Speeding Dumbo: Pushing Asynchronous BFT Closer to Practice

Bingyong Guo*§, Yuan Lu*§, Zhenliang Lu†§, Qiang Tang†§, Jing Xu*§ and Zhenfeng Zhang*§
*Institute of Software, Chinese Academy of Sciences
†School of Computer Science, The University of Sydney
{bingyong2017,luyuan}@iscas.ac.cn, zhlu9620@uni.sydney.edu.au, qiang.tang@sydney.edu.au,{xujing,zhenfeng}@iscas.ac.cn

***Abstract*—Asynchronous BFT consensus can implement robust mission-critical decentralized services in the unstable or even adversarial wide-area network without relying on any form of timing assumption. Starting from the work of HoneyBadgerBFT (CCS 2016), several studies tried to push asynchronous BFT towards practice. In particular, in a recent work of Dumbo (CCS 2020), they redesigned the protocol backbone and used one multi-valued validated Byzantine agreement (MVBA) to replace n concurrent asynchronous binary agreement (ABA) protocols and dramatically improved the performance.**

Despite those efforts, asynchronous BFT protocols remain to be slow, and in particular, the latency is still quite large. There are two reasons contributing to the inferior performance: (1) the reliable broadcast (RBC) protocols still incur substantial costs; (2) the MVBA protocols are quite complicated and heavy, and all existing constructions need dozens of rounds and take the majority of the overall latency.

We first present a new construction of asynchronous BFT that replaces RBC instance with a cheaper broadcast component. It not only reduces the $\mathcal{O}(n^3)$ message complexity incurred by n RBCs to $\mathcal{O}(n^2)$, but also saves up to 67% communications (in the presence of a fair network scheduler). Moreover, our technical core is a new MVBA protocol, Speeding MVBA, which is concretely more efficient than all existing MVBAs. It requires only 6 rounds in the best case and expected 12 rounds in the worst case (by contrast, several dozens of rounds in the MVBA from Cachin et al. [12] and the recent Dumbo-MVBA [32], and around 20 rounds in the MVBA from Abraham et al. [4]). Our new technique of the construction might be of independent interests.

We implemented Speeding Dumbo and did extensive tests among up to 150 EC2 t2.medium instances evenly allocated in 15 AWS regions across the globe. The experimental results show that Speeding Dumbo reduces the latency to about a half of Dumbo's, and also doubles the throughput of Dumbo, through all system scales from 4 nodes to 150 nodes. We also did tests to benchmark individual components such as the broadcasts and the MVBA protocols, which may be of interests for future improvements.

I. Introduction

Following the explosive popularity of blockchain and decentralized applications [11, 35], an unprecedented demand is recently raised to deploy public ledger services for mission-critical applications over the global Internet among mutually distrustful participants [15]. As such, Byzantine fault-tolerant (BFT) protocols, e.g., BFT atomic broadcast, are gathering renewed attentions, since they are the core techniques for implementing the decentralized infrastructures.

In need of asynchronous BFT. Conventional BFT protocols were mainly considered for the benign in-house scenarios where the nodes are geographically close and well connected by stable communication links. In particular, many classic BFT protocols [17] rely on various timing assumptions (e.g., synchrony assumption that all messages will be delivered within a known time bound Δ or the weaker variant called partial synchrony). Unfortunately, those network assumption may not always hold in the wide-area network environment, which has a dynamic or even adversarial nature, because of fluctuating bandwidth, unreliable links, substantial delays, and network attacks. It was actually shown in [33] that PBFT (and many leader based protocols) cannot make any progress in an asynchronous network where the adversary can schedule messages as it likes. This challenges the applicability of those conventional BFT protocols in the open Internet.

Moreover, in practice, to ensure the synchrony assumptions to hold with a larger probability, one has to choose a conservative time parameter Δ which is usually larger than the actual network delay. This tactic brings performance degradation to the (partially) synchronous protocols. For example, Bitcoin assumes exaggerated parameter to survive in the adversarial environment [36], resulting in extremely slow block generation (on average 10 minutes); in many partially synchronous protocols such as PBFT and HotStuff [45], a malicious leader could always withhold their outgoing messages up to Δ, thus violating the desirable *responsiveness* property that the performance should only depend on the actual network delay. Some recent results [3, 37, 43] considered to make synchronous protocols to attain fast confirmation in optimistic cases (i.e., optimistic responsiveness), but remains to suffer from slower confirmation in the general cases with failures or corruptions.

In contrast, the fully asynchronous BFT protocols [4, 12, 21, 26, 32, 33] do not suffer from the aforementioned liveness issue and responsiveness problem, as they do not rely on any timing assumptions. Also, when actually building the fault-tolerate systems, asynchronous protocols do not require engineers to manually tune the time-out mechanism, which is often frustrating and error-prone.

In search of practical asynchronous BFT protocols. Unfortunately, essentially none of the asynchronous protocols have been widely deployed due to efficiency concerns. The seminal FLP "impossibility" [22] states that, no *deterministic* BFT

§Authors are listed alphabetically, Yuan & Zhenliang led the effort.

protocols exist in asynchronous networks. Since the 1980s, many attempts [1, 6, 7, 10, 13, 16, 18, 34, 38–40] aimed to design *randomized* asynchronous protocols to circumvent the "impossibility". However, most of those studies focused on theoretical feasibility, and unsurprisingly, have prohibitively high costs. Renewed attentions have emerged recently due to the need of deploying consensus to support decentralized applications in the open Internet. People started to wonder whether asynchronous consensus can ever be practical. We take a brief tour to dissect the cutting-edge practical asynchronous BFT protocols.

Increasing throughput via batching. One of the most notable recent efforts is HoneyBadgerBFT [33], in which the authors optimized certain classical protocols, and the implementation demonstrated promising performance (particularly throughput). It is noted that asynchronous atomic broadcast protocols built from a carefully optimized asynchronous common subset (ACS) protocol could reduce asymptotic communication complexity (especially when increasing the batch size), making it more efficient than the best previous protocol [12] that directly applied a trivial reduction from ACS to asynchronous multi-valued validated Byzantine agreement (MVBA) protocol.[1] Here, an ACS protocol is a variant of Byzantine agreement outputting a subset containing $n - f$ input values (where n is the # of nodes and f is the # of faulty nodes allowed), while an MVBA protocol is another variant of Byzantine agreement outputting one input that could come from a malicious node but satisfy some certain public predicate.

The ACS protocol in HoneyBadgerBFT was constructed and adapted from the classic protocol of Ben-Or et al. [8]. As shown in Fig. 1, it begins with n parallel reliable broadcasts (RBCs), each of which lets a node broadcast its input value to the whole network. Then, all nodes participate in n (asynchronous) binary agreement (ABA) protocol instances to determine whether they output each individual's input value, namely, pick up at least $n - f$ completed broadcasts as the final ACS output.

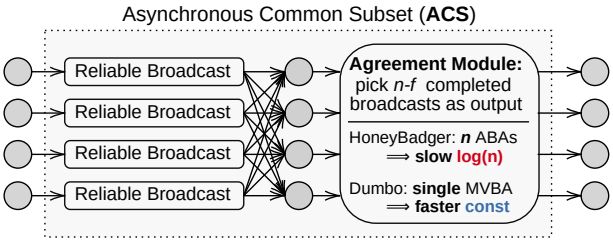

Fig. 1: The very high-level execution flow of the core ACS protocol in HoneyBadgerBFT and Dumbo.

Striving for a constant running time. Though showing promising performance at a small scale, it was noted in a very recent work [26] that HoneyBadgerBFT does not smoothly scale to support even a moderate size of nodes. The major bottleneck observed is that when a large number of randomized ABA protocols are executed concurrently, it is almost certain there is a very slow instance, the running time of which will depend on n. Indeed, as shown via experiments [26], the second phase of running n ABA protocols takes a dominating portion of the whole execution time in HoneyBadgerBFT, cf. Fig. 2.

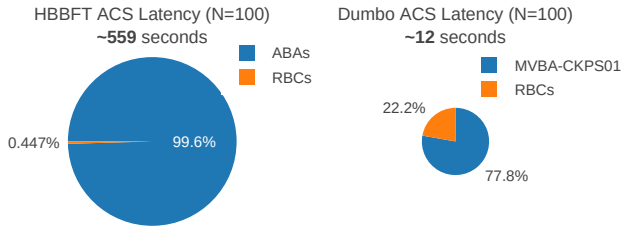

Fig. 2: Latency breakdown of HoneyBadgerBFT (HBBFT) and Dumbo (with nearly zero payload) among 100 WAN nodes distributed in 15 AWS regions across five continents. Different from the experiments in HBBFT[33], all tests in this paper include the cost for verifying threshold signatures.

The authors in [26] presented two new ACS protocols called Dumbo1 and Dumbo2 that can substantially reduce the # of needed ABA instances to k and $O(1)$, respectively, where k is a small security parameter. In particular, in their Dumbo2 protocol, instead of using n ABAs, they used a single MVBA, which only invokes 2 or 3 ABA instances on average [12] to decide the final ACS output.

More importantly, Dumbo2 (called Dumbo later for brevity) reclaimed the glory of MVBA as the core component for practical ACS, and it overcame the expensive communication of using MVBA directly to agree on transactions. To minimize MVBA's communication blow-up, RBCs are augmented with a succinct proof to broadcast transaction batches, and thus a proper predicate can be defined to let MVBA be invoked only on the small RBC proofs instead of large transaction batches (in analogy to the conventional wisdom of hybrid encryption). Basically, the proofs can facilitate a predicate that once a subset of inputs are output by one honest node, every other honest node will eventually receive them from RBCs to output.

Indeed, the RBC-then-MVBA paradigm proposed in Dumbo [26] brings about both asymptotic complexity reduction of running time and an order-of-magnitude improvement on practical performance, cf. Fig. 2 and [26] for details.

Remaining efficiency obstacles. Despite those recent progresses, existing asynchronous BFT protocols are still unsatisfying, especially regarding their high *latency*. As shown in previous experiments [26], when running among 100 global AWS t2.medium nodes (without failures), HoneyBadgerBFT incurs a basic latency [2] as large as several minutes on average. For this reason, it was suggested to deploy asynchronous atomic broadcast only in the settings that favor throughput over latency [33]. Though Dumbo is much faster, it still incurs a high basic latency about a dozen seconds. In contrast, the state-of-the-art (partially) synchronous protocols (e.g., HotStuff) can attain a latency even smaller than 1 second in the same benign setting. This naturally posts the following question:

Can we push the asynchronous consensus further, to have

[1] Cachin et al. [12] essentially constructed an ACS in a simple manner as follows: all nodes sign and multicast their input values, so they can solicit $n - f$ value-signature pairs from different nodes to invoke an MVBA instance that is with an external validity condition and can return $n - f$ values signed by distinct nodes as the output set of ACS.

[2] The basic latency of ACS is defined as the execution time when the payload size is about zero. The metric depicts the baseline of how fast the protocol can be, and the situation may be even worse when payload size gets larger.

even larger throughput and much smaller latency, thus it can be deployed in broader real-world settings?

We first examine the remaining efficiency obstacles:

(1) *Costs of RBCs cannot be ignored*. The improvements of Dumbo over HoneyBadgerBFT focus on the "agreement" phase: since in HoneyBadgerBFT, the costs of RBCs only took an ignorable portion in the overall basic latency. But in Dumbo, now the agreement part got simplified, the cost of concurrently executing n RBCs becomes clearly visible, cf. Fig. 2. The message complexity of each RBC instance is quadratic, and therefore n RBCs would incur a cubic message complexity, which is clearly not optimal, and may lag the protocol.

(2) *MVBA is cumbersome*. Though Dumbo demonstrated that when an MVBA is invoked with small-sized inputs, it is much faster than n parallel ABAs, it remains to be a very heavy and slow building block (for its power to trivially imply a full-fledged atomic broadcast [12]). Indeed, from the experimental data shown in Fig. 2, it is clear to see that MVBA still takes the majority of the latency of Dumbo ACS. We remark that this challenge cannot be easily resolved by simply replacing CKPS01-MVBA [3] [12] used in Dumbo with recent MVBA constructions, e.g., VABA [4] or Dumbo-MVBA [32]. Those works focus on theoretical improvements on the asymptotic complexities, and both protocols are still complicated that involve multiple dozens of rounds. Since all those MVBA protocols are already with expected constant running time, it becomes a clear challenge to open up the big O in the asymptotic complexity to design a *concretely* more efficient MVBA protocol with fewer rounds to further extend the applicability of asynchronous BFT.

A. Our contributions

We answer the above efficiency question affirmatively, and present a much faster asynchronous atomic broadcast called Speeding Dumbo (or sDumbo for short). Following the remaining efficiency obstacles identified to overcome, we proceed as follows:

Reducing message complexity asymptotically. As we briefly elaborated above, the n concurrent invocations of RBC protocols already incur a substantial portion of the cost in ACS. Since RBC ensures a strong agreement that if any node outputs a value, all other honest nodes will output the same value, each RBC instance now incurs an $\mathcal{O}(n^2)$ message complexity. It follows immediately that both HoneyBadgerBFT and Dumbo at least have an $\mathcal{O}(n^3)$ message complexity.

We first reduce this cubic message complexity of ACS by a factor of n by using a weaker (thus cheaper) broadcast primitive, i.e., the provable broadcast (PB) recently defined by Abraham et al. [4] instead of RBC. A PB protocol is just a normal multi-cast with a succinct proof (realizable simply via a threshold signature), showing which could guarantee that sufficient honest nodes have received the value.

Previously, we can simply augment each RBC to attain a proof attesting that at least one honest node has received some value, so that we can be sure of that every other honest node will receive the same value. Now PB does not have this strong agreement property. Therefore, we add a recovery phase at the end of the sDumbo protocol, thus enabling a node to fetch whatever PB output that shall be outputted but was not yet received. The potential communication blow-up during the recovery phase is carefully handled using the technique from verifiable information dispersal [14]. As a result, for sufficiently large inputs, our new ACS framework also attains a concrete communication saving. If there is a fair network scheduler (benign cases) that never reorders messages, it can save up to 67% concrete communications relative to HoneyBadgerBFT and Dumbo.

Speeding MVBA: a compact MVBA protocol. At the core of Speeding Dumbo, it is a novel MVBA protocol (called Speeding MVBA or sMVBA) that is concretely more efficient (as few rounds as possible) than all existing MVBAs. [4]

The challenges of MVBA and current designs. An MVBA is a multi-valued Byzantine agreement protocol, whose constructions are usually quite complicated. They either use ABA as a blackbox to be repeated, e.g., the CKPS01-MVBA [12], or expanding each component in ABA construction, e.g., the AMS19-MVBA (sometimes also called VABA) [4]. The latter currently yields the minimum concrete number of rounds. Let us briefly overview how it works. See also Fig. 3.

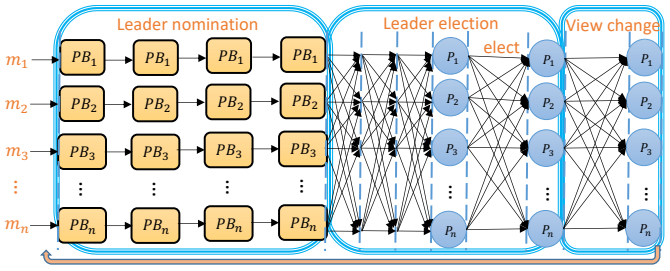

Fig. 3: The structure of AMS19 MVBA [4].

To maintain $\mathcal{O}(n^2)$ message/communication complexity (and thus linear complexity per node), it adopted a three-step "key-lock-commit" broadcast mechanism, which can be realized via 4 consecutive executions of provable broadcast (PB) protocols. To ensure expected constant running time, one extra "*completeness*" proof is generated from the last PB, so once the honest nodes receive enough completeness proofs to enter the "leader-election", it is asserted that sufficient number of "key-lock-commit" broadcasts have been completed.

Since each PB already has at least 2 rounds, plus several preparing rounds before leader-election and one round of view change, the AMS19-MVBA protocol already incurs more than a dozen rounds even in the optimistic case where there is no adversary and no repetition of views. In the adversarial case, the protocol would have about 20 expected rounds (and CKPS01-MVBA has even more). See Table I.

Intuitively, the *commit* proof alone (more precisely, any of the three has the same potential) can ensure at least $f+1$ honest nodes to terminate with the same output in the $2/3$ good cases

[3]Remark that [12] gave two MVBA constructions. We let CKPS01 refer to the MVBA with expected constant number of rounds throughout the paper.

[4]There were no experimental results in the original paper of those MVBA protocols. As a byproduct, we provided the first experimental comparisons of all those MVBA protocols, which may of independent interests for further practical optimizations.

TABLE I: Expected asynchronous rounds of MVBA protocols (approximated)

	Protocol			
	CKPS01[12]	AMS19[4]	LLTW20[32]	sMVBA
Adv case †	54	19.5	47	12
Best case ‡	12	13	18	6

†The "Adv Case" means the worst expected asynchronous rounds in the standard asynchronous model, i.e., with up to 1/3 corrupted nodes and adversarial network to arbitrarily schedule messages. ‡The "Best Case" means that all nodes are honest, there is a fair network scheduler that never reorders messages, and all nodes start to run the protocol simultaneously.

(where the leader election luckily chooses a node completes the 4-staged PB broadcast). While, the other two proofs handle the "dirty" work in the other $1/3$ unlucky situations. For *safety*, the *lock* proof is needed to ensure that if any honest node commits a value in a view, all honest nodes would "lock" themselves after view-change, so all 4-staged PBs in later views must carry the same value to convince them to participate; For *liveness*, once some honest node "locks" itself after some view, all honest nodes must have valid *keys* as part of their 4-staged PBs' input in all later views, which can "unlock" this locked node to participate, thus ensuring the protocol to always proceed despite some probably "locked" nodes.

A new compact protocol with fewer broadcasts. The elegant AMS19-MVBA protocol uses a quite natural (and seemingly tight) structure, so further compressing it requires us to break the box. We first reduce the PB instances to 2 and generates only two proofs called *lock* and *finish*. This immediately causes safety and liveness issues. It is possible that in one execution, one honest node A considers a selected node P_ℓ's broadcast has been finished and thus output the corresponding input value v_ℓ; while at the same time, the proofs have not reached another honest node B, and thus it will invoke view-change and start another view. Now we still have to ensure that node B would eventually output v_ℓ, either carrying the related proofs to enter the next view, or obtaining it from other honest nodes.

