an atomic broadcast from a weaker variant called asynchronous common subset (ACS) together with threshold encryption. An ACS essentially let each node output a common subset of all the node inputs. Formally, it satisfies:

- *Agreement.* If an honest node outputs a set V, then every honest node outputs V;
- *Validity.* If an honest node outputs a set V, then $|V| \geqslant n - f$ and V contains the inputs of at least $n - 2f$ honest nodes;
- *Totality.* If $n - f$ honest nodes have an input, then all honest nodes can produce an output.

Remark that there exists a simple conversion from ACS to atomic broadcast by adding threshold encryption, we refer the details in section 8.1 and [32].

Complexity measures. The practicality of BFT protocols depends heavily on their computational complexity. In this paper, we consider the following three metrics:

- *Message complexity:* the expected total number of messages that honest nodes generate during the protocol;
- *Communication complexity:* the expected total bit-length of messages that honest nodes generate during the protocol;
- *Time (round) complexity:* the expected number of rounds of communication before the protocol terminates.

Besides, note that we always consider $n = 3f + 1$ throughout this paper, hence, our BFT protocol is also an optimal resilience which just considers how many nodes may be corrupted.

3 PRELIMINARIES

We introduce definitions for some underlying building blocks.

Reliable broadcast (RBC**)** is a protocol running among a set of n nodes in which there is a node called sender whose aim is to broadcast a value to all the other nodes. More formally, an RBC protocol satisfies the following properties:

- *Agreement.* If any two honest nodes output v and v' respectively, then $v = v'$;
- *Totality.* If an honest node outputs v, then all honest nodes output v;
- *Validity.* If the sender is honest and inputs v, then all honest nodes output v.

Consistent broadcast (CBC) is similar to RBC, but it does not provide *Totality*.

Asynchronous binary agreement (ABA**)** is a special asynchronous Byzantine agreement protocol among n nodes. In an ABA protocol, each node has a single-bit (0/1) input, and their goal is to reach an agreement on the decided bit [14, 16, 33]. More formally, an ABA protocol has the following guarantees:

- *Agreement.* If any honest node decides the bit b, then every honest node decide b;
- *Termination.* If all honest nodes receive input, then every honest node decides a bit;
- *Validity.* If any honest node decides b, then at least one honest node received b as input.

Remark: As many previous works [14, 16, 32, 33], the *termination* property here for ABA only requires all honest node to *decide* (in the sense of outputting a value for further applications, while not halting the protocol). It is possible that they each decide a value, but some node still continues waiting messages in the protocol [32]. As in [32], we use "output" and "decide" interchangeably on a bit in ABA. Please see section 8.2 for details of concrete constructions of RBC, CBC and ABA [6] protocols.

Multi-valued Byzantine agreement (MVBA**):** The MVBA [2, 15] allows agreement on arbitrary values instead of being restricted to binary values. The protocol has a global, polynomial-time computable predicate Q known to all nodes, which is determined by the particular application. The basic idea of the protocol is that each party proposes a (different) value that contains certain validation information as input and outputs an value which satisfies the Q as the decision value. The protocol ensures that the decision value was proposed by at least one party. Each honest node only inputs a value to MVBA that satisfies Q.

More formally, an MVBA protocol satisfies the following properties except with negligible probability:

- *Termination.* If every honest node P_i inputs with an externally valid value v_i, then every honest node outputs a value;
- *External-Validity.* If an honest node outputs a value v, then $Q(v) = True$;
- *Agreement.* All honest nodes that terminate output the same value;
- *Integrity.* If all nodes are honest and if some nodes output v, then some nodes proposed v.

Threshold signature scheme: Let $0 \leqslant t \leqslant n$, A (t, n)-non interactive threshold signature scheme is a tuple of algorithms which involves n nodes and up to $t - 1$ node can be corrupted. The signature schema satisfies the following properties:

- *Unforgeability:* No polynomial-time adversary can forge a signature that can be verified correctly (by honest parties) of any message m without querying the signature algorithm;

[6] The original HoneyBadgerBFT [32] used an ABA protocol [33] which requires a strong common coin that cannot be realized by the threshold coin scheme [16]. The revised version of HB-BFT added a fix [31] of the ABA protocol (see also Alg. 7 without the "amendament"). In our experiments, we adopted this revised ABA.

- *Robustness:* When a message m is provided as the input of the signature algorithm, eventually all honest parties can get a signature of m that can be correctly verified.

A threshold signature scheme has the following algorithms:

- *Key generation algorithm:* $\mathsf{SigSetup}(1^\lambda, n, t) \rightarrow \{mpk, PK, SK\}$. Give a security parameter λ and generates a special public key mpk, a vector of public keys $PK := (pk_1, \cdots, pk_n)$, and a vector of secret keys $SK := (sk_1, \cdots, sk_n)$;
- *Signing algorithm:* $\mathsf{SigShare}_t(sk_i, m) \rightarrow \sigma_i$. On input a message m and a secret key share sk_i, this deterministic algorithm outputs a signature share σ_i;
- *Share verification algorithm:* $\mathsf{ShareVerify}_t(m, (i, \sigma_i)) \rightarrow 0/1$. Given a message m, a signature share σ_i and an index i, this deterministic algorithm outputs 1 or 0 depending on whether σ_i is a valid signature share generated by P_i or not. The *correctness* requirement needs that: for $\forall$ m and $i \in [n]$, $Pr[\mathsf{ShareVerify}_t(m, (i, \mathsf{SigShare}_t(sk_i, m))) = 1] = 1$;
- *Combining algorithm:* $\mathsf{Combine}_t(m, \{(i, \sigma_i)\}_{i \in S}) \rightarrow \sigma/\perp$. Given a message m, and a list of pairs $\{(i, \sigma_i)\}_{i \in S}$, where $S \subset [n]$ and $|S| = t$, this algorithm outputs either a signature σ for message m, or $\perp$ when $\{(i, \sigma_i)\}_{i \in S}$ contains ill-formed signature share (i, σ_i);
- *Signature verification algorithm:* $\mathsf{Verify}_t(m, \sigma) \rightarrow 0/1$. Given a message m and a signature σ, this algorithms outputs 1 or 0 depending on whether σ is a valid signature for m or not. The *correctness* requires that: for $\forall$ m, $S \subset [n]$ and $|S| = t$, $\Pr[\mathsf{Verify}_t(m, \mathsf{Combine}_t(m, \{(i, \sigma_i)\}_{i \in S})) = 1 \mid \forall i \in S, \mathsf{ShareVerify}_t(m, (i, \sigma_i)) = 1] = 1$.

$(1, \kappa, \epsilon)$-Committee election (CE): A CE protocol is executed among n nodes (identified from 1 through n). If at least $f + 1$ honest nodes participate, the protocol terminates with honest nodes output a κ-sized committee set C such that at least one of C is honest nodes. In particular, a protocol is said to be $(1, \kappa, \epsilon)$-committee election, if it satisfies the following properties except with negligible probability in cryptographic security parameter λ:

- *Termination.* If $f + 1$ honest nodes activate the protocol CE, all messages among honest nodes arrive, then all honest nodes output C;
- *Agreement.* Any two honest nodes output the same set C;
- *Validity.* If any honest node outputs C, (i) $|C| = \kappa$, (ii) the probability of every node $P_i \in C$ is same, and (iii) C contains at least one honest node with at least probability $1 - \epsilon$;
- *Unpredictability.* Before invocation by one honest node, the probability of the adversary to predict the returned committee is at most $1/\binom{n}{\kappa}$.

Remark that a $(1, \kappa, \epsilon)$-CE can be constructed directly from a threshold coin-tossing, which can be readily derived from threshold signatures. At least one honest node is elected in C with an overwhelming probability $1 - \epsilon - \mathsf{negl}(\lambda)$, where $\mathsf{negl}(\lambda)$ is a negligible function in cryptographic security parameter λ, and ϵ is $exp(-\Omega(\kappa))$ (c.f. Lemma 4.1 for details).

4 DUMBO1: A FAST ASYNCHRONOUS BFT PROTOCOL

In this section, we present our first ACS, which is called Dumbo1-ACS. Applying the same conversion from ACS to atomic broadcast [32] (adding threshold encryption), we can obtain a new atomic broadcast: Dumbo1. We will focus on the ACS protocol below.

