TARDIGRADE: An Atomic Broadcast Protocol for Arbitrary Network Conditions

Erica Blum[1], Jonathan Katz[1*], and Julian Loss[2**]

[1] University of Maryland
{erblum,jkatz2}@umd.edu
[2] lossjulian@gmail.com

Abstract. We study the problem of *atomic broadcast*—the underlying problem addressed by blockchain protocols—in the presence of a malicious adversary who corrupts some fraction of the n parties running the protocol. Existing protocols are either robust for any number of corruptions in a *synchronous* network (where messages are delivered within some known time Δ) but fail if the synchrony assumption is violated, or tolerate fewer than $n/3$ corrupted parties in an *asynchronous* network (where messages can be delayed arbitrarily) and cannot tolerate more corruptions even if the network happens to be well behaved.

We design an atomic broadcast protocol (TARDIGRADE) that, for any $t_s \geq t_a$ with $2t_s + t_a < n$, provides security against t_s corrupted parties if the network is synchronous, while remaining secure when t_a parties are corrupted even in an asynchronous network. We show that TARDIGRADE achieves optimal tradeoffs between t_s and t_a. Finally, we show a second protocol (UPGRADE) with similar (but slightly weaker) guarantees that achieves per-transaction communication complexity linear in n.

Keywords: Atomic broadcast · Byzantine agreement · Consensus.

1 Introduction

Atomic broadcast [4] is a fundamental problem in distributed computing that can be viewed as a generalization of *Byzantine agreement* (BA) [10,15]. Roughly speaking, a BA protocol allows a set of n parties to agree on a value *once*, even if some parties are *Byzantine*, i.e., corrupted by an adversary who may cause them to behave arbitrarily. In contrast, an atomic broadcast (ABC) protocol allows parties to repeatedly agree on values by including them a totally-ordered, append-only log maintained by all parties. (Formal definitions are given in Section 3. Note that ABC is not obtained by simply repeating a BA protocol multiple times; this point is discussed further below.) Atomic broadcast is used as a

[*] Work performed under financial assistance award 70NANB19H126 from the U.S. Department of Commerce, National Institute of Standards and Technology, and also supported in part by NSF award #1837517.

[**] Portions of this work were done while at the University of Maryland and Ruhr University Bochum.

building block for *state machine replication*, and has received renewed attention in recent years for its applications to blockchains and cryptocurrencies.

Different network models for atomic broadcast can be considered. In a *synchronous* network [**?**, **?**, 2, 8, 13], all messages are delivered within some known time Δ. In an *asynchronous* network [**?**, **?**], messages can be delayed arbitrarily. (Some work assumes the *partially synchronous* model [5], where messages are delivered within some time bound Δ that is unknown to the parties. We do not consider this model in our work.) Assuming a public-key infrastructure (PKI), atomic broadcast is feasible for $t_s < n$ adversarial corruptions in a synchronous network, but only for $t_a < n/3$ faults in an asynchronous network. A natural question is whether it is possible to design a protocol that can withstand strictly more than $n/3$ faults if the network happens to be synchronous, without entirely sacrificing security if the network happens to be asynchronous. More precisely, fix two thresholds t_a, t_s with $t_a \leq t_s$. Is it possible to design a *network-agnostic* atomic broadcast protocol that (1) tolerates t_s corruptions if it is run in a synchronous network and (2) tolerates t_a corruptions if it is run in an asynchronous network? Depending on one's assumptions about the probabilities of different events, a network-agnostic protocol could be preferable to either a purely synchronous protocol (which loses security if the network is asynchronous) or a purely asynchronous one (which loses security if there are $n/3$ or more faults).

We settle the above question in a model where there is a trusted dealer who distributes information to the parties in advance of the protocol execution:

- We present an atomic broadcast protocol, TARDIGRADE,[1] that achieves the above for any t_a, t_s satisfying $t_a + 2t_s < n$. We also prove that no atomic broadcast protocol can provide the above guarantees[2] if $t_a + 2t_s \geq n$, and so TARDIGRADE is optimal in terms of the thresholds it tolerates.
- We also describe a second protocol, UPGRADE, that is sub-optimal in terms of t_a, t_s but has asymptotic communication complexity comparable to state-of-the-art asynchronous atomic broadcast protocols (see Table 1).

To see how TARDIGRADE can be advantageous to either a fully synchronous or a fully asynchronous protocol, consider the following concrete example. Fix t_a, t_s with $t_a < n/3 \leq t_s$ and $2t_s + t_a < n$, and let $f(t)$ be the probability that the number of faults is strictly greater than t. Suppose $f(t_a) = 1/10$, $f(\lfloor \frac{n-1}{3} \rfloor) = 1/20$, and $f(t_s) = 0$, and the probability that the network delay ever exceeds the assumed bound is $p_\Delta = 1/10$. Then a purely synchronous protocol fails to provide security with probability $p_\Delta = 1/10$, while a purely asynchronous protocol fails to provide security with probability $f(\lfloor \frac{n-1}{3} \rfloor) = 1/20$. But TARDIGRADE (with parameters t_a, t_s) fails to provide security only with probability $f(t_s) + p_\Delta \cdot f(t_a) = 1/100$.

[1] Tardigrades, also called water bears, are microscopic animals known for their ability to survive in extreme environments.

[2] This does not contradict the existence of synchronous ABC protocols for $t_s < n$, since such protocols are insecure in an asynchronous setting even if no parties are corrupted.

Protocol	**Communication**	**Network model**
HoneyBadger [?]	$O(n \cdot \lvert\mathsf{tx}\rvert)$	Asynchronous
BEAT1 / BEAT2 [?]	$O(n^2 \cdot \lvert\mathsf{tx}\rvert)$	Asynchronous
Dumbo1 / Dumbo2 [?]	$O(n \cdot \lvert\mathsf{tx}\rvert)$	Asynchronous
TARDIGRADE	$O(n^2 \cdot \lvert\mathsf{tx}\rvert)$	Network-agnostic
UPGRADE	$O(n \cdot \lvert\mathsf{tx}\rvert)$	Network-agnostic

Table 1. Per-transaction communication complexity of ABC protocols, for transactions of length $|\mathsf{tx}|$, assuming infinite block size and suppressing dependence on the security parameter for simplicity.

Our work is inspired by work of Blum et al. [?], who show analogous results (with the same thresholds) for the simpler problem of Byzantine agreement. We emphasize that ABC is not realized by simply repeating a (multi-valued) BA protocol multiple times. In particular, the validity property of BA guarantees only that if a value is used as input by all honest parties then that transaction will be output by all honest parties. In the context of ABC, however, each honest party holds a local buffer containing multiple values called *transactions*. Transactions may arrive at arbitrary times, and there is no way to ensure that all honest parties will input the same transactions to some execution of an underlying BA protocol. (Although generic transformations from BA to ABC are known in other settings [3], no such transformation is known for the network-agnostic setting we consider.) Indeed, translating the approach of Blum et al. from BA to ABC introduces several additional challenges. In particular, as just noted, in the context of atomic broadcast there is no guarantee that honest parties ever use the same transaction, making it more challenging to prove liveness. A central piece of our construction is a novel protocol for the fundamental problem of *asynchronous common subset* (ACS). Our ACS protocol achieves non-standard security properties that turn out to be generally useful for constructing protocols in a network-agnostic setting; it has already served as a crucial ingredient in follow-up work [?] on network-agnostic secure computation.

1.1 Related Work

There is extensive prior work on both Byzantine agreement and atomic broadcast/SMR/blockchain protocols; we do not provide an exhaustive survey, but instead focus only on the most closely related works.

Miller et al. [?] already note that well-known SMR protocols that tolerate malicious faults (e.g., [2,8]) fail to achieve liveness in an asynchronous network. The HoneyBadger protocol [?] is designed for asynchronous networks, but only handles $t < n/3$ faults even if the network is synchronous.

Several of the most prominent blockchain protocols rely on synchrony [?,13]; Nakamoto consensus, in particular, relies on the assumption that messages will be delivered much faster than the time required to solve proof-of-work puzzles, and is insecure if the network latency is too high or nodes become (temporarily) partitioned from the network.

We focus on designing a single protocol that may be run in either a synchronous or asynchronous network while providing security guarantees in either case. Related work includes that of Malkhi et al. [?] and Momose and Ren [12], who consider networks that may be either synchronous or *partially* synchronous; Liu et al. [11], who design a protocol that tolerates a minority of malicious faults in a synchronous network and a minority of *fail-stop* faults in an asynchronous network; and Guo et al. [?] and Abraham et al. [?], who consider temporary disconnections between two synchronous network components.

A different line of work [?, ?, ?, ?] designs protocols with good *responsiveness*. Roughly, such protocols still require synchrony, but terminate in time proportional to the actual message-delivery time δ rather than the upper bound on the network-delivery time Δ. Kursawe [9] gives a protocol for an asynchronous network that terminates more quickly if the network is synchronous (but does not tolerate more faults in that case). Finally, other work [?, ?, 6, 14] considers a model where synchrony is available for some (known) period of time, and the network is asynchronous afterward.

1.2 Paper Organization

We describe our model in Section 2, and give formal definitions in Section 3. In Section 4, we describe a protocol for the asynchronous common subset (ACS) problem. Then, in Section 5, we show how to construct a network-agnostic atomic broadcast protocol (TARDIGRADE) achieving optimal security tradeoffs using ACS and other building blocks. In Section 6, we present a second atomic broadcast protocol (UPGRADE) that achieves per-transaction communication complexity linear in n at the cost of tolerating fewer corruptions.

2 Model

We consider protocols run by n parties $P_1, \ldots, P_n$, over point-to-point authenticated channels. Some fraction of these parties are controlled by an adversary, and may deviate arbitrarily from the protocol. For simplicity, we generally assume a static adversary who corrupts parties prior to the start of the protocol; in Section 5.6, however, we do briefly discuss how TARDIGRADE can be modified to tolerate an adaptive adversary who may corrupt parties as the protocol is executed. Parties who are not corrupted are called *honest*.

In our model, the network has two possible states. The state is fixed prior to the beginning of the execution; however, the state is not known to the honest parties. When the network is *synchronous*, all parties begin the protocol at the same time, parties' clocks progress at the same rate, and all messages are delivered within some known time Δ after they are sent. The adversary is able to adaptively delay and reorder messages arbitrarily (subject to the bound Δ). When the network is *asynchronous*, the adversary is able to delay messages for arbitrarily long periods of time (as long as all messages are eventually delivered). The parties still have local clocks in the asynchronous setting; however, in this case their clocks are only assumed to be monotonically increasing. In particular, parties' clocks are not necessarily synchronized, and they may start the protocol at different times.

We assume the network is either synchronous or asynchronous for the lifetime of the protocol. A more general model would consider a network that alternates between periods of synchrony and asynchrony. Our adaptively secure protocol (cf. Section 5.6) tolerates an asynchronous network that later becomes synchronous so long as the attacker does not exceed t_a corruptions until all iterations initiated while the network was asynchronous are complete, and does not exceed t_s corruptions overall. Handling a synchronous network that later becomes asynchronous is only interesting if some mechanism is provided to "uncorrupt" parties (as in the proactive setting). This is outside our model, and we leave treatment of this case as an interesting direction for future work.

We assume a trusted *dealer* who initializes parties with some information prior to execution of the protocol. Specifically, we assume the dealer distributes keys for threshold signature and encryption schemes, each secure for up to t_s corruptions. In a threshold signature scheme there is a public key pk, private keys $sk_1, \ldots, sk_n$, and (public) signature verification keys $(pk_1, \ldots, pk_n)$. Each party P_i receives sk_i, pk, and $(pk_1, \ldots, pk_n)$, and can use its secret key sk_i to create a signature share σ_i on a message m. A signature share from party P_i on a message m can be verified using the corresponding public verification key pk_i (and is called *valid* if it verifies successfully); for this reason, we can also view such a signature share as a signature by P_i on m. We often write $\langle m \rangle_i$ as a shorthand for the tuple (i, m, σ_i), where σ_i is a valid signature share on m with respect to P_i's verification key, and implicitly assume that invalid signature shares are discarded. A set of $t_s + 1$ valid signature shares on the same message can be used to compute a signature for that message, which can be verified using the public key pk; a signature σ on a message m is called *valid* if it verifies successfully with respect to pk. We always implicitly assume that parties use some form of domain separation when signing to ensure that signature shares are valid only in the context in which they are generated.

In a threshold encryption scheme, there is a public encryption key ek, (private) decryption keys $dk_1, \ldots, dk_n$, and public verification keys $vk_1, \ldots, vk_n$ that can be used, as above, to verify that a decryption share is correct (relative to a particular ciphertext). A party P_i can use its decryption key dk_i to generate a decryption share of a ciphertext c; any set of $t_s + 1$ correct decryption shares enable recovery of the underlying message m. Security requires that no collection of t_s parties can decrypt on their own.

We idealize the threshold signature and encryption schemes for simplicity, but they can be instantiated using any of several known protocols; in particular, we only require CPA-security for the threshold encryption scheme. We assume that signature shares and signatures have size $O(\kappa)$, where κ is the security parameter; this is easy to ensure using a collision-resistant hash function. We assume that encrypting a message m of length $|m|$ produces a ciphertext of length $|m| + O(\kappa)$, and that decryption shares have length $O(\kappa)$; these are easy to ensure using standard KEM/DEM mechanisms.

3 Definitions

In this section, we formally define atomic broadcast and relevant subprotocols. Throughout, when we say a protocol achieves some property, we include the case where it achieves that property with overwhelming probability in a security parameter κ. Additionally, in some cases we consider protocols where parties may not terminate even upon generating output; for this reason, we mention termination explicitly in our definitions when applicable.

Many of the definitions below are parameterized by a threshold t. This will become relevant in later sections, where we will often analyze a protocol's properties in a synchronous network with t_s corruptions, as well as in an asynchronous network with t_a corruptions.

3.1 Broadcast and Byzantine Agreement

A *reliable broadcast* protocol allows parties to agree on a value chosen by a designated sender. Honest parties are not guaranteed to terminate; hence, reliable broadcast is weaker than standard broadcast. However, if there is some honest party who terminates, then all honest parties terminate.

Definition 1 (Reliable broadcast). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where a designated sender $P^* \in \{P_1, \ldots, P_n\}$ begins holding input v^* and parties terminate upon generating output.*

- **Validity:** *Π is t-*valid *if the following holds whenever at most t parties are corrupted: if P^* is honest, then every honest party outputs v^*.*
- **Consistency:** *Π is t-*consistent *if the following holds whenever at most t parties are corrupted: either no honest party outputs a value, or all honest parties output the same value v.*

*If Π is t-valid and t-consistent, then we say it is t-*secure*.*

We reserve the term "broadcast" for reliable broadcast. When a party P_i sends a message m to all parties (over point-to-point channels), we say that P_i *multicasts m*.

Byzantine agreement (BA) is closely related to broadcast. In a BA protocol, there is no designated sender; instead, each party has their own input and the parties would like to agree on an output.

Definition 2 (Byzantine agreement). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in \{0, 1\}$.*

- **Validity:** *Π is t-*valid *if the following holds whenever at most t of the parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs v.*
- **Consistency:** *Π is t-*consistent *if whenever at most t parties are corrupted, every honest party outputs the same value $v \in \{0, 1\}$.*
- **Termination:** *Π is t-*terminating *if whenever at most t parties are corrupted, every honest party terminates with some output in $\{0, 1\}$.*

*If Π is t-valid, t-consistent, and t-terminating, then we say it is t-*secure*.*

3.2 Asynchronous Common Subset

Informally, a protocol for the *asynchronous common subset* (ACS) problem [**?**] allows n parties, each with some input, to agree on a subset of those inputs. (The term "asynchronous" in the name is historical, and one can also consider protocols for this task in the synchronous setting.)