To get around the challenge, we first add one round of *vote* step after leader election to facilitate the collection of sufficient *lock* proofs: even if one does not output in the current iteration, everyone has to bring the same value to next iteration. But now we lack the *key* proof as in the AMS19-MVBA, which could be used to "unlock" a locked message in the bad cases, we also add another *pre-vote* round before entering *vote* to determine whether to firmly "lock" the value. The added *pre-vote* step ensures that when a (potentially malicious) node wants to stop the selected value from being locked, it has to demonstrate a proof that a sufficient number of honest nodes acknowledgment that they have not received the *lock* proof corresponding to the selected value (and thus no honest node has output yet). This is realized by carefully embedding threshold signatures so that each node could indeed aggregate a proof on a same message without requiring one extra round or blowing out the communication. Moreover, this simpler structure allows us to further batch messages during the preparation before the "leader-election" phase.

Another simple but effective observation is that we could have a "short-cut" for the honest nodes to output right after the leader election, such that if they see the *finish* proof corresponding to the elected leader, they can immediately output the elected leader's input value. This further reduces the number of rounds in good cases.[5] See Fig. 5 in Sec. V.

The resulting sMVBA protocol requires only 6 rounds in the best case (compared to a dozen or more in the existing MVBAs), and 12 rounds in the fully adversarial case (compared to multiple dozens of rounds in the existing MVBA protocols), see Table I. The efficiency improvements not only reduce the communication rounds, but also save the computation cost on generating and verifying the proofs.

Implementation and extensive real-world evaluations. We implemented our protocols, and conducted extensive experiments among up to 150 Amazon EC2 t2.medium instances throughout 15 AWS regions across 5 continents.

(1) We evaluate the performance improvements brought by replacing RBCs with cheaper broadcasts alone. Due to the asymptotic improvement of message complexity, the basic latency (at nearly zero-payload) is significantly reduced by up to about 10% when the number of nodes is between 82 and 150. Also, the peak throughput is nearly doubled at all system scales from 4 nodes to 150 nodes, reflecting the saving of concrete communications.

(2) we compare sMVBA with the existing MVBAs regarding their latency in the WAN setting. Throughout all system scales, sMVBA can reduce at least 50% of the running time.

(3) we extensively test sDumbo, i.e., our new ACS framework instantiated by sMVBA, and compare it with the state-of-the-art asynchronous protocol Dumbo and the cutting-edge partially synchronous protocol HotStuff. At all system scales up to 150 nodes, our sDumbo protocol attains a peak throughput 2X-2.5X as much as Dumbo's. The latency is brought down significantly: for moderately large scale (e.g., 100 nodes), the latency is reduced to a few seconds from a dozens; for small scale (e.g., 16 nodes or less), the latency of sDumbo becomes at the same magnitude of HotStuff's, while Dumbo has a huge gap to HotStuff. In general, sDumbo demonstrates multi-fold improvements, and attains broader applicability with a better latency-throughput trade-off.

B. Other related work

To make asynchronous BFT protocols faster, there exist a few works [30, 41] including some very recent ones [25, 31] that consider adding an optimistic "fastlane" to the slow asynchronous atomic broadcast. The fastlane could simply be a fast leader-based deterministic protocol. This line of work is certainly interesting, however in the adversarial settings, the "fastlane" never succeeds, and the overall performance would be even worse than running the asymptotic atomic broadcast itself (since there is an extra asynchronous "fallback" mechanism). This paper, on the contrary, aims to directly improve asynchronous BFT atomic broadcast, and can be used together with these optimistic techniques to provide a better underlying pessimistic path. In addition, BEAT [21] cherry-picked concrete implementations for each component in HoneyBadgerBFT to demonstrate better performance in different settings. Very recently, Aleph [24] and DAG-rider [28] proposed to use direct acyclic graph for consensus besides

[5]The "short-cut" could further save rounds in practice, because it allows the nodes to output immediately after leader election without the need of completing the remaining *pre-vote* and *vote* rounds.

sequential ACS, which provides a theoretical alternative for implementing asynchronous atomic broadcast besides sequentially executing ACS. There are also interesting works on asynchronous distributed key generation [2, 19, 29], which could be helpful to remove the private setup phase in all recent asynchronous BFT protocols.

II. Models and Problem Statement

We first describe the models and formal definitions.

A. System and threat models

Established identities and trusted setup. There are n designated nodes, each of which has a unique identity $P_i \in \{P_i\}_{i\in[n]}$, where $[n]$ denotes the integers $\{1, 2, \ldots, n\}$. We assume that all involved threshold cryptosystems such as threshold signatures are properly set up, such that all participating nodes can get and only get their own secret keys in addition to all relevant public keys. Remark that the setup of threshold cryptosystems can be done through distrusted key generation (DKG) protocols [2, 19, 27, 29] or a trusted dealer.

Adversary model. We assume that there are up to f Byzantine fault nodes ($3f + 1 \leq n$), and consider these faulty nodes are fully controlled by a probabilistic polynomial-time bounded adversary [12, 33], i.e., the adversary can get all faulty nodes' internal states and also can let these nodes arbitrarily misbehave during the protocol execution, and the adversary can perform some probabilistic computing steps bounded by polynomials in the number of message bits generated by honest nodes. In addition, we might consider static corruptions, namely, the adversary chooses up to f nodes to corrupt before the execution.[6]

Asynchronous network. We consider asynchronous message-passing network made of fully meshed authenticated point-to-point (p2p) channels. In this network, the message delivery over the channels are fully determined by the adversary, namely, the adversary can arbitrarily delay and reorder messages and fully determines when the receiver can get the message. Nevertheless, the messages sent between honest nodes will eventually be delivered to the destinations without tampering, i.e., adversary cannot drop or modify this message among honest nodes.

B. Design goals

Asynchronous atomic broadcast. Our goal is to design an efficient atomic broadcast protocol among n nodes against f static corruptions in an asynchronous network. In an atomic broadcast protocol, each node has an implicit input transaction buffer (a.k.a. backlog) and outputs a sequence of transactions. Formally, the atomic broadcast satisfies the following properties with all but negligible probability:

- *Agreement.* If one honest node outputs a value v, then every honest node outputs v;
- *Total-order.* If any 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 $i \leq \min(j, j')$;
- *Liveness.* If a value v is input to $n - f$ honest nodes, then it is output by some honest node reasonably fast, e.g., in at most polynomial # of asynchronous rounds.

Asynchronous common subset (ACS). The main building block of atomic broadcast is asynchronous common subset (ACS), see Appendix A and [33] for the details of the conversion from ACS and threshold encryption to atomic broadcast. In an ACS protocol, each node has a (different) input, and their goal is to let each node output a common subset covering $n-f$ nodes' inputs [33]. More formally, ACS has the following properties with all but negligible probability:

- *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 $|V| \geq n - f$ and V contains the inputs of at least $n - 2f$ honest nodes;
- *Termination.* If $n-f$ honest nodes have an input, then all honest nodes can produce an output.

Complexity measures. In this paper, we mainly consider the following three metrics during the protocol:

- *Message complexity* [12]: the expected total number of messages that generated by honest nodes;
- *Communication complexity* [12]: the expected total number of bits exchanged among honest nodes;
- *Round complexity (running time)* [16]: the expected (asynchronous) rounds of communication before the protocol terminates.

Besides, we always consider $n = 3f + 1$ throughout the paper, namely, our BFT protocol is optimally resilient against $n/3$ Byzantine corruptions [23], and its scale size n is only parameterized by the number of nodes that may be corrupted.

III. Notations and Preliminaries

Notations. Through the paper, we let $|m|$ be the bit-length of protocols' input, λ be the cryptographic security parameter that captures the size of (threshold) signature and the length of hash value. $\mathcal{H}$ denotes a collision-resistant hash function. We also consider established threshold signature consisting of a tuple of algorithms $(\mathsf{SignShare}_t, \mathsf{VerifyShare}_t, \mathsf{Combine}_t, \mathsf{VerifyThld}_t)$, where the subscript t in notations represents threshold, cf. Appendix C for the syntax and securities of the primitive. A protocol message has a syntax of $(\mathsf{MsgType}, \mathsf{ID}, \cdots)$, where $\mathsf{MsgType}$ defines the message type and ID is the session identifier representing a specific protocol instance. Moreover, $\Pi[\mathsf{ID}]$ represents an instance of some protocol Π with a session identifier ID, and $y \leftarrow \Pi[\mathsf{ID}](x)$ means to invoke $\Pi[\mathsf{ID}]$ on input x and wait for its output y. Sometimes, $\Pi[\mathsf{ID}](x)$ itself also denotes the protocol output for brevity. Also, $\langle x, y\rangle$ denotes a string concatenating two strings x and y.

[6] Our implementations choose statically secure instantiations for efficiencies. We remark that static corruption is exactly the same adversarial model considered in the recent *practical* asynchronous BFT protocols including HoneyBadgerBFT [33] and Dumbo [26]. Moreover, our protocols *can* be adaptively secure if using adaptively secure threshold cryptosystems as components. Our techniques of simplifying protocol structure are critical under both static and adaptive corruptions, and once we have cheaper adaptive secure threshold cryptosystems, we can easily plug them in the concrete instantiations.

Multi-valued validated Byzantine agreement (MVBA) [4, 12] is a variant of Byzantine agreement with external validity, such that the participating nodes can agree on a value satisfying a global and polynomial-time computable predicate Q known by all of them (which can be concretely specified due to the application scenarios). Syntax-wise, each node in the protocol would take a (probably different) value validated by the predicate Q as input and can decide a common value satisfying Q as the output.

The protocol (with an explicit predicate Q) shall satisfy the following properties except with negligible probability:

- *Termination.* If all honest nodes are activated on an identifier ID with taking as input some values satisfying Q, then each honest node would output a value for ID;
- *External-Validity.* If an honest node outputs a value v for some ID, then v is valid for Q, i.e., $Q(v) = 1$;
- *Agreement.* All honest nodes would decide the same value as output for the same ID.

Leader election (Election) is a protocol among n nodes that can output an unpredictable index representing a node in the system. It satisfies the following properties with an overwhelming probability:

- *Termination:* If all honest nodes activate the protocol on an identifier ID, each honest node would output some index $\ell \in [n]$ for ID;
- *Agreement:* All honest nodes would output the same index ℓ for the same ID;
- *Unpredictability:* Before $t - f$ nodes invoke the protocol on an identifier ID (where f represents the number of corrupted nodes), no node can successfully predicate the output for ID except with $1/n$ probability (i.e., no better than guessing from $[n]$).

The (n, t) Election protocol can be instantiated by (n, t) non-interactive (unique) threshold signature under setup assumptions in the random oracle model [13]. The details of the classic construction is illustrated in Alg. 7 in Appendix G.

Provable broadcast (PB, adapted from [4]) is a broadcast protocol among n nodes with a designated node called sender and a global predicate function ValueValidation for validating the sender's input. The sender takes a tuple $m := (v, \pi)$ as input, where v is a value and π is a string to validate v according to ValueValidation. Then, each node would output a tuple $value := (v, \pi)$. The sender additionally outputs a tuple $lock := (h, \sigma)$, where h is the hash of some value v and σ is a threshold signature that aggregates $2f + 1$ signature shares for h. In addition, each node can invoke an $abandon$ interface during the execution to quit from the protocol. [7]

Besides the above syntax, a PB protocol with a session identifier ID satisfies the next properties with all but negligible probability:

- *Validity*. If any honest $\mathcal{P}_i$ outputs a tuple $value := (v, \pi)$, then $\mathsf{ValueValidation}(\mathsf{ID}, (v, \pi)) = 1$.
- *Termination*. If the sender $\mathcal{P}_s$ is honest and inputs $m := (v, \pi)$ satisfying $\mathsf{ValueValidation}(\mathsf{ID}, m) = 1$ and all honest nodes activate PB[ID] without invoking $abandon(\mathsf{ID})$, then all honest nodes would output $value := m$; additionally, the sender outputs a tuple $lock := (h, \sigma)$, where $h = \mathcal{H}(v)$ and $\mathsf{VerifyThld}_{(2f+1)}(\mathsf{ID}, h, \sigma) = 1$.
- *Provability*. If the sender can produce $lock := (h, \sigma)$ and $lock' := (h', \sigma')$ s.t. $\mathsf{VerifyThld}_{(2f+1)}(\mathsf{ID}, h, \sigma) = 1$ and $\mathsf{VerifyThld}_{(2f+1)}(\mathsf{ID}, h', \sigma') = 1$, then (1) $h = h'$ and (2) at least $f + 1$ honest nodes output $value := (v, \pi)$ satisfying that $\mathcal{H}(v) = h$ and $\mathsf{ValueValidation}(\mathsf{ID}, value) = 1$.
- *Abandon-ability*. An honest node does not deliver any message associated to ID after invoking $abandon(\mathsf{ID})$.

A construction of PB is deferred to Algorithm 6 in Appendix F, which costs only $\mathcal{O}(1)$ rounds, $\mathcal{O}(n)$ messages, and $\mathcal{O}(|m|n + \lambda n)$ bits.

IV. Warm-up: New ACS to Reduce Messages and Communications

In this section, we consider how to reduce the cost of the broadcast phase, and will present a new ACS framework using cheaper broadcast primitives and MVBA.

Overview of the new ACS framework. It is worth noting that RBC suffers from $\mathcal{O}(n^2)$ messages (actually for deterministic RBC with optimal $n/3$ tolerance, it is inherent according to Dolev-Reischuk bound [20] [8]), since it guarantees that if any node outputs a value, then all other nodes will output the same value. So n parallel RBCs might inherently incur huge cubic messages, and it becomes necessary to replace them by tighter broadcast primitives.

As illustrated in Fig. 4, we use a weaker (thus cheaper) broadcast primitive, i.e., the provable broadcast (PB) from Abraham et al. [4] to replace RBCs used in ACS. We do so for two considerations. First, the primitive is very compact in the sense of having only two rounds of communications, one is from the sender to all nodes to multicast its input m, the other round is to let each node to return a threshold signature share for m. As such, the sender can produce a full signature as a $lock$ "proof" to attest that the broadcast is completed, in the sense that at least $f + 1$ honest nodes have received the same value m. Second, since the $lock$ proof finally obtained by the PB sender cannot forged, it allows us to use a simple multicast to diffuse the proofs. [9] Now, each node can prepare a vector of $n - f$ unforgeable proofs for $n - f$ completed PBs. So a single constant-time MVBA can be used to efficiently pick up a vector of $n - f$ valid PB proofs to finalize the PBs batched in the final ACS output.

[7] Different from [4], here we consider the threshold signature for the hash value of the message instead of for the message itself, which is a natural extension in the presence of collision-resistance hash function and later might save communications when we leverage the provability to construct more efficient MVBA.

[8] Dolev and Reischuk actually demonstrated that the message complexity of deterministic Byzantine broadcast is $\mathcal{O}(f^2)$ for adversary controlling up to f nodes [20]. Their proof can be slightly tuned to prove the $\mathcal{O}(n^2)$ message lower-bound of deterministic RBC with $n/3$ tolerance, cf. Appendix B.

[9] Remark that multicasting the proof for PB immediately after running PB essentially gives us a strong variant of consistent broadcast [42].

However, an extra challenge appears immediately, since PB is not as strong as RBC to ensure that all honest nodes will eventually output upon any honest node outputs in it, causing a threat that a node might not eventually output in some PBs, even if the PBs are already selected by MVBA as part of the ACS output. To compensate the weakening of PB, we necessarily introduce a recovery phase to allow some nodes to retrieve missing PB outputs. We cautiously minimize the communication blowing-up during the phase by combining the idea of verifiable information dispersal from [14]: the sending nodes can commit the encoded fragments of a requested PB output to a Merkle tree, so the receiving node can verify that $f+1$ fragments are committed to the same Merkle root (thus can decode to the correct PB output). For sufficiently large input, our new ACS framework has a concrete communication cost at least same to that of Dumbo and HoneyBadgerBFT in the worst case, and if there is a fair network scheduler that never reorders messages, it can save up to 67% concrete communications relative to Dumbo and HoneyBadgerBFT.

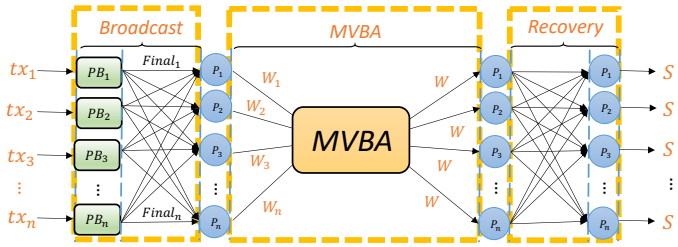

Fig. 4: The structure of the new ACS framework.

Constructing the new ACS framework. Now we describe the our new ACS framework that is centered around MVBA and PB. The formal description of the protocol is shown in Algorithm 1. Essentially, the protocol proceeds in three phases consisting of:

- *Broadcast*: (line 1-4). Upon every node $\mathcal{P}_i$ receives its input value v_i, it activates $\mathsf{PB}[\langle \mathsf{ID}, i\rangle]$ with using v_i as input, then waits for that $\mathsf{PB}[\langle \mathsf{ID}, i\rangle]$ outputs the $lock$ proof, after which it multicasts a Final message carrying $lock$.
- MVBA: (line 5-10). Each node $\mathcal{P}_i$ can wait for $n-f$ Final messages carrying $n-f$ valid $lock$s from distinct nodes, such that it can prepare a vector W_i including the $lock$s, and takes W_i as the input to participate into MVBA. Finally, the MVBA will decide a vector $\overline{W}$ consisting of $n-f$ valid $lock$ proofs for PBs along with the corresponding indices of these PBs.
- *Recovery*: (line 11-36). When a node $\mathcal{P}_i$ outputs in MVBA and gets $\overline{W}$, it can check whether it has received outputs in all PBs associated to the valid $lock$ proofs carried by $\overline{W}$. In case it has outputted in all PBs solicited by $\overline{W}$, it can output the set covering the PBs' outputs; otherwise, it multicasts a CallHelp message to request the missing PB outputs from other nodes. At the same time, each node $\mathcal{P}_j$ receiving $\overline{W}$ from MVBA listens to incoming CallHelp messages, and makes best effort to help the CallHelp message sender: for each PB that is requested by the CallHelp sender and also outputs to $\mathcal{P}_j$, the node $\mathcal{P}_j$ computes erasure codes of this PB output and a Merkle tree to commit the coded fragments; then via a Help message, $\mathcal{P}_j$ returns these Merkle tree roots, all j^{th} erasure code fragements and all j^{th} Merkle branches, to the CallHelp sender. Finally, the CallHelp sender can receive enough valid Help messages allowing it to recover all missing PB outputs and then output in ACS.

