Network-Agnostic State Machine Replication

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

[1] University of Maryland College Park
[2] George Mason University

Abstract. We study the problem of *state machine replication* (SMR)—the underlying problem addressed by blockchain protocols—in the presence of a malicious adversary who can corrupt some fraction of the parties running the protocol. Existing protocols for this task assume either a *synchronous* network (where all messages are delivered within some known time Δ) or an *asynchronous* network (where messages can be delayed arbitrarily). Although protocols for the latter case give seemingly stronger guarantees, in fact they are incomparable since they (inherently) tolerate a lower fraction of corrupted parties.

We design an SMR protocol that is *network-agnostic* in the following sense: if it is run in a synchronous network, it tolerates t_s corrupted parties; if the network happens to be asynchronous it is resilient to $t_a \leq t_s$ faults. Our protocol achieves optimal tradeoffs between t_s and t_a.

1 Introduction

State machine replication (SMR) is a fundamental problem in distributed computing [17, 18, 30] that can be viewed as a generalization of *Byzantine agreement* (BA) [19, 29]. Roughly speaking, a BA protocol allows a set of n parties to agree on a value *once*, whereas SMR allows those parties to agree on an infinitely long *sequence* of values with the additional guarantee that values input to honest parties are eventually included in the sequence. (See Section 3 for formal definitions. Note that SMR is not obtained by simply repeating a BA protocol multiple times; see further discussion in Section 1.1.) Moreover, these properties should hold even in the presence of some fraction of corrupted parties who may behave arbitrarily. SMR protocols are deployed in real-world distributed data centers, and the problem has received renewed attention in the context of *blockchain* protocols used for cryptocurrencies and other applications.

Existing SMR protocols assume either a *synchronous network*, where all messages are delivered within some publicly known time bound Δ, or an *asynchronous network*, where messages can be delayed arbitrarily. Although it may appear that protocols designed for the latter setting are strictly more secure, this is not the case because they also (inherently) tolerate a lower fraction of corrupted parties. Specifically, assuming a public-key infrastructure is available to the parties, SMR protocols tolerating up to $t_s < n/2$ adversarial corruptions are possible in a synchronous network, but in an asynchronous network SMR is achievable only for $t_a < n/3$ faults (see [7]).

We study here so-called *network-agnostic* SMR protocols that offer meaningful guarantees regardless of the network in which they are run. That is, fix thresholds t_a, t_s with $0 \leq t_a < n/3 \leq t_s < n/2$. We seek to answer the following question: is it possible to construct an SMR protocol that (1) tolerates t_s (adaptive) corruptions if the network is synchronous, and moreover (2) tolerates t_a (adaptive) corruptions even if the network is asynchronous? We show that the answer is positive iff $t_a + 2t_s < n$.

Our work is directly inspired by recent results of Blum et al. [4], who study the same problem but for the simpler case of Byzantine agreement. We match their bounds on t_a, t_s; since SMR

* **Email:** erblum@cs.umd.edu

implies BA, even in the network-agnostic setting we consider (cf. Section 6.1), their impossibility result implies that the thresholds we obtain are optimal for our setting as well. While the high-level structure of our SMR protocol resembles the high-level structure of their BA protocol, in constructing our protocol we need to address several technical challenges (mainly due to the stronger liveness property required for SMR; see the next section) that do not arise in their work. Of additional interest, we also extend their impossibility result to show that it holds in a *proof-of-work* setting.

1.1 Related Work

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

As argued by Miller et al. [24], the well-known SMR protocols that tolerate malicious faults (e.g., [6, 15]) require at least partial synchrony in order to achieve liveness. Their HoneyBadger protocol [24] was designed specifically for fully asynchronous networks, but can only handle $t < n/3$ faults even if run in a synchronous network. Blockchain protocols are typically analyzed assuming synchrony [11, 25]; Nakamoto consensus, in particular, assumes that messages will be delivered much faster than the time required to solve proof-of-work puzzles.

We emphasize that SMR is *not* realized by simply repeating a (multi-valued) BA protocol multiple times. In particular, the validity property of BA only guarantees that if a value is input by all honest parties then that value will be output by all honest parties. In the context of SMR the parties each hold multiple inputs in a local buffer (where those inputs may arrive at arbitrary times), and there is no way to ensure that all honest parties will select the same value as input to some execution of an underlying BA protocol. Although generic techniques for compiling a BA protocol into an SMR protocol are known [7], those compilers are not network-agnostic and so do not suffice to solve our problem.

Our work focuses on protocols being run in a network that may be either synchronous or fully asynchronous. Other work looking at similar problems includes that of Malkhi et al. [23], who consider networks that may be either synchronous or *partially synchronous*; Liu et al. [20], 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. [12] and Abraham et al. [2], who consider temporary disconnections between two synchronous network components.