4.1 Dumbo1-ACS

High level overview. The core of our design is to reduce the number of ABA instances needed in an ACS execution. As briefly elaborated in the Introduction, the reason that HoneyBadgerBFT (and also BEAT) needs n instances of ABA is due to the following reason: the peers need to agree what to do for each peer's input via one ABA instance. Observe that after the first RBC phase, each peer is prepared with a subset of inputs.

Instead of investing an ABA for each input, we would let a small number κ of "aggregators" to nominate which *subset* of inputs to output (based on what it has already received). In this way, each ABA instance is now used to let the nodes to determine whether they agree on the i-th nominated subset S_i. We remark again that the nomination procedure is also using

RBC, however, the inputs are just indices-set S_i instead of the actual data load. Also the nominator/committee election is just one coin-tossing at best, thus the overhead is minimal compared to the saved ABA instances.

Note that κ instances of ABA protocol are still needed, otherwise one honest node may decide to follow S_i while the other honest node may decide to follow S_j, as each of them indeed receives the corresponding input values but ends up violating the agreement.

In slightly more detail, as illustrated in Figure 3 in Introduction, our Dumbo1-ACS includes two phases of RBC, denoted as *data*-RBC and *index*-RBC respectively. The data-RBC instances are executed by the nodes to broadcast their inputs. κ leaders will then be selected. The index-RBC instances are only initiated by the selected members when they have received $n - f$ values from those data-RBC instances. Each index-RBC is used to broadcast the indexes indicating which $n - f$ values that a selected member has already received. In the last phase, an honest node will input 1 to the i-th ABA instance if it has already received S_i (with size $n - f$) and all the corresponding values in the data-RBC instances.

Committee election. The election of those committees/nominators is standard practice, i.e., randomly choosing κ nodes to ensure the probability that one of them is honest with an overwhelming probability in κ. Usually, the probability that none of κ random peers to be honest is at most $(1/3)^\kappa$ (see Lemma 4.1). In practice, we could let system designer to choose $\kappa = min\{\kappa_0, f + 1\}$ in a way that $(1/3)^{\kappa_0} \leq \epsilon_0$ for any small ϵ_0 he likes. If the number of faulty nodes f is small, we can simply set $\kappa = f + 1$ to guarantee to include one honest node. Here we describe a committee election construction, the details of this election procedure CE is illustrated in the following Algorithm 1, the threshold coin-tossing scheme is the underlying Algorithm of CE. Threshold coin-tossing scheme is a tuple of algorithms/protocols, including a private function CShare_i, and two public functions: CShareVerify and CToss. Informally, given $f + 1$ validated coin shares which generated by CShare_i, the function CToss returns a unique and pseudorandom set of size κ. See section 8.2 for details of threshold coin-tossing scheme.

Algorithm 1 Committee election (CE): for party P_i

1: **Local variables initialization:** $\Sigma \leftarrow \{\}$
2: **upon** CE(id) **do**
3: $\quad \sigma_i \leftarrow \mathsf{CShare}_i(id)$;
4: $\quad$ **send** (SHARE, id, σ_i) to all parties;
5: $\quad$ **wait** until $|\Sigma| = f + 1$
6: $\quad\quad$ **return** CToss(id, Σ);
7: **upon** receiving (SHARE, id, S_j) from P_j for the first time **do**
8: $\quad$ **if** CShareVerify(id, j, σ_j)=true **then**
9: $\quad\quad \Sigma \leftarrow \Sigma \cup \{\sigma_j\}$;

Construction of Dumbo1-ACS. Now we describe the construction of our Dumbo1-ACS. The detailed process of Dumbo1-ACS is shown in Algorithm 2. As illustrated in Figure 3 in Introduction, our Dumbo1-ACS includes two phases of RBC, the first phase is to broadcast value, the second phase is to broadcast indices.

The Dumbo1-ACS protocol composed of five logical phases, the detailed protocol proceeds as follows:

- *Value broadcast*: (line 02). All nodes P_i input their value v_i to RBC_i protocol.
- *Committee election*: (line 03-03). All nodes participate CE protocols to select committee C, and take the committee member' identities into a set CMIS.
- *Indices broadcast*: (line 06-09). When the committee members have received $n - f$ Value message from distinct RBC instances, then they will broadcast these indexes of $n - f$ Value messages through RBC.
- *ABA phase*: (line 10-17). When one honest node has already received Index message (Index,S_j) from committee member P_j and all corresponding Value message from the RBC instances, then input 1 to the ABA_j instance; if one honest node gets an output from any ABA_j and $j \in$ CMIS, then input 0 to other ABA instance which no input has been provided to yet.

- *Output phase*: (line 18-25). When a node terminates in all κ instances of ABA, for any $x \in$ CMIS, if ABA_x outputs 1, then the node waits Index message from RBC_x and gets a set S, and further waits Value message to get v_j for all $j \in S$ from RBC_j and finally outputs $\{v_j\}_{j \in S}$.

Algorithm 2 The Dumbo1-ACS protocol (for party P_i) in consecutive epoch r

1: **Let** $\{\text{RBC}_j\}_n$ refer to n instances of the reliable broadcast protocol, where P_j is the sender of RBC_j, and ABA_j refer to the ABA instance corresponding committee member P_j. **Initial:** Committee member identities set CMIS = $\emptyset$.
2: **Input** (Value, v_i) to RBC_i; ▷ data-RBC
3: **Invoke** Committee Election protocol CE(r);
4: **wait** until Committee: $\{P_{j_1}, P_{j_2}, \cdots, P_{j_\kappa}\} \leftarrow \text{CE}(r)$
5: CMIS $\leftarrow \{j_1, j_2, \cdots, j_\kappa\}$;
6: **if** $P_i \in$ Committee **then**
7: **wait** until receiving $n - f$ Value messages $\{(\text{Value}, v_{i_1}), (\text{Value}, v_{i_2}), \cdots, (\text{Value}, v_{i_{n-f}})\}$ from distinct RBC instance
8: let $S_i = \{i_1, i_2, \cdots, i_{n-f}\}$;
9: input (Index, S_i) to RBC_i; ▷ index-RBC
10: **upon** receiving Index message (Index, S_j) from committee member P_j and $|S_j| = n - f$ **do**
11: **if** no input has been provided to ABA_j **then**
12: **wait** until all (Value, $v_x)_{x \in S_j}$ have received
13: input 1 to ABA_j;
14: **upon** receiving 1 from any ABA_j and $j \in$ CMIS **do**
15: **for** x: $x \in \{\text{CMIS} - j\}$ **do**
16: **if** no input has been provided to ABA_x **then**
17: input 0 to ABA_x;
18: **upon** all κ ABA instances have completed **do**
19: **for** x: $x \in$ CMIS **do**
20: **if** ABA_x output 1 **then**
21: **wait** Index message: (Index, S_x) $\leftarrow \text{RBC}_x$
22: $S \leftarrow S \cup S_x$;
23: **for** $j \in S$ **do**
24: **wait** Value message: (Value, v_j) $\leftarrow \text{RBC}_j$
25: **Output** $\cup_{j \in S}\, v_j$.

4.2 Security analysis

Dumbo1 realizes an atomic broadcast via the combination of ACS and threshold encryption. To prove the security of Dumbo1, we need to go in two steps: (1) a reduction from atomic broadcast to ACS and threshold encryption; (2) to show our new constructed Dumbo1-ACS protocol indeed satisfies the ACS properties.

The proof for Step 1 has been given in [32], for more details please refer to section 8.1. Here we focus mainly on the Step 2 and prove Dumbo1-ACS satisfies all properties of ACS.

Lemma 4.1. *(Validity of* CE.*) If* $n = 3f + 1$, $\kappa \leqslant f$ *and* CE(id) *returns a set* C*, then the set* C *containing at least one honest nodes except with* $exp(-\Omega(\kappa))$ *probability.*

Proof. Due to the pseudo-randomness, hence, the total case is $\binom{n}{\kappa}$ of random choose κ nodes, and the total case is $\binom{f}{\kappa}$ of the set C containing no honest nodes. Let p is the probability of the set C containing no honest nodes. So, we have

$$p = \frac{\binom{f}{\kappa}}{\binom{n}{\kappa}} = \frac{f!(n-\kappa)!}{(f-\kappa)!n!} \leqslant (\frac{1}{3})^\kappa = exp(-\Omega(\kappa)).$$

□

Remark: If f is small, we can simply set $\kappa = f + 1$. Algorithm 1 satisfies the *Termination, Agreement and Unpredictability* properties of CE follows from the properties of threshold coin-tossing.