Algorithm 1 New ACS framework with identifier ID for P_i

Let $\{\mathsf{PB}[\langle \mathsf{ID}, j\rangle]\}_{j\in[n]}$ refer to n instances of provable broadcast protocol, where $\mathcal{P}_j$ is the sender of $\mathsf{PB}[\langle \mathsf{ID}, j\rangle]$; the ValueValidation function of PB always return $true$ for any input; the Q of MVBA be the following predicate: $Q_{\mathsf{ID}}[\{(s_1,h_1,\sigma_1),\cdots,(s_n,h_n,\sigma_n)\}] \equiv$ (exist at least $n-f$ distinct $i\in[n]$, such that $s_i \neq \perp$ and $\mathsf{VerifyThld}_{2f+1}(\langle\langle \mathsf{ID}, s_i\rangle, h_i\rangle, \sigma_i) = 1$).

Initialization: $W_i = \{(s_1,h_1,\sigma_1),\cdots,(s_n,h_n,\sigma_n)\}$, where $(s_j,h_j,\sigma_j) \leftarrow (\perp,\perp,\perp)$ for all $1\le j\le n$; $M_i = \{(H^1,b_i^1,m_i^1),\cdots,(H^n,b_i^n,m_i^n)\}$, where $(H^j,b_i^j,m_i^j) \leftarrow (\perp,\perp,\perp)$ for all $1\le j\le n$; $HS\leftarrow\emptyset$; $FS=0$; $Ready = false$.

▷ Broadcast

1: **upon** receiving input value v_i **do**
2: input v_i to $\mathsf{PB}[\langle \mathsf{ID}, i\rangle]$
3: **upon** receiving $lock := \langle h_i, \sigma_i\rangle$ from $\mathsf{PB}[\langle \mathsf{ID}, i\rangle]$ **do**
4: multicast $(\mathsf{Final}, \mathsf{ID}, h_i, \sigma_i)$ ▷ Multicast PB proof

▷ Wait for $n-f$ PB proofs to run MVBA

5: **upon** receiving $(\mathsf{Final}, \mathsf{ID}, h_j, \sigma_j)$ from $\mathcal{P}_j$ for the first time **do**
6: **if** $\mathsf{VerifyThld}_{(2f+1)}(\langle\langle \mathsf{ID}, j\rangle, h_j\rangle, \sigma_j) = 1$ **then**
7: $(s_j,h_j,\sigma_j) \leftarrow (j,h_j,\sigma_j)$, where $(s_j,h_j,\sigma_j)\in W_i$
8: $FS = FS+1$
9: **if** $FS = n-f$ **then**
10: propose W_i for the MVBA[ID]

▷Recovery

11: **upon** the MVBA[ID] return $\overline{W} = \{(\bar{s}_1,\bar{h}_1,\bar{\sigma}_1),\cdots,(\bar{s}_n,\bar{h}_n,\bar{\sigma}_n)\}$ **do**
12: **for** $1\le j\le n$ and $\bar{s}_j \neq \perp$ **do**
13: **if** $value := v_j$ was not yet receive from $\mathsf{PB}[\langle \mathsf{ID}, j\rangle]$ s.t. $\bar{h}_j = \mathcal{H}(v_j)$ **then**
14: $HS \leftarrow HS \cup j$
15: $Ready \leftarrow true$
16: **if** $Ready = true$ and $|HS| > 0$ **then** ▷ Trigger CallHelp
17: multicast $(\mathsf{CallHelp}, \mathsf{ID}, HS)$
18: **for** $\ell \in HS$ **do**
19: **wait** for $f+1$ valid Help messages carrying $(H^\ell, b_j^\ell, m_j^\ell)$ with same H^ℓ
20: **interpolate** the $f+1$ leaves $\{m_j^\ell\}$ to reconstruct v_ℓ
21: **let** $S\subset[n]$ be the set of $\bar{s}_j \neq \perp$ for all $1\le j\le n$
22: **output** $\cup_{j\in S} v_j$ ▷ Output

23: **upon** receiving $(\mathsf{CallHelp}, \mathsf{ID}, HS_k)$ from node $\mathcal{P}_k$ for the first time **do**
24: **wait** until $Ready = true$ ▷ Wait for the output of MVBA
25: **for** any $j\in HS_k$ **do**
26: **if** already receiving $value{:=}v_j$ from $\mathsf{PB}[\langle \mathsf{ID}, j\rangle]$ and $\bar{h}_j = \mathcal{H}(v_j)$ **then**
27: $\{m_k^j\}_{k\in[n]} \leftarrow (f+1,n)$-erasure coding applied to v_j
28: **let** MT^j be a Merkle tree root computed over $\{m_k^j\}_{k\in[n]}$
29: **let** b_i^j is the i^{th} Merkle tree branch
30: $(H^j,b_i^j,m_i^j) \leftarrow (MT^j,b_i^j,m_i^j)$, where $(H^j,b_i^j,m_i^j)\in M_i$
31: send $(\mathsf{Help}, \mathsf{ID}, M_i)$ to $\mathcal{P}_k$
32: **for** each $j\in[n]$ **do** $(H^j,b_i^j,m_i^j) \leftarrow (\perp,\perp,\perp)$ ▷ Re-initialize M_i

33: **upon** receiving $(\mathsf{Help}, \mathsf{ID}, M_j)$ from node P_j for the first time **do**
34: parse M_j as $\{(H^1,b_j^1,m_j^1),\cdots,(H^n,b_j^n,m_j^n)\}$
35: **for** $1\le k\le n$ and $b_j^k \neq \perp$ **do**
36: **check** that b_j^k is valid for root H^k and leaf m_j^k, otherwise discard

Complexities. The complexities of our new ACS framework can be briefly analyzed in a modular way, by considering the costs of n parallel PBs, an MVBA instance, and a simple recovery phase.

As shown in Table II, the message complexity is reduced to $\mathcal{O}(n^2)$. This is mainly because n tight PBs replace n RBCs in Dumbo to realize a cheaper broadcast phase costing only $\mathcal{O}(n^2)$ messages, while the extra recovery phase at most incurs n^2 Help messages and n^2 CallHelp messages, and the MVBA instance can be easily instantiated with expected quadratic messages [4, 12, 32].

TABLE II: Comparison for performance metrics of ACS

	Complexity		
Protocol	Round	Communication	Message
HBBFT	$\mathcal{O}(\log n)$	$\mathcal{O}(n^2\|m\| + \lambda n^3 \log n)$	$\mathcal{O}(n^3)$
Dumbo1	$\mathcal{O}(\log \kappa)$	$\mathcal{O}(n^2\|m\| + \lambda n^3 \log n)$	$\mathcal{O}(n^3)$
Dumbo2	$\mathcal{O}(1)$	$\mathcal{O}(n^2\|m\| + \lambda n^3 \log n)$	$\mathcal{O}(n^3)$
This work	$\mathcal{O}(1)$	$\mathcal{O}(n^2\|m\| + \lambda n^3 \log n)$	$\mathcal{O}(n^2)$

The protocol's running time (a.k.a. asynchronous rounds) is expected constant, as the single randomized module, MVBA, can be implemented to be expected constant time [4, 12, 32].

The framework's communication complexity is asymptotically $\mathcal{O}(n^2|m|+\lambda n^3 \log n)$, where $|m|$ is the bit-length of each node's ACS input and λ is the size of security parameter (and it is easy to see when $|m| \geq \lambda n \log n$, it is optimal linear communication per ACS input). The n PBs cost $|m|n^2 + \mathcal{O}(\lambda n^2)$ bits, and all multicasts of PBs' *lock* proofs cost $\mathcal{O}(\lambda n^2)$ bits. If the MVBA module is instantiated by CKPS01 [12] or AMS19 [4], it would cost $\mathcal{O}(\lambda n^3)$ bits. Besides, all n^2 CallHelp messages might incur at most $\mathcal{O}(n^3)$ overall bits, and all n^2 Help messages would cost at most $2|m|n^2+\mathcal{O}(\lambda n^3 \log n)$ bits. So in the worst case, the overall communication cost of the new ACS design is $3|m|n^2+\mathcal{O}(\lambda n^3 \log n)$, which is at least same to that of Dumbo and HoneyBadgerBFT. In case there is a fair network scheduler that never reorders messages, no CallHelp and Help messages would be exchanged among honest nodes, so the overall communication becomes only $|m|n^2+\mathcal{O}(\lambda n^3 \log n)$, which saves up to 67% concrete communications for sufficiently large $|m|$ (e.g., $|m| > \lambda n \log n$).[10]

Security intuition. The new ACS framework implements all properties of ACS (cf. Appendix D for deferred proofs). Its securities can be intuitively understood as follows:

- *The termination* of the new ACS framework immediately follows the termination of PB and MVBA, along with the provability of PB that also ensures an honest node that multicasts a Help can eventually receive enough CallHelp to recover its missing PB outputs to form ACS output.
- *The agreement* follows the agreement of MVBA and the provability of PB, because the agreement of MVBA ensures any two honest nodes to output the same $\overline{W}$ including the same unforgeable PB proofs attesting that more than $f+1$ honest nodes do output the same values in these PBs, so no matter an honest node directly output in the PBs or recover the PBs' outputs through the recovery phase, its ACS output must be same to any other honest node's.
- *The validity* is trivial, because the external validity of MVBA ensures that it output a $\overline{W}$ that containing $n-f$ valid PB proofs from distinct PB instances.

V. Speeding MVBA:

[10]Broadcasts are normally completed after MVBA finishes, because MVBA is started only after $n-f$ broadcasts are actually completed. Even in the extreme-case where the recovery phase is always triggered, the price of recovery is only 2 rounds, while our latency improvements are still more.

Toward a Speeding Dumbo

This section deals with the expensive MVBA module in ACS. We propose a novel MVBA, called Speeding MVBA (sMVBA) that are substantially faster than all existing ones.

Overview of sMVBA**.** At the very high-level, sMVBA is a concretely more compact MVBA design following the beautiful asynchronous "view-change" methodology from Abraham et al. in [4], which allows to directly design MVBA with a tighter structure from a variant of common coin. To make a shorter critical path, we develop a new two-phase Yes-No voting technique to ensure an output in a view must be the only possible output in later views.

As explained in Introduction and illustrated in Fig. 3, AMS19 [4] relies on **key-lock-commit** proofs to facilitate asynchronous "view-change", and these three proofs need a 4-staged PB protocol to generate.

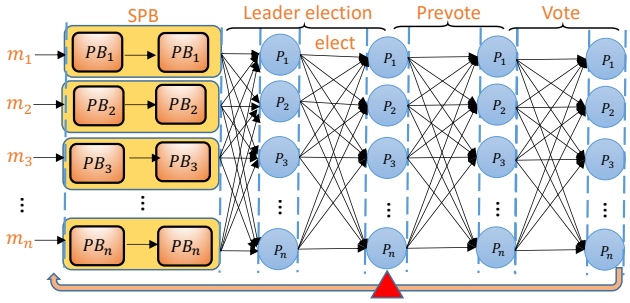

Fig. 5: sMVBA structure triangle is a short-cut for nodes to output (such that they might skip the remaining protocol execution after leader election).

As shown in Fig. 5, our sMVBA protocol would invoke the leader election earlier than AMS19, in order to enjoy faster terminations in the $2/3$ good cases where an already completed broadcast is elected. Namely, after 2-staged PBs (which gives a strong provable broadcast or SPB, to be formally defined below) instead of 4-staged PBs, sMVBA would finish the broadcasts and start the leader election phase. Nevertheless, we still need a mechanism to ensure safety and liveness in the other $1/3$ worse case as if [4] uses **lock** and **key** proofs. To this end, we introduce a simple two-phase "Yes"-"No" voting after leader election. In the 1st "Pre-vote" round, the nodes vote on a binary variable (e.g., either "Yes" or "No") according to whether they output in the leader's SPB. In the 2nd main "Vote" round, a node further votes on "Yes/No", i.e., whether to output or not, based on whether it collects any "Yes" or solicit enough "No" during the first vote. Finally, once an honest node solicits enough "Yes" in the 2nd vote phase, it decides to output, which also ensures that all honest nodes must at least receive a valid proof on "Yes" during the 2nd vote phase, and more importantly, no corrupted node can forge a proof on "No".

Thanks to the two-phase Yes-No voting, we do not have to explicitly "lock" any node as if AMS19, and just specify the external predicate function of SPB to ensure either input valid "Yes" proof or valid "No" proof: For valid "Yes", the predicate shall also check the proof binds a value (which is the potential output of earlier views); For valid "No", we do not worry what input it binds, because there certainly is no honest node did output before. As such, if any node outputs in some view, then in all later views, all honest nodes would use the same value as

their valid SPB input, and no corrupted node can use a value different from it to pass SPB's validity checking. In this way, the two-round "Yes"-"No" voting handles the "dirty" work in the $1/3$ worse case to ensure that the protocol satisfies safety and liveness all the time.

A. Preparation: lifting PB stronger

One key idea of constructing sMVBA is to enter the leader election phase soon after the sender delivers a single proof (instead of three) with using two sequential PBs. To be precise, we expect to compose the two PBs to be a strong PB (SPB) that can be formalized as follow.

Strong provable broadcast (SPB) protocol is a strengthened variant of PB. In SPB, there is a designated sender and global predicate CheckValue for validating the sender's input. The sender would broadcast a value v along with a validation string π to all nodes. Each node would output a tuple Lock $:= (v, \sigma_1)$, and the sender would additionally output a tuple Finish $:= (v, \sigma_2)$. Both of σ_1 and σ_2 shall be valid threshold signatures for $\mathcal{H}(v)$. And we also say the SPB has completed, if the sender did output a valid Finish tuple.

Besides the above syntax, an SPB protocol (that can be activated on an explicit session identifier ID) shall satisfy the following properties with all but negligible probability:

- *Validity*. Same to that of PB.
- *Termination*. If the sender $\mathcal{P}_s$ is honest and inputs m:=(v_s, π) satisfying CheckValue(ID, m) = 1 and all honest nodes activate SPB[ID] without invoking $abandons$(ID), then each honest node would output a valid Lock tuple of v_s and σ_1 s.t. VerifyThld$_{(2f+1)}(\langle\langle \text{ID}, 1\rangle, \mathcal{H}(v_s)\rangle, \sigma_1)$ = 1. Additionally, the sender $\mathcal{P}_s$ outputs a valid Finish tuple of v_s and σ_2 s.t. VerifyThld$_{(2f+1)}(\langle\langle \text{ID}, 2\rangle, \mathcal{H}(v_s)\rangle, \sigma_2)$ =1.
- *Provability*. If the sender can produce a valid Finish $:= (v_s, \sigma_2)$ output, then (1) for any two valid Finish $:= (v_s, \sigma_2)$ and Finish$' := (v'_s, \sigma'_2)$, then $v_s = v'_s$; (2) at least $f + 1$ honest nodes did output valid Lock with the same value v_s.
- *Abandon-ability*. Same to that of PB.

Remark. Once an honest node outputs Lock, no honest node would output a Lock carrying a different value, and valid Finish attests that at least $f + 1$ honest nodes did output valid Lock.

Details of SPB **protocol.** The detailed SPB protocol is shown in Algorithm 2. Slightly informally, it proceeds as follows:

1) *Initialization* (line 1-2). All honest nodes activate two PBs.
2) *First* PB (line 3-5). The sender $\mathcal{P}_s$ passes its input (v_s, π) to the first PB instance and waits for it return a $lock$ message.
3) *Second* PB (line 6-9). Upon the first PB returns $lock := (h_1, \sigma_1)$ to $\mathcal{P}_s$, the sender passes (v_s, σ_1) to the second PB instance as input, and waits for the second PB returns $lock$ to output it as Finish. For every node, once receiving $value := (v_s, \sigma_1)$ from the second PB, output Lock $:= value$.

Algorithm 2 SPB subprotocol with identifier ID, sender $\mathcal{P}_s$, and a input validation predicate CheckValue

```
1: activate PB[⟨ID, 1⟩] with ValueValidation defined in Line 10
2: activate PB[⟨ID, 2⟩] with ValueValidation defined in Line 11

3: if P_i = P_s then
4:   upon receiving input value (v_s, π) s.t. CheckValue(ID, v_i, π) = 1 do
5:     input (v_s, π) to PB[⟨ID, 1⟩] and wait until it outputs lock := (h_1, σ_1)
6:     input (v_s, σ_1) to PB[⟨ID, 2⟩] and wait until it outputs lock := (h_2, σ_2)
7:     output Finish := (v_s, σ_2)                          ▷ Produce Finish

8: upon PB[⟨ID, 2⟩] outputs value := (v_s, σ_1) do
9:   output Lock := (v_s, σ_1)                              ▷ Produce Lock

   procedure ValueValidation(⟨ID, j⟩, v_s, Σ):
10:  if j=1 then return CheckValue(ID, v_s, Σ)
11:  if j=2 then return VerifyThld_(2f+1)(⟨⟨ID, 1⟩, H(v_s)⟩, Σ)
   procedure abandons(ID):
12:  for j=1,2 do abandon[⟨ID, j⟩]
```

Security intuition of SPB **protocol**. The security of the simple SPB construction in Algorithm 2 is easy to see: *Validity* and *Abandon-ability* follows immediately from those two properties of PB. For *provability*, it is naturally an extension of PB's, because breaking it directly break PB's. For *termination*, it is also immediate, because of the termination of PB, along with the fact that the predicate function of both PB instances are properly specified, so an honest sender can always pass valid inputs to them.

Complexities of SPB **protocol**. Because the SPB sequentially runs two PBs, its complexities are asymptotically same to those of PB, that means, its message complexity is $\mathcal{O}(n)$ and its communication complexity is $\mathcal{O}(|m|n + \lambda n)$.

B. Constructing Speeding MVBA

Here we are ready to present the details of sMVBA. Its basic idea is inspired by the view-change methodology of AMS19-MVBA [4] but has a more compact structure to run n parallel SPBs (which are 2-stage PBs instead of 4-stage PBs) before the leader election. With $2/3$ chance, the elected SPB instance was completed, i.e. the SPB produced a Finish, then the input value of the SPB will be output in the current view. In other worse cases, sMVBA leverages a compact two-phase Yes-No voting paradigm to guarantee: if some honest node outputs value v in a view R, then in any view R$' >$ R, the valid input value to SPBs must be v, any other values will be rejected by the CheckValue predicates of SPBs.