A slightly different line of work [21, 22, 26, 27] looks at designing protocols with good *responsiveness*. Roughly speaking, this means that the protocol still requires the network to be synchronous, but terminates more quickly if the actual message-delivery time is lower than the known upper bound Δ. Kursawe [16] designed a protocol for an asynchronous network that terminates more quickly if the network is synchronous, but does not tolerate more faults in the latter case.

Finally, other work [3, 8, 9, 28] considers a model where synchrony is available for some (known) limited period of time, but the network is asynchronous afterward.

1.2 Paper Organization

We define our model in Section 2, before giving definitions for the various tasks we consider in Section 3. In Section 4 we describe a network-agnostic protocol for the asynchronous common subset (ACS) problem. The ACS protocol is used as a sub-protocol of our main result, the network-agnostic SMR protocol, which is described and analyzed in Section 5.

In Section 6 we prove a lower bound showing that the thresholds we achieve are tight for network-agnostic SMR protocols, even when the protocol may rely on proofs of work. (This improves on the analogous result by Blum et al. [4], who do not consider proofs of work.) Toward this result, we show that a (network-agnostic) SMR protocol can be used to construct a (network-agnostic) BA protocol with the same thresholds, a result that may be of independent interest.

2 Model

Setup assumptions and notation. We consider a network of n parties $P_1, \ldots, P_n$ who communicate over point-to-point authenticated channels. We assume that the parties have established a public-key infrastructure prior to the protocol execution. That is, we assume that all parties hold the same vector $(pk_1, \ldots, pk_n)$ of public keys for a digital-signature scheme, and each honest party P_i holds the honestly generated secret key sk_i associated with pk_i. A *valid signature* σ *on* m *from* P_i is one for which $\mathsf{Vrfy}_{pk_i}(m, \sigma) = 1$. We treat signatures as ideal (i.e., perfectly unforgeable) for simplicity.

We also implicitly assume that parties use some form of domain separation when signing (e.g., use unique session IDs) to ensure that signatures are valid only for the context in which they are generated, and cannot be used in any other context.

Where applicable, we use κ to denote the security parameter for a protocol.

Adversarial model. We consider the security of our protocols in the presence of an adversary who can *adaptively* corrupt some number of parties. The adversary may coordinate the behavior of corrupted parties, and cause them to deviate arbitrarily from the protocol. Note, however, that our claims about adaptive security are only with respect to the property-based definitions found in Section 3, not with respect to a simulation-based definition (cf. [10, 13]). Finally, we assume that the adversary is able to choose corrupted parties' keys arbitrarily.

Network model. We consider two possible settings for the network. In a *synchronous* network, all messages are delivered within some known time Δ after they are sent, but the adversary can reorder and delay messages subject to this bound. (As a consequence, the adversary can potentially be *rushing*, i.e., it can wait to receive all incoming messages in a round before sending its own messages.) In this setting, we also assume all parties begin the protocol at the same time, and that parties' clocks progress at the same rate. When we say the network is *asynchronous*, we mean that the adversary can delay messages for an arbitrarily long period of time, though messages must eventually be delivered. We do not make any assumptions on parties' local clocks in the asynchronous case.

The network is either synchronous or asynchronous for the duration of the protocol (although we stress that the honest parties do not know which is the case).

3 Definitions

Although we are ultimately interested in state machine replication, our main protocol relies on various sub-protocols for different tasks. We therefore provide relevant definitions here.

Throughout, when we say that a protocol achieves some property, we mean that it achieves that property with overwhelming probability.

3.1 Useful Sub-Protocols

Throughout this section we consider protocols where, in some cases, parties may not terminate (even upon generating output); for this reason, we mention termination explicitly in the definitions. Honest parties are those who are not corrupted by the end of the execution.

Reliable broadcast. A *reliable broadcast* protocol allows parties to agree on a value chosen by a designated sender. In contrast to the stronger notion of *broadcast*, here honest parties might not terminate (but, if so, then none of them terminate).

Definition 1 (Reliable broadcast). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where a designated party $P^* \in \{P_1, \ldots, P_n\}$ 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 terminates, or else all honest parties output the same value $v \in \{0,1\}^*$.*

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

Byzantine agreement. A *Byzantine agreement* protocol allows parties who each hold some initial value to agree on an output value. We define a notion of Byzantine agreement that is weaker than usual, in that we do not require parties to terminate upon generating 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 the following holds whenever at most t of the parties are corrupted: every honest party outputs the same value $v \in \{0,1\}$.*

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

As an additional property (external to the definition of security), we say that an n-party BA protocol is *t-terminating* if it is guaranteed to terminate whenever at most t parties are corrupted.