Theorem 4.2. *With except $exp(-\Omega(\kappa))$ probability, the Dumbo1-ACS protocol satisfies Agreement, Validity, and Totality of ACS, assuming the underlying* RBC, CE *and* ABA *protocols are secure.*

Proof. *Agreement:* To prove that Dumbo1-ACS satisfies the agreement property, we show that when an honest node outputs V, then every honest node outputs V. Assume that an honest node P has a output $V = \{v_j\}_{j \in S}$.

The indices S must be contained in some index sets. W.l.o.g, we assume S is included only one index set S_k, (Index, S_k) was received in some RBC (denoted as RBC_k), then the node must have received 1 in the corresponding ABA instance (denoted as ABA_k). Due to the *agreement* property of ABA, all honest nodes will also receive 1 in ABA_k. Hence, all honest nodes will wait an Index message output from RBC_k, due to *totality* and *agreement* of RBC, so all other honest nodes will receive same Index message (Index, S_k).

On the other hand, due to the *validity* of ABA, at least one honest node P' inputs 1 to ABA_k. It implies that this honest node must have received Index message (Index, S_k) and these Value messages $\{\text{Value}, v_j\}_{j \in S_k}$ corresponding to the index set S_k. The *totality* and *agreement* of RBC now can ensure that all other honest nodes including P will receive $\{\text{Value}, v_j\}$ for any $j \in S_k$.

Hence, every honest node outputs $\{v_j\}_{j \in S_k} = \{v_j\}_{j \in S} = V$.

Validity: To prove that Dumbo1-ACS satisfies the validity property, we show that $|V| \geqslant n - f$ and V contains the input of at least $n - 2f$ honest nodes when an honest node outputs a set V.

If an honest node P_i outputs a set $V = \{v_j\}_{j \in S}$, W.l.o.g, we assume S is included only one index set S_k, (Index, S_k) was received in index-RBC_k. According to the Algorithm 2, we can know ABA_k return 1, due to the *validity* of ABA, at least one honest node (say P') inputs 1 to ABA_k. It implies that P' must have received Index message (Index, S_k) and all Value message $\{\text{Value}, v_j\}_{j \in S_k}$ corresponding the index set S_k, where $|S_k| = n - f$.

The *totality* and *agreement* of RBC now can ensure that all honest nodes including P_i will receive $\{\text{Value}, v_j\}_{j \in S_k}$, where $|S_k| = n - f$. So, $V = \{v_j\}_{j \in S_k}$. Hence, we have $|V| \geqslant n - f$. Notice that there are at most f byzantine nodes, there must be at least $n - 2f$ inputs from honest nodes in set V.

Totality: To prove that Dumbo1-ACS satisfies the totality property, we show that all honest nodes produce an output if $n - f$ honest nodes have an input.

Since $n - f$ honest nodes have an input, according to the *validity* of RBC, hence, every committee member can receive $n - f$ Value messages from distinct RBC instance. So, every committee member can activate the second phase RBC instance. Besides, according to the CE protocols, there exists at least an honest node (say P_i) belongs to the committee.

Next, we will prove that at least one ABA instance returns 1.

Firstly, suppose all ABA instances output 0, in this case, line 14-17 will never execute, that is to say, 0 will never input to any ABA instance by honest nodes. However, according to the *validity* of ABA, at least one honest node inputs 0 to ABA, which induces contradiction.

Secondly, since the committee member P_i can activate the second phase RBC_i instance, if all honest nodes have not received 1 from any ABA all the times, in this case, all honest nodes can receive valid Index message from P_i (the *validity* of RBC) and corresponding Value message from RBC instances (the *totality* of RBC), next, all honest nodes input 1 to ABA_i. Again according to the *validity* of ABA, the ABA_i will return 1 to all.

Hence, there exists at least one ABA (say ABA_k) instance returning 1. Due to *validity* of ABA, at least one honest node (say P') inputs 1 to ABA_k. It implies that such an honest node must have received Index message (Index, S_k) and all Value message $\{\text{Value}, v_j\}_{j \in S_k}$ corresponding the index set S_k. The *totality* and *agreement* of RBC now can ensure that all honest nodes will receive $\{\text{Value}, v_j\}_{j \in S_k}$. Hence, all honest nodes can produce an output $\{v_j\}_{j \in S_k}$. □

5 DUMBO2: A FASTER ASYNCHRONOUS BFT PROTOCOL

In this section, we present a further improved ACS protocol Dumbo2-ACS, which reduces the number of ABA instances to constant, thus guarantees termination within a constant running time. We show a new construction of ACS using RBC and

MVBA. More interestingly, our new method demonstrates an innovative use of MVBA can actually lead to more efficient ACS, which was considered less promising than using RBC and ABA in [32].

5.1 Dumbo2-ACS

High level overview. As discussed above, Dumbo1 improves HB-BFT in the sense that reduces the number of ABA instances to κ, but still they all need to be run. In order to minimize the usage of ABA, we need to (1) prepare each peer with a vector of inputs from enough peer nodes; (2) find a way to identify and output one of them. The former is easy that an RBC phase already achieves it. The latter inspire us to re-examine the possibility of MVBA which outputs only one input (not necessarily from an honest node, but satisfies some condition). As explained in Introduction (and also Table 1 in Sec. 6), current MVBA constructions were considered impractical for ACS due to its high communication complexity. However, if we examine all the terms in the communication complexity, the dominating term changes when message size becomes small, MVBA could even over-perform! More importantly, MVBA [15] only needs to run three consecutive ABA instances in expectation.

Considering the conventional wisdom of *hybrid encryption*, the heavier public key encryption is only used to hide a short session key, while the actual (potentially large) message will be encrypted using symmetric key encryption using the session key. In analog to that, we similarly use indices as input to invoke MVBA. Now we have one more challenge that MVBA may output one such index-set from a dishonest node. To resolve this, we propose *provable* RBC that further outputs a succinct proof s.t., whoever produces such a proof, it guarantees that all honest nodes will receive the input value.

Provable reliable broadcast (PRBC**).** A natural way to obtain such a (succinct) proof is to get acknowledgement from enough nodes, which can be realized via threshold signing on the RBC identifier. The PRBC protocol with an identifier id, and a verify algorithm Verify is denoted PRBC_{id}. Formally, an PRBC_{id} protocol satisfies the following properties except a negligible probability:

- *Agreement.* If one honest node outputs v, another honest node outputs v', then $v = v'$;
- *Totality.* If any node outputs a pair (id, σ) and $\text{Verify}(id, \sigma) = 1$, then all honest nodes output a value v and (id, σ);
- *Validity.* If the sender is honest and inputs v, then all honest nodes output v and valid string (id, σ);
- *Succinctness.* The length (size) of valid string σ is independent with the length of value v.

Algorithm 3 The PRBC_{id} protocol with epoch r (for party P_i, where the sender is P_s and $id = \langle r, s \rangle$)

1: **Let** RBC_{id} refer to the instance of the reliable broadcast protocol, where P_s is the sender of RBC_{id}; $\{DS_s\} = \emptyset$.
2: **if** $P_i = P_s$ **then**
3: **upon** receiving input value v_s **do**
4: input $\{\text{Value}, v_s\}$ to RBC_{id};
5: **upon** receiving Value message $\{\text{Value}, v\}$ from RBC_{id} **do**
6: $\sigma_{is} \leftarrow \text{SigShare}_{f+1}(sk_i, id)$;
7: multicast $(\text{Done}, id, \sigma_{is})$;
8: **upon** receiving a Done message $(\text{Done}, id, \sigma_{js})$ from node P_j for the first time **do**
9: **if** $\text{ShareVerify}_{f+1}(id, (j, \sigma_{js})) = 1$ **then**
10: $DS_s \leftarrow DS_s \cup \{j, \sigma_{js}\}$;
11: **upon** $|DS_s| = f + 1$ **do**
12: $\sigma_s \leftarrow \text{Combine}_{f+1}(id, DS_s)$;
13: **return** $(\text{Finish}, id, \sigma_s)$.

A PRBC can be constructed from RBC and threshold signature, it is shown in Algorithm 3. The PRBC protocol can be decomposed in three logical phases, the details are as follows:

- Value broadcast phase: (line 02-04). If the node is sender, then the node P_s inputs the value v_s to RBC protocol.
- Output value phase: (line 05-07). If honest nodes receive a value from sender, then the nodes send a threshold share signature of id to all.