Construction of sMVBA. The formal description of sMVBA is shown in Algorithm 3, 4, and 5. Algorithm 3 focuses on how to act upon some internal states are changed due to receiving new messages in Algorithm 5. We describe the process for handling incoming messages of sMVBA in Algorithm 5. Slightly informally, it has five logic phases proceeding as follows:

1) SPB *phase* (line 1-4). The n nodes broadcast their own input values v through n concurrent SPB instances, where each node $\mathcal{P}_i$ is the designated sender of SPB[$\langle$id, R, $i\rangle$]. Here id is the identifier of the sMVBA protocol, and R represents the current view.
2) *Election phase* (line 5-9). Upon the nodes realize that enough Finish for the SPB instances (i.e., $2n/3$) have been produced, they can make sure that enough input values (i.e., $2n/3$) have been broadcasted and "locked" across the network, then the nodes enter the

election phase with multicasting a Done message and invoking the Election protocol. More precisely, each node multicasts a Done message together with the Election protocol's SHARE message at the same time. There could be some honest node that did not receive enough Finish for SPBs but receive at least $f+1$ Done messages from distinct nodes, it also enters the election phase to multicast a Done message together with the Election's SHARE message (cf. Alg. 5 line 9-11). Then, the honest nodes wait for $2f+1$ Done-SHARE messages to abandon all SPB instances and obtain the common pseudo-random number $\ell \in [n]$. Here ℓ represents that the randomly elected leader in the current view is $\mathcal{P}_\ell$.

3) PreVote *phase* (line 10-16). Upon the Election returns ℓ, the honest node checks whether a valid Finish message associated to ℓ was received: if yes, it then multicasts a Halt message carrying the valid Finish, after which it immediately enters a *short-cut* to output and exit the protocol; else (i.e., no corresponding Finish received), it multicasts $(\mathsf{PreVote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$ if $\mathsf{SPB}[\langle \mathsf{id}, \mathsf{R}, \ell \rangle]$ already returns Lock, otherwise, multicasts $(\mathsf{PreVote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{No})$.
4) Vote *phase* (line 17-21). All honest nodes wait for $2f+1$ valid PreVote messages, if these $2f+1$ PreVote has a valid $(\mathsf{PreVote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$, then multicast $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$, else multicast $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{No})$. Here a valid $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{No})$ message contains a proof that is a threshold signature aggregated from $2f+1$ $(\mathsf{PreVote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{No})$.
5) Finish *phase* (line 22-35). The honest nodes wait for $2f+1$ valid Vote messages, then combine them to generate a Finish proof and multicast it via a Halt message, and output the corresponding value and halt; else if receive $2f+1$ valid $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{No})$ messages, compute σ_{VN} and take σ_{VN} with the input value of the current SPB into next view; else, if a node receives $2f+1$ valid Vote messages including two types message, i.e. both $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$ and $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{No})$, takes the value v_ℓ and proof $\pi = (\mathsf{Yes}, \mathsf{R}, \sigma_1)$ from $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes}, v_\ell, \sigma_1, \rho_{2,i})$ into the next view as its SPB input.

Security intuition of sMVBA. The protocol described in Algorithm 3-5 realizes MVBA among n nodes against an adversary controlling $f \leq \lfloor \frac{n-1}{3} \rfloor$ nodes (cf. Appendix E for detailed proofs). Its securities can be intuitively understood as:

- *Termination:* Suppose no honest node outputs yet before a view R, all honest nodes must have valid input (may include a proof from an earlier view) to enter view R and at least $n-f$ SPBs can be completed in the view. With at least $2/3$ probability, Election can select a completed SPB instance whose corresponding Finish proof has been generated. In such good case, every honest node can either receive $2f+1$ valid Vote-Yes messages (otherwise, the adversary can break the unforgeability of threshold signature to forge a valid Vote-No message) or receive a valid Halt message (sent from another node that outputs in the view), thus can output in the view. Hence, with at least $2/3$ probability, all honest nodes terminate in view R.

Algorithm 3 sMVBA protocol with identifier id for node $\mathcal{P}_i$: main process, cf. Alg. 5 for message handlers (*a process that handles incoming messages and changes variables in Alg.3 to take responses*) and Alg.4 for SPB predicate

Initialization: let $\mathsf{R} \leftarrow 1$, $\pi \leftarrow \{\}$
for every $view$ $\mathsf{R} \geq 1$: $D\text{-}flag_{\mathsf{R}} \leftarrow 0$, $Y\text{-}flag_{\mathsf{R}} \leftarrow 0$, $N\text{-}flag_{\mathsf{R}} \leftarrow 0$, $\mathsf{PN}_{\mathsf{R}} \leftarrow \{\}$, $\mathsf{YFin}_{\mathsf{R}} \leftarrow \{\}$, $\mathsf{NFin}_{\mathsf{R}} \leftarrow \{\}$, $F_{\mathsf{R}} \leftarrow \{\}$, $S_{\mathsf{R}} \leftarrow \{\}$, $Ready_{\mathsf{R}} \leftarrow 0$

1: **repeat**:
2: **initialize** $\mathsf{SPB}[\langle \mathsf{id}, \mathsf{R}, j \rangle]$ with the designated sender $\mathcal{P}_j$ for each $j \in [n]$
3: **wait** until receiving input v_i s.t. $\mathsf{CheckValue}(\langle \mathsf{id}, \mathsf{R}, i \rangle, v_i, \pi) = 1$
▷Broadcast phase
4: pass (v_i, π) into $\mathsf{SPB}[\langle \mathsf{id}, \mathsf{R}, i \rangle]$ as input
▷ Election phase: wait for $n-f$ FIN messages
5: **wait** until $D\text{-}flag_{\mathsf{R}} = 1$
6: **multicast** $(\mathsf{Done}, \mathsf{id}, \mathsf{R})$
7: **wait** until receiving $2f+1$ $(\mathsf{Done}, \mathsf{id}, \mathsf{R})$ from distinct nodes
8: $abandons(\langle \mathsf{id}, \mathsf{R}, j \rangle)$ for each $j \in [n]$
9: $l \leftarrow \mathsf{Election}[\langle \mathsf{id}, \mathsf{R} \rangle]$
▷ PreVote phase
10: **wait** until $Ready_{\mathsf{R}} = 1$
11: **if** delivers Lock from $\mathsf{SPB}[\langle \mathsf{id}, \mathsf{R}, \ell \rangle]$ **then**
12: parse Lock as (v_ℓ, σ_1)
13: **multicast** $(\mathsf{PreVote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes}, v_\ell, \sigma_1)$
14: **else**
15: $\rho_{pn,i} \leftarrow \mathsf{SignShare}_{(2f+1)}(sk_i, \langle \mathsf{No}, \langle \mathsf{id}, \mathsf{R}, \ell \rangle, null \rangle)$
16: **multicast** $(\mathsf{PreVote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{No}, null, \rho_{pn,i})$
▷ Vote phase
17: **wait** until $Y\text{-}flag_{\mathsf{R}} = 1$ or $N\text{-}flag_{\mathsf{R}} = 1$
18: **if** $Y\text{-}flag_{\mathsf{R}} = 1$ **then**
19: **multicast** $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes}, v_\ell, \sigma_1, \rho_{2,i})$
20: **else**
21: **multicast** $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{No}, null, \sigma_{\mathsf{PN}}, \rho_{vn,i})$
▷ Finish phase: wait for $2f+1$ Vote messages
22: **wait** until $|\mathsf{YFin}_{\mathsf{R}}| + |\mathsf{NFin}_{\mathsf{R}}| = 2f+1$
23: **if** $|\mathsf{YFin}_{\mathsf{R}}| = 2f+1$ **then** ▷ All Votes are YES
24: $\sigma_2 \leftarrow \mathsf{Combine}_{(2f+1)}(\langle \langle \mathsf{id}, \mathsf{R}, \ell \rangle, 2 \rangle, \mathcal{H}(v_\ell))$, $\mathsf{YFin}_{\mathsf{R}}$)
25: **Let** Finish $:= (v_\ell, \sigma_2)$ ▷ Output and halt
26: **multicast** $(\mathsf{Halt}, \mathsf{id}, \mathsf{R}, \mathsf{Finish})$, **output** v_ℓ and **then** halt
27: **else**
28: **if** $|\mathsf{NFin}_{\mathsf{R}}| = 2f+1$ **then** ▷ All Votes are NO
29: $\sigma_{\mathsf{VN}} \leftarrow \mathsf{Combine}_{(2f+1)}(\langle \mathsf{UnLocked}, \langle \mathsf{id}, \mathsf{R}, \ell \rangle \rangle, \mathsf{NFin}_{\mathsf{R}})$
▷ $|\pi| \geq 1$, i.e. new π is the current π union with $(\mathsf{No}, \mathsf{R}, \sigma_{\mathsf{VN}})$
30: $\pi \leftarrow \pi \cup (\mathsf{No}, \mathsf{R}, \sigma_{\mathsf{VN}})$
31: $v_i \leftarrow v_i$, $\mathsf{R} \leftarrow \mathsf{R}+1$
32: **else** ▷ Including YES and NO
▷ $|\pi| = 1$, i.e. reset $\pi = \{(\mathsf{Yes}, \mathsf{R}, \sigma_1)\}$
33: $\pi \leftarrow (\mathsf{Yes}, \mathsf{R}, \sigma_1)$
34: $v_i \leftarrow v_\ell$, $\mathsf{R} \leftarrow \mathsf{R}+1$
35: **until** halt

Remark: in Election (Alg. 7), a node first multicasts a SHARE message, then waits for $2f+1$ SHARE messages to compute a coin. We use Election in a non-blackbox manner, namely, in line 6, multicast (Done, id, R) and SHARE messages at the same time. So in line 9, it shall have received $2f+1$ SHARE messages already, and can just compute l accordingly instead of invoking the Election protocol one more time.

Besides, if all honest nodes do not output in the current view, the proofs (aggregated from $2f+1$ Vote-No or obtained from Vote-Yes) can ensure that they will carry *valid* inputs into the next view to repeat the above procedure.
More importantly, it is possible that only some honest nodes output in view R. In the case, all nodes would eventually output as well, because once the first node outputs, it will multicast a Halt message, which can convince all other nodes to output and halt in one more round. Putting these together, all honest nodes can terminate in expected constant running time.

- *Agreement:* Agreement is slightly more involved. Suppose no honest node already outputs yet before a view R, and $\mathcal{P}_i$ is the first honest node that outputs v in the view R. In case another honest node $\mathcal{P}_j$ outputs in the same view, the output of $\mathcal{P}_i$ and $\mathcal{P}_j$ must be

same. This is because the output condition in view R is to receive a valid FIN message from the selected leader, or to receive a valid Halt message, or to receive $2f+1$ valid Vote-Yes messages. All three cases need to verify a valid proof for the selected SPB in the current view. As such, the *provability* of SPB would ensure that if any two nodes would output in the same view, their output is same to the value broadcasted by the selected SPB.

Next, we consider the other case that $\mathcal{P}_j$ outputs in a view $\mathsf{R}' > \mathsf{R}$. Conditioned on $\mathcal{P}_i$ outputs v in view R, every honest node (including $\mathcal{P}_j$) must receive at least one valid $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$ in the view R (thus taking v as next view' input); and more importantly, it is infeasible for anyone to forge a valid $(\mathsf{No}, \mathsf{R}, \sigma_{\mathsf{VN}})$ message, because the adversary cannot forge the threshold signature σ_{VN} as it has at most $2f$ shares of it. The CheckValue function of SPB is further specified to ensure: without valid σ_{VN}, a node cannot send $v' \neq v$ via SPB in later views (otherwise, the validity of SPB is broken). So the only possible output of $\mathcal{P}_j$ in view $\mathsf{R}' > \mathsf{R}$ is same to the output of $\mathcal{P}_i$ in view R.

- *External-Validity:* it trivially holds because each SPB broadcast input value needs to satisfy CheckValue.

Algorithm 4 sMVBA protocol with identifier id for node $\mathcal{P}_i$: the external predicate of SPB, cf. Alg 3 for the main protocol process

function CheckValue($\langle \mathsf{id}, \mathsf{R}, j \rangle, v, \pi$):

▷ $|\pi| = 0$, i.e. initialized validation string $\pi = \{\}$

1: **if** R $= 1$ and $|\pi| = 0$ **then** return $\mathsf{Q}(v)$

// Either received one Vote message that contains Yes or R $= 2$ and no node outputs in R $= 1$, then $|\pi| = 1$, i.e. $\pi = \{(\mathsf{Yes}, \mathsf{R}, \sigma_1)\}$ or $\{(\mathsf{No}, 1, \sigma_{\mathsf{VN}})\}$

2: **if** $|\pi := (*, \mathsf{R}-1, \rho)| = 1$ **then** ▷ $|\pi| = 1$

3: $a \leftarrow \mathsf{VerifyThld}_{(2f+1)}(\langle\langle\langle \mathsf{id}, \mathsf{R}-1, \mathsf{Election}[\langle \mathsf{id}, \mathsf{R}-1\rangle]\rangle, 1\rangle, \mathcal{H}(v)\rangle, \rho)$

4: $b \leftarrow \mathsf{VerifyThld}_{(2f+1)}(\langle \mathsf{UnLocked}, \langle \mathsf{id}, 1, \mathsf{Election}[\langle \mathsf{id}, 1\rangle]\rangle\rangle, \rho)$ // R $= 2$

5: return $(a \vee b) \wedge \mathsf{Q}(v)$

▷ $|\pi| > 1$

6: **if** R > 2 and $|\pi| > 1$ and $(\mathsf{No}, \mathsf{R}-1, \rho) \in \pi$ **then**

// No node ouputs before view R, then $\pi = \{(\mathsf{No}, 1, \rho), \cdots, (\mathsf{No}, \mathsf{R}-1, \rho')\}$

7: **if** for $0 < k < \mathsf{R}-1$: $(\mathsf{No}, k, \rho_k) \in \pi$ **then**

8: **for** $k \in [1 : \mathsf{R}-1] : (\mathsf{No}, k, \rho_k) \in \pi$ **do**

9: $b \leftarrow \mathsf{VerifyThld}_{(2f+1)}(\langle \mathsf{UnLocked}, \langle \mathsf{id}, k, \mathsf{Election}[\langle \mathsf{id}, k\rangle]\rangle\rangle, \rho_k)$

10: **if** $b = 0$ **then** return 0

11: return $\mathsf{Q}(v)$

// Some node ouputs in view r, and the SPB$[\langle \mathsf{id}, k, \mathsf{Election}[\langle \mathsf{id}, k\rangle]\rangle]$ did not complete for view k from $r+1$ to R -1, then $\pi = \{(\mathsf{Yes}, r, \rho_r), (\mathsf{No}, r+1, \rho_{r+1}), \cdots, (\mathsf{No}, \mathsf{R}-1, \rho_{\mathsf{R}-1})\}$

12: **if** $\exists\ 0 < r < \mathsf{R}-1$: $(\mathsf{Yes}, r, \rho_r) \in \pi$ **then**

13: **if** for $r < k < \mathsf{R}-1$: $(\mathsf{No}, k, \rho_k) \in \pi$ **then**

14: **for** $k \in [r+1 : \mathsf{R}-1] : (\mathsf{No}, k, \rho_k) \in \pi$ **do**

15: $b \leftarrow \mathsf{VerifyThld}_{(2f+1)}(\langle \mathsf{UnLocked}, \langle \mathsf{id}, k, \mathsf{Election}[\langle \mathsf{id}, k\rangle]\rangle\rangle, \rho_k)$

16: **if** $b = 0$ **then** return 0

17: $a \leftarrow \mathsf{VerifyThld}_{(2f+1)}(\langle\langle\langle \mathsf{id}, r, \mathsf{Election}[\langle \mathsf{id}, r\rangle]\rangle, 1\rangle, \mathcal{H}(v)\rangle, \rho_r)$

18: return $a \wedge \mathsf{Q}(v)$

19: **else** return 0

Complexities of sMVBA. Similar to [4], sMVBA costs expected constant asynchronous rounds and expected $\mathcal{O}(n^2)$ messages, where each message is not larger than $\mathcal{O}(\lambda)$ bits if assuming that the input is $\mathcal{O}(\lambda)$-bit. We elaborate the difference between the concrete expected running time (rounds) of sMVBA and AMS19-MVBA [4] in Table I. Two settings are considered, namely, the adversarial case capturing 1/3 Byzantine corruptions in the fully asynchronous network, and the best case capturing all honest nodes and a fair network scheduler that never reorders messages. No matter the setting,

Algorithm 5 sMVBA with identifier id for node $\mathcal{P}_i$: the protocol message handlers, cf. Alg 3 for the main protocol process that might take responses if certain messages are received by this algorithm

1: **upon** SPB$[\langle \mathsf{id}, \mathsf{R}, i\rangle]$ delivers Finish **do**

2: **multicast** (FIN, id, R, Finish) ▷ Multicast FIN

3: **upon** receiving (FIN, id, R, Finish) from $\mathcal{P}_j$ for the first time **do**

4: parse Finish as (v_j, σ_2)

5: **if** $\mathsf{VerifyThld}_{(2f+1)}(\langle\langle\langle \mathsf{id}, \mathsf{R}, j\rangle, 2\rangle, \mathcal{H}(v_j)\rangle, \sigma_2) = 1$ **then**

6: $F_{\mathsf{R}} \leftarrow F_{\mathsf{R}} \cup (j, \mathsf{Finish})$

7: **if** $|F_{\mathsf{R}}| = n - f$ and $D\text{-}flag_{\mathsf{R}} = 0$ **then** ▷ Wait for $n - f$ FIN

8: $D\text{-}flag_{\mathsf{R}} = 1$ ▷ Start Election phase

9: **upon** receiving $f+1$ (Done, id, R) from distinct nodes **do**

10: **if** $D\text{-}flag_{\mathsf{R}} = 0$ **then**

11: $D\text{-}flag_{\mathsf{R}} = 1$ ▷ Start Election phase

12: **upon** Election$[\langle \mathsf{id}, \mathsf{R}\rangle]$ return ℓ **do** ▷ Short-cut to output

13: **for** $(k, \mathsf{Finish}) : (k, \mathsf{Finish}) \in F_{\mathsf{R}}$ **do**

14: **if** $k = l$ **then**

15: parse Finish as (v_k, σ_2)

16: **multicast** (Halt, id, R, Finish), **Output** v_k and **then** halt

17: $Ready_{\mathsf{R}} \leftarrow 1$

18: **upon** receiving (Halt, id, R, Finish) **do** ▷ Amplify speeding, ensure liveness

19: parse Finish as (v_k, σ_2)

20: **if** $k = \mathsf{Election}[\langle \mathsf{id}, \mathsf{R}\rangle]$ **then**