Definition 3 (ACS). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each P_i begins holding input $v_i \in \{0, 1\}^*$, and parties output sets of size at most n.*

- **Validity:** *Π is t-*valid *if the following holds whenever at most t parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs $\{v\}$.*
- **Consistency:** *Π is t-*consistent *if whenever at most t parties are corrupted, all honest parties output the same set S.*
- **Liveness:** *Π is t-*live *if whenever at most t parties are corrupted, every honest party generates output.*

*If Π is t-consistent, t-valid, and t-live, we say it is t-*secure*.*

For our analysis, it will be helpful to define a few additional properties.

Definition 4 (ACS properties). *Let Π be as above.*

- **Set quality:** *Π has t-*set quality *if the following holds whenever at most t parties are corrupted: if an honest party outputs a set S, then S contains the input of at least one honest party.*
- **Validity with termination:** *Π is t-*valid with termination *if, whenever at most t parties are corrupted and every honest party's input is equal to the same value v, then every honest party outputs $\{v\}$ and terminates.*
- **Termination:** *Π is t-*terminating *if whenever at most t parties are corrupted, every honest party generates output and terminates.*

3.3 Atomic Broadcast

Protocols for *atomic broadcast* (ABC) allow parties to maintain agreement on an ever-growing, ordered log of *transactions*. An atomic broadcast protocol does not terminate but instead continues indefinitely. We model the local log held by each party P_i as a write-once array $\mathsf{Blocks}_i = \mathsf{Blocks}_i[1], \mathsf{Blocks}_i[2], \ldots$. Each $\mathsf{Blocks}_i[j]$ is initially set to a special value $\perp$. We say that P_i *outputs a block in iteration* j when P_i writes a set of transactions to $\mathsf{Blocks}_i[j]$; similarly, for each i, j such that $\mathsf{Blocks}_i[j] \neq \perp$, we refer to $\mathsf{Blocks}_i[j]$ as the *block output by* P_i *in iteration* j. For convenience, we let $\mathsf{Blocks}_i[k : \ell]$ denote the contiguous subarray $\mathsf{Blocks}_i[k], \ldots, \mathsf{Blocks}_i[\ell]$ and let $\mathsf{Blocks}_i[: \ell]$ denote the prefix $\mathsf{Blocks}_i[1 : \ell]$.

For simplicity, we imagine that each party P_i has a local buffer buf_i, and that transactions are added to parties' local buffers by some mechanism external to the protocol (e.g., via a gossip protocol). Whenever P_i outputs a block, they delete from their buffer any transactions that have already been added to their log. We emphasize that a particular transaction tx may be provided as input to different parties at arbitrary times, and may be provided as input to some honest parties but not others.

Definition 5 (Atomic broadcast). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$ who are provided with transactions as input and locally maintain arrays* Blocks *as described above.*

- **Completeness:** *Π is t-*$\mathsf{complete}$ *if the following holds whenever at most t parties are corrupted: for all $j > 0$, every honest party outputs a block in iteration j.*
- **Consistency:** *Π is t-*$\mathsf{consistent}$ *if the following holds whenever at most t parties are corrupted: if an honest party outputs a block B in iteration j then all honest parties output B in iteration j.*
- **Liveness:** *Π is t-*live *if the following holds whenever at most t parties are corrupted: if every honest party is provided a transaction* tx *as input, then every honest party eventually outputs a block that contains* tx.

If Π is t-consistent, t-live, and t-complete, then we say it is t-secure.

In the above definition, a transaction tx is only guaranteed to be contained in a block output by an honest party if *every* honest party receives tx as input. A stronger definition might require that a transaction is output even if only a *single* honest party receives tx as input; however, it is easy to achieve the latter from the former by requiring honest parties to forward new transactions they receive to the rest of the parties in the network.

4 ACS with Higher Validity Threshold

A key component of our atomic broadcast protocol is an ACS protocol for asynchronous networks that is secure when the number of corrupted parties is below a fixed threshold t_a, and guarantees validity up to a higher threshold t_s. More precisely, fix $t_a \leq t_s$ with $t_a + 2 \cdot t_s < n$; we show a t_a-secure ACS protocol that achieves t_a-termination, t_s-validity with termination, and t_a-set quality. Throughout this section, we assume an asynchronous network. (Of course, the protocol achieves the same guarantees in a synchronous network.)

Our protocol is adapted from the ACS protocol of Ben-Or et al. [**?**] (later adapted by Miller et al. [**?**]), which is built using subprotocols for reliable broadcast and Byzantine agreement. We present our construction in two steps: first, we describe an ACS protocol $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ (cf. Figure 1) that is t_a-secure and has t_a-set quality, but is non-terminating. Then, we construct a second protocol $\Pi^{t_a,t_s}_{\mathsf{ACS}}$ (cf. Figure 2) that uses $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ as a subprotocol. $\Pi^{t_a,t_s}_{\mathsf{ACS}}$ inherits security and set quality from $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$, and additionally achieves t_a-termination and t_s-validity with termination.

Protocol $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$. At a high level, an execution of $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ involves one instance of reliable broadcast and one instance of Byzantine agreement per party P_i, denoted Bcast_i and BA_i, respectively. Informally, Bcast_i is used to broadcast P_i's input v_i, and BA_i is used to determine whether P_i's input will be included in the final output. When a party receives output v'_i from Bcast_i, they input 1 to BA_i. Once a party has received output from $n - t_a$ broadcasts, they input 0 to any BA instances they have not yet initiated. Each party keeps track of which BA instances have output 1 using a local variable $S^* := \{i : \mathsf{BA}_i \text{ output } 1\}$. At the end of the protocol, if a party observes a majority value v in the set of values $\{v'_i\}_{i \in S^*}$, it outputs the singleton set $\{v\}$; otherwise, it outputs $\{v'_i\}_{i \in S^*}$, i.e., the set of all values broadcast by parties in S^*.

We assume an ABA subprotocol that is secure for $t_a < n/3$ corruptions and has communication complexity $O(n^2)$, such as the ABA protocol of Mostéfaoui et al. [**?**]. We also assume an asynchronous reliable broadcast protocol Bcast that is t_s-valid and t_a-consistent with communication complexity $O(n^2 |v|)$. It is straightforward to adapt Bracha's (asynchronous) reliable broadcast protocol [**?**] to achieve these properties; an example construction can be found in the full version of the paper.

Lemma 1. *Fix t_s, t_a with $t_a + 2 \cdot t_s < n$, and assume there are at most t_s corrupted parties during some execution of $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$. If an honest party P_i outputs a set S_i, then $\exists v_j \in S_i$ such that v_j was input by an honest party P_j.*

Proof. We show that P_i's output S_i always includes a value that was output from an execution of Bcast where the corresponding sender is honest. The lemma follows from t_s-validity of Bcast.

Suppose P_i generates output due to Event 1, so S_i is a singleton set $\{v\}$. P_i must have received v as output from at least $n - t_s$ executions of $\{\mathsf{Bcast}_i\}$. Because $n - 2t_s > t_a \geq 0$, at least one of those corresponds to an honest sender.

Next, suppose P_i generates output due to Event 2. Again, S_i is a singleton set $\{v\}$. P_i must have seen at least $\lfloor \frac{|S^*|}{2} \rfloor + 1$ broadcast instances terminate with output v, and furthermore $|S^*| \geq n - t_a$. Therefore, P_i has seen at least $\lfloor \frac{n-t_a}{2} \rfloor + 1 \geq \lfloor \frac{2t_s}{2} \rfloor + 1 > t_s$ executions of $\{\mathsf{Bcast}_i\}$ terminate with output v. Since there are at most t_s corrupted parties, at least one of those executions must correspond to an honest sender.

Finally, suppose P_i generates output due to Event 3, so $S_i = \{v'_i\}_{i \in S^*}$. Since there are at most t_s corrupted parties and $|S^*| - t_s \geq n - t_a - t_s > t_s \geq 0$, at least one party in S^* is honest. □

$\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$

- Set $\mathsf{commit} := \mathsf{false}$ and $S^* := \emptyset$.
- Run Bcast_j as the sender with input v_j, and for each $i \neq j$ run Bcast_i with P_i as the sender.
- Upon Bcast_i terminating with output v'_i: if P_j has not yet begun running BA_i then begin running it with input 1.
- Upon BA_i terminating with output 1: add i to S^*.
- Upon setting $|S^*|$ to $n - t_a$: for any BA_i that P_j has not yet begun running, begin running BA_i with input 0.

Predicates:

$C_1(v)$: at least $n - t_s$ executions $\{\mathsf{Bcast}_i\}_{i\in[n]}$ have output v.

C_1: $\exists v$ for which $C_1(v)$ is true.

$C_2(v)$: $|S^*| \geq n - t_a$, all executions $\{\mathsf{BA}_i\}_{i\in[n]}$ have terminated, and a strict majority of the executions $\{\mathsf{Bcast}_i\}_{i\in S^*}$ have output v.

C_2: $\exists v$ for which $C_2(v)$ is true.

C_3: $|S^*| \geq n - t_a$, all executions $\{\mathsf{BA}_i\}_{i\in[n]}$ have terminated, and all executions $\{\mathsf{Bcast}_i\}_{i\in S^*}$ have terminated.

Output conditions:

(Event 1) If $C_1(v) = \mathsf{true}$ for some v and $\mathsf{commit} = \mathsf{false}$ then:
set $\mathsf{commit} := \mathsf{true}$ and output $\{v\}$.

(Event 2) If $C_1 = \mathsf{false}$, $C_2(v) = \mathsf{true}$ for some v, and $\mathsf{commit} = \mathsf{false}$ then:
set $\mathsf{commit} := \mathsf{true}$ and output $\{v\}$.

(Event 3) If $C_1 = C_2 = \mathsf{false}$, $C_3 = \mathsf{true}$, and $\mathsf{commit} = \mathsf{false}$ then:
set $\mathsf{commit} := \mathsf{true}$ and output $\{v'_i\}_{i\in S^*}$.

Fig. 1. An ACS protocol, from the perspective of party P_j with input v_j.

Lemma 2. *If $t_a + 2 \cdot t_s < n$, then $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$ is t_s-valid.*

Proof. Assume at most t_s parties are corrupted, and all honest parties have the same input v. By t_s-validity of Bcast, at least $n - t_s$ executions of $\{\mathsf{Bcast}_i\}$ (namely, those for which the sender is honest) will eventually output v. It follows that all honest parties eventually set $C_1(v) = \mathsf{true}$, at which point they will output $\{v\}$ if they have not already generated output. It only remains to show that there is no other set an honest party can output.

If an honest party generates output S due to Events 1 or 2, then S is a singleton set. Since all honest parties have input v, Lemma 1 implies $S = \{v\}$.

To conclude, we show that no honest party can generate output due to Event 3. Assume toward a contradiction that some honest party P generates output due to Event 3. Then P must have seen Bcast_i terminate (say, with output v_i) for all $i \in S^*$. Since also $|S^*| \geq n - t_a > 2t_s$, a majority of those executions $\{\mathsf{Bcast}_i\}_{i\in S^*}$ correspond to honest senders and so (by t_s-validity of Bcast) resulted in output v. But then $C_2(v)$ would be true for P, and P would not generate output due to Event 3. □

Lemma 3. *Fix $t_a \le t_s$ with $t_a + 2 \cdot t_s < n$, and assume at most t_a parties are corrupted during an execution of $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$. If honest parties P_1, P_2 output sets S_1, S_2, respectively, then $S_1 = S_2$.*

Proof. Say P_1 generates output due to event i and P_2 generates output due to event j, and assume without loss of generality that $i \le j$. We consider the different possibilities.

First, assume $i = 1$ so Event 1 occurs for P_1 and $S_1 = \{v_1\}$ for some value v_1. We have the following sub-cases:

- If Event 1 also occurs for P_2, then $S_2 = \{v_2\}$ for some value v_2. P_1 and P_2 must have each seen some set of at least $n - t_s > n/2$ executions of $\{\mathsf{Bcast}_i\}$ output v_1 and v_2, respectively. The intersection of these sets is non-empty; thus, t_a-consistency of Bcast implies that $v_1 = v_2$ and hence $S_1 = S_2$.
- If Event 2 occurs for P_2, then once again $S_2 = \{v_2\}$ for some v_2. P_2 must have $|S^*| \ge n - t_a$, and must have seen at least $\left\lfloor \frac{|S^*|}{2} \right\rfloor + 1 \ge \left\lfloor \frac{n-t_a}{2} \right\rfloor + 1$ executions of $\{\mathsf{Bcast}_i\}$ output v_2. Moreover, P_1 must have seen at least $n - t_s$ executions of $\{\mathsf{Bcast}_i\}$ output v_1. Since
$$n - t_s + \left\lfloor \frac{n - t_a}{2} \right\rfloor + 1 \ge n - t_s + \left\lfloor \frac{2t_s}{2} \right\rfloor + 1 > n, \tag{1}$$
these two sets of executions must have a non-empty intersection. But then t_a-consistency of Bcast implies that $v_1 = v_2$ and hence $S_1 = S_2$.
- If Event 3 occurs for P_2 then P_2 must have seen all executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ terminate, where $|S^*| \ge n - t_a$. We know P_1 has seen at least $n - t_s$ executions $\{\mathsf{Bcast}_i\}_{i \in [n]}$ output v_1, and so (by t_a-consistency of Bcast) there are at most t_s executions $\{\mathsf{Bcast}_i\}_{i \in [n]}$ that P_2 has seen terminate with a value other than v_1. The number of executions of $\{\mathsf{Bcast}_i\}_{i \in S^*}$ that P_2 has seen terminate with output v_1 (which is at least $(n - t_a) - t_s > t_s$) is thus strictly greater than the number of executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ that P_2 has seen terminate with a value other than v_1 (which is at most t_s). But then $C_2(v_1)$ would be true for P_2. We conclude that Event 3 cannot occur for P_2.

Next, assume $i = j = 2$, so Event 2 occurs for P_1 and P_2. Then $S_1 = \{v_1\}$ and $S_2 = \{v_2\}$ for some v_1, v_2. Both P_1 and P_2 must have seen all executions $\{\mathsf{BA}_i\}_{i \in [n]}$ terminate. By t_a-consistency of BA, they must therefore agree on S^*. P_1 must have seen a majority of the executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ output v_1; similarly, P_2 must have seen a majority of the executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ output v_2. Then t_a-consistency of Bcast implies $v_1 = v_2$.