- Output signature phase: (line 08-13). If nodes received $f+1$ valid threshold share signature of id, they can combine these share signature into a threshold signature σ of id, then output the σ.

Construction of Dumbo2-ACS. Now we give the construction of our Dumbo2-ACS protocol, the details of which are shown in Algorithm 4. We denoted MVBA_r as MVBA protocol with identification r. As illustrated in Figure 4 in Introduction, the Dumbo2-ACS includes two part: PRBC and MVBA.

We will use $W = \{(s_1, \sigma_1), (s_2, \sigma_2), \cdots, (s_n, \sigma_n)\}$ as the input to MVBA_r for each node. In particular, s_i is put in W if the corresponding σ_i on s_i is received. The predicate Q of the MVBA_r will output 1 if at least $n-f$ distinct i satisfy $s_i \neq \perp$ in W, and for each (s_i, σ_i) of W, it is a valid output of PRBC, i.e., $\mathsf{Verify}_{f+1}(\langle r, s_i \rangle, \sigma_i) = 1$ if $s_i \neq \perp$. The Dumbo2-ACS protocol can be decomposed in three logical phases, the detailed protocol proceeds as follows:

- Value broadcast phase: (line 03-04). All nodes P_i input their value v_i to PRBC protocol, and wait for $n-f$ distinct Finish messages.
- MVBA phase: (line 08-10). Upon receiving $n-f$ distinct Finish messages, then invoke the MVBA protocol and wait to get an output $\overline{W}$ from MVBA.
- Output phase: (line 11-13). All honest nodes wait Value message from PRBC according to the $\overline{W}$.

Algorithm 4 The Dumbo2-ACS protocol (for party P_i) in consecutive epoch r

1: **Let** $\{\mathsf{PRBC}_{\langle r,j \rangle}\}_n$ refer to n instance of provable reliable broadcast protocol, where P_j is the sender of $\mathsf{PRBC}_{\langle r,j \rangle}$, and the Q be the following predicate:
$Q_r[\{(s_1, \sigma_1), (s_2, \sigma_2), \cdots, (s_n, \sigma_n)\}] \equiv$ (at least $n-f$ distinct i satisfy $s_i \neq \perp$ and $\mathsf{Verify}_{f+1}(\langle r, s_i \rangle, \sigma_i) = 1$).
2: **Initial:** $W = \{(s_1, \sigma_1), (s_2, \sigma_2), \cdots, (s_n, \sigma_n)\}$, where $(s_j, \sigma_j) \leftarrow (\perp, \perp)$ for all $1 \leqslant j \leqslant n$; $FS = 0$.
3: **upon** receiving input value v_i **do**
4: input $\{\mathsf{Value}, v_i\}$ to $\mathsf{PRBC}_{\langle r,i \rangle}$;
5: **upon** receiving a Finish message $(\mathsf{Finish}, \langle r, j \rangle, \sigma_j)$ **do**
6: $(s_j, \sigma_j) \leftarrow (j, \sigma_j)$;
7: $FS = FS + 1$;
8: **upon** $FS = n - f$ **do**
9: propose W for the MVBA_r;
10: **wait** the MVBA_r to return $\overline{W} = \{(\bar{s}_1, \bar{\sigma}_1), (\bar{s}_2, \bar{\sigma}_2)), \cdots, (\bar{s}_n, \bar{\sigma}_n)\}$
11: Let $S \subset [n]$ be the set of $\bar{s}_j \neq \perp$ for $1 \leqslant j \leqslant n$.
12: Wait until receive v_j from $\mathsf{PRBC}_{\langle r,j \rangle}$ for all $j \in S$.
13: Finally output $\cup_{j \in S} v_j$.

5.2 Security Analysis

Intuition: Similarly with the Dumbo1-ACS, when an honest node outputs a subset of values, it implies that the node has received the corresponding index subsets. Due to the termination and agreement of the MVBA, all honest nodes will receive the same index-set W.

Besides, the external-validity of MVBA ensures that the index-set W satisfies the predicate Q, which means that the input message corresponding to each index will be received by all honest nodes in the PRBC, and the size of subsets is at least $n-f$.

Like in Dumbo1, we also focus on the proof of Dumbo2-ACS. Note that the property of *succinctness* follows from proper choice of threshold signature scheme whose signature size is λ.

Lemma 5.1. *The Algorithm* 3 *satisfies the Agreement, Validity and Totality properties of* PRBC*, assuming the underlying* RBC *and threshold signature scheme are secure.*

Proof. *Agreement:* If one honest node outputs v and another honest node outputs v', according to the *agreement* property of RBC, we have $v = v'$.

Totality: If any node outputs string (id, σ) and $\mathsf{Verify}_{f+1}(id, \sigma) = 1$, it implies that at least one honest has received a value v from the sender P_s. If not, at most f share signatures on id will be generated; hence, it's impossible for any malicious node to outputs a valid threshold signature. Otherwise, it will violate the *unforgeability* property of threshold signature scheme. Due to *totality* property of RBC, all honest nodes eventually output v.

Validity: If the sender is honest and inputs v, from the *validity* property of RBC, all honest nodes output v. Besides, according to the Algorithm 3, after receiving a value from the sender, each honest node will multicast a share signature of id to all, so each honest node can receive at least $f + 1$ valid share signatures. Hence, all honest nodes can output a valid string (id, σ). □

Theorem 5.2. *With except negligible probability, the Dumbo2-ACS protocol satisfies the Agreement, Validity, and Totality properties of ACS, assuming the underlying* PRBC *and* MVBA *are secure.*

Proof. *Agreement:* To prove that Dumbo2-ACS satisfies the agreement property, we need to prove that when an honest node outputs V, then every honest node outputs V.

Assume an honest node P_i outputs $V = \{v_j\}_{j \in S}$. It implies that for any $j \in S$, $\bar{s}_j \neq \perp$ in $\overline{W}$, following *external-validity* of MVBA, we know there is a valid proof (threshold signature) $\bar{\sigma}_j$ for $\bar{s}_j$. Hence, according to the *totality* of PRBC, all honest nodes including P_i output v_j. Besides, the *agreement* of MVBA ensures all honest nodes have the same S. So every honest node also outputs $V = \{v_j\}_{j \in S}$.

Validity: To prove that Dumbo2-ACS satisfies the validity property, we show that $|V| \geqslant n - f$ and V contains at least $n - 2f$ inputs from honest nodes when an honest node outputs a set V.

If an honest node P_i outputs a set $V = \{v_j\}_{j \in S}$. Due to the predicate Q of MVBA, (1) for any $\bar{s}_i \in \overline{W}$, there is a corresponding valid threshold signature $\bar{\sigma}_i$ if $\bar{s}_i \neq \perp$, (2) at least have $n - f$ distinct $\bar{s}_i \neq \perp$. Hence, according to the *totality* of PRBC, P_i outputs the data set V including $n - f$ values.

Note that there are at most f byzantine nodes, hence, there must be at least $n - 2f$ values from honest nodes' input in set V.

Totality: To prove that Dumbo2-ACS satisfies the totality property, we show that all honest nodes produce an output if $n - f$ honest nodes have an input.

According to the *validity* of PRBC: if a sender i is honest, then all honest nodes can receive its input v_i and (i, σ_i). Now that $n - f$ honest nodes have an input, thus every honest node can receive at least $n - f$ distinct valid pairs (id, σ_{id}). Hence, every honest node can receive at least $n - f$ distinct Finish messages and define an externally valid value W_i as input to MVBA. Following *agreement* and *termination* of MVBA, all honest nodes can get the same output $\overline{W}$ from MVBA.

Besides, the value $\overline{W}$ satisfies the predicate Q due to the *external-validity* property of MVBA. Hence, for any $\bar{s}_i \neq \perp$, there is a corresponding valid threshold signature $\bar{\sigma}_i$. Let S be the set of $\bar{s}_i \neq \perp$ for all $1 \leqslant i \leqslant n$. The *totality* of PRBC now can ensure that all honest nodes will receive $\{v_i\}_{i \in S}$. □

6 EFFICIENCY ANALYSIS

Now let us summarize the efficiency of the Dumbo protocols. Throughout the paper, we consider $|m|$ denotes the message size, λ is the security parameter for the cryptographic primitives and also denotes the size of (threshold) signature.