21: **if** $\mathsf{VerifyThld}_{(2f+1)}(\langle\langle\langle \mathsf{id}, \mathsf{R}, k\rangle, 2\rangle, \mathcal{H}(v_k)\rangle, \sigma_2) = 1$ **then**

22: **multicast** (Halt, id, R, Finish), **Output** v_k and **then** halt

23: **upon** receiving (PreVote, id, R, ℓ, Yes, v_ℓ, σ_1) from $\mathcal{P}_j$ for the first time **do**

24: **if** $\mathsf{VerifyThld}_{(2f+1)}(\langle\langle\langle \mathsf{id}, \mathsf{R}, \ell\rangle, 1\rangle, \mathcal{H}(v_\ell)\rangle, \sigma_1)$=1 and $N\text{-}flag_{\mathsf{R}}$=0 **then**

25: $\rho_{2,i} \leftarrow \mathsf{SignShare}_{(2f+1)}(sk_i, \langle\langle\langle \mathsf{id}, \mathsf{R}, \ell\rangle, 2\rangle, \mathcal{H}(v_\ell)\rangle)$

26: $Y\text{-}flag_{\mathsf{R}} \leftarrow 1$ ▷ Receive one valid (PreVote, id, R, ℓ, Yes)

27: **upon** receiving (PreVote, id, R, ℓ, No, $null, \rho_{pn,j}$) from $\mathcal{P}_j$ for the first time **do**

28: **if** $\mathsf{VerifyShare}_{(2f+1)}(\langle\langle \mathsf{No}, \langle \mathsf{id}, \mathsf{R}, \ell\rangle, null\rangle, (j, \rho_{pn,j})) = 1$ **then**

29: $\mathsf{PN}_{\mathsf{R}} \leftarrow \mathsf{PN}_{\mathsf{R}} \cup (j, \rho_{pn,j})$

30: **if** $|\mathsf{PN}_{\mathsf{R}}| = 2f+1$ and $Y\text{-}flag_{\mathsf{R}} = 0$ **then**

31: $\sigma_{\mathsf{PN}} \leftarrow \mathsf{Combine}_{(2f+1)}(\langle \mathsf{No}, \langle \mathsf{id}, \mathsf{R}, \ell\rangle, null\rangle, \mathsf{PN}_{\mathsf{R}})$

32: $\rho_{vn,i} \leftarrow \mathsf{SignShare}_{(2f+1)}(sk_i, \langle \mathsf{UnLocked}, \langle \mathsf{id}, \mathsf{R}, \ell\rangle\rangle)$

33: $N\text{-}flag_{\mathsf{R}} \leftarrow 1$ ▷ Receive $2f+1$ valid (PreVote, id, R, ℓ, No)

34: **upon** receiving (Vote, id, R, ℓ, Yes, $v_\ell, \sigma_1, \rho_{2,j}$) from $\mathcal{P}_j$ for the first time **do**

35: **if** $\mathsf{VerifyThld}_{(2f+1)}(\langle\langle\langle \mathsf{id}, \mathsf{R}, \ell\rangle, 1\rangle, \mathcal{H}(v_\ell)\rangle, \sigma_1) = 1$ **then**

36: **if** $\mathsf{VerifyShare}_{(2f+1)}(\langle\langle\langle \mathsf{id}, \mathsf{R}, \ell\rangle, 2\rangle, \mathcal{H}(v_\ell)\rangle, (j, \rho_{2,j}))$ =1 **then**

37: $\mathsf{YFin}_{\mathsf{R}} \leftarrow \mathsf{YFin}_{\mathsf{R}} \cup (j, \rho_{2,j})$ ▷ The set of (Vote, id, R, ℓ, Yes)

38: **upon** receiving (Vote, id, R, ℓ, No, $null, \sigma_{\mathsf{PN}}, \rho_{vn,j}$) from $\mathcal{P}_j$ for the first time **do**

39: **if** $\mathsf{VerifyThld}_{(2f+1)}(\langle \mathsf{No}, \langle \mathsf{id}, \mathsf{R}, \ell\rangle, null\rangle, \sigma_{\mathsf{PN}}) = 1$ **then**

40: **if** $\mathsf{VerifyShare}_{(2f+1)}(\langle \mathsf{UnLocked}, \langle \mathsf{id}, \mathsf{R}, \ell\rangle\rangle, (j, \rho_{vn,j}))$ =1 **then**

41: $\mathsf{NFin}_{\mathsf{R}} \leftarrow \mathsf{NFin}_{\mathsf{R}} \cup (j, \rho_{vn,j})$ ▷ The set of (Vote, id, R, ℓ, No)

sMVBA has two PBs less than AMS19 before leader election (4 rounds in each view), and we also compress the messages needed to stop PBs (2 rounds in each view). In the adversarial case, our two-phase voting slightly increases one round in each view; while in the best case with a fair network scheduler, we develop a short-cut to output as soon as the leader election returns, which skips the voting. In sum, sMVBA overall saves 5 rounds in each view under the adversarial setting (which is then amplified to 7.5 because both protocols need on average 3/2 views to terminate in this worst case), and saves overall 7 rounds in the best case.

C. Speeding Dumbo: put everything together

We can instantiate our new ACS framework in Section IV by sMVBA to get a super speeding asynchronous common subset (ACS), i.e., Speeding Dumbo (sDumbo), from which there are several simple conversions to build fully-fledged asynchronous atomic broadcast. Through the paper, we adopt a simple conversion from Miller et al. [33] with using threshold encryption, cf. Appendix A for details about the reductions from atomic broadcast to ACS.

VI. Implementation and Evaluations

We implemented and evaluated sDumbo (and sMVBA) in the WAN setting. Along the way, we also implemented and made systematic comparisons with several typical MVBAs including AMS19 [4] and CKPS01 [12], and the state-of-the-art asynchronous protocol Dumbo [26]. A typical partially-synchronous HotStuff [45] is also implemented and deployed in the same setting as a "reference-point" in our evaluations.

Implementation details. All protocols are written as single-process Python 3 program, and are forked from the open source code at https://github.com/initc3/HoneyBadgerBFT-Python/. The p2p channels among nodes are established using unauthenticated TCP sockets. The concurrent tasks are handled by gevent library. Boldyreva's pairing-based threshold signature [9] and Baek et al.'s hybrid threshold encryption [5] are adopted for implementing threshold signature and threshold encryption, respectively, whose setups are conduced by us - the implementer. Different from [33] that optimistically skip the verification for threshold signatures, our tests verify all signatures, which better reflects the actual performance in the worse cases, e.g., with corruptions. We also implement the two-chain version of HotStuff (with a stable non-faulty leader) in about 200 lines Python code for fair comparison. Same to [45], we use ECDSA to form quorum certificates in HotStuff.

Test environment. The experiments are conducted among EC2 t2.medium instances evenly distributed in 15 AWS regions: Virginia, Ohio, California, Oregon, Canada, Mumbai, Seoul, Singapore, Sydney, Tokyo, Frankfurt, London, Ireland, Stockholm and San Paulo. [11] Note that t2.medium is not supported in Stockholm, so we adopted t3.medium, which has a similar configuration. All tests scale up to 150 instances. A transaction in our tests is a string of 250 bytes, which approximates the size of basic Bitcoin transactions with one input and two outputs. The batch size (also called the system load sometimes) represents the number of transactions proposed by all nodes in an one-shot agreement, and will vary from 1 to 3×10^6.

Result highlights. Key information from our experiments are:

(1) Our end result, sDumbo, significantly reduces the latency of Dumbo by about 50% and attains a throughput twice as Dumbo's, almost despite system size, cf. Table III;

(2) For some small scales (e.g., 10 or 16 nodes), sDumbo reduces the latency to the same magnitude of HotStuff's in the same WAN environment, though the latency gap between HotStuff and Dumbo was much larger, cf. Table IV.

TABLE III: Improvements of latency and throughput

	Basic Latency (sec)			Peak Throughput (tx/s)		
Scale	Dumbo	sDumbo		Dumbo	sDumbo	
$n = 4$	0.97	0.46	↓52.6%	14,087	34,958	↑148%
$n = 16$	3.1	1.7	↓45.2%	18,586	39,662	↑113%
$n = 82$	17.8	9.2	↓48.3%	10,584	24,564	↑132%
$n = 121$	33.7	21	↓37.7%	7,163	17,143	↑139%

[11] We remark that all our tests were with modest and identical configurations across the globe to fairly compare different protocols. The experiments for HotStuff in [45] used high-end configurations within one AWS region. Thus, we show HotStuff results different from it [45].

TABLE IV: Comparison to HotStuff at small scales (n= 10 or 16)

	Latency (sec) @ Throughput = 2500 tps				
Scale	HotStuff (HS) [45]	Dumbo [26]		sDumbo	
$n = 10$	0.48	3.9	8 x HS	1.8	3.7 x HS
$n = 16$	0.41	4.2	10 x HS	2.1	5 x HS

A. Performance of each module

We first evaluate the performance of replacing RBCs by PBs and the performance of the new sMVBA protocol, respectively, to understand our modular improvements.

PB **broadcasts v.s. RBC broadcasts**. To evaluate the improvement brought by replacing RBCs by PB in our new ACS framework in Section IV, we use the same CKPS01 MVBA [12] to implement our new ACS and Dumbo, respectively, and evaluate them with varying batch sizes in the WAN setting (for n = 4, 31 and 82).

Fig. 6 shows the dependency of throughput on batch size. When the batch size approximates 5×10^4 (as shown by the red dotted line), the benefit of using PBs becomes very significant, and finally nearly doubles the throughput of Dumbo if the batch size keeps on increasing. Namely, Dumbo (instantiated by RBC+CKPS01) exhausts bandwidth much more quickly than our new ACS framework (instantiated by PB+CKPS01). This reflects our analysis in Section IV that our new ACS framework is much more communication conserving than Dumbo in the presence of a fair network scheduler.

Fig. 6 also shows the basic latency (i.e., the latency when inputting only one transaction) of the protocols using two different broadcast modes. The performance of new broadcast mode using PBs is better, and its merit gradually appears when the system scale gets larger. Because the system load is nearly zero, the result mainly reflects the improvement brought by the reduction of message complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.

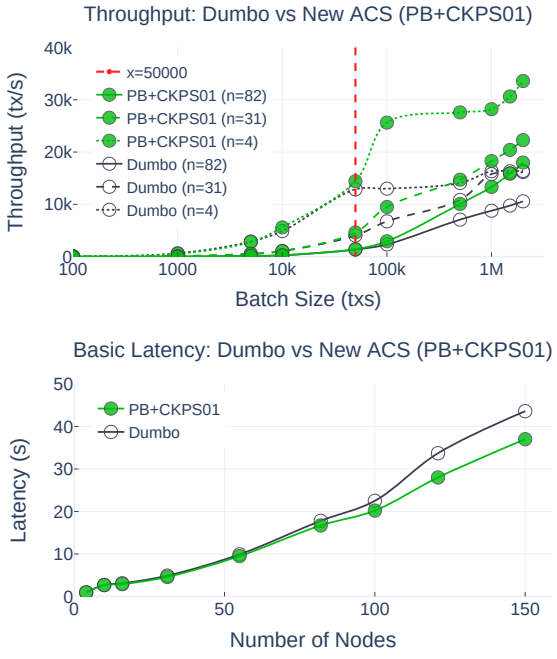

Fig. 6: New ACS (PB+CKPS01) v.s. Dumbo (RBC+CKPS01)

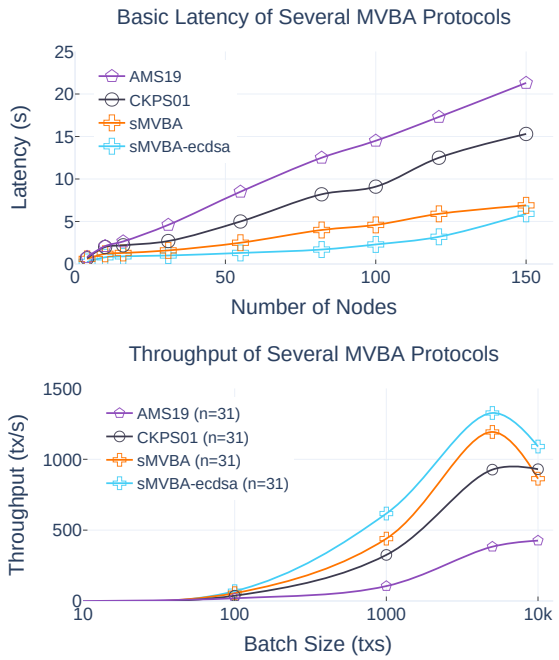

Fig. 7: Basic latency & throughput of several MVBA protocols

sMVBA v.s. State-of-the-art MVBAs. We also evaluated the performance of sMVBA and two other representative MVBA protocols, including CKPS01 [12] and AMS19 [4]. Considering that MVBA is mainly used in ACS to agree on a small vector of index-certificate pairs in our context, we set its input to one transaction, thus testing the basic latency. In Fig. 7, the three solid lines represent AMS19 [4], CKPS01 [12] and our sMVBA, respectively, from top (i.e., high latency) to bottom (i.e., low latency). Clearly, sMVBA is much faster, which is consistent to our analysis about the concrete communication rounds of these MVBA protocols.

Fig. 7 also plots the throughput of *directly* running MVBAs to agree on transactions (for $n = 31$). Our sMVBA protocol has the best throughput performance among the tested MVBA protocols, which reflects the reduced latency of sMVBA from another perspective. However, the peak throughput of directly running MVBAs is only about 1k-1.5k tx/s. Such inferior throughput is expected because of the high communication complexity of MVBAs (pointed out in HoneyBadgerBFT [33]). Thus, better reductions to MVBAs such as sDumbo become needed so that MVBA can be invoked only with small inputs.

Remarkably, if we implement quorum proofs by ECDSA,[12] the performance of sMVBA can be further improved. This is because the computation efficiency of ECDSA is superior to that of Boldyreva's threshold signature (e.g., a few microseconds v.s a few milliseconds with a t2.medium instance). Therefore, even if ECDSA signatures cannot be aggregated, using them to implement the quorum proofs still achieves considerable improvement in latency (for the tested system scales not larger than 150 nodes). Taking the observation, we would stick with ECDSA later for the quorum proofs in the sDumbo implementation, while still using Boldyreva's threshold signature for implementing common coin and random leader election.

[12]We trivially concatenate $2f + 1$ ECDSA signatures.

B. *Performance of Speeding Dumbo*

Putting our new ACS framework and sMVBA together, we have our Speeding Dumbo (sDumbo). We extensively evaluate it and compared it with Dumbo and HotStuff.

Basic latency. We set the input batch size as 1 transaction (i.e., nearly zero) to test the basic latency of sDumbo (and Dumbo) and show the results in Fig. 9. Referring to the results in Fig. 7, it is easy to see that the improvement is consistent with the benefits of sMVBA. Though the basic latency of both Dumbo and sDumbo is increasing with larger system scales, the improvement made by sDumbo is always significant. Especially when $n < 100$, the basic delay of sDumbo is about only half of that of Dumbo.

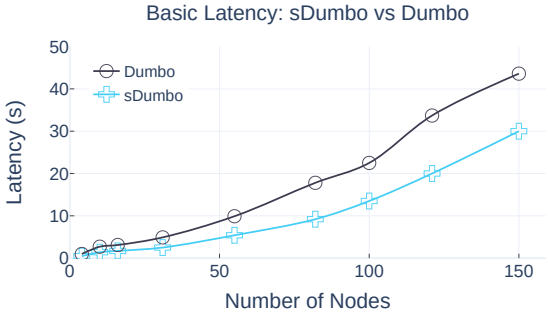

Fig. 8: Basic latency of sDumbo and Dumbo

Throughput-latency trade-off. Fig. 9 shows the throughput-latency trade-offs in sDumbo, Dumbo, and HotStuff. All the trends are roughly L-shaped, indicating that all of them eventually become network-bound. sDumbo not only achieves higher peak throughput, but also has a latency that is always smaller than that of Dumbo at the same throughput. Moreover, the latency gap between sDumbo and HotStuff is greatly narrowed at some small scales, which was much larger when comparing Dumbo with HotStuff. This evidence indicates that our multi-fold improvements greatly expand the applicability of asynchronous BFT protocol.

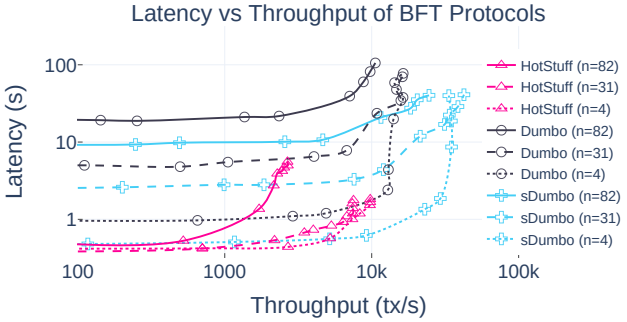

Fig. 9: Latency vs Throughput of several BFT protocols

Throughput at varying scales and batch sizes. We then evaluate the throughput of sDumbo at different system scales with varying batch sizes. Along the way, we compare sDumbo to Dumbo and HotStuff, and plot the comparison in Fig. 10.