Finally, consider the case where $j = 3$ (so Event 3 occurs for P_2) but $i > 1$ (so P_1 generates output due either to Event 2 or 3). As above, t_a-consistency of BA ensures that P_1 and P_2 agree on S^*. Moreover, P_2 must have seen all executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ terminate, but without any value being output by a majority of those executions. But then t_a-consistency of Bcast implies that P_1 also does not see any value being output by a majority of those executions, and so Event 2 cannot occur for P_1; thus, Event 3 must have occurred for P_1. Therefore, t_a-consistency of Bcast implies that P_1 outputs the same set as P_2. □

Lemma 4. *If $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$, then $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ is t_a-live.*

Proof. It follows easily from t_a-security of Bcast and BA that if any honest party generates output then all honest parties generate output, so consider the case where no honest parties have (yet) generated output. Let H denote the indices of the honest parties. By t_s-validity of Bcast, all honest parties eventually see the executions $\{\mathsf{Bcast}_i\}_{i \in H}$ terminate, and so all honest parties input a value to the executions $\{\mathsf{BA}_i\}_{i \in H}$. By t_a-security of BA, all honest parties eventually see those executions terminate and agree on their outputs. There are now two cases:

- If all executions $\{\mathsf{BA}_i\}_{i \in H}$ output 1, then it is immediate that all honest parties have $|S^*| \geq n - t_a$.
- If BA_i outputs 0 for some $i \in H$, then (by t_a-validity of BA) some honest party P must have used input 0 when running BA_i. But then P must have seen at least $n - t_a$ other executions $\{\mathsf{BA}_i\}$ output 1. By t_a-consistency of BA, this implies that all honest parties see at least $n - t_a$ executions $\{\mathsf{BA}_i\}$ output 1, and hence have $|S^*| \geq n - t_a$.

Since all honest parties have $|S^*| \geq n - t_a$, they all execute $\{\mathsf{BA}_i\}_{i \in [n]}$. Once again, t_a-termination of BA implies that all those executions will eventually terminate. Finally, if $i \in S^*$ for some honest party P then P must have seen BA_i terminate with output 1; then t_a-validity of BA implies that some honest party used input 1 when running BA_i and hence has seen Bcast_i terminate. It follows that P will see Bcast_i terminate. As a result, we see that every honest party can (at least) generate output due to Event 3. □

Lemma 5. *If $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$, then $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ has t_a-set quality.*

Proof. If an honest party P outputs $S = \{v\}$ due to Event 1, then P has seen at least $n - t_s$ executions $\{\mathsf{Bcast}_i\}$ terminate with output v. Of these, at least $n - t_s - t_a > 0$ must correspond to honest senders. By t_s-validity of Bcast, those honest parties must have all had input v, and so set quality holds. Alternatively, say P outputs a set $\{v\}$ due to Event 2. Then P must have $|S^*| \geq n - t_a$, and at least $\lfloor \frac{|S^*|}{2} \rfloor + 1 \geq \lfloor \frac{n - t_a}{2} \rfloor + 1 > t_a$ of the executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ output v. At least one of those executions must correspond to an honest party, and that honest party must have had input v (by t_s-validity of Bcast); thus, set quality holds. Finally, if P output a set S due to Event 3, then S contains every value output by $\{\mathsf{Bcast}_i\}_{i \in S^*}$ with $|S^* \geq n - t_a$. Since S^* must contain at least one honest party, set quality follows as before. □

Theorem 1. *Fix t_a, t_s with $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ is t_a-secure and t_s-valid, and has t_a-set quality.*

Proof. Lemma 2 proves t_s-validity. Lemmas 3 and 4 together prove t_a-liveness and t_a-consistency, and Lemma 6 proves t_a-set quality. □

Lemma 6. *If $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$, then $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ has t_a-set quality.*

Proof. If an honest party P outputs $S = \{v\}$ due to Event 1, then P has seen at least $n - t_s$ executions $\{\mathsf{Bcast}_i\}$ terminate with output v. Of these, at least $n - t_s - t_a > 0$ must correspond to honest senders. By t_s-validity of Bcast, those honest parties must have all had input v, and so set quality holds. Alternatively, say P outputs a set $\{v\}$ due to Event 2. Then P must have $|S^*| \geq n - t_a$, and at least $\lfloor \frac{|S^*|}{2} \rfloor + 1 \geq \lfloor \frac{n-t_a}{2} \rfloor + 1 > t_a$ of the executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ output v. At least one of those executions must correspond to an honest party, and that honest party must have had input v (by t_s-validity of Bcast); thus, set quality holds. Finally, if P output a set S due to Event 3, then S contains every value output by $\{\mathsf{Bcast}_i\}_{i \in S^*}$. Since $|S^*| \geq n - t_a$, S^* must contain at least one honest party, and so set quality follows as before. □

Protocol $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. Protocol $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$ does not guarantee termination. We transform $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$ to a terminating ACS protocol $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ using digital signatures. The parties first run $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$. When a party P_i generates output S_i from that protocol, it then notifies the other parties by multicasting a signature share$\langle \mathsf{commit}, S_i \rangle_i$ on S_i. Any party who receives enough signature shares to form a signature—or receives a signature directly—multicasts the signature to all other parties, outputs the corresponding set, and terminates.

$\Pi_{\mathsf{ACS}}^{t_a,t_s}$

- Run $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$ using input v_j.
- Upon receiving output S_j from $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$, multicast $\langle \mathsf{commit}, S_j \rangle_j$.
- Upon receiving $t_s + 1$ signature shares of (commit, S), form a signature σ on (commit, S), multicast $(\mathsf{commit}, S, \sigma)$, output S, and terminate.
- Upon receiving a valid signature σ of (commit, S), multicast $(\mathsf{commit}, S, \sigma)$, output S, and terminate.

Fig. 2. A terminating ACS protocol, from the perspective of party P_j with input v_j.

Lemma 7. *$\Pi_{\mathsf{ACS}}^{t_a,t_s}$ is t_a-terminating.*

Proof. If one honest party terminates $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ then all honest parties will eventually receive a valid signature and thus terminate $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. But as long as no honest parties has yet terminated, t_a-liveness of $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$ implies that all honest parties will generate output from $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$; moreover, t_a-consistency of $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$ implies that all those outputs will be equal to the same set S. So the $n - t_a \geq t_s + 1$ honest parties will send signature shares on S to all parties, which means that all honest parties will terminate. □

Lemma 8. *Fix t_a, t_s with $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ is t_a-secure, t_a-terminating, and t_s-valid with termination, and has t_a-set quality.*

Proof. Lemma 7 implies that $\Pi^{t_a,t_s}_{\mathsf{ACS}}$ is t_a-live as well as t_a-terminating. If an honest party outputs a set S from $\Pi^{t_a,t_s}_{\mathsf{ACS}}$, then (as long as at most t_s parties are corrupted) at least one honest party must have output S from $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$. Thus, $\Pi^{t_a,t_s}_{\mathsf{ACS}}$ inherits t_a-set quality, t_a-consistency, and t_s-validity (without termination) from $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ (cf. Theorem 1). It is straightforward to extend t_s-validity to t_s-validity with termination using an identical argument as in Lemma 7. □

Communication complexity of $\Pi^{t_a,t_s}_{\mathsf{ACS}}$. Let $|v|$ be the size of each party's input. Recall that each instance of Bcast has communication complexity $O(n^2\,|v|)$, and each instance of BA has cost $O(n^2)$. Since the inner protocol $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ consists of n parallel instances of Bcast and BA, the cost of the inner protocol is $O(n^3\,|v|)$. In the remaining steps, each party sends a set of size at most n plus a signature share (or signature) to everyone else, contributing an additional $O(n^2 \cdot (n\,|v| + \kappa))$ communication. The total communication for $\Pi^{t_a,t_s}_{\mathsf{ACS}}$ is thus $O(n^3\,|v| + n^2\kappa)$.

5 Network-Agnostic Atomic Broadcast

In this section, we show our main result: for any $t_s \geq t_a$ with $t_a + 2t_s < n$, an atomic broadcast protocol that is t_s-secure in a synchronous network and t_a-secure in an asynchronous network.

5.1 Technical Overview

At a high level, each iteration of the protocol consists of four main steps. First, there is an information-gathering phase in which each party sends its input to all other parties, and waits for a fixed amount of time to receive inputs from others. Any party who receives enough inputs during the first phase will use them as input to a synchronous *block agreement* (BLA) protocol $\Pi^{t_s}_{\mathsf{BLA}}$. If the network is synchronous and at most t_s parties are corrupted, the BLA subprotocol will output a set of inputs that contains sufficiently many honest parties' inputs. The BLA subprotocol is run for a fixed amount of time, with the timeout chosen to ensure that (with high probability) it will terminate before the timeout if the network is synchronous. This brings us to the third phase, in which parties run the ACS protocol $\Pi^{t_a,t_s}_{\mathsf{ACS}}$. If a party received output from the BLA protocol before the timeout, they will use that as their input to the ACS subprotocol; otherwise, they wait until they have received sufficiently many inputs from other parties and use those. The final phase occurs once parties have received output from the ACS protocol. The parties use that output to form the next block.

The BLA and ACS protocols are designed to have complementary security properties. In particular, if the network is synchronous then the BLA protocol will ensure that all honest parties use the same input value B in the ACS protocol. This is exactly why $\Pi^{t_a,t_s}_{\mathsf{ACS}}$ has t_s-validity with termination: so that that, in this case, all parties will be in agreement on the singleton set $\{B\}$ after running $\Pi^{t_a,t_s}_{\mathsf{ACS}}$. On the other hand, if the network is not synchronous and at most t_a parties are corrupted, it is possible that $\Pi^{t_s}_{\mathsf{BLA}}$ will not succeed, and parties may input different values to $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. However, in this case t_a-security of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ ensures that the parties will agree on a set of values $B = \{\beta_1, \beta_2, \dots\}$. Moreover, the output-quality property ensures that at least a constant fraction of the values in B were contributed by honest parties.

5.2 Block Agreement

We use a block-agreement protocol to agree on objects that we call *pre-blocks.* (The name alludes to their role in our eventual atomic broadcast protocol, where they will serve as an intermediate between parties' raw inputs and the final blocks.) A pre-block is a vector of length n whose ith entry is either $\perp$ or a message along with a valid signature by P_i on that message. The *quality* of a pre-block is defined as the number of entries that are not $\perp$; we say that a pre-block is a *k-quality pre-block* if it has quality at least k.

Definition 6 (Block agreement). *Let Π be a protocol executed by parties $P_1, \dots, P_n$, where parties terminate upon generating output.*

- **Validity:** *Π is t-*valid *if whenever at most t of the parties are corrupted and every honest party's input is an $(n-t)$-quality pre-block, then every honest party outputs an $(n-t)$-quality pre-block.*
- **Consistency:** *Π is t-*consistent *if whenever at most t of the parties are corrupted, every honest party outputs the same pre-block B.*

*If Π is t-valid and t-consistent, then we say it is t-*secure*.*

A synchronous block-agreement protocol can be constructed using a straightforward adaptation of the *synod protocol* by Abraham et al. [1]. (For completeness, a construction and security analysis can be found in the full version.)

Theorem 2. *Fix a maximum input length $|m|$. There is a block-agreement protocol Π_{BLA} with communication complexity $O(n^3\kappa^2 + n^2\kappa|m|)$ that is t-secure for any $t < n/2$ when run in a synchronous network and terminates in time $5\kappa\Delta$.*

5.3 A Network-Agnostic Atomic Broadcast Protocol

We now describe our atomic broadcast protocol TARDIGRADE (cf. Figure 3), parameterized by thresholds t_s and t_a. Let L denote a desired maximum *block size*, i.e., the maximum number of transactions that can appear in a block. At a high level, parties agree on each new block via the following steps. First, each party P_i chooses a set V_i of L/n transactions from among the first L transactions in its local buffer. (We assume without loss of generality that parties always have at least L transactions in their buffer, since they can always pad their buffers with null transactions.) Next, P_i encrypts V_i using a (t_s, n)-threshold encryption scheme to give a ciphertext μ_i. (As in HoneyBadger [**?**], transactions are encrypted to limit the adversary's ability to selectively censor certain transactions.) Each party signs its ciphertext and multicasts it, then waits for a fixed period of time to receive signed ciphertexts from the other parties. Whenever a party receives a signed ciphertext during this time, they add it to a pre-block. Any party who forms an $(n - t_s)$-quality pre-block in this way within the time limit will input that pre-block to Π_{BLA}. The parties then wait for another fixed period of time to see whether Π_{BLA} outputs an $(n - t_s)$-quality pre-block. If a party receives an $(n-t_s)$-quality pre-block as output from Π_{BLA} within this time limit, it inputs that pre-block to the ACS protocol $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. Otherwise, if some party does not receive suitable output within the time limit, it inputs a pre-block containing the signed ciphertexts it received directly from other parties. (In this case, if a party has not received enough signed ciphertexts to form an $(n - t_s)$-quality pre-block, it waits for additional ciphertexts to arrive before inputting its pre-block to $\Pi_{\mathsf{ACS}}^{t_a,t_s}$.) At this point, each party waits for $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ to output a set of pre-blocks. The output of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ is passed into a subroutine $\mathsf{ConstructBlock}$ that performs threshold decryption for each ciphertext in each pre-block in the set, and combines the resulting transactions into a final block.

Each party begins iteration k when its local clock reaches time $T_k := \lambda\cdot(k-1)$, where $\lambda > 0$ is a *spacing parameter*. (The value of λ is irrelevant for the security proofs, but can be tuned to achieve better performance in practice; see further discussion in Section 5.4.) If the network is synchronous, parties' clocks are synchronized and so all parties begin each iteration at the same time. If the network is asynchronous, we do not have this guarantee. In either case, parties do not necessarily finish agreeing on block k prior to starting iteration $k' > k$, and so it is possible for parties to be participating in several iterations in parallel.

We implicitly assume that messages in each iteration, including messages corresponding to the various subprotocols, carry an identifier for the corresponding iteration so that parties know the iteration to which it belongs. Importantly, the executions of Π_{BLA} and $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ associated with a particular iteration are entirely separate from those of other iterations.

Theorem 3 (Completeness and consistency). *Fix t_a, t_s with $t_a \leq t_s$ and $t_a+2\cdot t_s < n$. Then $\Pi_{\mathsf{ABC}}^{t_a,t_s}$ is t_a-complete/consistent when run in an asynchronous network, and t_s-complete/consistent when run in a synchronous network.*

Proof. First, consider the case where at most t_s parties are corrupted and the network is synchronous. In the beginning of each iteration k, each honest party multicasts a set of transactions, and so every honest party can form an $(n-t_s)$-quality pre-block by time $T_k + \Delta$. Thus, every honest party starts running Π_{BLA} at time $T_k + \Delta$ using an $(n - t_s)$-quality pre-block as input. By t_s-security of Π_{BLA} in a synchronous network (note $t_s < n/2$), with overwhelming probability every honest party outputs the same $(n - t_s)$-quality pre-block β^* from Π_{BLA} by time $T_k + \Delta + 5\kappa\Delta$. Therefore, each honest party inputs β^* to $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. By t_s-validity with termination of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$, every honest party obtains the same output B^* from $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. So all honest parties eventually receive $n - t_s > t_s$ valid decryption shares for each ciphertext in each pre-block of B^*, and they all output the same block.