Efficiency of Dumbo1. Firstly, let's go through the process of the Dumbo1-ACS. According to the process of Algorithm 2, the message exchange appears in four places. First, all parties participate n concurrent RBC instances to broadcast input values to all, the second phase is all parties to participate the committee election, then the third phase is all parties to participate κ concurrent RBC instances to broadcast indexes message to all, the last phase is all parties to participate κ concurrent ABA instances to agree on whose indices to adopt. We present the following observation from [8]. Suppose $X_1, X_2, \ldots, X_n$ be independent random variables such that for every $1 \leqslant i \leqslant n$, $Pr[X_i > j] = q^j$ $(0 < q < 1)$. If $Y = max\{X_i\}$, then $EXP[Y] = O(\log n)$.

Hence, the expected *time complexity* of Dumbo1 is $O(\log \kappa)$ due to the κ ABA instances (and all other concurrent RBC instances have constant rounds in total). *Message complexity* is still $O(n^3)$ as the added components κ (κ is normally substantially

smaller than n) more RBC instances cost no more than $O(\kappa \cdot n^2)$, while there are also some reductions due to smaller # of ABA instances. Regarding *communication complexity*, the data-RBC instances generate $O(n^2|m| + \lambda n^3 \log n)$ bits communication, while the added index-RBC instances generate only $O(n^2\lambda)$ bits communication, (ignoring the reduced part due to ABA reduction). Hence, the communication complexity of Dumbo1-ACS is still $O(n^2|m| + \lambda n^3 \log n)$ as before.

Efficiency of Dumbo2. Similarly, here we also go through the process of Dumbo2-ACS. From the Algorithm 4, the message exchange appears in two places. First, all parties participate n concurrent PRBC instances, the second phase an MVBA instance.

The expected *time complexity* of Dumbo2 is $O(1)$ due to all concurrent PRBC instances have constant rounds in total and the running time of MVBA is also constant. *Message complexity* still keeps same with the HoenyBadgerBFT and is $O(n^3)$, due to it needs n PRBC instances (incur $O(n^3)$ message) and the MVBA message complexity is $O(n^2)$. Since it needs n PRBC instances, hence, the *communication complexity* of the Dumbo2 was dominated by concurrent n PRBC instances and up to $O(n^2|m| + \lambda n^3 \log n)$. In fact, the MVBA phase only generates $O(\lambda n^3)$ bit communication.

As shown in Table 1, the table summarizes the asymptotic performance of ACS of Dumbo with ACS of several other atomic broadcast protocols in the asynchronous setting:

Table 1. Detailed performance metrics of ACS.

	Complexity‡		
Protocol	Time	Communication	Message
HB-BFT/BEAT0	$O(\log n)$	$O(n^2\|m\| + \lambda n^3 \log n)$	$O(n^3)$
BEAT1/BEAT2	$O(\log n)$	$O(n^3\|m\| + \lambda n^3)$	$O(n^3)$
Dumbo1	$O(\log \kappa)$	$O(n^2\|m\| + \lambda n^3 \log n)$	$O(n^3)$
Dumbo2	$O(1)$	$O(n^2\|m\| + \lambda n^3 \log n)$	$O(n^3)$

‡ Time means expected running time (or communication rounds). One may notice that the communication complexities here look different with that in [32]: here the communication and message complexity both refer to the *total* complexity for the whole ACS with all terms, while in [32] they calculated complexity per transaction, and "ignored" the terms they considered small for *large-size* input.

7 EXPERIMENTAL EVALUATIONS

We implement Dumbo1, Dumbo2 (the full fledged atomic broadcast) and HoneyBadgerBFT, and deploy them on Amazon AWS. Note that we used same parameters and environment as HB-BFT. We carry out a serial of experiments in various settings with different system scales and input sizes. The results demonstrate that we have significant improvements on both latency and throughput over HBBFT, especially when n gets moderately large. Some example comparisons are given in Table 2 collected from random nodes.

	Basic Latency (s)					Throughput (tx/s)				
System Scale	HB-BFT [32]	Dumbo1		Dumbo2		HB-BFT [32]	Dumbo1		Dumbo2	
n=32	70	19	↓ 73%	7.5	↓ 89%	8430	11313	↑ 34%	15121	↑ 79%
n=64	240	49	↓ 80%	14	↓ 94%	4453	12111	↑ 172%	18692	↑ 320%
n=100	491	90	↓ 78%	24	↓ 95%	1934	8814	↑ 356%	17767	↑ 819%

Table 2. Improvements of latency and throughput

Implementation details. Our prototypes of Dumbo1 and Dumbo2 are implemented in Python, part of which were developed from the implementation of HoneyBadgerBFT provided by [31, 32]. Each node runs on a separate EC2 instance. At the beginning of the program, nodes establish communication channels with each other through unauthenticated TCP sockets. All nodes behave honestly by default. In the implementation of Dumbo1 and Dumbo2, the ABA instantiation was the “corrected” version, see [31] and Alg. 7 below in the section 8.2 for details.

We implement Boldyreva’s pairing-based threshold signature scheme [12] on MNT224 curve for threshold signature (also for random-number generation, coin-tossing and committee-election). For threshold encryption, we adopt the threshold

encryption scheme from Baek and Zheng [6] using SS512 symmetric bilinear group. These threshold cryptography schemes were implemented with Charm [3] Python wrappers for PBC library [28]. To implement Reed-Solomon codes, we use the zfec library [39].

Evaluation. We deploy the protocols on Amazon EC2 services, run them on 100 Amazon EC2 t2.medium instances uniformly distributed from 10 different regions (Tokyo, Singapore, Mumbai, Stockholm, Paris, Frankfurt, St. Paulo, California, Virginia and Central Canada) across the globe, each with two virtual CPUs and 4GB memory. We carry out several groups of tests in different system scale, varying the batch size B from 4 to 2×10^6 transactions. We assume the size of each transaction are 250 bytes[7] and use the parameter $n = 4f$. Besides, we set the error parameter $\epsilon = 10^{-8}$ to determine the size of committee, which is sufficiently small in practice to ensure error-free operation within ten years even if suppose the time is 1 second for each epoch. Note that we always try to make instances distributed geographically heterogeneous to simulate the practical WAN environment. E.g., When 8 nodes are tested, they will be located in 8 different areas. Significant delay and fluctuation therefore commonly exist in the communication channels. It is natural that if more instances are located in the same area, the efficiency of the protocols would be higher.

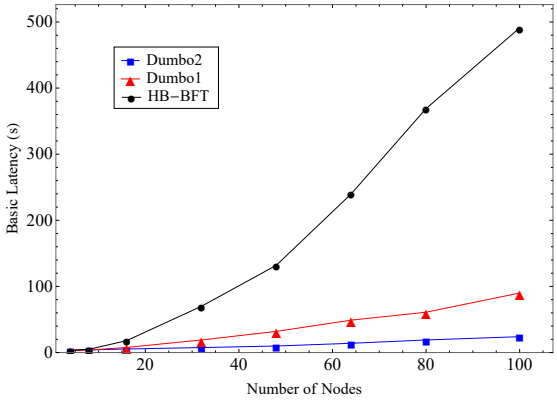

Fig. 6. Basic latency of Dumbo1/2 and HoneyBadgerBFT

Latency. Similar as HB-BFT, latency is defined as the average time interval between the time the first node starts the protocol and when the $(n - f)$-th node gets the result. Latency is related to the process of protocol, as well as influenced by the batch size of the input. We also consider basic latency by letting each nodes propose only one transaction.

In Figure 6, we show the basic latency of the protocols with different system sizes n. It increases when n increases. When n is very small, the basic latency of the three protocols is almost the same. However, the latency of HB-BFT increases much faster than Dumbo1 and Dumbo2 when n gets larger. For example, when $n = 100$, the basic latency of HB-BFT has been up to 500 seconds while that of Dumbo1 is only about 80 seconds and Dumbo2 is only about 20 seconds. The reason lies in that when n gets larger, the latency improvement from reducing number of ABA instances becomes more and more significant.

Throughput. Throughput is defined as the number of transactions committed per second. We use different batch sizes to test the protocols, and shows the relationship between throughput and batch size in Figure 7.