As illustrated in the first three subgraphs in Fig. 10, sDumbo attains multi-fold improvements relative to Dumbo, disregarding the system scale and batch size. First, sDumbo directly inherits the advantages of using PBs, which helps it

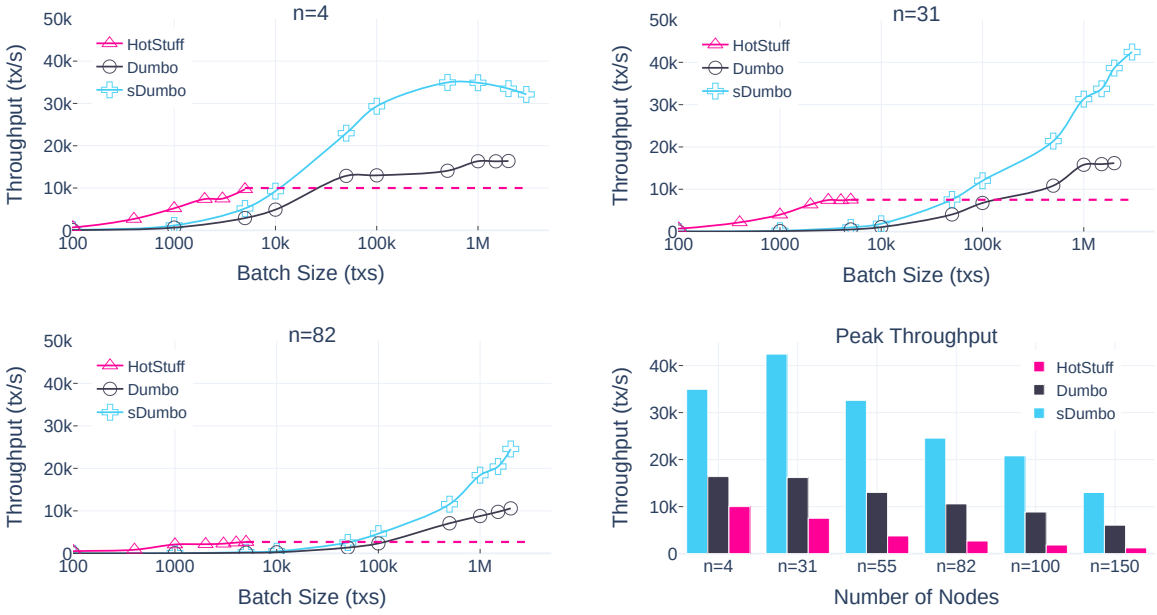

Fig. 10: Throughput of several BFT protocols

finally achieve a throughput twice as Dumbo's in the larger batch sizes. Moreover, carefully comparing Fig. 10 with Fig. 6, we notice that sDumbo noticeably outperforms Dumbo after the batch size reaches 10,000 (about 50,000 in Fig. 6), demonstrating that sMVBA further reduces the cost of the entire BFT protocol. HotStuff is unsurprisingly faster than both asynchronous protocols, if the system load is very small. But with larger batch size, HotStuff soon reaches its throughput peak, which is much smaller than sDumbo's or Dumbo's. This is in line with our expectation: HotStuff is naturally faster because it is a simple deterministic protocol, but its peak throughput is seriously restricted by the leader's bandwidth.

The peak throughputs of all BFT protocols are comprehensively shown in the last subgraph of Fig. 10. Although the throughput of all protocols decreases with scaling up, sDumbo always maintains the highest peak throughput among them.

VII. Conclusion and Future Work

We propose Speeding Dumbo (sDumbo) to continue pushing forward the performance of asynchronous BFT consensus protocols: we first reduce the message complexity to optimal; together with the recent major progresses [26, 33], we have an asynchronous consensus protocol that is asymptotically optimal for all major metrics including round complexity, communication complexity (when the batch size is moderate) and message complexity. We then design a *compact* MVBA protocol (dubbed sMVBA, that can be of independent interests and use), s.t. its concrete round complexity is reduced from multiple dozens to a dozen or fewer. Extensive experiments in the WAN setting demonstrate that sDumbo doubles the throughput and halves the latency in comparison to the state-of-the-art asynchronous BFT consensus Dumbo [26].

Nevertheless, many questions in the area remain to be explored. For example, like existing performant asynchronous BFT [26, 33], we also rely on costly threshold encryption to mitigate the censorship attack targeting on certain transactions. Can we resolve the threat with minimum cost? It is also challenging to evaluate the *worst-case* performance of asynchronous BFT in the adversarial network environment. Can we create more versatile benchmarks to capture the network fluctuations and attacks in the global Internet, thus telling the robustness of existing asynchronous BFT protocols?

Acknowledgment

The authors would like to thank our shepherd Kapil Singh and the anonymous reviewers for their valuable comments. Bingyong and Jing were supported in part by the National Key R&D Program of China (No. 2020YFB1005801), and the National Natural Science Foundation of China (No. 62172396). Yuan is supported in part by the National Natural Science Foundation of China (No. 62102404).

References

[1] I. Abraham, D. Dolev, and J. Y. Halpern, "An almost-surely terminating polynomial protocol for asynchronous byzantine agreement with optimal resilience," in *Proceedings of the twenty-seventh ACM symposium on principles of distributed computing*, 2008, pp. 405–414.

[2] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu, "Reaching consensus for asynchronous distributed key generation," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, 2021, p. 363âĂŞ373.

[3] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync hotstuff: Simple and practical synchronous state machine replication," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 106–118.

[4] I. Abraham, D. Malkhi, and A. Spiegelman, "Asymptotically optimal validated asynchronous byzantine agreement," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 337–346.

[5] J. Baek and Y. Zheng, "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, 2003, pp. 1491–1495.

[6] M. Ben-Or, "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, 1983, pp. 27–30.

[7] M. Ben-Or and R. El-Yaniv, "Resilient-optimal interactive consistency in constant time," *Distributed Computing*, vol. 16, no. 4, pp. 249–262, 2003.

[8] M. Ben-Or, B. Kelmer, and T. Rabin, "Asynchronous secure computations with optimal resilience," in *Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing*. ACM, 1994, pp. 183–192.

[9] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *International Workshop on Public Key Cryptography*. Springer, 2003, pp. 31–46.

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

[11] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, 2014.

[12] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Annual International Cryptology Conference*. Springer, 2001, pp. 524–541.

[13] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: practical asynchronous byzantine agreement using cryptography," in *19th Annual ACM Symposium on Principles of Distributed Computing*, 2000.

[14] C. Cachin and S. Tessaro, "Asynchronous verifiable information dispersal," in *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*. IEEE, 2005, pp. 191–201.

[15] C. Cachin and M. Vukolic, "Blockchain consensus protocols in the wild (keynote talk)," in *Proc. DISC 2017*.

[16] R. Canetti and T. Rabin, "Fast asynchronous byzantine agreement with optimal resilience," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, pp. 42–51.

[17] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[18] M. Correia, N. F. Neves, and P. Veríssimo, "From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures," *The Computer Journal*, vol. 49, no. 1, pp. 82–96, 2006.

[19] S. Das, Z. Xiang, and L. Ren, "Asynchronous data dissemination and its applications," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[20] D. Dolev and R. Reischuk, "Bounds on information exchange for byzantine agreement," *Journal of the ACM (JACM)*, vol. 32, no. 1, pp. 191–204, 1985.

[21] S. Duan, M. K. Reiter, and H. Zhang, "Beat: Asynchronous bft made practical," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 2028–2041.

[22] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process." Massachusetts Inst of Tech Cambridge lab for Computer Science, Tech. Rep., 1982.

[23] G. Bracha, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[24] A. Gągol, D. Leśniak, D. Straszak, and M. Świętek, "Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 214–228.

[25] R. Gelashvili, L. Kokoris-Kogias, A. Sonnino, A. Spiegelman, and Z. Xiang, "Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback," *arXiv preprint arXiv:2106.10362*, 2021.

[26] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo: Faster asynchronous bft protocols," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 803–818.

[27] A. Kate, Y. Huang, and I. Goldberg, "Distributed key generation in the wild," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 377, 2012.

[28] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is dag," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, 2021, pp. 165–175.

[29] E. Kokoris Kogias, D. Malkhi, and A. Spiegelman, "Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures." in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1751–1767.

[30] K. Kursawe and V. Shoup, "Optimistic asynchronous atomic broadcast," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2005, pp. 204–215.

[31] Y. Lu, Z. Lu, and Q. Tang, "Bolt-dumbo transformer: Asynchronous consensus as fast as pipelined bft," *arXiv preprint arXiv:2103.09425*, 2021.

[32] Y. Lu, Z. Lu, Q. Tang, and G. Wang, "Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited," in *Proceedings of the 39th Symposium on Principles of Distributed Computing*, 2020, pp. 129–138.

[33] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 31–42.

[34] A. Mostéfaoui, H. Moumen, and M. Raynal, "Signature-free asynchronous binary byzantine consensus with t< n/3, o (n2) messages, and o (1) expected time," *Journal of the ACM (JACM)*, vol. 62, no. 4, p. 31, 2015.

[35] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[36] R. Pass and E. Shi, "Rethinking large-scale consensus," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, 2017, pp. 115–129.

[37] Pass, Rafael and Shi, Elaine, "Thunderella: Blockchains with optimistic instant confirmation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 3–33.

[38] A. Patra, A. Choudhary, and C. Pandu Rangan, "Simple and efficient asynchronous byzantine agreement with optimal resilience," in *Proceedings of the 28th ACM symposium on Principles of distributed computing*, 2009, pp. 92–101.

[39] A. Patra and C. P. Rangan, "Communication optimal multi-valued asynchronous byzantine agreement with optimal resilience," in *International Conference on Information Theoretic Security*, 2011, pp. 206–226.

[40] M. O. Rabin, "Randomized byzantine generals," in *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*. IEEE, 1983, pp. 403–409.

[41] H. V. Ramasamy and C. Cachin, "Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast," in *International Conference On Principles Of Distributed Systems*. Springer, 2005, pp. 88–102.

[42] M. K. Reiter, "Secure agreement protocols: Reliable and atomic group multicast in rampart," in *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, 1994, pp. 68–80.

[43] N. Shrestha, I. Abraham, L. Ren, and K. Nayak, "On the optimality of optimistic responsiveness," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 839–857.

[44] L. Yang, S. J. Park, M. Alizadeh, S. Kannan, and D. Tse, "Dispersedledger: High-throughput byzantine consensus on variable bandwidth networks," *arXiv preprint arXiv:2110.04371*, 2021.

[45] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus in the lens of blockchain," *arXiv preprint arXiv:1803.05069*, 2018.

APPENDIX A
SIMPLE CONVERSIONS FROM ACS TO ASYNCHRONOUS ATOMIC BROADCAST

Given ACS, there are several simple ways to asynchronous atomic broadcast. Here we introduce a couple of them:

- The most simple conversion from ACS to atomic broadcast can be found in the seminal work of Cachin et al. [12]. That is to sequentially execute ACS instances: in each ACS, a node simply proposes the transactions in front of its transaction pool (a.k.a., transaction backlog or buffer) as ACS input, and then every node removes the ACS output from its local transaction pool and repeats the process in the next ACS instance. In worst case, the nodes might propose redundant transactions to ACS, such that the ACS output probably can only cover a small number of transactions instead of a large batch.
- Observing the redundant communications in Cachin et al.'s approach, Miller et al. [33] proposed a way of using ACS for batching consensus at the help of threshold encryption. The main idea is still to sequentially execute ACS instances, but it also adds threshold encryption (resp. threshold decryption) before (resp. after) the ACS instance. Now, instead of deterministically proposing the transactions in front of transaction pool as ACS input, each node now randomly choose $1/n$ transactions out of B transactions in front of backlog (where $B > n$ is a batch size parameter), and uses threshold encryption to encrypt the chosen transactions, takes the ciphertext as ACS input, after the ACS output a set of ciphertexts, all nodes collectively perform threshold decryption to decrypt the ACS output to form a block to output in atomic broadcast. This would minimize the chance that different nodes propose overlapped inputs in ACS, thus improving the communication efficiency of the reduction from atomic broadcast to ACS by an (n) order while preserving the critical liveness.

We adopt the latter conversion from Miller et al. to construct asynchronous atomic broadcast throughout the paper, because of its asymptotically better communication efficiency. Nevertheless, once some novel reductions from atomic broadcast to ACS (or MVBA) can be invented with demonstrated security and efficiency (e.g., [44]), we can naturally instantiate them with using our ACS (or MVBA) result.

APPENDIX B
ADAPTING DOLEV-REISCHUK BOUND TO DETERMINISTIC RELIABLE BROADCAST

Recall that the properties of reliable broadcast are agreement, totality and validity. Here agreement means that if any two honest nodes output, they must output the same value; totality requires that if any honest node outputs, all honest nodes would output; validity ensures that if the sender is honest, then all honest nodes would output. The latter two properties are "eventual" as they are conditioned on that the adversary eventually delivers all messages sent among honest nodes. Here at a high level, we explain how to slightly adapt the proof of Dolev-Reischuk bound [20] and thus prove the $\mathcal{O}(f^2)$ message lower-bound of deterministic reliable broadcasts with respect to any adversary controlling up to f nodes.

First consider the next execution. The adversary controls a set of $f/2$ nodes (excluding the sender). Let V denote these $f/2$ corrupted nodes and U denote the else honest nodes. The adversary specifies the corrupted nodes in V to behave nearly same to be honest, but with only two exceptions: (i) the corrupted nodes ignore the earliest $f/2$ messages sent to them; (ii) they stop sending messages to each other. Since the sender is not corrupted, for validity and agreement to hold, all nodes in U must output the value same to the sender's input. If only $(f/2)^2$ or less messages are sent to the nodes in V, then we can assert that at least one node $\mathcal{P}_x$ in V receives $f/2$ or less messages.

Then consider the following slightly different execution. The adversary makes some minor changes to her strategy, in particular, it no longer corrupts $\mathcal{P}_x$ but instead corrupts all nodes that send messages to $\mathcal{P}_x$ in the former execution. Note that these nodes send messages to $\mathcal{P}_x$ in the former execution are all in the set U and their number is at most $f/2$, so the adversary controls at most f nodes in the current execution. Now the adversary prevents the newly corrupted nodes in U from sending messages to $\mathcal{P}_x$.

Observe that there exists some node $\mathcal{P}_y$ in the set U that is honest in both executions, and $\mathcal{P}_y$ must receive the same messages and thus output the same value in the two executions. For totality and agreement to hold, all honest nodes in the latter execution, including $\mathcal{P}_x$, must output the value same to $\mathcal{P}_y$'s output (because $\mathcal{P}_y$ outputs). But $\mathcal{P}_x$ unfortunately receives no message in the second execution, causing it impossible to remain agreement anyhow, because the output of other honest nodes can always be opposite to its. As such, it is impossible to guarantee all three properties of a deterministic reliable broadcast protocol against a byzantine adversary controlling up to f nodes, if the number of sent messages is at most $(f/2)^2$. I.e., when f is optimal as equal to $n/3$, any deterministic reliable broadcast needs $\mathcal{O}(n^2)$ messages to be secure.

APPENDIX C
NON-INTERACTIVE THRESHOLD SIGNATURE

Non-interactive (n, t) **threshold signature (**TSIG**)** is a tuple consisting of the following algorithms or protocols:

- *Key generation:* $\mathsf{KeyGen}(1^\lambda, n, t) \rightarrow \{mpk, PK, SK\}$. Given a security parameter λ, the algorithm or protocol generates a master 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:* $\mathsf{SignShare}_t(sk_i, m) \rightarrow \sigma_i$. On input a message m and a secret key share sk_i, this deterministic algorithm outputs a signature share σ_i;
- *Share verification:* $\mathsf{VerifyShare}_t(m, (i, \sigma_i)) \rightarrow 0/1$. Given a message m, a signature share σ_i, an index i and the implicit mpk and PK, this deterministic algorithm outputs 1 if σ_i is correctly computed by $\mathsf{SignShare}_t(sk_i, m)$, otherwise, it outputs 0;
- *Combining algorithm:* $\mathsf{Combine}_t(m, \{(i, \sigma_i)\}_{i\in S}) \rightarrow \sigma/\perp$. Given 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 $\perp$ only if $\{(i, \sigma_i)\}_{i \in S}$ contains any invalid signature share (i, σ_i) for m (that cannot pass share verification);

- *Signature verification:* $\mathsf{VerifyThld}_t(m, \sigma) \rightarrow 0/1$. Given a message m, a signature σ and the implicit mpk and PK, this algorithms outputs 1 if σ is a valid signature for m, otherwise, it outputs 0.

The (n, t) threshold signature satisfies the next properties:

- *Unforgeability:* For any message m, polynomial-time adversary can forge a signature that can be verified (by honest nodes), unless $t - f$ honest nodes send their signature shares for m to the adversary, where f represents the number of nodes corrupted by the adversary;
- *Robustness:* When a message m is provided as the input to sign, it is infeasible for the adversary to generate t valid signature shares such that the combining algorithm cannot aggregate them to form a valid full signature.

APPENDIX D
DEFERRED PROOFS FOR THE NEW ACS FRAMEWORK

Here down below are the deferred proofs for the securities of our new ACS framework presented in Algorithm 1.

Proof outline: To prove *Termination*, we need to ensure that all honest nodes would not get stuck in any phase of the protocol, namely, all honest nodes can enter and then exit MVBA (Lemma 1) and all honest nodes can base on the MVBA output to recover any broadcasted values that shall be output but they do not receive (Lemma 3), thus always making the output. For *Agreement*, we first prove that all nodes get the same output after the MVBA (Lemma 2), and then show that all honest nodes would receive the same broadcasted value corresponding to each PB, no matter they directly receive the value from PB or recover the value via the recovery phase (Lemma 4). For *Validity*, the external validity of MVBA and the provability of PB ensure that the final output can solicit at least $n - 2f$ honest nodes' input values.

Remark that the following lemmas might be conditioned on secure MVBA and PB, collision-resistant hash and $(f + 1, n)$ erasure-coding, and hold with all but negligible probability.[13] We might omit these conditions in the lemmas for brevity.

Lemma 1: All honest nodes would invoke the MVBA instance, and then get some output $\overline{W}$ from MVBA.

Proof: There are at least $n - f$ honest nodes, each of which can eventually complete a PB as the sender and then multicast the corresponding $lock$ proof. That means, all honest node can at least receive $n - f$ valid $lock$ proofs multicasted by these honest nodes, thus invoking the MVBA instance with valid inputs, otherwise, the termination property of PB would be violated. After all honest nodes invoke MVBA with valid inputs, they all will get the output $\overline{W}$ of MVBA, otherwise, the termination property of MVBA would be violated. ■

Lemma 2: All honest nodes would get the same $\overline{W}$ output by MVBA s.t. $\mathsf{VerifyThld}_{(2f+1)}(\langle \mathsf{ID}, \ell \rangle, h_\ell, \bar{\sigma}_\ell) = 1$ for each $(\bar{s}_\ell, \bar{h}_\ell, \bar{\sigma}_\ell) \in \overline{W}$.

Proof: The output $\overline{W}$ of any two nodes is the same because of the agreement of MVBA. Due to Lemma 1, all nodes output some $\overline{W}$. So all honest nodes would output the same $\overline{W}$, otherwise, either Lemma 1 or the agreement of MVBA is violated. Moreover, every element in the unique output $\overline{W}$ satisfies $\mathsf{VerifyThld}_{(2f+1)}(\langle \mathsf{ID}, \ell \rangle, h_\ell, \bar{\sigma}_\ell) = 1$, because the external validity condition of MVBA is specified to reject any $\overline{W}$ that does not satisfy this. ■

Lemma 3: If an honest node does not get the output value of $\mathsf{PB}[\langle \mathsf{ID}, \ell \rangle]$ when MVBA returns $\overline{W}$ containing $(\bar{s}_\ell, \bar{h}_\ell, \bar{\sigma}_\ell)$, the node still can recover the output value v_ℓ of $\mathsf{PB}[\langle \mathsf{ID}, \ell \rangle]$ via the recovery phase s.t. $\mathcal{H}(v_\ell) = \bar{h}_\ell$.