$\Pi_{\mathsf{ABC}}^{t_a,t_s}$

For each iteration $k = 1, 2, \ldots$ do:

- At time $T_k = \lambda\cdot(k-1)$: sample $V \leftarrow \mathsf{ProposeTxs}(L/n, L)$ and encrypt V using pk to produce a ciphertext μ. Multicast $(\mathsf{input}, \langle\mu\rangle_j)$.
- Upon receiving a signed input $(\mathsf{input}, \langle\mu\rangle_i)$ from P_i (for iteration k):
  - If this is the first input received for iteration k, create a new pre-block $\beta_j^k := (\bot, \ldots, \bot)$ and set $\mathsf{ready}_k := \mathsf{false}$.
  - If $\beta_j^k[i] = \bot$: set $\beta_j^k[i] := \langle\mu\rangle_j$.
  - If B_j^k is an $(n - t_s)$-quality pre-block and $\mathsf{ready}_k = \mathsf{false}$, set $\mathsf{ready}_k := \mathsf{true}$.
- At time $T_k + \Delta$: if $\mathsf{ready}_k = \mathsf{true}$, run Π_{BLA} using input β_j^k.
- At time $T_k + \Delta + 5\kappa\Delta$:
  - Terminate Π_{BLA} (if it has not already terminated). If Π_{BLA} had output an $(n-t_s)$-quality pre-block β^*, run $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ using input β^*. Else, wait until $\mathsf{ready}_k = \mathsf{true}$ and then run $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ using input β_j^k.
  - When $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ terminates with output B^*, run $\mathsf{ConstructBlock}(B^*)$ to produce a block B. Then set $\mathsf{Blocks}[k] := B$ and delete from buf_j any transactions that appear in $\mathsf{Blocks}[k]$.

$\mathsf{ProposeTxs}(\ell, M)$: choose a set V of ℓ values $\{\mathsf{tx}_1, \ldots, \mathsf{tx}_\ell\}$ uniformly (without replacement) from the first M values in buf_j, then output V.

$\mathsf{ConstructBlock}(B^*)$: participate in threshold decryption for each unique ciphertext μ in each pre-block $\beta \in B^*$. Once all decryptions have finished, output the set B of all unique transactions obtained.

Fig. 3. Our atomic broadcast protocol TARDIGRADE, from the perspective of party P_j.

The case where the network is asynchronous and at most t_a parties are corrupted is similar. In each iteration, each honest party multicasts a set of transactions and so every honest party eventually receives input from at least $n - t_a \geq n - t_s$ distinct parties and can form an $(n - t_s)$-quality pre-block. This means that every honest party eventually runs $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ using an $(n - t_s)$-quality pre-block as input. By t_a-security of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$, all honest parties eventually receive the same output B^* from $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. So all honest parties will eventually receive $n - t_a > t_s$ valid decryption shares for each ciphertext in each pre-block of B^*, and they all output the same block. □

In what follows, we let $\mathsf{Blocks}[k]$ denote the block output by honest parties in iteration k. We now turn our attention to liveness. We begin by proving a bound on the number of honest parties who contribute transactions to a block. Formally, we say that an honest party P_i *contributes transactions* to a block $B := \mathsf{ConstructBlock}(B^*)$ if there is a pre-block $\beta \in B^*$ such that $B[i] \neq \bot$. Using this lower bound, we show that any transaction that is at the front of most honest parties' buffers will eventually be output with overwhelming probability. Liveness follows by arguing that any transaction that is in the buffer of all honest parties will eventually move to the front of most honest parties' buffers.

Lemma 9. *Fix t_a, t_s with $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$, and assume at most t_a parties are corrupted and the network is asynchronous, or at most t_s parties are corrupted and the network is synchronous. Then in an execution of $\Pi_{\mathsf{ABC}}^{t_a,t_s}$, for any block B output by an honest party, at least $n - (t_s + t_a)$ honest parties contributed transactions to B.*

Proof. First, consider the case where at most t_a parties are corrupted and the network is asynchronous. As shown in the proof of Theorem 3, every honest party executes $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ using an $(n - t_s)$-quality pre-block as input. Thus, the input of every honest party to $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ contains at least $n - (t_s + t_a)$ ciphertexts created by honest parties. By t_a-set quality of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$, the output of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ contains some honest party's input and the lemma follows.

Next, consider the case where at most t_s parties are corrupted and the network is synchronous. As shown in the proof of Theorem 3, every honest party executes $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ using the same $(n-t_s)$-quality pre-block β as input. By t_s-validity with termination of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$, all honest parties output $B^* = \{\beta\}$ from $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. Because β is $(n-t_s)$-quality, it contains at least $(n-2t_s)$ honest parties' ciphertexts; the lemma follows. □

Lemma 10. *Assume the conditions of Lemma 9. Consider an iteration k and a transaction* tx *such that, at the beginning of iteration k, all but at most t_s honest parties have* tx *among the first L transactions in their buffers. Then for any $r > 0$,* tx *is in* $\mathsf{Blocks}[k : k + r]$ *except with probability at most* $(1 - 1/n)^{r+1}$.

Proof. By Lemma 9, at least $n - (t_s + t_a)$ honest parties contribute transactions to $\mathsf{Blocks}[k]$. So even if t_s parties are corrupted, at least one of the $n - 2t_s$ honest parties who have tx among the first L transactions in their buffers contributes transactions to $\mathsf{Blocks}[k]$. That party fails to include tx in the set V of transactions it chooses with probability $\binom{L-1}{L/n}/\binom{L}{L/n} = 1 - \frac{1}{n}$, and so tx is in $\mathsf{Blocks}[k]$ except with probability at most $1 - \frac{1}{n}$. (Note that this does not take into account the fact that the adversary may be able to choose which honest parties contribute transactions to B. However, because the parties encrypt their transactions, the adversary's choice has no effect on the calculation.) If tx does not appear in $\mathsf{Blocks}[k]$, then we can repeat the argument in all successive iterations $k + 1, \ldots, k + r$ until it does. □

Theorem 4 (Liveness). *Fix $t_a \leq t_s$ with $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{ABC}}^{t_a,t_s}$ is t_a-live in an asynchronous network, and t_s-live in a synchronous network.*

Proof. Suppose all honest parties have received a transaction tx. If, at any point afterward, tx is not in some honest party's buffer then tx must have already been included in a block output by that party (and that block will eventually be output by all honest parties). If all honest parties have tx in their buffers, then they each have a finite number of transactions ahead of tx. By completeness, all honest parties eventually output a block in each iteration. Additionally, by Lemma 9, at least $n - (t_s + t_a)$ honest parties' inputs are incorporated into each block, and so in each iteration all but at most t_s honest parties each remove at least L/n transactions from their buffers. It follows that eventually all but at most t_s honest parties will have tx among the first L transactions in their buffers. Once that occurs, Lemma 10 implies that tx is included in the next κ blocks except with probability negligible in κ. □

The above shows that a transaction received by all honest parties is eventually output. This is the standard notion of liveness in asynchronous networks. When working in a synchronous model, on the other hand, it is common to analyze liveness in more concrete terms. We provide such an analysis in the full version of the paper.

5.4 Efficiency and Choice of Parameters

The communication cost per iteration is dominated by the cost of the ACS and BLA subprotocols. Both ACS and BLA are run on pre-blocks, which have size $L \cdot |\mathsf{tx}| + O(n \cdot \kappa)$. Thus, each execution of BLA incurs cost $O(n^3\kappa^2 + n^2 L|\mathsf{tx}|\kappa)$, and an execution of ACS incurs cost $O(n^4\kappa + n^3 L|\mathsf{tx}|)$. The overall communication per block is therefore $O(n^4\kappa + n^3\kappa^2 + n^3 L|\mathsf{tx}| + n^2 L|\mathsf{tx}|\kappa)$.

At the beginning of every iteration, each honest party uniformly selects L/n transactions from among the first L transactions in its buffer. The following lemma shows that the expected number of *distinct* transactions they collectively choose is $O(L)$:

Lemma 11. *Assume the conditions of Lemma 9. In any iteration of $\Pi_{\mathsf{ABC}}^{t_a,t_s}$, the expected number of distinct transactions contributed by honest parties to the block B output by the honest parties in that iteration is at least $L/4$.*

Proof. The expectation is minimized when all honest parties have the same L transactions as the first L transactions in their buffers, so we assume this to be the case. As in Lemma 10, for some particular such transaction tx, the probability that some particular honest party fails to include tx in the set V of transactions it chooses is $1 - \frac{1}{n}$. Since, by Lemma 9, at least $n - (t_s + t_a) > n/3$ honest parties contribute transactions to B, the probability that none of those parties choose tx is at most $\left(1 - \frac{1}{n}\right)^{n/3} \leq e^{-1/3} < 3/4$, and so tx is chosen by at least one of those parties with probability at least $1/4$. (Once again, we do not take into account the fact that the adversary may be able to choose which honest parties contribute transactions because honest parties encrypt the transactions they choose.) The lemma follows by linearity of expectation. □

Because each block contains $O(L)$ transactions, the communication cost per transaction is $O((n^4\kappa + n^3\kappa^2)/L + n^3|\mathsf{tx}| + n^2|\mathsf{tx}|\kappa)$. So for $L = \Theta(n\kappa)$, the amortized communication cost per transaction is $O(n^3|\mathsf{tx}| + n^2|\mathsf{tx}|\kappa)$.

We remark that although each block contains at least $L/4$ distinct transactions in expectation, it is possible that some of those transactions are not new, i.e., they may have already been included in a previous block. This is possible because honest parties may sample their input in some iteration before having finished outputting blocks in all previous iterations. Thus, the actual communication cost per transaction may be higher than what we computed above. In general, the amount of overlap between blocks will depend on the spacing parameter λ as well as the actual network conditions and the parties' local clocks. If λ is too small, some space in each block may be wasted on redundant transactions; however, setting λ to be too large could introduce unnecessary delays in a synchronous network. Understanding how different choices of λ affect our protocol's performance in various network conditions is an interesting challenge for future work.

5.5 Optimality of Our Thresholds

We show that our protocol achieves the optimal tradeoff between the security thresholds. This result does not follow immediately from the impossibility result of Blum et al. [**?**] for network-agnostic Byzantine agreement because reductions from BA to atomic broadcast do not trivially translate to the network-agnostic setting; however, the main ideas of their proof readily extend to the case of atomic broadcast.

Lemma 12. *Fix t_a, t_s, n with $t_a + 2t_s \geq n$. If an n-party atomic broadcast protocol is t_s-live in a synchronous network, then it cannot also be t_a-consistent in an asynchronous network.*

Proof. Assume $t_a + 2t_s = n$ and fix an ABC protocol Π. Partition the n parties into sets S_0, S_1, S_a where $|S_0| = |S_1| = t_s$ and $|S_a| = t_a$. Consider the following experiment:

- Choose uniform $m_0, m_1 \leftarrow \{0,1\}^\kappa$. At global time 0, parties in S_0 begin running Π holding only m_0 in their buffer, and parties in S_1 begin running Π holding only m_1 in their buffer.
- All communication between parties in S_0 and parties in S_1 is blocked. All other messages are delivered within time Δ.
- Create virtual copies of each party in S_a, call them S_a^0 and S_a^1. Parties in S_a^b begin running Π (at global time 0) with their buffers containing only m_b, and communicate only with each other and parties in S_b.

Compare this experiment to a hypothetical execution E_{sync} of Π in a synchronous network, in which parties in S_1 are corrupted and simply abort, and the remaining parties are honest and initially hold only (uniformly chosen) m_0 in their buffer. The views of parties $S_0 \cup S_a^0$ in the experiment are distributed identically to the views of the honest parties in E_{sync}. Thus, t_s-liveness of Π implies that in the experiment, all parties in S_0 include m_0 in some block. Moreover, since parties in S_0 never receive information about m_1, they include m_1 in any block with negligible probability. By a symmetric argument, in the experiment, all parties in S_1 include m_1 in some block, and include m_0 in any block with negligible probability.

Now, consider a hypothetical execution E_{async} of Π, this time in an asynchronous network. In this execution, parties in S_0 and S_1 are honest while parties in S_a are corrupted. The parties in S_0 and S_1 initially hold $m_0, m_1 \leftarrow \{0,1\}^\kappa$, respectively. The corrupted parties interact with parties in S_0 as if they are honest and have m_0 in their buffer, and interact with parties in S_1 as if they are honest and have m_1 in their buffer. Meanwhile, all communication between parties in S_0 and S_1 is delayed indefinitely. The views of the honest parties in this execution are distributed identically to the views of $S_0 \cup S_1$ in the above experiment, yet the conclusion of the preceding paragraph shows that t_a-consistency is violated with overwhelming probability. □

5.6 Adaptive Security

Our analysis of TARDIGRADE assumes a static adversary who must choose the set of corrupted parties prior to the start of the protocol. In fact, TARDIGRADE is not secure against an adaptive adversary, since an adaptive adversary can prevent Π_{BLA} from terminating within time $5\kappa\Delta$ by corrupting the parties who are chosen as leaders. It is possible to modify TARDIGRADE to achieve adaptive security by suitably modifying Π_{BLA} in a relatively standard way: rather than choosing a leader who acts as the only proposer, each party will act as the proposer for one instance of the propose protocol, and a leader is then chosen retroactively after all instances terminate. Designing an adaptively secure network-agnostic atomic broadcast protocol with improved communication complexity is an interesting direction for future work. (Note that the committee-based approach in the following section is not adaptively secure.)

6 Improving Complexity using Committees

In this section, we describe an extension to TARDIGRADE that achieves lower amortized communication complexity in the presence of a static adversary. The improved protocol, UPGRADE, achieves expected communication complexity per transaction that is *linear* in n; specifically, it has expected per-transaction communication complexity $O(n\kappa|\mathsf{tx}| + \kappa^2|\mathsf{tx}|)$. This is made possible by delegating the most expensive steps of TARDIGRADE to a small committee.

To prove security for TARDIGRADE, we often used the fact that any sufficiently large subset of parties contained at least some minimum number of honest parties. We cannot assume this about the committees in UPGRADE, as the committee may be constant size, and in particular may be less than the number of corrupted parties. Instead, we prove that UPGRADE is secure in a setting with $O(\epsilon)$ fewer corrupted parties, where ϵ is a positive constant parameter of the protocol. More formally, fix t_s, t_a as before, and fix $\hat{t}_s$ such that $\hat{t}_s \leq (1-2\epsilon) \cdot t_s$ (for some $\epsilon > 0$); with probability $1 - e^{-O(\epsilon^2\kappa)}$, the improved ABC protocol is $\hat{t}_s$-secure in a synchronous network and t_a-secure in an asynchronous network. (Unless otherwise mentioned, all of the claims in this section hold with this probability.)

As in TARDIGRADE, we assume a trusted dealer who sets up threshold encryption and signature schemes. During the setup phase, the dealer also selects a *committee* $C \subset \{P_1, \ldots, P_n\}$ of size $O(\kappa)$ and provides each committee member $P_i \in C$ with a special credential π_i that proves P_i is on the committee. (For example, π_i might be a signature $\langle i \rangle_D$ on the index i that can be checked against the dealer's public key.) We also assume that there is a collision-resistant hash function $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ known to all the parties.

6.1 Committee-Based Reliable Broadcast

We briefly describe a committee-based reliable broadcast protocol that will prove useful in our improved ACS construction. The basis for the committee-based protocol is a plain reliable broadcast protocol Bcast that is t_s-valid and t_a-consistent with communication complexity $O(n^2\,|v|)$ a hash of the sender's input. (An example construction can be found in the full version.) The sender sends their input v individually to each of the committee members. If the hash output by the reliable broadcast matches this value, the committee members propagate v to all parties.

$\Pi^{t_a,t_s}_{\mathsf{BB+}}(v)$

Throughout, let $t_\kappa := \lfloor \frac{(1-\epsilon)\cdot\kappa\cdot t_s}{n} \rfloor$.