As shown in Figure 7, the throughput of protocols get larger with the increase of batch size if the bandwidth and computing resources are sufficient. Note that the throughput reaches its peak at certain point and then may decrease if the batch size continues to grow, due to the limitation of the bandwidth or computing resources, see the result when $n = 8$. More importantly, when n gets larger, the advantage of Dumbo1/2 become more and more significant. For example, when batch size is 2×10^6 and $n = 100$, Dumbo2 running for one minute and achieve throughput more than 17000 transactions/s, which is 9 times as much as that in HoneyBadgerBFT.

[7] sufficient to contain an ECDSA signature, two public keys, and a typical Bitcoin transaction

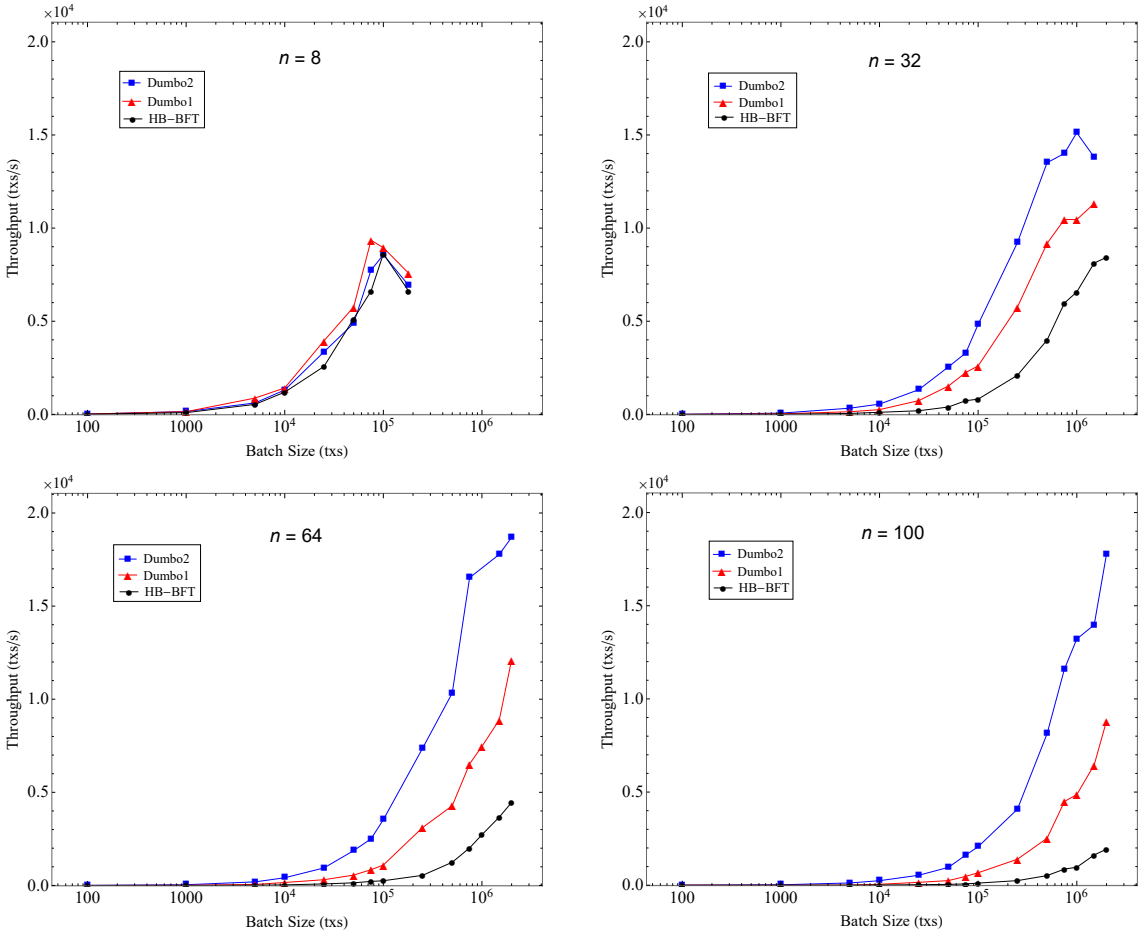

Fig. 7. Throughput of Dumbo1/2 and HoneyBadgerBFT

Trade-off of latency and throughput. Figure 8 shows the relationship between latency and throughput in different settings ($n = 8/32/64$). For all protocols, latency grows with the increase of throughput, but the growth rate is obviously accelerating. Combined with our analysis of Fig. 7, this reflects that bandwidth (and other resources) are gradually consumed with the increase of system load. Another observation is when n is properly large, Dumbo1/2 can provide much higher throughput at the same cost of latency, which means Dumbo1/2 have better scalability so that are more applicable to larger systems.

8 DETAILS OF SUBPROTOCOLS

8.1 From ACS to atomic broadcast

In HoneyBadgerBFT, nodes receive "transactions" as input value and store them in their buffers. The protocol proceeds in *epochs*. At the start of each epochs, nodes choose a certain number of txs randomly from their buffer as their inputs to the atomic broadcast, and at the end of each epochs, a final set of txs for this epoch will be chosen. The final agreements are the union of all received txs decided by n ABA instance. So to improve efficiency, in HB-BFT, nodes randomly choose txs from their buffers instead of sequentially to avoid duplication. A naive attempt of atomic broadcast is sequentially invoking multiple ACS instances. the Censorship Resilience property is not satisfied. That is because the adversary knowing the inputs of all RBC instances can prevent some *tx* from being output controlling the network.

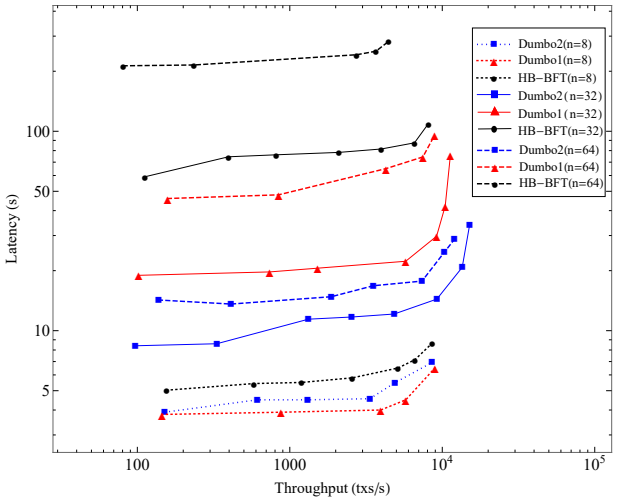

Fig. 8. Throughput vs. Latency

To avoid this, HoneyBadgerBFT adopted threshold encryption: all inputs are first encrypted before provided to the ACS; the agreement will be decrypted only after it has been output by the ACS. By this approach, the adversary has nothing to target. For the detailed description building an atomic broadcast from ACS is as follows:

Given an ACS and the already-setup $(f+1, n)$ threshold encryption, atomic broadcast can be trivially instantiated [32]. In specific, let each node P_i keep a buffer of values called buf_i. Also let TPE.Enc, TPKE.DecShare and TPKE.Dec represent the relevant algorithms of a threshold encryption scheme, and let PK and SK_i to denote the public key and the node P_i's private key of the above threshold encryption scheme (see [32] for the concrete definitions of the threshold encryption scheme). The main part of the algorithm at each node P_i will proceed as follows (with a consecutively increasing parameter r to denote the epoch number):

(1) Random selection and encryption: let *proposed* be a random selection of $\lfloor B/n \rfloor$ values from the first B elements of buf, then encrypt $x := \mathsf{TPKE.Enc}(PK, proposed)$.
(2) Agreement on ciphertexts: pass x as input to ACS; then receive $\{v_j\}_{j\in S}$ from ACS, where S is a subset of all nodes.
(3) Decryption: for each $j \in S$: let $e_j := \mathsf{TPKE.DecShare}(SK_i, v_j)$, then multicast $\mathsf{DEC}(r, j, i, e_j)$, then wait to receive at least $f+1$ value of the form $\mathsf{DEC}(r, j, k, e_{j,k})$, decode $y_j := \mathsf{TPKE.Dec}(PK, \{(k, e_{j,k})\}_{f+1})$.
(4) Output the block: let $\mathsf{block}_r := sort(\bigcup_{j\in S}\{y_i\})$, where *sort* represents a pre-specified rules to order values, then $\mathsf{buf} := \mathsf{buf} - \mathsf{block}_r$.