Proof: Due to Lemma 2, every $(\bar{s}_\ell, \bar{h}_\ell, \bar{\sigma}_\ell) \in \overline{W}$ satisfies $\mathsf{VerifyThld}_{(2f+1)}(\langle \mathsf{ID}, \ell \rangle, h_\ell, \bar{\sigma}_\ell) = 1$. So even if some honest node does not get the output value of $\mathsf{PB}[\langle \mathsf{ID}, \ell \rangle]$, at least $f + 1$ honest nodes shall already receive the same v_ℓ s.t. $\mathcal{H}(v_\ell) = \bar{h}_\ell$, otherwise, the provability of PB is violated. As such, the honest node can at least receive $f + 1$ valid $(H^\ell, b_j^\ell, m_j^\ell)$ carrying the same Merkle root H^ℓ, because the honest nodes holding v_ℓ will send them through the Help messages during the recovery phase. Then, the node can recover a value v'_ℓ corresponding to $\mathsf{PB}[\langle \mathsf{ID}, \ell \rangle]$ from these $f + 1$ valid $(H^\ell, b_j^\ell, m_j^\ell)$. Moreover, the Merkle root H^ℓ must be computed by some honest nodes (because at most f nodes are corrupted), so $v'_\ell = v_\ell$ due to the collision resistance of Merkle tree and the correctness of erasure code. ■

Lemma 4: For any $(\bar{s}_\ell, \bar{h}_\ell, \bar{\sigma}_\ell) \in \overline{W}$ output by MVBA, any two honest nodes can receive the same corresponding value v_ℓ (either directly from $\mathsf{PB}[\langle \mathsf{ID}, \ell \rangle]$ or from the recovery phase).

Proof: Lemma 2 states that each $(\bar{s}_\ell, \bar{h}_\ell, \bar{\sigma}_\ell) \in \overline{W}$ satisfying $\mathsf{VerifyThld}_{(2f+1)}(\langle \mathsf{ID}, \ell \rangle, h_\ell, \bar{\sigma}_\ell) = 1$, so at least $f + 1$ honest nodes can directly get the corresponding value v_ℓ from the $\mathsf{PB}[\langle \mathsf{ID}, \ell \rangle]$ instance. For any other honest nodes that do not receive v_ℓ from $\mathsf{PB}[\langle \mathsf{ID}, \ell \rangle]$, Lemma 3 states that they can recover the same v_ℓ. So no matter the honest nodes receive v_ℓ in which manner, v_ℓ is unique. ■

Theorem 1: The ACS protocol presented in Algorithms 1 satisfies *Termination*, *Agreement* and *Validity*, conditioned on secure MVBA and PB, collision-resistant hash function, and $(f + 1, n)$ erasure-code.

Proof: Termination is immediate from Lemma 1 and 3. Because Lemma 1 states that all honest nodes would get the output of MVBA, and Lemma 3 states that every honest node can receive the broadcasted values corresponding to all elements in $\overline{W}$, and thus must terminate.

Agreement is immediate from Lemma 2 and 4. Because Lemma 2 states that all honest nodes get the same $\overline{W}$, and Lemma 4 states that for every element in $\overline{W}$, all honest nodes can get the same value in their final output set.

Validity is immediate from the external validity of MVBA and the properties of PB. Because the external validity condition of MVBA is specified to pick a $\overline{W}$ vector containing at

[13] In a $(f + 1, n)$ erasure-coding scheme, a value can be encoded into n fragments in a deterministic manner, and it later can be recovered from any $f + 1$ fragments. The MVBA and PB protocols used in the paper also rely on secure non-interactive threshold signature.

least $(n-f)$ valid PB *lock* proofs. Out of the $(n-f)$ *lock* proofs in $\overline{W}$, at least $n-2f$ are related to honest PB senders, so the final output is a superset of at least $n-2f$ honest nodes' input values. ■

APPENDIX E
DEFERRED PROOFS FOR SPEEDING MVBA

We now prove that the sMVBA protocol presented in Algorithms 3, 4 and 5 satisfies the *Termination*, *Agreement* and *Validity* properties of MVBA. Besides these standard MVBA properties proposed in [12], we will also prove that sMVBA satisfies an extra *Quality* property as a by-product. The *Quality* property was recently proposed in [4] and it ensures that the output is from honest nodes with a constant probability (e.g. 1/2), while MVBA without *Quality* might face an issue that the output is always proposed by some corrupted nodes.

Termination proof outline: First, as a basic preparation, we need to make sure all honest nodes do not get stuck in any phase of the protocol (Lemma 6). However, not being stuck is not enough for termination, it could be that all honest nodes repeat iterating in each view of the protocol but never output, so we further argue that: (1) If all honest nodes enter a view, all of them output in the view with a constant probability (Lemma 7, 8 show the protocol terminate in each view with probability $2/3$); (2) If no honest node outputs and halts in the current view R, we will further argue that all honest nodes have a *valid* value as input to enter next view to repeat the previous process (Lemma 6 and Corollary 1); (3) If not all honest nodes enter the current view R (e.g., some of them already output), we demonstrate that all honest nodes still can eventually output (i.e., Case-II of Theorem 2 shows all other honest nodes can terminate once receiving some valid Halt message).

Put these together, the protocol can make sure terminate with expected constant views. Below are the detailed proofs.

Lemma 5: If all honest nodes start the sMVBA with externally valid values in view R, then: if an honest node invokes Election$[\langle \mathsf{id}, \mathsf{R}\rangle]$, then at least $2f+1$ SPB$[\langle \mathsf{id}, \mathsf{R}, j\rangle]$ instances have completed (i.e., $2f+1$ or more SPB instances have sent valid Lock proofs to at least $f+1$ honest nodes), assuming the SPB protocol is secure.

Proof: Suppose an honest node $\mathcal{P}_i$ invokes Election$[\langle \mathsf{id}, \mathsf{R}\rangle]$, i.e., mulitcast a SHARE message, then it means that (1) $\mathcal{P}_i$ receives $2f+1$ valid FIN messages from distinct nodes, or (2) $\mathcal{P}_i$ receives $f+1$ Done messages from distinct nodes.

For case 1, since $\mathcal{P}_i$ already receives $2f+1$ distinct valid FIN messages that contain $2f+1$ distinct valid Finish. Considering the provability of SPB, at least $2f+1$ SPB's Lock proofs have reached at sufficient $f+1$ honest nodes, otherwise, the provability of at least one SPB instance is violated.

For case 2, $\mathcal{P}_i$ receives $f+1$ Done messages, only if at least one Done message is sent by some honest node $\mathcal{P}_j$ (as the adversary corrupts at most f nodes). This indicates $\mathcal{P}_j$ either has received $2f+1$ distinct valid FIN messages or has received $f+1$ distinct Done messages. If $\mathcal{P}_j$ receives $2f+1$ valid FIN, it is reduced to case 1; otherwise, it is another case 2, while we can repeatedly reduce the case, until reducing it to that the first honest nodes multicasts a valid Done message, i.e., making it completely reduced to case 1. ■

Lemma 6: Assuming that the underlying threshold signature, SPB and Election are all secure, then: if all honest nodes start the sMVBA with externally valid values in view R, then no honest nodes will be stuck in any phase of current view R (in case the short-cut output condition in line 16 Algorithm 5 is not added); besides, if no honest node outputs and halts in current view R, then all honest nodes will also have externally valid value at next view R+1.

Proof: Assuming that all honest nodes start the sMVBA with externally valid values in view R, no honest node has output and halted before view R, and each of them would invoke a SPB instance with valid input.

So we consider the following two cases in the current view R: (1) if no honest node abandons SPB, from the *termination* of SPB, at least $n-f$ honest nodes can receive the valid Finish from their SPB instances and then multicast the Finish to all honest nodes, hence all honest nodes can enter the election phase to multicast a Done message and invoke the Election protocol; (2) if some honest node abandons the SPB instances, the node must receive $2f+1$ valid Done messages from distinct nodes, which can attest at least $f+1$ honest nodes multicast the Done message, by the code, all honest nodes can receive at least $f+1$ Done messages and enter the election phase to multicast a Done message and invoke the Election protocol. As such, all nodes would multicast Done message and invoke leader election in either case.

Hence, all honest nodes will invoke Election$[\langle \mathsf{id}, \mathsf{R}\rangle]$. Due to the *termination* of Election, all honest nodes can exit Election phase and then enter the PreVote and Vote phase. Then, since the short-cut output condition is turned off, all honest nodes would multicast a valid PreVote message, and therefore every honest node can receive at least $2f+1$ valid PreVote messages: if all received PreVote messages are valid PreVote-No, the honest node can aggregate them due to the robustness of threshold signature, and then multicast Vote-No; else, the honest node simply multicasts Vote-Yes. So every honest node can receive at least $2f+1$ valid Vote messages, and again: if all received Vote messages are valid Vote-No, the honest node can aggregate them due to the robustness of threshold signature, and take the aggregated signature to the next view to form a valid input regarding SPB's external predicate in the next view; else, it either outputs, or receives a Vote-Yes message containing a proof that can be taken into the next view to form a valid SPB input. ■

It is immediate to have the next corollary from Lemma 6.

Corollary 1: Suppose all honest nodes start the sMVBA protocol with externally valid values, and no honest node outputs and halts after the view R, then all honest nodes will also have externally valid value at view R+1 for any $\mathsf{R} \geq 1$, assuming the underlying threshold signature, SPB and Election are secure.

Lemma 7: Suppose all honest nodes participate in some view R, at the time when someone learns the view's leader $\ell = $ Election$[\langle \mathsf{id}, \mathsf{R}\rangle]$, if a valid Finish $:= (v_\ell, \sigma_2)$ satisfying VerifyThld$_{(2f+1)}(\langle\langle\langle \mathsf{id}, \mathsf{R}, \ell\rangle, 2\rangle, \mathcal{H}(v_\ell)\rangle, \sigma_2) = 1$ has been produced, then all honest nodes will output the same value v_ℓ and halt in view R (in case of no short-cut added), assuming the underlying threshold signature, SPB and Election are secure.

Proof: Since the threshold of Election is $2f+1$, if Election returns ℓ, there must be at least $f+1$ honest nodes have already invoked leader election with multicasting a Done message in the current view R (thus all honest nodes will participate in the Election), otherwise, the unpredictability of leader election is violated. At the moment when Election returns ℓ, if someone can produce a valid Finish $:= (v_\ell, \sigma_2)$, it implies that at least $f+1$ honest nodes has already delivered a valid Lock $:= (v_\ell, \sigma_1)$ message, otherwise, the *provability* of SPB is violated.

Hence, at least $f+1$ honest nodes, who have received the valid Lock $:= (v_\ell, \sigma_1)$, will multicast valid $(\mathsf{PreVote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes}, v_\ell, \sigma_1)$ message. So once any node solicits $2f+1$ PreVote messages, there must contain a valid $(\mathsf{PreVote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes}, v_\ell, \sigma_1)$ message. Thus, all honest nodes will multicast the $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$ message.

Due to the threshold of σ_{PN} is $2f+1$, in order to generate a valid σ_{PN} to form valid $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{No})$, each node needs to receive $2f+1$ valid $(\mathsf{PreVote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{No})$ messages from distinct nodes. So it is computationally infeasible for the adversary to generate valid $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{No})$, because at least $f+1$ honest nodes send $(\mathsf{PreVote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$. So $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$ is the only computable Vote message.

Moreover (when no short-cut is added), all honest nodes will multicast $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$ message, and thus all of them can receive at least $2f+1$ $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$ messages without receiving any valid $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{No})$. So all honest nodes will output value v_ℓ and halt in view R. ■

Lemma 8: Lemma 7 also holds if the short-cut (i.e. the output condition in line 16 Algorithm 5) is added.

Proof: While proving Lemma 7, we see that if someone can produce a valid Finish $:= (v_\ell, \sigma_2)$ associated to the elected SPB, $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$ is the only valid form of Vote in the current view R. We therefore add a short-cut to enable faster output in this case. I.e., if any honest node $\mathcal{P}$ receives the valid Finish, it can skip the PreVote and Vote phases, and multicasts a Halt message carrying the Finish, then immediately outputs with halt; other honest nodes either output the same value in the current view R if receiving $2f+1$ $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$, or they at worst cannot get enough $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$. In the latter worse case, recall that $\mathcal{P}$ already multicasts the Halt message carrying Finish, the honest nodes still can receive Finish and then output the value associated to the selected SPB in view R according to Finish instead of according to $2f+1$ $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$. ■

Theorem 2: **Termination**. If all honest nodes take some values that are externally valid due to Q, then each honest node would output a value, assuming that the underlying threshold signature, SPB and Election are secure.

Proof: We prove that the theorem is true in the following two cases, respectively.

Case 1. Suppose no honest node outputs and halts before view R, since all honest nodes start the sMVBA with externally valid values, according to Corollary 1, all honest nodes still have externally valid values in current view R, then from Lemma 6, no honest nodes will be stuck in any phase of cueernt view R.

Due to any honest node will not be stuck in any phase of current view R, so each nodes can invoke the $\mathsf{Election}[\langle \mathsf{id}, \mathsf{R} \rangle]$, from Lemma 5, at least $n-f$ SPB instances have completed, suppose $\mathsf{Election}[\langle \mathsf{id}, \mathsf{R} \rangle]$ returns ℓ, then the probability that $\mathsf{SPB}[\langle \mathsf{id}, \mathsf{R}, \ell \rangle]$ have completed is $p = 2/3$, which also means the $\mathsf{SPB}[\langle \mathsf{id}, \mathsf{R}, \ell \rangle]$ produces a valid Finish $:= (v_\ell, \sigma_2)$, it also implies that with probability $2/3$, all honest nodes will output value v_ℓ and halt in current view R according to Lemma 8.

Now, we prove that the protocol terminates after sequentially running in continuous views. Let the event E_k represents that the protocol does not terminate when $\mathsf{Election}[\langle \mathsf{id}, k \rangle]$ has been invoked, so the probability of the event E_k is less than $(1-p)^k$. It is clear that $\lim_{k\to\infty} \Pr[E_k] \leq \lim_{k\to\infty}(1-p)^k = 0$, so the protocol eventually halts. Moreover, let K be the random variable that the protocol just terminates when $k = K$, so $E[K] \leq \sum_{K=1}^{\infty} K(1-p)^{K-1}p = 1/p = 3/2$, indicating the protocol terminates in expected constant time.

Case 2. Suppose some honest node $\mathcal{P}$ outputs and halts before view R. By line 26 in Algorithm 3 and lines 16 and 22 in Algorithm 5, it must multicast a valid Halt message. Due to lines 18-22 in Algorithm 5, any honest node can immediately output, once it receives the valid Halt message sent by $\mathcal{P}$. So in this case, if some honest nodes get stuck in the next view $\mathsf{R}' > \mathsf{R}$, the Halt message sent by $\mathcal{P}$ still can be eventually delivered and thus ensure all honest nodes to output. ■

Agreement proof outline: The agreement needs to ensure that any two honest nodes $\mathcal{P}_i$ and $\mathcal{P}_j$ output the same. (1) We first prove that if some $\mathcal{P}_i$ node outputs in view R after receiving a valid Finish from the elected leader, then $\mathcal{P}_j$ must output the same value in the current view R: our two-phase Yes-No voting via threshold signature prevents the forgery of any fake Vote-No message, thus ensuring all nodes to receive sufficient Vote-Yes messages to output elected leader's SPB input in the current view R (Lemma 9). (2) Then, we prove if $\mathcal{P}_i$ outputs in view R because of receiving a valid Halt message ($2f+1$ valid $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$), then $\mathcal{P}_j$ either outputs the same value in the current view R or outputs the same value in some later view $\mathsf{R}' > \mathsf{R}$. This is because $\mathcal{P}_j$ must receive an unforgeable SPB proof to either output in the current view, or use as the next view's input, cf. Lemma 10 (resp. Lemma 11).

Lemma 9: Suppose all honest nodes participate in view R and $\mathsf{Election}[\langle \mathsf{id}, \mathsf{R} \rangle] = \ell$, if an honest node $\mathcal{P}_i$ has received a valid $(\mathsf{FIN}, \mathsf{id}, \mathsf{R}, \mathsf{Finish})$ message from the elected leader $\mathcal{P}_\ell$ when it learns ℓ, then all honest nodes output the same value v_ℓ in the current view R.

Proof: Since an honest node $\mathcal{P}_i$ has received a valid $(\mathsf{FIN}, \mathsf{id}, \mathsf{R}, \mathsf{Finish})$ message from the elected leader $\mathcal{P}_\ell$ when it learns the output of Election is ℓ and the threshold of Election is $2f+1$, then we can consider the following two cases:

Case 1. The elected node $\mathcal{P}_\ell$ has produced the valid Finish when $f+1$ honest nodes invoke leader election;

Case 2. The elected node $\mathcal{P}_\ell$ has not produced the valid Finish when $f+1$ honest nodes invoke ℓ (but it produces the valid Finish later).

Case 1 follows Lemma 8, because it states that if the valid Finish is produced when anyone learns ℓ, all honest nodes would output the same value in the current view.

For case 2, we would discuss by the next two situations: (2.1) at least $f+1$ honest nodes have received Lock proofs when $f+1$ honest nodes learn ℓ; (2.2) less than $f+1$ honest nodes have received Lock proofs when $f+1$ honest nodes learn ℓ. In case (2.1), at least $f+1$ honest nodes will multicast $(\mathsf{PreVote}, \mathsf{Yes})$, then it can make sure all honest nodes would multicast $(\mathsf{Vote}, \mathsf{Yes})$, so all honest nodes would output the same value and halt in current R, the analysis of which is similar to the proving for Lemma 7 and 8 and can be reduced to the unforgeability of threshold signature. In case (2.2), since an honest node abandons all SPB instances after it learns ℓ, so once $f+1$ honest nodes learn ℓ, then at least $f+1$ honest nodes have abandoned all SPB instances. Due to the abandonability of SPB, at most f honest nodes can receive valid Lock proof since then, and thus it becomes computationally infeasible for anyone to produce a valid Finish, otherwise, it breaks the provability of SPB to forge a Finish while only f honest parties receive valid Lock. In sum, case (2.2) has a negligible probability to occur, and case (2.1) ensures that all honest parties to output the same value in the current view. ■

Lemma 10: Suppose all honest nodes participate in view R and $\mathsf{Election}[\langle \mathsf{id}, \mathsf{R}\rangle] = \ell$, if an honest node $\mathcal{P}_i$ outputs v_ℓ in view R because of receiving $2f+1$ valid $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$ messages from distinct nodes, then any other honest node either outputs the same value v_ℓ in the current view R or takes value v_ℓ into view R' and v_ℓ is the unique valid SPB input in view R' for any $\mathsf{R}' > \mathsf{R}$, assuming the underlying threshold signature, SPB and Election are secure.