- If $P_i = P^*$: send input v to each $P_j \in C$, and input $h = H(v)$ to $\Pi^{t_s}_{\mathsf{BB}}$.
- Run $\Pi^{t_s}_{\mathsf{BB}}$.
- If $P_i \in C$ and $\Pi^{t_s}_{\mathsf{BB}}$ has output h': upon receiving a message v' from P^* or (v', h', π_j) from some $P_j \in C$, if $H(v') = h'$, multicast (v', h', π_i).
- Upon receiving $(v'', H(v''), \pi_j)$ from at least $t_\kappa + 1$ distinct $P_j \in C$ (even if $\Pi^{t_s}_{\mathsf{BB}}$ has not yet output a value), output v'' and terminate.

Fig. 4. A reliable broadcast protocol for sender P^* and committee C, from the perspective of party P_i.

The security analysis uses standard techniques for broadcast; for completeness, proofs can be found in the full version.

Communication complexity of $\Pi^{t_a,t_s}_{\mathsf{BB+}}$. Running the inner broadcast on hashes of size $O(\kappa)$ has communication complexity $O(n^2\kappa)$, while sending the value, hash, and credential to all parties costs $O(n\kappa(|m|+\kappa))$. Thus, sending a message of size $|m|$ using the 'wrapped' reliable broadcast costs $O(n^2\kappa + n|m|\kappa + n\kappa^2)$, while sending it using the inner reliable broadcast alone costs $O(n^2|m|)$.

6.2 Committee-Based ACS

We can construct a committee-based ACS protocol (Figure 5) by making two minor changes to the basic ACS protocol introduced in Section 4. First, the inner (non-terminating) ACS protocol is modified to use the committee-based broadcast described in Section 6.1. Because broadcast is used opaquely by the inner ACS protocol, this change does not require any special modifications, and the claims previously proven about the inner ACS protocol still hold. Second, the termination wrapper is modified so that only the members of the committee send the output in its entirety. Upon outputting a set S from the inner (non-terminating) ACS subprotocol, each committee member P_i multicasts S and $\langle \mathsf{commit}, H(S)\rangle_i$, along with the credential they received from the dealer. The other parties will echo signature shares and hashes, but not the set S itself.

$\Pi^{t_a,t_s}_{\mathsf{ACS+}}(v_j)$

Throughout, let $t_\kappa := \lfloor \frac{(1-\epsilon)\cdot\kappa\cdot t_s}{n} \rfloor$.

- Input v_j to $\Pi^{t_a,t_s}_{\mathsf{ACS}}$.
- If $P_j \in C$: upon receiving output S_j from $\Pi^{t_a,t_s}_{\mathsf{ACS}}$, compute $h := H(S_j)$ and multicast $(S_j, \langle \mathsf{commit}, h\rangle_j, \pi_j)$.
- Upon receiving at least $t_\kappa + 1$ valid signature shares $\sigma_i = \langle \mathsf{commit}, h\rangle_i$ from distinct parties in C on the same value h, form a combined signature σ for h and multicast (σ, h).
- Upon receiving a valid combined signature σ for some h, multicast σ.
- Upon holding S, σ such that σ is a combined signature for h from parties in C and S is a set such that $H(S) = h$, output S and terminate.

Fig. 5. A terminating ACS protocol with predetermined committee C, shown from the perspective of party P_j with input v_j.

The proof that $\Pi^{t_a,t_s}_{\mathsf{ACS+}}(v)$ is t_a-secure and has $\hat{t}_s$-validity with termination is very similar to the security proof for the basic ACS protocol, so we omit it.

Communication complexity of $\Pi^{t_a,t_s}_{\mathsf{ACS+}}$. As before, let $|m|$ represent the size of parties' inputs. When instantiated using the committee-based broadcast protocol from Section 6.1, the communication complexity of the inner ACS protocol is $O(n^3\kappa + n^2|m|\kappa + n^2\kappa^2)$. Moving on to the rest of the protocol, we see that the committee members multicast their output, a signature share, and the credential they received from the dealer. (Note that the signature share is for a hash of the output rather than the entire output.) Since the signature share, credential, and hash are each of size $O(\kappa)$, this step contributes $O(n \cdot \kappa(n \cdot |m| + \kappa)) = O(n^2\kappa \cdot |m| + n\kappa^2)$. Next, all parties multicast a combined signature of size $O(\kappa)$, for a total cost of $O(n^2\kappa)$. All together, the total cost of the improved ACS protocol is $O(n^3\kappa + n^2|m|\kappa + n^2\kappa^2)$.

6.3 An ABC Protocol with Improved Communication Complexity

Here, we give an overview of UPGRADE. Because the high-level techniques are similar to TARDIGRADE, we will highlight the key differences between the two protocols and defer further details to the full version.

The first (and simplest) difference is that wherever TARDIGRADE would run an instance of the plain ACS protocol, UPGRADE runs the improved version described in Section 6.2. The second difference concerns how parties choose and share their inputs, and how those inputs are combined to form a final block. At the beginning of the protocol, when parties choose a set of transactions to input, they will now also choose a second, larger input set, which is encrypted and sent only to the committee members. The committee members form the large ciphertexts into a separate pre-block, which is used to construct the final block in case ACS outputs only one small pre-block is output. Sending a large pre-block all-to-all is costly, so the committee members also form a placeholder called a *block pointer*. A block pointer contains a hash of a large pre-block and a combined signature on that hash by members of the committee. In most steps, the block pointer can be sent in place of the large pre-block. Although forming and sharing the block pointer adds some extra communication, we are able to significantly increase the expected number of distinct transactions.

References

1. I. Abraham, S. Devadas, D. Dolev, K. Nayak, and L. Ren. Efficient synchronous Byzantine consensus, 2017. Available at `https://eprint.iacr.org/2017/307`.
2. M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186. USENIX Association, 1999.
3. M. Correia, N. Neves, and P. Veríssimo. From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. *Comput. J.*, 49:82–96, 01 2006.
4. F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. *Information and Computation*, 118(1):158–179, 1995.
5. C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, Apr. 1988.
6. M. Fitzi and J. B. Nielsen. On the number of synchronous rounds sufficient for authenticated byzantine agreement. In I. Keidar, editor, *Distributed Computing*, pages 449–463. Springer Berlin Heidelberg, 2009.
7. J. Katz and C.-Y. Koo. On expected constant-round protocols for byzantine agreement. *JCSS*, 75(2):91–112, 2009.
8. R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative byzantine fault tolerance. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 45–58. ACM, 2007.
9. K. Kursawe. Optimistic byzantine agreement. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, SRDS '02, page 262. IEEE Computer Society, 2002.
10. L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine generals problem. *ACM Trans. Programming Language Systems*, 4(3):382–401, 1982.
11. S. Liu, P. Viotti, C. Cachin, V. Quema, and M. Vukolic. XFT: Practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 485–500, Savannah, GA, Nov. 2016. USENIX Association.

12. A. Momose and L. Ren. Multi-threshold byzantine fault tolerance. In *28th Conference on Computer and Communications Security (CCS)*, 2021. Available at `https://eprint.iacr.org/2017/307`.
13. R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology—Eurocrypt 2017, Part II*, volume 10211 of *LNCS*, pages 643–673. Springer, 2017.
14. A. Patra and D. Ravi. On the power of hybrid networks in multi-party computation. *IEEE Transactions on Information Theory*, 64(6):4207–4227, 2018.
15. M. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

A Useful Sub-Protocols

A.1 Reliable Broadcast with Higher Validity Threshold

In this section, we present a concrete protocol (cf. Protocol 6) that can be used to instantiate the broadcast subprotocol needed for our ACS construction. Our protocol is based on Bracha's (asynchronous) reliable broadcast protocol [**?**], but allows for a more general tradeoff between consistency and validity.

$\Pi^{t_s}_{\mathsf{BB}}(v^*)$

- Set $\mathsf{ready} = \mathsf{false}$.
- If $P_i = P^*$: multicast v^*.
- Upon receiving initial value v^* from P^*, multicast (echo, v^*).
- Upon receiving (echo, v^*) messages on the same value v^* from $n - t_s$ distinct parties: if $\mathsf{ready} = \mathsf{false}$, set $\mathsf{ready} = \mathsf{true}$ and multicast (ready, v^*).
- Upon receiving (ready, v^*) messages on the same value v^* from $t_s + 1$ distinct parties: if $\mathsf{ready} = \mathsf{false}$, set $\mathsf{readied} = \mathsf{true}$ and multicast (ready, v^*).
- Upon receiving (ready, v^*) messages on the same value v^* from $n - t_s$ distinct parties: output v^* and terminate.

Fig. 6. A reliable broadcast protocol with sender P^*, from the perspective of party P_i.

Lemma 13. *If $t_s < n/2$ then $\Pi^{t_s}_{\mathsf{BB}}$ is t_s-valid.*

Proof. Assume there are at most t_s corrupted parties, and the sender is honest. All honest parties receive the same value v^* from the sender, and consequently send (echo, v^*) to all other parties. Since there are at least $n - t_s$ honest parties, all honest parties receive (echo, v^*) from at least $n - t_s$ different parties, and as a result send (ready, v^*) to all other parties. By the same argument, all honest parties receive (ready, v^*) from at least $n - t_s$ parties, and so can output v^* (and terminate). Fix any $v \neq v^*$; to complete the proof, we argue that no honest party will output v. Note first that no honest party will send (echo, v). Thus, any honest party receives (echo, v) from at most t_s other parties. Since $t_s < n - t_s$, no honest party will ever send (ready, v). By the same argument, this shows that any honest party receives (ready, v) from at most t_s other parties, and hence will not output v. □

Lemma 14. *Fix $t_a \leq t_s$ with $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{BB}}^{t_s}$ is t_a-consistent.*

Proof. Suppose at most t_a parties are corrupted, and that an honest party P_i outputs v. Then P_i must have received (ready, v) from at least $n - t_s$ distinct parties, at least $n - t_s - t_a \geq t_s + 1$ of whom are honest. Thus, all honest parties receive (ready, v) from at least $t_s + 1$ distinct parties, and so all honest parties send (ready, v) to everyone. It follows that all honest parties receive (ready, v) from at least $n - t_a \geq n - t_s$ parties, and so can output v as well. To complete the proof, we argue that an honest party cannot output $v' \neq v$. We argued above that every honest party sends (ready, v) to everyone. Since $t_a < t_s + 1$, each honest party must have sent (ready, v) in response to receiving (echo, v) from at least $n - t_s$ distinct parties. If some honest party outputs v' then, arguing similarly, every honest party must have received (echo, v') from at least $n - t_s$ distinct parties. But this is a contradiction, since an honest party sends only a single echo message but $2 \cdot (n - t_s) - t_a > n$. □

A.2 A Block-Agreement Protocol

In this section, we show how to construct a block-agreement protocol that can be used in our atomic broadcast protocol in Section 5. Throughout this section, we assume the network is synchronous and at most $t < n/2$ parties are corrupted.

Recall from Definition 6 that block agreement is used to agree on pre-blocks, which are vectors of length n such that the i^{th} entry is either a valid signed message by P_i or simply $\perp$. A *k-quality pre-block* is a pre-block with at least k non-$\perp$ entries. Honest parties are assumed to input $(n - t)$-quality pre-blocks, and ignore any pre-blocks with quality less than $n - t$.

The structure of our block agreement protocol is inspired by the *synod protocol* of Abraham et al. [1]. We begin by defining a subprotocol $\Pi_{\mathsf{Propose}}^{P^*}$ (see Figure 7) in which a designated party P^* serves as a *proposer.* A tuple (r, β, C) is called a *round r vote* for a pre-block β if either:

- $r = 0$ and $C = \emptyset$, or
- $r > 0$ and C is a set of at least $t + 1$ signed messages $\langle \mathsf{commit}, r_i, \beta \rangle_i$ from distinct parties such that $r_i \geq r$.

When the exact value of r is unimportant, we simply refer to the tuple as a *vote.*

At the start of the propose protocol, the proposer waits to receive a set V of signed votes on valid pre-blocks such that $|V| \geq t + 1$. Then, the proposer determines a safe pre-block to propose from among these votes by finding the vote (r^*, β^*, C^*) in V such that r^* is greater than or equal to the round number of all other votes in V (breaking ties by lowest party index). The proposer then multicasts a *proposal* message $\langle \mathsf{propose}, (r^*, \beta^*, C^*), V\rangle_*$. An honest party who receives a proposal will consider it valid if all of the following hold:

- the signatures on the propose message and on each vote in V are valid,
- β^* is a valid pre-block,
- there is a round r^* vote for β^* in V,
- $|V|$ contains at least $t+1$ votes,
- r^* is greater than or equal to the round number of all votes in V.

If any of these conditions are not met, the proposal is not considered valid.

$\Pi^{P^*}_{\mathsf{Propose}}(r, \beta, C)$

1. (All parties) At time 0: send $\mathsf{vote}_i := \langle \mathsf{vote}, (r, \beta, C)\rangle_i$ to P^*.
2. (Only proposer) Until time Δ: Set $V = \emptyset$. Upon receiving a vote vote_j from party P_j on a valid pre-block: if this is the first such message received from P_j during this round, add vote_j to V.
3. (Only proposer) At time Δ, if $|V| \geq t$, find the vote (r^*, β^*, C^*) in V such that r^* is greater than or equal to the round number of all votes in V (breaking ties by lowest party index), and multicast $\langle \mathsf{propose}, (r^*, \beta^*, C^*), V\rangle_*$.
4. (All parties) At time 2Δ, if a valid $m = \langle \mathsf{propose}, (r^*, \beta^*, C^*), V\rangle_*$ has been received from P^*, multicast m. Otherwise, output $\perp$.
5. (All parties) At time 3Δ: let m_j denote the valid proposal forwarded by P_j (if any). If there exists m_j such that $m_j \neq m$, output $\perp$. Otherwise, output the pre-block β^* carried by the proposal m.

Fig. 7. A protocol $\Pi^{P^*}_{\mathsf{Propose}}$ parameterized by threshold t and designated proposer P^*, from the perspective of party P_i.

We first show that any two honest parties who generate output in this protocol agree on their output.

Lemma 15. *If honest parties P_i and P_j output $\beta_i, \beta_j \neq \perp$, respectively, in an execution of $\Pi^{P^*}_{\mathsf{Propose}}$, then $\beta_i = \beta_j$.*

Proof. If P_i outputs $\beta_i \neq \perp$, then P_i must have received a valid proposal message for β_i by time 2Δ. That message is forwarded by P_i to P_j, and hence P_j either outputs $\perp$ (if the proposals do not match) or the same value β_i. □

Next, we show that if each honest party P_i inputs a vote (r_i, β, C_i) on the same pre-block β, and no honest party ever receives a vote (r', β', C') such that $r' \geq \min_i\{r_i\}$ and $\beta' \neq \beta$, then any honest party who outputs a value other than $\perp$ outputs β.

Lemma 16. *If the input of each honest party P_i to $\Pi^{P^*}_{\mathsf{Propose}}$ is a round r_i-vote on the same valid pre-block β, and if no honest party ever receives a round r'-vote on $\beta' \neq \beta$ with $r' \geq \min_i\{r_i\}$, then every honest party outputs either β or $\bot$.*

Proof. Consider an honest party P who outputs $\beta \neq \bot$. That party must have received a valid proposal message from P^*, which in turn must contain a vote (r_i, β, C_i) from at least one honest party P_i. Under the assumptions of the lemma, any other vote (r', β', C') contained in the proposal message with $r' \geq r_i$ has $\beta' = \beta$. It follows that P outputs β. □

Finally, we show that when P^* is honest then all honest parties do indeed generate output.