Asynchronous common subset (ACS). 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 $\{v\}$.*
- **Liveness:** *Π is t-live if whenever at most t of the parties are corrupted, every honest party produces output.*
- **Consistency:** *Π is t-consistent if whenever at most t parties are corrupted, all honest parties output the same set S.*
- **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 inputs of at least $t+1$ honest parties.*

3.2 State Machine Replication

Protocols for *state machine replication* (SMR) allow parties to maintain agreement on an ever-growing, ordered sequence of *blocks*, where a block is a set of values called *transactions*. An SMR protocol does not terminate but instead continues indefinitely. We model the sequence of blocks output by a party P_i via a write-once array $\mathsf{Blocks}_i = \mathsf{Blocks}_i[0], \mathsf{Blocks}_i[1], \ldots$ maintained by P_i, each entry (or *slot*) of which is initially equal to $\perp$. We say that P_i *outputs a block in slot* j when P_i writes a block to $\mathsf{Blocks}_i[j]$; if $\mathsf{Blocks}_i[j] \neq \perp$ then we refer to $\mathsf{Blocks}_i[j]$ as the *block output by* P_i *in slot* j. We do not require that honest parties output a block in slot $j-1$ before outputting a block in slot j.

It is useful to define a notion of *epochs* for each party. (We stress that these are not global epochs; instead, each party maintains a local view of its current epoch.) Formally, we assume that each party P_i maintains a write-once array $\mathsf{Epochs}_i = \mathsf{Epochs}_i[0], \mathsf{Epochs}_i[1], \ldots$, each entry of which is initialized to 0. We say P_i *enters epoch* j when it sets $\mathsf{Epochs}_i[j] := 1$, and require:

- For $j > 0$, P_i enters epoch $j-1$ before entering epoch j.
- P_i enters epoch j before outputting a block in slot j.

An SMR protocol is run in a setting where parties asynchronously receive inputs (i.e., transactions) as the protocol is being executed; each party P_i stores transactions it receives in a local buffer buf_i. We imagine these transactions as being provided to parties by some mechanism external to the protocol (which could involve a gossip protocol run among the parties themselves), and make no assumptions about the arrival times of these transactions at any of the parties.

Definition 4 (State machine replication). *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 *and* Epochs *as described above.*

- **Consistency:** *Π is t-*consistent *if the following holds whenever at most t parties are corrupted: for all j, if an honest party outputs a block B in slot j then all parties that remain honest output B in slot j.*
- **Strong liveness:** *Π is t-*live *if the following holds whenever at most t parties are corrupted: for any transaction* tx *for which every honest party received* tx *before entering epoch j, every party that remains honest outputs a block that contains* tx *in some slot $j' \leq j$.*
- **Completeness:** *Π is t-*complete *if the following holds whenever at most t parties are corrupted: for all $j \geq 0$, every party that remains honest outputs some block in slot j.*

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

Our liveness definition is stronger than usual, in that we require a transaction tx that appears in all honest parties' buffers by epoch j to be included in a block output by each honest party in some slot $j' \leq j$. (Typically, liveness only requires that each honest party eventually outputs a block containing tx.) This stronger notion of liveness is useful for showing that SMR implies Byzantine agreement (see Section 6.1), and is achieved by our protocol.

In our definition, a transaction tx is only guaranteed to be contained in a block output by an honest party if *all* honest parties receive tx as input. A stronger definition would be to require this to hold even if only a *single* honest party receives tx as input. It is easy to achieve the latter from the former, however, by simply having honest parties gossip all transactions they receive to the rest of the network.

4 An ACS Protocol with Higher Validity Threshold

Throughout this section, we assume an asynchronous network.

Fix $t_a \leq t_s$ with $t_a + 2 \cdot t_s < n$. We show here an ACS protocol that is t_a-secure, and achieves validity even for t_s corruptions. Our construction follows the high-level approach taken by Miller et al. [24], who devise an ACS protocol based on sub-protocols for reliable broadcast and Byzantine agreement. In our case we need a reliable broadcast protocol that achieves validity for $t_s \geq n/3$ faults, and in Section 4.1 we show such a protocol. We then describe and analyze our ACS protocol in Section 4.2.

4.1 Reliable Broadcast with Higher Validity

In Figure 1, we present a variant of Bracha's (asynchronous) reliable broadcast protocol [5] that allows for a more general tradeoff between consistency and validity. Specifically, the protocol is parameterized by a threshold t_s; for any $t_a \leq t_s$ with $t_a + 2 \cdot t_s < n$, the protocol achieves t_a-consistency and t_s-validity.