Theorem 8.1. *(Adapted from [32]) The above protocol is atomic broadcast (i.e., it satisfies totality, agreement and censorship resilience except for negligible probability), conditioned on the underlying ACS and threshold encryption satisfy their definitions respectively.*

8.2 Instantiations of Building Blocks

Here we gave concrete instantiations of the underlying building blocks. Note that the techniques from [22] cherry-picking those components could be applied easily to further optimize our protocol implementation.

Reliable broadcast algorithm (RBC) [32]: The detail of process of RBC shown in Algorithm 5. We remark that in RBC, the erasure code and Merkle-tree are adopted to reduce the communication. In this article, we adopt a $(n-2f, n)$-erasure code scheme in all scenarios, which tolerates the maximal adversary boundary and helps honest nodes recover the message efficient. Note that the RBC message complexity is $O(n^2)$ and the communication complexity is $O(n|m| + \lambda n^2 \log n)$ in expectation.

Algorithm 5 Reliable broadcast (RBC) for party P_i with sender P_s

if $P_i = P_{\text{sender}}$ and received input v **then**
  let $\{s_j\}_{j\in[n]}$ be the blocks of $(n-2f, n)$-erasure coding applied to v;
  let h be a Merkle tree root computed over $\{s_j\}$;
  send $\text{VAL}(h, b_j, s_j) := \{\text{VAL}, h, b_j, s_j\}$ to each party P_j, where b_j is the j^{th} Merkle tree branch;
upon receiving $\text{VAL}(h, b_j, s_j)$ from P_{sender} **do**
  multicast $\text{ECHO}(h, b_j, s_j) := \{\text{ECHO}, h, b_j, s_j\}$;
upon receiving $\text{ECHO}(h, b_j, s_j)$ from P_j **do**
  check that b_j is a valid Merkle branch for root h and leaf s_j, otherwise discard;
upon receiving valid $\text{ECHO}(h, \cdot, \cdot)$ message from $n - f$ distinct parties **do**
  interpolate s'_j from any $n - 2f$ leaves received;
  recompute Merkle root h' and if $h' \neq h$ then abort;
  if $\text{READY}(h) := \{\text{READY}, h\}$ has not yet been sent, multicast $\text{READY}(h)$;
upon receiving $f + 1$ matching $\text{READY}(h)$ messages **do**
  if READY has not yet been sent, multicast $\text{READY}(h)$
upon receiving $2f + 1$ matching $\text{READY}(h)$ messages **do**
  wait for $n - 2f$ ECHO messages, then decode v.

Consistent broadcast (CBC**) [15]:** The CBC is a weaker version of RBC that has no totality. The detail of process of CBC is illustrated in Algorithm 6. Note that the CBC message complexity is $O(n)$ and the communication complexity is $O(n(|m| + \lambda))$ in expectation.

Algorithm 6 Consistent broadcast algorithm (CBC) for party P_i with sender P_s

if $P_i = P_s$ and received input value v **then**
  multicast message (SEND, v);
  upon receiving (ECHO, σ_j) from P_j for the first time **do**
    if $\text{ShareVerify}_{2f+1}(s, v, (j, \sigma_j)) = 1$ **then**
      $DS = DS \cup \{j, \sigma_j\}$;
  upon $|DS| = 2f + 1$ **do**
    $\sigma \leftarrow \text{Combine}_{2f+1}(s, v, DS)$;
    multicast (Finish, v, σ);
upon receiving a SEND message (SEND, v) from P_s for the first time **do**
  $\sigma_i \leftarrow \text{SigShare}_{2f+1}(sk_i, s, v)$;
  send message (ECHO, σ_i) to P_s;
upon receiving a Finish message (Finish, v, σ) from P_s for the first time **do**
  if $\text{Verify}_{2f+1}(s, v, \sigma) = 1$ **then**
    output v.

Asynchronous binary agreement (ABA**) [11, 29, 32, 33]:** The detail of process of ABA is illustrated in Algorithm 7. Note that the running time of one ABA is $O(1)$ in expectation, but that of n concurrent ABA is $O(\log n)$ in expectation. Besides, the ABA message complexity is $O(n^2)$ and the communication complexity is $O(n^2\lambda)$ in expectation. Two important remarks are in place:

Termination: deciding (outputting) v.s halting. As we mentioned briefly in section 3, the termination of ABA only requires *all* honest parties to decide/output a bit. The instantiation in Alg. 7 (without the amendment) indeed satisfies this (see more detailed analysis below). Admittedly, it is possible that some honest party decides a bit in round r, but still waits for messages

Algorithm 7 Asynchronous binary agreement (ABA) for party P_i :

1: **Initialization:** Upon receiving input b_{input}, set $est_0 = b_{input}$, $r = 0, e = 0$, $decided$=false, $values_r = \emptyset$ for $r = \{0, 1, 2, \dots\}$ and proceed in consecutive rounds number r for each consecutive epoch:
2: multicast $\mathsf{Val}_r(est_r) := \{\mathsf{Val}, r, est_r\}$ to all;
3: **upon** receiving $\mathsf{Val}_r(v)$ messages from $f + 1$ nodes **do**
4: **if** $\mathsf{Val}_r(v)$ has not been sent **then**
5: multicast $\mathsf{Val}_r(v)$;
6: **upon** receiving $\mathsf{Val}_r(v)$ messages from $2f + 1$ nodes **do**
7: $values_r = values_r \cup \{v\}$;
8: **wait** until $values_r \neq \emptyset$;
9: multicast $\mathsf{AUX}_r[\omega] := \{\mathsf{AUX}, r, \omega\}$, where $\omega \in values_r$;
10: **wait** until at least $n - f$ $\mathsf{AUX}_r[x]$ messages have been received, such that $val_r \subseteq values_r$ where val_r is the set of values x carried by these $n - f$ messages;
11: multicast $\mathsf{CONF}_r[values_r] := \{\mathsf{CONF}, r, values_r\}$;
12: **wait** until at least $n - f$ $\mathsf{CONF}_r[S]$ messages have been received, such that $S_r \subseteq values_r$ where $S_r = \bigcup S$ of set S carried by these $n - f$ messages;
13: $s \leftarrow coin_r()$; // see e.g., [16, 32]
14: **if** $S_r = \{b\}$ **then**
15: **if** $b = s\%2$ **then**
16: **if** $decided$=false **then**
once decide(b), later modules of the ACS could be invoked
17: decide(b); ▷ decide but not exit
18: $decided$ = true;
19: **else**
if (i). decide(b) in round r; (ii). $coin_{r'}\%2 = b$ in round $r' > r$
20: halt; ▷ exit
21: $est_{r+1} \leftarrow b$;
22: **else**
23: $est_{r+1} \leftarrow s\%2$;
continue looping from line 2
24: $r = r + 1$; goto line 2;

Amendment

replace line 16-20 with line 25-26
25: **if** $\mathsf{FINISH}(b) := \{\mathsf{FINISH}, b\}$ was not yet sent **then**
26: multicast $\mathsf{FINISH}(b)$;
include the following instructions before line 24
27: **upon** receiving $f + 1$ $\mathsf{FINISH}(v)$ from distinct nodes **do**
28: **if** $\mathsf{FINISH}(v)$ was not yet sent **then**
29: multicast $\mathsf{FINISH}(v)$;
30: **upon** receiving $2f + 1$ $\mathsf{FINISH}(v)$ from distinct nodes **do**
31: decide(v) and **halt**. ▷ decide and exit

in round $r + 1$ (while other honest nodes may exit already) in the ABA protocol. This may waste some thread when the peer concurrently runs multiple threads after he decides. However, note that, once an honest party decides a bit, actions (invoke or not) on other modules of the ACS protocol is determined already, thus the safety of the ACS still holds.

As pointed out in the elegant work of [11, 29, 34], there could be a stronger termination condition that all honest parties not only decide, but also exit/halt the ABA instance. Following [11, 29, 34], we also give an amendment of Alg. 7, which satisfies the stronger termination condition. Since we mainly examine the effectiveness of our ABA reduction techniques to see advantages over HB-BFT, we use the same ABA instantiation (Alg. 7) in all those experiments (as revised HB-BFT [31]). Exploring further optimizations within ABA and its practical impact would be interesting future questions.

All honest nodes deciding in Alg.7. Algorithm 7 is the fixed ABA protocol as in [31]. In the original HB-BFT [32] (without line 11 and line 12 in Alg. 7), it is possible for the adversary to trick some nodes to never decide. Concretely, the adversary can reconstruct the coin value before delivering certain messages, so that he can arrange the delivered messages to a party P_i and makes his condition always fail in line 15, thus P_i will repeat forever without deciding anything.