Lemma 17. *If each honest party P_i inputs a vote (r_i, β_i, C_i) on some valid pre-block β_i to $\Pi^{P^*}_{\mathsf{Propose}}$, and P^* is honest, then there is some $(n-t)$-quality pre-block $\beta \neq \bot$ such that every honest party outputs β.*

Proof. P^* will receive at least $t+1$ votes from honest parties, and so sends a valid proposal message on some $(n-t)$-quality pre-block β to all honest parties. (It is possible for the set V to include votes from dishonest parties, but these votes must be on $(n-t)$-quality pre-blocks.) Since P^* is honest, and the adversary cannot forge signatures on other proposals behalf of P^*, this is the only valid proposal message the honest parties will receive. Therefore, all honest parties output $\beta \neq \bot$. □

We now present a protocol Π^{r}_{GC} (Figure 8) that uses $\Pi^{P^*}_{\mathsf{Propose}}$ to achieve a form of graded consensus on pre-blocks. As in the protocol of Abraham et al. [1], we rely on an atomic leader-election mechanism Leader with the following properties: On receiving input r from a majority of parties, Leader chooses a uniform leader $P^* \in \{1, \ldots, n\}$ and sends (r, P^*) to all parties. This ensures that if less than half of all parties are corrupted, then at least one honest party must call Leader with input r before the adversary can learn the identity of the leader. A leader-election mechanism tolerating any $t < n/2$ faults can be realized (in the synchronous model with a PKI) based on general assumptions [7]; it can also be realized more efficiently using a threshold unique signature scheme.

Below, we refer to a message $\langle \mathsf{commit}, r, \beta \rangle_i$ as a *valid commit message* from P_i on a pre-block β if the quality of β is at least $(n-t)$ and the associated signature is valid. Commit messages are used to construct *notify messages* $(\mathsf{notify}, r, \beta, C)$. A notify message $(\mathsf{notify}, r, \beta, C)$ is *valid* if β is an $(n-t)$-quality pre-block and C is a set of valid commit messages such that (1) all commit messages carry the same pre-block β, (2) C contains messages from at least $t+1$ distinct parties, and (3) for each $c_i = \langle \mathsf{commit}, r_i, \beta \rangle_i \in C$ the round number r_i is greater than or equal to r.

We refer to the value g in a tuple (β, C, g) as the *grade*.

Lemma 18. *Assume that the input of each honest party P_i to Π^{r}_{GC} is a vote on the same $(n-t)$-quality pre-block β. If no honest party ever receives a round*

$\mathit{\Pi}^r_{\mathsf{GC}}$

At time 0: Set $C' = \emptyset$. Call $\mathsf{Leader}(r)$ and let (r, P^*) denote the output. Run $\mathit{\Pi}^{P^*}_{\mathsf{Propose}}$ using input (r, β, C).

At time $3\mathit{\Delta}$: Let β^* denote the output of $\mathit{\Pi}^{P^*}_{\mathsf{Propose}}$. If $\beta^* \neq \bot$, multicast $\langle \mathsf{commit}, r, \beta^* \rangle_i$. Until time $4\mathit{\Delta}$, upon receiving a valid commit message $c_j = \langle \mathsf{commit}, r_j, \beta_j \rangle_j$ from P_j, if this is the first such message received from P_j, add c_j to C'.

At time $4\mathit{\Delta}$: If there is a subset $C'' \subseteq C'$ of commit messages on the same pre-block β' such that (a) $|C''| \geq t + 1$, and (b) for each $c_j = \langle \mathsf{commit}, r_j, \beta' \rangle_j \in C''$, $r_j \geq r$, then multicast $(\mathsf{notify}, r, \beta', C'')$, output $(\beta', C'', 2)$, and terminate.

At time $5\mathit{\Delta}$: If a valid notify message $(\mathsf{notify}, r, \beta^*, C^*)$ has been received, output $(\beta^*, C^*, 1)$ and terminate (if there is more than one such message, choose arbitrarily). Otherwise, output $(\bot, \bot, 0)$ and terminate.

Fig. 8. A graded consensus protocol from the perspective of party P_i with input (r, β, C).

r' vote on $\beta' \neq \beta$ such that r' is greater than or equal to the smallest round number carried by an honest parties' input in step 1 of $\mathit{\Pi}^r_{\mathsf{GC}}$, then (1) no honest party sends a commit message on $\beta' \neq \beta$ and (2) any honest party who outputs a nonzero grade outputs β.

Proof. By Lemma 16, every honest party outputs either β or $\bot$ in every execution of $\mathit{\Pi}_{\mathsf{Propose}}$ in step 1. It follows that no honest party P_i sends a commit message on $\beta' \neq \beta$, proving the first part of the lemma. Since at most t parties are corrupted, this means an honest party will receive fewer than $t+1$ valid commit messages on anything other than β; it follows that if an honest party outputs grade $g = 2$ then that party outputs $(\beta, C, 2)$.

Arguing similarly, no honest party will receive a valid notify message on anything other than β. Hence each honest party that outputs grade 1 outputs $(\beta, C', 1)$. □

Lemma 19. *If an honest party outputs (β, C, g) such that $g \neq 0$ in an execution of $\mathit{\Pi}^r_{\mathsf{GC}}$, then no honest party sends a commit message on $\beta' \neq \beta$.*

Proof. Say an honest party outputs (β, C, g) where g is nonzero. That party must have received a valid notify message on β. Therefore, C must contain signatures from at least $t + 1$ distinct parties. It follows that at least one honest party P must have sent a commit message on β. This means that P must have received β as its output from $\mathit{\Pi}^{P^*}_{\mathsf{Propose}}$. By Lemma 15, this means the pre-block output by any other honest party from $\mathit{\Pi}^{P^*}_{\mathsf{Propose}}$ is either β or $\bot$. □

Lemma 20. *If some honest party outputs (β, C, g) with grade $g = 2$ in an execution of $\mathit{\Pi}^r_{\mathsf{GC}}$, then each honest party P_i outputs (β_i, C_i, g_i) such that $\beta_i = \beta$ and $g > 0$.*

Proof. Say an honest party P outputs (β, C, g) such that $g = 2$. By Lemma 19, this means no honest party sent a commit message on $\beta' \neq \beta$; it is thus impossible for any honest party to output $\beta' \neq \beta$ with a nonzero grade. Since P sends a valid notify message on β to all honest parties before terminating, every honest party will output β with a nonzero grade. □

Lemma 21. *During an execution of Π^r_{GC}, the event that every honest party outputs the same $(n-t)$-quality pre-block β with a grade of 2 occurs with probability at least $1/2$.*

Proof. The leader P^* is honest with probability at least $\frac{n-t}{n} > 1/2$. If the leader is honest, agreement on an $(n-t)$-quality pre-block β follows from Lemma 20. Therefore, it remains to show that whenever the leader is honest, every honest party outputs grade 2.

Assume P^* is honest. Lemma 17 implies that every honest party receives the same pre-block $\beta \neq \perp$ as output from $\Pi^{P^*}_{\mathsf{Propose}}$. Thus, every honest party sends a valid commit message on β by time 3Δ. Consequently, each honest party P_j receives $n-t$ commit messages on the same pre-block β before time 4Δ. This causes them to output with grade $g = 2$. □

In Figure 9 we describe the complete block-agreement protocol Π_{BLA}. Note that parties do not terminate upon generating output; instead, parties terminate after participating in all κ rounds of the protocol.

```
$\Pi_{\mathsf{BLA}}(\beta)$

Set $(\beta', C') = (\beta, \emptyset)$ and set $r = 0$.
While $r < \kappa$:
    · At time $5r \cdot \Delta$: run $\Pi^r_{\mathsf{GC}}$ using input $(r, \beta', C')$.
    · At time $5(r+1) \cdot \Delta$: let $(\beta^*, C^*, g)$ denote the output of $\Pi^r_{\mathsf{GC}}$. If $g > 0$,
      set $(\beta', C') = (\beta^*, C^*)$. Additionally, if $g = 2$, output $\beta'$. Set $r = r + 1$.
```

Fig. 9. A block-agreement protocol Π_{BLA} with security parameter κ, from the perspective of party P_i.

Lemma 22. *If $t < n/2$, then Π_{BLA} is t-secure.*

Proof. Assume at most t parties are corrupted during an execution of Π_{BLA}. Termination follows trivially from the protocol description, as parties terminate after κ fixed-length rounds.

Let r^* be the first round in which some honest party outputs a pre-block β. We first show that in every subsequent round, the following hold: (1) every honest party P_i uses as its input in step 1 a round r_i vote on β; and (2) the adversary cannot construct a round r' vote on $\beta' \neq \beta$ for any $r' \geq \min_i\{r_i\}$.

Consider some honest party P_i who outputs a pre-block β in round r^*. P_i must have received a valid notify message for β during the graded consensus subprotocol for that round. By Lemma 20, this means every honest party received a valid notify message for β in the same execution of Π^r_{GC}, and so claim (1) holds in iteration r^*+1. Moreover, Lemma 19 implies that no honest party sent a commit message on $\beta' \neq \beta$ in the execution of Π^r_{GC}, and so claim (2) also holds in iteration r^*+1. Lemma 18 implies, inductively, that the two claims will continue to hold in every subsequent iteration. Thus, any other honest party P_j who generates output in Π_{BLA} also outputs β, regardless of whether they generate output in round r^* or a later round. This proves t-consistency.

Lemma 21 shows that in each iteration of Π_{BLA}, with probability at least $1/2$ there is an $(n-t)$-quality pre-block β such that all honest parties output β in that iteration. Thus, after κ iterations all honest parties have generated $(n-t)$-quality output except with negligible probability. This proves t-validity. □

Communication complexity of block agreement. During the propose subprotocol, parties send and receive votes. Recall that a vote is a tuple (r, β, C), where r is a constant, β is an input to the BLA protocol, and C is a set of $O(n)$ signatures σ_i on $(\mathsf{commit}, r_i, \beta)$. Because r_i is not necessarily equal to r_j for all σ_i, σ_j in C, the signatures cannot be combined into a single threshold signature. Thus, a vote is size $O(n\kappa + |m|)$, where $|m|$ denotes the size of parties' inputs. The most expensive step of the propose subprotocol requires all parties to send a vote to all other parties, resulting in an overall communication complexity for the propose subprotocol of $O(n^2(n\kappa + |m|)) = O(n^3\kappa + n^2|m|)$.

In the graded consensus subprotocol, the parties participate in one run of the propose subprotocol and send a constant number of all-to-all messages of size $O(n\kappa + |m|)$. Since both of these steps cost $O(n^3\kappa + n^2|m|)$, the overall communication complexity for one instance of graded consensus is the same as that of the propose protocol.

The BLA protocol runs κ iterations of the graded consensus protocol, for a total communication cost of $O(\kappa \cdot (n^3\kappa + n^2|m|)) = O(n^3\kappa^2 + n^2\kappa \cdot |m|)$.

B Proof of Lemma 12

Assume $t_a + 2t_s = n$ and fix an ABC protocol Π. Partition the n parties into sets S_0, S_1, S_a where $|S_0| = |S_1| = t_s$ and $|S_a| = t_a$. Consider the following experiment:

- Choose uniform $m_0, m_1 \leftarrow \{0,1\}^\kappa$. At global time 0, parties in S_0 begin running Π holding only m_0 in their buffer, and parties in S_1 begin running Π holding only m_1 in their buffer.
- All communication between parties in S_0 and parties in S_1 is blocked. All other messages are delivered within time Δ.
- Create virtual copies of each party in S_a, call them S_a^0 and S_a^1. Parties in S_a^b begin running Π (at global time 0) with their buffers containing only m_b, and communicate only with each other and parties in S_b.

Compare this experiment to a hypothetical execution E_{sync} of Π in a synchronous network, in which parties in S_1 are corrupted and simply abort, and the remaining parties are honest and initially hold only (uniformly chosen) m_0 in their buffer. The views of parties $S_0 \cup S_a^0$ in the experiment are distributed identically to the views of the honest parties in E_{sync}. Thus, t_s-liveness of Π implies that in the experiment, all parties in S_0 include m_0 in some block. Moreover, since parties in S_0 never receive information about m_1, they include m_1 in any block with negligible probability. By a symmetric argument, in the experiment, all parties in S_1 include m_1 in some block, and include m_0 in any block with negligible probability.

Now, consider a hypothetical execution E_{async} of Π, this time in an asynchronous network. In this execution, parties in S_0 and S_1 are honest while parties in S_a are corrupted. The parties in S_0 and S_1 initially hold $m_0, m_1 \leftarrow \{0,1\}^\kappa$, respectively. The corrupted parties interact with parties in S_0 as if they are honest and have m_0 in their buffer, and interact with parties in S_1 as if they are honest and have m_1 in their buffer. Meanwhile, all communication between parties in S_0 and S_1 is delayed indefinitely. The views of the honest parties in this execution are distributed identically to the views of $S_0 \cup S_1$ in the above experiment, yet the conclusion of the preceding paragraph shows that t_a-consistency is violated with overwhelming probability.

C Concrete Liveness Analysis

Imagine an external observer watching the protocol, with a clock running at rate ρ. (The observer's clock is not visible to the honest parties and is not assumed to be synchronized with parties' local clocks.) Let ρ_i denote the (possibly variable) rate at which P_i's local clock runs relative to the observer's clock.

Fix some finite interval $I = [\mathsf{Start}, \mathsf{End}]$ during an execution of the protocol. From the perspective of the observer, it is possible to identify bounds $\rho_{min}(I)$, $\rho_{max}(I)$ on the skew of honest parties clocks during interval I, so that for all honest P_i, $\rho_{min} \leq \rho_i \leq \rho_{max}$. The observer can also determine an upper bound $\delta(I)$ such that any message sent by time $T \in [\mathsf{Start}, \mathsf{End} - \delta]$ is delivered by time $T + \delta$. (Note that in an asynchronous network, $\delta(I)$ may be significantly greater than Δ.) Lastly, we let β_{max} denote the maximum number of transactions in any honest party's buffer during a given interval. We emphasize that ρ_{min}, ρ_{max}, δ, and β_{max} do not need to be known by the honest parties, and are used only for the analysis.

For each i and each k, let $\mathsf{Start}_{i,k}$ and $\mathsf{End}_{i,k}$ be the time according to the observer's clock when P_i begins iteration k and when P_i outputs block k, respectively, and let $I_{i,k}$ denote the interval $[\mathsf{Start}_{i,k}, \mathsf{End}_{i,k}]$. (By completeness of the protocol, $\mathsf{End}_{i,k}$ is well-defined for all i and k.)

The claims below apply in either setting (t_a corruptions in an asynchronous network, or t_s corruptions in a synchronous network); however, in a synchronous network the bounds are naturally simpler because we have $\rho_{min} = \rho_{max} = \rho$ and $\delta = \Delta$.

Lemma 23. *For any iteration k, the number of new blocks started by honest party P_i during the interval $I_{i,k} := [\mathsf{Start}_{i,k}, \mathsf{End}_{i,k}]$ is at most $\tau := \frac{\rho_{max}}{\lambda} \cdot \left(\frac{5\kappa\Delta+\Delta+A(\delta,\kappa)}{\rho_{min}}\right)$ (with overwhelming probability), where ρ_{min}, ρ_{max}, δ, β_{max}, and τ are measured by an external observer over the interval $I_{i,k}$, and $A(\delta, \kappa)$ is an upper bound such that the local running time of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ for P_i during the interval $I_{i,k}$ is at most $A(\delta, \kappa)$, with overwhelming probability in κ.*

Proof. Let ρ_i be the rate of P_i's local clock (or an upper bound on the rate, if it is variable). Each honest party P_i begins a new block every λ clock ticks, as measured by their local clock. Thus, the number of new blocks started by an honest party P_i during the interval $I_{i,k}$ is the length of $I_{i,k}$ (in global time) divided by λ/ρ_i.