Proof: Since $\mathcal{P}_i$ receives $2f+1$ valid $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$ messages from distinct nodes, it means that at least $f+1$ honest nodes multicast valid $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$ message, hence, for any nodes, it is computationally infeasible to generate a valid σ_{VN} proof which is due to the threshold of σ_{VN} is $2f+1$, it also means any $(\mathsf{No}, \mathsf{R}, \sigma_{\mathsf{VN}})$ is invalid. Hence, for any other honest node $\mathcal{P}_j$, it will either (1) output in the same view, or (2) receive at least one valid $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes}, v_\ell, \sigma_1, \rho_{2,j})$ message out of $n-f$ received valid Vote messages and thus use v_ℓ and $\pi = \{(\mathsf{Yes}, \mathsf{R}, \sigma_1)\}$ to enter view $\mathsf{R}+1$, where $\mathsf{VerifyThld}_{(2f+1)}(\langle\langle\langle \mathsf{id}, \mathsf{R}, \ell\rangle, 1\rangle, \mathcal{H}(v_\ell)\rangle, \sigma_1) = 1$.

For case 1, suppose some node $\mathcal{P}_j$ also outputs in the same view, it then: (1.1) receives $2f+1$ valid $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \ell, \mathsf{Yes})$ messages, or (1.2) receives a valid $(\mathsf{FIN}, \mathsf{id}, \mathsf{R}, \mathsf{Finish})$ message, or (1.3) receives a valid $(\mathsf{Halt}, \mathsf{id}, \mathsf{R}, \mathsf{Finish})$ message. In case (1.1), $\mathcal{P}_j$ would output v_ℓ, otherwise there exists an honest node sends two different $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}, \mathsf{Yes})$ messages to $\mathcal{P}_i$ and $\mathcal{P}_j$. In case (1.2), Lemma 9 makes sure $\mathcal{P}_j$ would output the elected SPB's input value v_ℓ. In case (1.3), the valid $(\mathsf{Halt}, \mathsf{id}, \mathsf{R}, \mathsf{Finish})$ message contains a valid Finish proof regarding the elected SPB, and if $\mathcal{P}_j$ outputs a value different to the elected SPB's input, the provability of SPB is broken.

For case 2, suppose some node $\mathcal{P}_j$ takes value (v_j, π) to enter view $\mathsf{R}+1$, where $v_j \neq v_\ell$. From previous analysis we know: it doesn't exist any valid $(\mathsf{No}, \mathsf{R}, \sigma_{\mathsf{VN}})$ message, hence, $(\mathsf{Yes}, \mathsf{R}, \sigma_1') \in \pi$, it also implies that $|\pi| = 1$. If (v_j, π) is a valid value in view $\mathsf{R}+1$, then $\mathsf{CheckValue}(\langle \mathsf{id}, \mathsf{R}+1, j\rangle, v_j, \sigma_1') = 1$, it also means that $\mathsf{VerifyThld}_{(2f+1)}(\langle\langle\langle \mathsf{id}, \mathsf{R}, \ell\rangle, 1\rangle, \mathcal{H}(v_j)\rangle, \sigma_1') = 1$.

Since $\mathsf{Election}[\langle \mathsf{id}, \mathsf{R}\rangle] = \ell$ and $\mathsf{VerifyThld}_{(2f+1)}(\langle\langle\langle \mathsf{id}, \mathsf{R}, \ell\rangle, 1\rangle, \mathcal{H}(v_\ell)\rangle, \sigma_1)$=1, following *provability* of SPB, we have $v_j = v_\ell$ and v_ℓ is the unique valid value in the view $\mathsf{R}+1$.

Considering in view $\mathsf{R}+2$, some honest nodes $\mathcal{P}_i$ still have not output, and according to Algorithm 3 and 5, the valid π only have the next two possible cases:

Case 2.1. $\pi = \{(\mathsf{Yes}, \mathsf{R}, \sigma_1), (\mathsf{No}, \mathsf{R}+1, \sigma_{\mathsf{VN}_{\mathsf{R}+1}})\}$

Case 2.2. $\pi = \{(\mathsf{Yes}, \mathsf{R}+1, \sigma_1')\}$.

For case (2.1), due to $(\mathsf{Yes}, \mathsf{R}, \sigma_1) \in \pi$ and according to the Alg. 4 and previous analysis, the value v_ℓ is the unique value which satisfies $\mathsf{CheckValue}(\langle \mathsf{id}, \mathsf{R}+2, i\rangle, v_\ell, \pi) = 1$.

For case (2.2), suppose $\mathsf{Election}[\langle \mathsf{id}, \mathsf{R}+1\rangle] = \ell'$ in view $\mathsf{R}+1$. Due to $\pi = \{(\mathsf{Yes}, \mathsf{R}+1, \sigma_1')\}$, it implies that node $\mathcal{P}_i$ received a valid $(\mathsf{Vote}, \mathsf{id}, \mathsf{R}+1, \ell', \mathsf{Yes}, v_{\ell'}, \sigma_1', \rho_{2,j}')$ message, it also means that node $\mathcal{P}_{\ell'}$ produced a valid Lock message. Hence, $\mathcal{P}_{\ell'}$ has a valid input $v_{\ell'}$ in view $\mathsf{R}+1$. However, from previous analysis, we know the v_ℓ is the unique valid input value in view $\mathsf{R}+1$, hence, $v_\ell = v_{\ell'}$, so, the σ_1' is a proof corresponding to the value v_ℓ.

In sum, the value v_ℓ is the unique value in both cases which satisfies the CheckValue predicate. For any $\mathsf{R}' > \mathsf{R}+2$, a similar analysis can be done and prove v_ℓ is the unique valid value regarding the predicate of SPBs in view R'. ■

Lemma 11: Suppose all honest nodes participate in view R and $\mathsf{Election}[\langle \mathsf{id}, \mathsf{R}\rangle] = \ell$, if an honest node $\mathcal{P}_i$ outputs v_ℓ because of receiving a valid $(\mathsf{Halt}, \mathsf{id}, \mathsf{R}, \mathsf{Finish})$ message s.t. $\mathsf{Finish} := (v_\ell, \sigma_2)$, then all honest nodes either output the same value v_ℓ in current view R, or take value v_ℓ into view R' as v_ℓ is the unique valid SPB input in view R' for any $\mathsf{R}' > \mathsf{R}$, assuming the underlying threshold signature, SPB and Election are secure.

Proof: Since all honest nodes participate in view R and an honest node $\mathcal{P}_i$ receives a valid $(\mathsf{Halt}, \mathsf{id}, \mathsf{R}, \mathsf{Finish})$, then it means the Finish was produced in the current view R either (1) by the sender $\mathcal{P}_\ell$ or (2) by aggregating $2f+1$ valid Vote-Yes.

For case 1, we can know all honest nodes output the same value in current view by the same analysis with Lemma 9.

For case 2, at least $f+1$ honest nodes multicast valid Vote-Yes messages, so it is computationally infeasible to generate a valid σ_{VN} proof which is due to the threshold of σ_{VN} is $2f+1$, it also means any $(\mathsf{No}, \mathsf{R}, \sigma_{\mathsf{VN}})$ message is invalid. Then, similar to the proving to Lemma 10, the honest nodes either output the same value in the current view, or take value v_ℓ into view R' as v_ℓ is the unique valid SPB input in view R' for any $\mathsf{R}' > \mathsf{R}$. ■

Lemma 12: Suppose all honest nodes participate in view R and $\mathsf{Election}[\langle \mathsf{id}, \mathsf{R}\rangle] = \ell$, if some honest nodes output in the current view R, they output the same value v_ℓ, while for all other honest nodes who have not output in view R, v_ℓ is the unique valid SPB input in view R' for any $\mathsf{R}' > \mathsf{R}$, assuming the underlying threshold signature, SPB and Election are secure.

Proof: Suppose no honest node outputs before view R, and there exists an honest node $\mathcal{P}_i$ that outputs a value v_ℓ in view R due to one of the following three cases: (1) $\mathcal{P}_i$ has received a valid $(\mathsf{FIN}, \mathsf{id}, \mathsf{R}, \mathsf{Finish})$ message from the elected leader $\mathcal{P}_\ell$ when it learns ℓ; (2) the node $\mathcal{P}_i$ does not receive any

valid Finish corresponding to SPB[$\langle$id, R, $\ell\rangle$], but it receives $2f+1$ valid (Vote, id, R, ℓ, Yes) messages from distinct nodes; (3) the node $\mathcal{P}_i$ receives a valid (Halt, id, R, Finish) message. Lemma 9 guarantees the lemma to be hold in case 1, Lemma 10 makes sure the lemma to be hold in case 2, and Lemma 11 ensures the lemma to be hold in case 3. ■

Theorem 3: **Agreement**. If any two honest nodes output, their output values would be same, assuming the underlying threshold signature, SPB and Election are secure.

Proof: Now suppose that an honest node $\mathcal{P}$ is the first node that outputs some value v in some view R. For any other honest node $\mathcal{P}'$, if $\mathcal{P}'$ also outputs in view R, the first part of Lemma 12 ensures its output to be v; if $\mathcal{P}'$ outputs in some view $\mathsf{R}' > \mathsf{R}$, the second part of Lemma 12 ensures that v is the only possible output of $\mathcal{P}'$, because any $v' \neq v$ will be rejected by SPBs after view R. ■

Theorem 4: **External-Validity**. If an honest node outputs a value v, then v is valid for Q, i.e., $Q(v) = 1$, assuming the underlying SPB is secure.

Proof: According to Algorithm 2, when a node inputs v into SPB, it always satisfies $\mathsf{Q}(v) = true$, otherwise, the value cannot be output by SPB due to the external validity of SPB. Therefore, the external-validity trivially holds. ■

A remark on adaptive security: Our sMVBA (and sDumbo) can also be adaptively secure, given threshold signature that is unforgeable against adaptive adversaries, which is similar to many existing asynchronous BFT protocols [4, 12, 32]. The leader election primitive can also be secure against adaptive adversary, as long as it is built from adaptively secure threshold signature. Similarly, SPB and PB enjoy termination (as long as the sender has not been corrupted), satisfy provability (if using adaptively secure threshold signature).

Considering that the adaptive adversary corrupting at most f nodes, that means there are at least $n - f$ nodes that are always honest during the protocol execution (called forever-honest nodes). So in sMVBA, the senders of at least $2f + 1$ SPB instances are forever-honest nodes, if they do not abandon SPB, they all will complete the SPB and send $n - f$ Finish messages to all so-far honest nodes; else, some forever-honest node abandons SPB because of receiving $n-f$ FIN messages, which indicates at least $n - 2f \geq f + 1$ forever-honest nodes multicast FIN messages thus all so-far honest nodes can receive at least $f + 1$ FIN messages. In both cases, all so-far honest nodes enter the leader election phase, then the termination and agreement of Election protocol guarantee all of them to output the same index ℓ. Also, these so-far honest nodes will not be stuck in any PreVote and Vote phase mainly because threshold signature is still robust and at least $n-f$ forever-honest nodes will participate. Again, the leader election is still unpredictable (which can be induced by (adaptive) unforgeability of the threshold signatures), thus helping all so-far honest nodes terminate in expected constant time. The agreement of sMVBA can be similarly argued as in Theorem 3. The external-validity also trivially holds. Similarly, we can lift for sDumbo.

Proving the extra quality property: sMVBA can also satisfy the *Quality* property [4, 32] as a by-product. The actual quality guarantee is subtly different between the adaptive corruption case and the static corruption case: when the adversary is static (resp. adaptive), the quality would ensure that the sMVBA output was proposed by the adversary with at most 1/2 (resp. 2/3) probability. Below are the detailed proofs.

Lemma 13: **Quality**. If an honest node outputs v, the probability that v was proposed by the static (resp. adaptive) adversary is at most 1/2 (resp. 2/3), assuming the underlying threshold signature, SPB and Election are secure.

Proof: Since the threshold of Election is $2f + 1$, the adversary learns the output of Election, only if at least $f + 1$ honest nodes invoke Election. This further implies that: when the adversary learns the elected leader, at least $2f + 1$ SPB instances have been completed, because Lemma 5 states: once an honest node invokes Election, at least $2f + 1$ distinct SPB instances have completed. Note that if Election[$\langle$id, R$\rangle$] returns ℓ and the sender $\mathcal{P}_\ell$ has completed the SPB[$\langle$id, R, $\ell\rangle$] protocol, the SPB instance also produces a valid Finish, and all honest nodes can output value v_ℓ in view R due to Lemma 8.

Then we discuss quality in the static and adaptive adversary models, separately. We begin with the static case. Let us consider the following cases for the output ℓ of Election:

Case 1: The elected sender $\mathcal{P}_\ell$ is corrupted and completes the SPB protocol. The probability of this case at most is $1/3$.

Case 2: The elected sender $\mathcal{P}_\ell$ is honest and completes the SPB protocol. The probability of this case at least is $1/3$.

Case 3: The elected sender $\mathcal{P}_\ell$ is corrupted and has not completed SPB yet, then the protocol might be repeated to enter the next view or might output. The probability of this case and case 1 is at most $1/3$.

Case 4: The elected sender $\mathcal{P}_\ell$ is (so-far) honest and has not completed SPB yet, then the protocol might be repeated to enter the next view or might output. The probability of this case and case 3 at most is $1/3$.

In case of facing static corruptions, the output can be controlled by the adversary in case 1 and 3. So the worst-case quality occurs, if the adversary maximizes the probability of case 1. Namely, the adversary makes that all corrupted nodes' SPBs completed and delays the completeness of f honest nodes' SPBs when Election is invoked. Hence, the probability that the output v is proposed by the static adversary would be at most $\sum_{k=1}^{\infty}(1/3)^k = 1/2$.

Then we analyze the case of adaptive adversaries. Different from a static adversary that can propose the output in case 1 and 3, an adaptive adversary can also propose the output in case 4. This is because it can learn the leader election result before $f+1$ honest nodes abandon SPBs, so it can adaptively corrupt the so-far-honest elected leader in case 4 and control the elected node to quickly finish SPB. As a result, an adaptive adversary might propose the output with 2/3 probability. ■

APPENDIX F
DEFERRED PB CONSTRUCTION

Algorithm 6 presents a simple construction of PB, given secure non-interactive threshold signature. Assuming that the input size of PB is $\mathcal{O}(|m|)$ bit, the complexities of PB can be analyzed as: it has two asynchronous rounds, where each round including $\mathcal{O}(n)$ messages and $\mathcal{O}(1)$ rounds; in addition,

in the first round, the size of message is $\mathcal{O}(|m|)$-bit; in the second round, the size of message is $\mathcal{O}(\lambda)$-bit. So the overall bits of the $\mathcal{O}(n)$ messages in PB are $\mathcal{O}(|m|n + \lambda n)$.

Algorithm 6 The PB protocol with identifier ID (for node $\mathcal{P}_i$, where the sender is $\mathcal{P}_s$)

Initialization: let $S_s = \{\}$

if $\mathcal{P}_i = \mathcal{P}_s$ **then**
  upon receiving input value (v_s, π) **do**
    multicast $(\mathsf{Value}, \mathsf{ID}, v_s, \pi)$
  upon receiving $(\mathsf{Echo}, \mathsf{ID}, \rho_j)$ from $\mathcal{P}_j$ for the first time **do**
    if $\mathsf{VerifyShare}_{(2f+1)}(\langle \mathsf{ID}, h_s \rangle, (j, \rho_j))$ =1 **then**
      $S_s \leftarrow S_s \cup \{j, \rho_j\}$
      if $|S_s| = 2f + 1$ **then**
        $\sigma_s \leftarrow \mathsf{Combine}_{2f+1}(\langle \mathsf{ID}, h_s \rangle, S_s)$
        output $lock := (h_s, \sigma_s)$
upon receiving $(\mathsf{Value}, \mathsf{ID}, v_s, \pi)$ from sender $\mathcal{P}_s$ for the first time **do**
  if $\mathsf{ValueValidation}(\mathsf{ID}, v_s, \pi) = true$ **then**
    output $value := \langle v_s, \pi \rangle$
    let $h_s = \mathcal{H}(v_s)$
    $\rho_i \leftarrow \mathsf{SignShare}_{2f+1}(sk_i, \langle \mathsf{ID}, h_s \rangle)$
    send $(\mathsf{Echo}, \mathsf{ID}, \rho_i)$ to $\mathcal{P}_s$

procedure $abandon(\mathsf{ID})$:
  halt PB[ID]

APPENDIX G
DEFERRED Election CONSTRUCTION

Algorithm 7 presents a random leader election protocol due to Cachin et al. [13] from non-interactive unique threshold signature in the random oracle model. In the protocol, every node only has to multicast a threshold signature share via a SHARE message, and then waits for $2f + 1$ such messages from distinct nodes to compute the elected leader. Note that we might use the protocol in a non-blackbox manner (and we present its blackbox properties mainly for easier understanding of our sMVBA construction). The security of sMVBA actually is reduced to non-interactive unique threshold signature and the random oracle (instead of the blackbox properties of the random leader election protocol).

Algorithm 7 $(n, 2f + 1)$ Leader Election (Election): for node $\mathcal{P}_i$

Setup: $\mathcal{P}_i$ obtains its private key share sk_i and all public keys msk and PK for a $(n, 2f + 1)$ threshold signature scheme from a trusted dealer or some distributed key generation protocol.

Initialization: $\Sigma \leftarrow \{\}$
upon the protocol is activate on identifier ID **do**
  $\sigma_i \leftarrow \mathsf{SignShare}_{(2f+1)}(sk_i, \mathsf{ID})$
  multicast $(\mathsf{SHARE}, \mathsf{ID}, \sigma_i)$
  wait until $|\Sigma| = 2f + 1$
    return $\mathcal{H}(\mathsf{Combine}_{(2f+1)}(\mathsf{ID}, \Sigma))$ % $n + 1$, where $\mathcal{H}$ is a random oracle
upon receiving $(\mathsf{SHARE}, \mathsf{ID}, \sigma_j)$ from $\mathcal{P}_j$ for the first time **do**
  if $\mathsf{ShareVerify}_{(2f+1)}(\mathsf{ID}, (j, \sigma_j)) = 1$ **then**
    $\Sigma \leftarrow \Sigma \cup \{(j, \sigma_j)\}$