Now an extra round of CONF message was added, it guarantees that among honest nodes whose value S_r in round r contains only one bit after line 12, then they must be the same bit (honest parties could also have $\{0, 1\}$). The bad case that some honest node has 1 and some honest node has 0 after line 12 is taken place only when both $\mathsf{CONF}_r[0]$ and $\mathsf{CONF}_r[1]$ was multicasted for $2f + 1$ times in line 11. This means at least one honest user multicasted both $\mathsf{CONF}_r[0]$ and $\mathsf{CONF}_r[1]$, which leads to a contradiction.

In this way, we can derive the following two key lemmas: (1) at least one honest node would decide a bit; (2) if one honest node decides v in a round r, then all honest nodes would decide the same value v in a later round $r' > r$. To see lemma (1): in every two rounds, there must be one honest user who has one bit as value in line 14 (if all of them have $\{0, 1\}$, then they all assign the coin value as input for next round), thus he will have 1/2 probability to decide. To see lemma (2): since one honest node decides a bit in round r, all other honest nodes either decide the same bit, or have $\{0, 1\}$ as S_r, which will become a same bit (the coin value) in next round; The above analysis is similar to that in [11, 29, 33, 34].

Threshold coin-tossing: We assume the trust third party (dealer) have an unpredictable pseudo-random generator (PRG) $G : R \rightarrow \{1, \cdots, n\}^s$, that is know only to the dealer, which gets a string $r \in R$ and returns a set $\{S_1, S_2, \cdots, S_s\}$ size of s, where $1 \leqslant S_i \leqslant n$.

At the beginning of the protocol, the dealer gives a private function CShare_i to every node P_i, and two public functions: CShareVerify and CToss. Informally, given $f + 1$ validated coin shares, the function CToss returns a unique and pseudorandom set [2]. Formally, the following properties are satisfied except with negligible probability:

- For all $i : 1 \leqslant i \leqslant n$ and for every string r, $\mathsf{CShareVerify}\ (r, i, \sigma) = true$ if and only if $\sigma = \mathsf{CShare}_i(r)$;
- If P_i is honest, then it is infeasible for the adversary to compute $\mathsf{CShare}_i(r)$;
- For every string r, $\mathsf{CToss}(r, \Sigma) = G(r)$ iff $|\Sigma| \geqslant f + 1$ and $\forall \sigma \in \Sigma$, $\exists$ a party P_i s.t. $\mathsf{CShareVerify}\ (r, i, \sigma) = true$.

9 CONCLUSIONS AND FUTURE WORKS

We propose two efficient asynchronous BFT protocols named Dumbo1 and Dumbo2, each of which has an asymptotically and practically better construction of ACS. The experiments on 100 AWS EC2 instances dispersed in 4 continents of the world show that our schemes outperform HoneyBadgerBFT, known as the first practical asynchronous BFT, by several times. For example Dumbo2 can even achieve throughput of tens of thousands in a system with hundreds of nodes, meanwhile the delay is around 20 seconds (instead of more than 8 minutes running HB-BFT in the same environment).

Our core technical contributions include two methods reducing the number of randomized ABA instances. We remark that Dumbo2 deviates the design methodology of ACS as in [32] and turn back to MVBA which was not considered optimal in building ACS. Our construction of Dumbo2 draws an analogy to the widely used conventional wisdom of hybrid encryption.

We remark that we haven't done any optimization, all improvements are demonstrated in a basic instantiation. There are multiple ways to further improve our protocols, e.g., applying the techniques from BEAT to choose the best instantitions. More importantly, we may further reduce the (message or communication) complexity and push the asynchronous atomic broadcast towards optimal. We leave them as interesting open problems.

ACKNOWLEDGMENTS

We thank anonymous reviewers for pointing out the problem in the ABA of [32] and in our previous description, and various other valuable comments. We also thank Yuan Lu for fruitful discussions. The authors are all supported in part by JD Digits via the JDD-NJIT-ISCAS Joint Blockchain Lab. Qiang Tang is also supported in part by a Google Faculty Award; and Zhenfeng Zhang is also supported in part by the National Key R&D Program of China (No. 2017YFB0802500).

REFERENCES

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. 2019. Synchronous Byzantine Agreement with Expected $O(1)$ Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience. In *International Conference on Financial Cryptography and Data Security*. Springer, 320–334.

[2] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 337–346.

[3] Joseph A Akinyele, Christina Garman, Ian Miers, Matthew W Pagano, Michael Rushanan, Matthew Green, and Aviel D Rubin. 2013. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering* 3, 2 (2013), 111–128.

[4] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. 2010. Prime: Byzantine replication under attack. *IEEE Transactions on Dependable and Secure Computing* 8, 4 (2010), 564–577.

[5] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. 2013. Rbft: Redundant byzantine fault tolerance. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 297–306.

[6] Joonsang Baek and Yuliang Zheng. 2003. Simple and efficient threshold cryptosystem from the gap diffie-hellman group. In *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, Vol. 3. IEEE, 1491–1495.

[7] Michael Ben-Or. 1983. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*. ACM, 27–30.

[8] Michael Ben-Or and Ran El-Yaniv. 2003. Resilient-optimal interactive consistency in constant time. *Distributed Computing* 16, 4 (2003), 249–262.

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

[10] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. 2014. State machine replication for the masses with BFT-SMaRt. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 355–362.

[11] Erica Blum, Jonathan Katz, and Julian Loss. 2019. Synchronous Consensus with Optimal Asynchronous Fallback Guarantees. In *Theory of Cryptography Conference*. Springer, 131–150.

[12] Alexandra Boldyreva. 2003. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In *International Workshop on Public Key Cryptography*. Springer, 31–46.

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

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

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

[16] 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.

[17] Christian Cachin and Jonathan A Poritz. 2002. Secure intrusion-tolerant replication on the Internet. In *Proceedings International Conference on Dependable Systems and Networks*. IEEE, 167–176.

[18] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[19] 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.

[20] Flaviu Cristian, Houtan Aghili, Raymond Strong, and Danny Dolev. 1986. *Atomic broadcast: From simple message diffusion to Byzantine agreement*. International Business Machines Incorporated, Thomas J. Watson Research Center.

[21] Danny Dolev, Michael J Fischer, Rob Fowler, Nancy A Lynch, and H Raymond Strong. 1982. An efficient algorithm for Byzantine agreement without authentication. *Information and Control* 52, 3 (1982), 257–274.

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

[23] 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.

[24] Juan A Garay and Aggelos Kiayias. 2018. SoK: A Consensus Taxonomy in the Blockchain Era. *IACR Cryptology ePrint Archive* 2018 (2018), 754.

[25] Klaus Kursawe and Victor Shoup. 2005. Optimistic asynchronous atomic broadcast. In *International Colloquium on Automata, Languages, and Programming*. Springer, 204–215.

[26] Leslie Lamport. 1998. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)* 16, 2 (1998), 133–169.

[27] 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.

[28] Ben Lynn. 2007. *On the implementation of pairing-based cryptosystems*. Ph.D. Dissertation. Stanford University Stanford, California.

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

[30] Dahlia Malkhi, Kartik Nayak, and Ling Ren. 2019. Flexible Byzantine Fault Tolerance. *arXiv preprint arXiv:1904.10067* (2019).

[31] Andrew Miller. 2018. Bug in ABA protocol's use of Common Coin 59. https://github.com/amiller/HoneyBadgerBFT/issues/59. (2018). Online Forum.

[32] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 31–42.

[33] Achour Mostefaoui, Hamouma Moumen, and Michel Raynal. 2014. Signature-free asynchronous byzantine consensus with $t < n/3$ and $O(n^2)$ messages. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*. ACM, 2–9.

[34] 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), 31.

[35] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX} {ATC} 14)*. 305–319.

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

[37] HariGovind V Ramasamy and Christian Cachin. 2005. Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast. In *International Conference On Principles Of Distributed Systems*. Springer, 88–102.

[38] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. 2009. Spin one's wheels? Byzantine fault tolerance with a spinning primary. In *2009 28th IEEE International Symposium on Reliable Distributed Systems*. IEEE, 135–144.

[39] Zooko Wilcox-O'Hearn. 2008. Zfec 1.4. 0. *Open source code distribution: http://pypi. python. org/pypi/zfec* (2008).