We would like to find an upper bound on the length of $I_{i,k}$ for all honest P_i. The most significant contributors to the length of $I_{i,k}$ are the running time of Π_{BLA} and $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. The local running time of Π_{BLA} is at most $5\kappa\Delta + \Delta$ for any honest party, because P_i will timeout at this time if Π_{BLA} has not yet output. Thus, the running time of Π_{BLA} for P_i according to the observer's clock is at most $\frac{5\kappa\Delta+\Delta}{\rho_i}$. By assumption, $\rho_{min} \leq \rho_i$ for all P_i, and so the global running time of Π_{BLA} for any honest party is at most $\frac{5\kappa\Delta+\Delta}{\rho_{min}}$. Similarly, the running time of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ from the observer's perspective is bounded above by $\frac{A(\delta,\kappa)}{\rho_{min}}$ (with overwhelming probability in κ).

We can simply add the bounds for Π_{BLA} and $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ together to get an upper bound on the entire length of the interval $I_{i,k}$. Plugging this bound into the expression we had originally, we have the following bound on the number of new blocks started by any P_i during the interval $I_{i,k}$, which holds with overwhelming probability in κ:

$$\frac{|I_{i,k}|}{\lambda/\rho_i} \leq \frac{\frac{5\kappa\Delta+\Delta}{\rho_{min}} + A(\delta,\kappa)}{\frac{\lambda}{\rho_{max}}} = \frac{\rho_{max}}{\lambda} \cdot \left(\frac{5\kappa\Delta + \Delta + A(\delta,\kappa)}{\rho_{min}}\right) \tag{2}$$

This completes the proof. □

The following lemma concerns the overall progress of the honest parties.

Lemma 24. *Let t denote the number of dishonest parties during an execution of $\Pi_{\mathsf{ABC}}^{t_a,t_s}$, and let tx be a transaction that has been received by each honest P_i by time $\mathsf{Start}_{i,k}$. Let ρ_{min}, ρ_{max}, δ, β_{max}, and τ be bounds as described above over the interval $I^H_{k,k+c_x\cdot\tau} := [\min_{P_i\in H}(\mathsf{Start}_{i,k}), \max_{P_j\in H}(\mathsf{End}_{j,k+c})]$. Then with overwhelming probability in the security parameter κ, there are at least $n-t$ honest parties P_i such that P_i removes at least β_{max} transactions from their buffer during the interval $[\mathsf{Start}_{i,k}, \mathsf{Start}_{i,k+c_x\cdot\tau}]$, where $c_x := \frac{\beta_{max}}{L/n} \cdot \frac{n-t}{n-(t_s+t_a)}$.*

Proof. By Lemma 23, we know that every honest party P_i has output block k by time $\mathsf{Start}_{i,k+\tau}$. Therefore, by time $\mathsf{Start}_{i,k+\tau}$, P_i has removed from their buffer any transactions that are included in $\mathsf{Blocks}[k]$. In particular, if P_i's input was included in block k, then P_i must have removed at least L/n transactions from the front of their buffer between time $\mathsf{Start}_{i,k}$ and $\mathsf{Start}_{i,k+\tau}$.

Next, we can extend this argument to apply to sets of honest parties. Recall from Lemma 9 that at least $n-(t_s+t_a)$ honest parties' inputs are included in each block. Let $S^*_{k+c\cdot\tau}$ denote the set of honest parties whose inputs are included in block $k+c\cdot\tau$ $(c=0,1,2,\dots)$. For each $P_i \in S^*_{k+c\cdot\tau}$, notice that P_i must have selected L/n transactions from their buffer as input at time $\mathsf{Start}_{i,k+c\cdot\tau}$, and those transactions were included in block $k+c\cdot\tau$. Therefore, P_i must have removed at least L/n transactions from their buffer at some point during the interval $[\mathsf{Start}_{i,k+c\cdot\tau}, \mathsf{Start}_{i,k+(c+1)\tau}]$.

Consider a sequence of sets S^*_k, $S^*_{k+\tau}$, $S^*_{k+2\tau}$, $\dots$, defined as above. Suppose the adversary is able to choose S^* in each iteration, subject to the constraint that each S^* must contain at least $n-(t_s+t_a)$ honest parties. We would like to find an upper bound on number of iterations needed to ensure that all but t_s of the honest parties have tx among the first L transactions in their buffer. For convenience, assume that each honest parties initially has exactly β_{max} transactions in their buffer ahead of tx, and assume without loss of generality that parties $P_1,\dots,P_{n-t}$ are honest. In the worst case, the adversary chooses the honest parties for each set in the sequence in a round robin fashion, i.e.:

$$S^*_{k+c\cdot\tau} = \{P_i \mid 1+c\cdot(n-(t_s+t_a)) \le i \le c+c\cdot(n-(t_s+t_a)) \mod (n-t)\}. \quad (3)$$

Let $c_x := \frac{\beta_{max}}{L/n}\cdot\frac{n-t}{n-(t_s+t_a)}$, and consider the sequence of sets S^*_k, $\dots$,$S^*_{k+c_x\cdot\tau}$ determined according to the round robin strategy. All together, each honest party is in at least $\lfloor\frac{(n-(t_s+t_a))\cdot c_x}{n-t}\rfloor = \lfloor\frac{\beta_{max}}{L/n}\rfloor$ distinct sets in the sequence. Therefore, each honest party P_i has removed at least $\lfloor\frac{\beta_{max}}{L/n}\rfloor\cdot L/n = \beta_{max}$ transactions from their buffer during the interval $[\mathsf{Start}_{i,k}, \mathsf{Start}_{i,k+c_x\cdot\tau}]$. □

D Atomic Broadcast with Improved Complexity

D.1 Protocol Description

This section contains a detailed description of the improved atomic broadcast protocol, UPGRADE, which was deferred from Section 6.3. Pseudocode for the protocol is presented in Protocol 1, and related utilities are defined in Figure 2.

At the beginning of the protocol, when parties choose a set of transactions to input, they will now also choose a second, larger input set, which is encrypted and sent only to the committee members. The committee members use these ciphertexts to form a separate pre-block. (To distinguish between the two, we refer to a pre-block composed of smaller ciphertexts as a 'small pre-block' and a pre-block composed of larger ciphertexts as a 'large pre-block.') Sending this large pre-block all-to-all would increase the communication complexity beyond what we can afford. Thus, the committee members create a placeholder called a *block pointer* that can be sent in its place at various points during the protocol. A block pointer for a large pre-block β^{lg} consists of a hash $H(\beta^{lg})$ and a combined signature by the committee members on $H(\beta^{lg})$. We say that a block pointer $\tau = (h, \sigma)$ is *well-formed* if σ is a valid combined signature on h. Committee members will only create signature shares for a hash h if if they have received a pre-block β^{lg} such that $h = H(\beta^{lg})$; thus, a block pointer acts as a promise that a corresponding pre-block exists. Now, when parties provide input to BLA and ACS, they will input a pair (β^{sm}, τ) where β^{sm} is a small pre-block and τ is a block pointer.[3] We refer to a pair (β^{sm}, τ) as a *block share*, and say that a block share (β^{sm}, τ) is *well-formed* if β^{sm} is an $(n - \hat{t}_s)$-quality pre-block and τ is a well-formed block pointer. Because a block pointer is only size $O(\kappa)$, running BLA and ACS on block shares is only slightly more expensive than running BLA and ACS on small pre-blocks alone.

In the event that ACS outputs only a single block share $\{(\beta^{sm}, \tau)\}$, the committee members will wait to receive the large pre-block β^{lg} that matches τ, at which point they will send β^{lg} in full to all parties. In this case, the parties will use β^{lg} to construct the final block. Alternatively, if the output of ACS contains multiple block shares, the committee members do not need to send β^{lg} at all. Instead, the parties will use the small pre-blocks to form the final block.

To simplify the protocol description, we implicitly assume a means of domain separation between iterations. More precisely, we assume that parties can distinguish messages belonging to iteration k from messages belonging to iteration k' for any $k \neq k'$ in some way that does not rely on arrival time (e.g. by including a tag in each message), and a protocol message belonging to iteration k is only ever used in iteration k.

Communication complexity of $\Pi^{t_a,t_s}_{\mathsf{ABC^+}}$. We compute the communication complexity of each step of the protocol. First, each party P_i samples a set of L/n transactions, and divides this set into n subsets of equal size. The j^{th} subset is encrypted and sent to party P_j. Each small ciphertext is size $O(L|\mathsf{tx}|/n^2 + \kappa)$, so the total cost of this step is $O(n^2\kappa + L|\mathsf{tx}|)$. Separately, each party also chooses a large sample of L/n transactions, encrypts it using the committee's public key, and sends the resulting ciphertext of size $O(L|\mathsf{tx}|/n + \kappa)$ to only the committee members. The total cost of this step is $O(n\kappa(L|\mathsf{tx}|/n + \kappa)) = O(n\kappa^2 + L|\mathsf{tx}|\kappa)$.

As before, each party gathers the small ciphertexts they receive into a pre-block. Meanwhile, the members of the committee gather large ciphertexts into a separate pre-block. A small pre-block is size $O(L|\mathsf{tx}|/n + n\kappa)$, and a large pre-block is size $O(L|\mathsf{tx}| + n\kappa)$.

A member of the committee P_j who collects a large pre-block β will send $(\beta, \langle H(\beta)\rangle_j, \langle j\rangle_D)$ to all other members of the committee. These messages also have size $O(L|\mathsf{tx}| + n\kappa)$, because the size of the signatures is absorbed by the size of the large pre-block. Committee members will echo the first message $(\beta', \langle H(\beta')\rangle_i, \langle i\rangle_D)$ that they receive from each other committee member $P_i \in C$ (with their own signature and credential) until they either receive t_κ signatures on the same hash or they receive a block pointer directly, at which point they will

[3] ACS is agnostic to the type of input it receives, so it can be used as-is. BLA is not technically agnostic to the type of input, however, we can simply amend the definition of a valid proposal to include a well-formed block pointer.

multicast the block pointer and stop echoing signatures. Sending and echoing the large pre-blocks among the committee members incurs cost $O(\kappa^2(L|\mathsf{tx}|+n\kappa)) = O(n\kappa^3 + L|\mathsf{tx}|\kappa^2)$. A block pointer has size $O(\kappa)$, and so κ committee members multicasting a block pointer contributes only $O(n\kappa^2)$ communication.

Recall that in original version of the protocol, parties input pre-blocks to both BLA and ACS. In the new version, the inputs are block shares of size $O(L|\mathsf{tx}|/n + n\kappa)$. Previously, we showed that the communication complexity of BLA is $O(n^3\kappa^2 + n^2|m|\kappa)$ and the communication complexity of the improved ACS is $O(n^3\kappa + n^2|m|\kappa + n^2\kappa^2)$. Setting $|m| = O(L|\mathsf{tx}|/n + n\kappa)$, the communication complexity of both protocols simplifies to $O(n^3\kappa^2 + n\kappa L|\mathsf{tx}|)$.

After the parties receive a set of block shares as output from ACS, they must use threshold decryption to reveal which transactions will be included in the block. If the output of ACS contains only one block share, then the final block will be constructed using the block pointer. In this case, each committee member multicasts the corresponding large pre-block (waiting to receive it if necessary), as well as one decryption share for each of the $O(n)$ large ciphertexts in the block indicated by the block pointer, for a total of $O(n^2\kappa^2 + n\kappa L|\mathsf{tx}|)$. Conversely, if the output of ACS contains more than one block share, the final block will be constructed from the small pre-blocks. In this case, each committee member sends decryption shares for each of the ciphertexts in each of the small pre-blocks. The output contains up to $O(n)$ pre-blocks, each containing $O(n)$ ciphertexts. Therefore, each committee member sends $O(n^2)$ decryption shares to each party, for a total cost of $O(n\kappa(n^2\kappa)) = O(n^3\kappa^2)$.

Having computed the contribution of each step, we can see that the dominating terms arise from BLA, ACS, and forming the block pointer, for a total of $O(n^3\kappa^2 + n\kappa L|\mathsf{tx}| + n\kappa^3 + L|\mathsf{tx}|\kappa^2)$ per block.

In order to compute the amortized cost of a block per transaction, we first need to compute a lower bound on the expected number of distinct transactions per block. If the final block is constructed from a large pre-block indicated by a block pointer, then it must contain samples of size L/n from at least $n-(t_s+t_a)$ honest parties – precisely the same as the basic protocol. In this case, by Lemma 11, we know that the expected number of distinct transactions is at least $L/4$. If the final block is instead constructed from small pre-blocks, the calculation is not exactly the same as before, but the steps are largely the same. In this case, the final block will contain at least $n-(t_s+t_a)$ pre-blocks input by honest parties, and each of these pre-blocks contains samples of size L/n^2 from at least $n-(t_s+t_a)$ honest parties, for a total of $\frac{(n-(t_s+t_a))^2}{n^2} \cdot L$ honestly chosen transactions. As before, for a given transaction tx, the probability that some particular honest party's sample does not include tx is $1-\frac{1}{n^2}$. By Lemma 9, at least $n-(t_s+t_a) > n/3$ honest parties contribute transactions to B, and so the probability that none of those parties choose tx is at most $\left(1-\frac{1}{n^2}\right)^{n/3} \leq e^{-1/9} < 9/10$; thus, tx is chosen by at least one of those parties with probability at least $1/10$. (As before, the fact that parties' inputs are encrypted means that the adversary's ability to choose which ciphertexts are included in a block is irrelevant.) The lemma follows by linearity of expectation.

We have shown that each block contains at least $O(L)$ distinct transactions in expectation. Setting $L = n^2\kappa$, we obtain an amortized cost per transaction of $O(n\kappa|\mathsf{tx}| + \kappa^2|\mathsf{tx}|)$.

Security analysis. The improved protocol is secure for up to $\hat{t}_s$ corrupted parties in a synchronous network and up to t_a corrupted parties in an asynchronous network, where $n > 2t_s + t_a$ and $t_a \leq \hat{t}_s \leq (1 - 2\epsilon) \cdot t_s$. Security for the improved protocol can be argued very similarly to the original, so we will limit our attention to details that differ from the original.

Let us begin with the consistency property. First, consider the case in which the network is synchronous and there are at most $\hat{t_s}$ corrupted parties. In this case, any honest party's encrypted inputs will be received by all honest parties within time Δ, and so all honest parties hold an $n - \hat{t}_s$-quality small pre-block by time $T_k + \Delta$. Likewise, the honest committee members are able to assemble a block pointer and forward it to all parties by time $T_k + 4\Delta$. Thus, all parties input a well-formed block share to Π_{BLA}.

Π_{BLA} is secure for any $t < n/2$ parties in a synchronous network, and so we can rely on its security properties. In particular, consistency of Π_{BLA} ensures that all honest parties output the same well-formed block share (β^{sm}, τ) by time $T_k + 4\Delta + 5\kappa\Delta$. Therefore, all honest parties input (β^{sm}, τ) to the improved ACS protocol. By $\hat{t}_s$ validity of the ACS protocol, all honest parties receive (β^{sm}, τ) as output from ACS. By the security of the underlying threshold encryption scheme, all honest parties output the same block after running the block construction utility, as desired.

In the case that the network is asynchronous and there are most t_a corrupted parties, consistency of the improved protocol follows immediately from t_a-security of the ACS subprotocol.

Turning our attention to liveness, we note that the lower bounds on the number of honest parties' inputs included in the eventual output still hold when the inputs are block shares rather than sets of ciphertexts, because $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ is agnostic of the type of input value. Furthermore, as we showed earlier in this section, both the new protocol and original protocol produce blocks containing at least $O(L)$ distinct transactions in expectation. Thus, the arguments we used to prove liveness of the original protocol can be straightforwardly adapted to the improved protocol.

Finally, we remark that the proof of Theorem 3 can be repurposed almost verbatim to prove completeness of the improved protocol, using the security properties of the block agreement subprotocol and improved ACS subprotocol.

D.2 Proofs

Let $\chi_{s,n}$ denote the distribution that samples a subset of the n parties, where each party is included independently with probability s/n. We can use standard Chernoff bounds to prove the following useful facts about the composition of committees.

Protocol 1 $\Pi^{t_a,t_s}_{\mathsf{ABC^+}}$: an ABC protocol, described from the perspective of P_j.

```
for $k \in [1, 2, \ldots]$ do
    committee members:
        create a new large pre-block $\beta^{lg}_{j,k} = (\bot, \ldots, \bot)$ and set flag $\mathsf{ready}^{lg}_k = \mathsf{false}$
        upon receiving a large signed sample $(\mathsf{large}, \langle \nu \rangle_i)$ from $P_i$:
            if $\beta^{lg}_{j,k}[i] = \bot$: set $\beta^{lg}_{j,k}[i] = \nu_i$
            if $\beta^{lg}_{j,k}$ is $(n - \hat{t}_s)$-quality and $\mathsf{ready}^{lg}_k = \mathsf{false}$:
                set $\mathsf{ready}^{lg}_k = \mathsf{true}$ and send $(\beta^{lg}_{j,k}, \langle H(\beta^{lg}_{j,k}) \rangle_j, \langle j \rangle_D)$ to each $P_i \in C$
        while $\tau_k = \bot$:
            upon receiving the first valid $(\beta, \langle H(\beta) \rangle_i, \langle i \rangle_D)$ from $P_i \in C$:
                    send $(\beta, \langle H(\beta) \rangle_j, \langle j \rangle_D)$ to each $P_i \in C$
            upon receiving valid $(\beta', \langle H(\beta') \rangle_j, \langle j \rangle_D)$ (for the same $\beta'$)
              from $t_\kappa + 1$ distinct $P_j \in C$:
                combine signature shares into a signature $\sigma$ for $H(\beta')$
                set $\tau_k = (H(\beta'), \sigma)$ and multicast $\tau_k$
            upon receiving well-formed $\tau = (h, \sigma)$:
                set $\tau_k = \tau$ and multicast $\tau_k$
        upon receiving output $B^*$ from $\Pi^{t_a,t_s}_{\mathsf{ACS}}$, if $B^* = \{(\beta^{sm}_k, \tau^*)\}$:
            wait to receive $\beta$ such that $\tau^*$ points to $\beta$, then multicast $\beta$
    all parties:
        create a new small pre-block $\beta^{sm}_{j,k} := (\bot, \ldots, \bot)$ and block pointer $\tau_k = \bot$
        set flag $\mathsf{ready}^{sm}_k = \mathsf{false}$
        upon receiving a small signed sample $(\mathsf{small}, \langle \mu \rangle_i)$ from $P_i$:
            if $\beta^{sm}_{j,k}[i] = \bot$: set $\beta^{sm}_{j,k}[i] = \mu_i$
            if $\beta^{sm}_{j,k}$ is $(n - \hat{t}_s)$-quality and $\mathsf{ready}^{sm}_k = \mathsf{false}$: set $\mathsf{ready}^{sm}_k = \mathsf{true}$
        upon receiving a block pointer $\tau = (h, \sigma)$ from $P_i \in C$:
            if $\tau_k = \bot$: set $\tau_k = \tau$
        at time $T_k := \lambda \cdot (k - 1)$:
            choose a sample $V \leftarrow \mathsf{ProposeTxs}(L/n, L)$
            partition $V_j$ into $n$ sets $V_{j,1}, \ldots, V_{j,n}$ of size $L/n^2$, encrypt each set to
                form ciphertexts $\mu_{j,1}, \ldots, \mu_{j,n}$, and send $(\mathsf{small}, \langle \mu_{j,i} \rangle_j)$ to each $P_i$.
            choose a sample $W_j \leftarrow \mathsf{ProposeTxs}(L/n, L)$, and encrypt it to form
                a ciphertext $\mu$. Send $(\mathsf{large}, \langle \mu \rangle_j)$ to each $P_i \in C$.
        at time $T_k + 4\Delta$:
            if $\mathsf{ready}^{sm}_k = \mathsf{true}$ and $\tau \neq \bot$: input $(\beta^{sm}_{j,k}, \tau)$ to $\Pi_{\mathsf{BLA}}$
        at time $T_k + 4\Delta + 5\kappa\Delta$:
            stop running $\Pi_{\mathsf{BLA}}$
            if $\Pi_{\mathsf{BLA}}$ has output well-formed $(\beta^{sm}_k, \tau^*)$: input $(\beta^{sm}_k, \tau^*)$ to $\Pi^{t_a,t_s}_{\mathsf{ACS}}$
            else: wait for $\mathsf{ready}^{sm}_k = \mathsf{true}$ and $\tau \neq \bot$, then input $(\beta^{sm}_{j,k}, \tau)$ to $\Pi^{t_a,t_s}_{\mathsf{ACS}}$
            wait for $\Pi^{t_a,t_s}_{\mathsf{ACS}}$ to output $B^*$
            if $B^* = \{(\beta^{sm}_k, \tau^*)\} = 1$ :
                wait to receive $\beta^{lg}_k$ such that $\tau^*$ points to $\beta^{lg}_k$
                $B_{out} \leftarrow \mathsf{ConstructBlock}(\{\beta^{lg}_k\})$
            else:
                $B_{out} \leftarrow \mathsf{ConstructBlock}(\{\beta^{sm}_k \mid (\beta^{sm}_k, \tau) \in B^*\})$
            set $\mathsf{Blocks}[k] = B_{out}$ and delete all transactions $\mathsf{tx} \in \mathsf{Blocks}[k]$ from $\mathsf{buf}_j$
end for
```

Figure 2 Utilities for $\Pi_{\mathsf{ABC}}^{t_a,t_s}$, described from the point of view of party P_j

```
function $\mathsf{ProposeTxs}(\ell, M)$:
    choose $\ell$ values $v_1, \ldots, v_\ell$ uniformly at random (without replacement) from
        the first $M$ values in $\mathsf{buf}_j$
    output $\{v_1, \ldots, v_\ell\}$
function $\mathsf{ConstructBlock}(B)$:                ▷ $B$ is expected to be a set of pre-blocks
    for each pre-block $\beta$ in $B$, for each unique ciphertext $\mu$ in $\beta$:
        participate in threshold decryption for $\mu$
    upon completing decryption of all ciphertexts $\mu_1, \mu_2, \ldots$:
        output the set of transactions $B_{out} = \{\mathsf{tx} \mid \mathsf{tx} \in \mu_i\}$
```

Lemma 25 (Chernoff bound). *Let $X_1, ..., X_n$ be independent Bernoulli random variables with parameter p. Let $X := \sum_i X_i$, so $\mu := E[X] = p \cdot n$. Then, for $\delta \in [0, 1]$:*

- $\Pr[X \leq (1 - \delta) \cdot \mu] \leq e^{-\delta^2 \mu/2}$.
- $\Pr[X \geq (1 + \delta) \cdot \mu] \leq e^{-\delta^2 \mu/(2+\delta)}$.

Lemma 26. *Fix $s \leq n$ and $0 < \epsilon < 1/3$. If $C \leftarrow \chi_{s,n}$, then:*

1. *C contains fewer than $(1 + \epsilon) \cdot s$ parties except with probability $e^{-\frac{\epsilon^2 s}{2+\epsilon}}$.*
2. *C contains more than $(1 - \epsilon) \cdot s$ parties except with probability $e^{-\frac{\epsilon^2 s}{2}}$.*
3. *If there are at most $\hat{t}_s \leq (1 - 2\epsilon) \cdot t_s$ corrupted parties, then C contains fewer than $(1 - \epsilon) \cdot s \cdot \frac{t_s}{n}$ corrupted parties except with probability at most $e^{-\epsilon^2 s/(4-6\epsilon)}$.*
4. *If there are at most t_a corrupted parties, then C contains more than $(1 - \epsilon) \cdot s \cdot t_s/n$ honest parties except with probability at most $e^{-\frac{\epsilon^2 s}{3}}$.*

Proof. Let $H \subseteq [n]$ be the indices of the honest parties. Let X_j be the Bernoulli random variable indicating if $P_j \in C$, so $\Pr[X_j = 1] = s/n$. Define $Z_1 = \sum_j X_j$, $Z_2 := \sum_{j \notin H} X_j$, and $Z_3 := \sum_{j \in H} X_j$. Then:

1. Since $E[Z_1] = s$, setting $\delta = \epsilon$ in Lemma 25 yields
$$\Pr\left[Z_1 \geq (1 + \epsilon) \cdot s\right] \leq e^{-\epsilon^2 s/(2+\epsilon)}. \tag{4}$$
2. Using the other half of Lemma 25, setting $\delta = \epsilon$ yields
$$\Pr\left[Z_1 \leq (1 - \epsilon) \cdot s\right] \leq e^{-\epsilon^2 s/2}. \tag{5}$$
3. Since $E[Z_2] \leq \hat{t}_s \cdot s/n \leq (1 - 2\epsilon) \cdot t_s \cdot s/n$ and $t_s/n < 1/2$, setting $\delta = \frac{\epsilon}{1-2\epsilon}$ in Lemma 25 yields
$$\Pr\left[Z_2 \geq \frac{(1 - \epsilon) \cdot t_s \cdot s}{n}\right] \leq -e^{\epsilon^2 s/(4-6\epsilon)}. \tag{6}$$

4. Assuming that there are at most t_a corrupted parties, then $E[Z_3] \geq (n - t_a) \cdot s/n$. Thus, plugging in $\delta = \epsilon$, we have

$$\Pr\left[Z_3 \leq (1-\epsilon)(n-t_a) \cdot s/n\right] \leq e^{-\epsilon^2 \frac{(n-t_a)s}{2n}}. \tag{7}$$

Next, using the fact that $t_s/n < n/2$ and $(n - t_a)/n > 2n/3$, we see that

$$(1-\epsilon)s \cdot t_s/n < (1-\epsilon)s/2 < (1-\epsilon)s \cdot 2/3 < (1-\epsilon)s \cdot (n-t_a)/n. \tag{8}$$

Thus, $\Pr\left[Z_3 \leq (1-\epsilon)s \cdot t_s/n\right] \leq \Pr\left[Z_3 \leq (1-\epsilon)(n-t_a) \cdot s/n\right]$. Putting these two pieces together, we have $\Pr[Z_3 \leq (1-\epsilon)s \cdot t_s/n] \leq e^{-\frac{\epsilon^2(n-t_a)s}{2n}} \leq e^{-\frac{\epsilon^2 s}{3}}$ (note the last step is only used to simplify the bound).

□

Lemma 27. *$\Pi_{\mathsf{BB}^+}^{t_a,t_s}$ is $\hat{t}_s$-valid.*

Proof. Consider an execution in which the sender P^* is honest and holds input v. P^* will input $h = H(v)$ to $\Pi_{\mathsf{BB}}^{t_s}$ and send v to each member of the committee. By t_s-validity of $\Pi_{\mathsf{BB}}^{t_s}$, every honest party eventually receives output h such that $h = H(v)$ from the inner broadcast. Thus, each honest committee member P_i eventually sends $(v, h, \langle i \rangle_D)$, upon either receiving v directly from P^*, or receiving a matching $(v, h, \langle j \rangle_D)$ from another committee member. Additionally, no honest committee member will ever send a message $(v', h', \langle i \rangle_D)$ such that $h' \neq h$ or $v' \neq v$.

By Lemma 26, except with negligible probability, there are not enough malicious parties on the committee to cause an honest party to output $v' \neq v$. Furthermore, Lemma 26 also states that there are at least $t_\kappa + 1$ honest parties on the committee with overwhelming probability, so all honest parties will eventually receive enough messages on (v, h) to output v and terminate. □

Lemma 28. *$\Pi_{\mathsf{BB}^+}^{t_a,t_s}$ is t_a-consistent.*

Proof. Suppose some honest party P_i outputs a value v. P_i must have received messages of form $(v, h, \langle j \rangle_D)$ from at least $t_\kappa + 1$ distinct members of the committee. With overwhelming probability, at least one of these messages was sent by an honest committee member. Call that committee member P_j. An honest committee member will only send such a message after receiving h as output from the inner broadcast. By t_a-consistency of the inner broadcast, all parties eventually receive h as output from the inner broadcast. Hence, no honest committee member will ever send a message $(v', h', \langle j \rangle_D)$ for $v' \neq v$, because this would imply that either there is a collision such that $H(v) = H(v')$ for $v \neq v'$, or that they received $h' \neq h$ from the inner broadcast. By Lemma 26, with overwhelming probability there are at most t_κ malicious parties on the committee, and so with this same probability no honest party will receive the $t_\kappa + 1$ signatures for $v' \neq v$ required to output $v' \neq v$ during this execution.

Finally, as long as some honest committee member sent $(v, h, \langle j \rangle_D)$, that message will eventually arrive at all parties. This will cause any honest committee member who is still running to echo it, if they have not already. Since we showed above that no honest party can terminate with output $v' \neq v$, we see that all parties will eventually receive enough messages of form $(v, h, \langle j \rangle_D)$, at which point they will output v and terminate. □

Lemma 29. *Let $t_a \leq \hat{t}_s$. Then $\Pi^{t_a,t_s}_{\mathsf{ACS}^+}(v)$ is t_a-terminating.*

Proof. Consider a point during an execution of $\Pi^{t_a,t_s}_{\mathsf{ACS}^+}$ such that no honest party has seen a combined signature, and therefore, no honest parties have terminated. While this holds, the inner loop $\Pi^{t_a,t_s}_{\mathsf{ACS}}$ remains live due to the t_a-liveness property. Thus, at any (global) time t_1 during the execution such that no honest party has yet seen a combined signature, there must be some (global) time $t_2 > t_1$ at which an honest party in the committee outputs a value in the inner loop. Additionally, for any two honest parties that receive sets B, B' as output from $\Pi^{t_a,t_s}_{\mathsf{ACS}}$, we have $B = B'$. By Lemma 26, with overwhelming probability, at least $t_\kappa + 1$ honest parties are on the committee. This means that eventually some honest party will receive at least $t_\kappa + 1$ signed outputs on the same $h = H(B)$ from members of the committee, at which point they will form a combined signature, forward it to all parties, and begin waiting to receive B. The combined signature will eventually be delivered to any honest party who is still running, and hence they eventually also start waiting to receive such B. Because at least one of the signatures in the combined signature must have been contributed an honest committee member P' (with overwhelming probability, due to Lemma 26), every party will eventually receive the set B from P' and terminate. □