Protocol $\Pi_{\mathsf{BB}}^{t_s}$

The sender P^* sends its input v^* to all parties. Then each party does:

- Upon receiving v^* from P^*, send (echo, v^*) to all parties.
- Upon receiving (echo, v^*) messages on the same value v^* from $n - t_s$ distinct parties, do: if (ready, v^*) was not yet sent, then send (ready, v^*) to all parties.
- Upon receiving (ready, v^*) messages on the same value v^* from $t_s + 1$ distinct parties, do: if (ready, v^*) was not yet sent, then send (ready, v^*) to all parties.
- Upon receiving (ready, v^*) messages on the same value v^* from $n - t_s$ distinct parties, output v^* and terminate.

Fig. 1. Bracha's reliable broadcast protocol, parameterized by t_s.

Lemma 1. *If $t_s < n/2$ then $\Pi_{\mathsf{BB}}^{t_s}$ 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).

To complete the proof, we also argue that honest parties cannot output $v \neq v^*$. Note first that no honest party will send (echo, v) for any $v \neq v^*$. Thus, any honest party will receive (echo, v) for some $v \neq v^*$ from at most t_s other parties. Since $t_s < n - t_s$, no honest party will ever send (ready, v) for any $v \neq v^*$. By the same argument, this shows that honest parties will receive (ready, v) for some $v \neq v^*$ from at most t_s other parties, and hence cannot output $v \neq v^*$. □

Lemma 2. *Let t_a be such that $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. 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) messages 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) messages from at least t_s+1 distinct parties, and so all honest parties send (ready, v) messages to everyone. It follows that all honest parties receive (ready, v) messages 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 honest parties cannot output $v' \neq v$. We argued above that all honest parties send (ready, v) to everyone. Let P be the first honest party to do so. Since $t_a < t_s+1$, that party must have sent (ready, v) in response to receiving (echo, v) messages from at least $n-t_s$ distinct parties. If some honest P_j outputs v' then, arguing similarly, some honest party P' must have received (echo, v') messages from at least $n-t_s$ distinct parties. But this is a contradiction, since honest parties send only a single echo message but $2\cdot(n-t_s)-t_a > n$. □

4.2 An ACS Protocol

In Figure 2 we describe an ACS protocol $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ that is parameterized by thresholds t_s, t_a, where $t_a \leq t_s$ and $t_a + 2\cdot t_s < n$. Our protocol relies on two sub-protocols: a reliable broadcast protocol Bcast that is t_s-valid and t_a-consistent (such as the protocol $\Pi_{\mathsf{BB}}^{t_s}$ from the previous section), and a Byzantine agreement protocol BA that is t_a-secure. (Since $t_a < n/3$, any asynchronous BA protocol secure for that threshold can be used.) Our ACS protocol runs several executions of these protocols as sub-routines, and so to distinguish between them we denote the ith execution by Bcast_i, resp., BA_i. We say that the executions $\mathsf{Bcast}_i, \mathsf{BA}_i$ *correspond to party* P_i.

As we will see in the analysis below, $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ is only t_a-live, not terminating (and in fact runs forever). Given that the SMR protocol itself runs indefinitely, it is reasonable to settle for an $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ protocol that runs forever but has bounded communication complexity; we prove that $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ has bounded communication complexity in Lemma 8 below). Likewise, the state for $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ may not be bounded, but since the state for $\Pi_{\mathsf{SMR}}^{t_s,t_a}$ is also unbounded, we consider this acceptable.

Lemma 3. *Say $t_s < n/2$ and at most t_s parties are corrupted. Then if an honest P_i uses input v_i in an execution of $\Pi_{\mathsf{ACS}}^{t_s,t_a}$, all honest parties receive output v_i from Bcast_i.*

Proof. The lemma follows from t_s-validity of Bcast. □

Before proceeding with the analysis, we note that because BA is t_a-secure even when the network is asynchronous, it remains t_a-consistent and t_a-valid once honest parties cease to participate after seeing C_1 become true. In the following lemmata, we will rely on this observation implicitly.

Lemma 4. *If $t_a + 2\cdot t_s < n$, then $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ is t_s-valid.*

Proof. Note that $t_s < n/2$. Say at most t_s parties are dishonest, and all honest parties have the same input v. Using Lemma 3, we see that at least $n-t_s$ executions of $\{\mathsf{Bcast}_i\}$ (namely, those for which P_i is honest) will result in v as output, and so all honest parties can take Exit 1 and output $\{v\}$. It is not possible for an honest party to take Exit 1 and output something other than $\{v\}$, since $t_s < n-t_s$. Thus, it only remains to show that if an honest party takes some other exit then it must also output $\{v\}$. Consider the two possibilities:

Exit 2: Suppose some honest party P takes Exit 2 and outputs $\{v'\}$. Then, for that party, $C_2(v')$ is true, and so P must have seen at least $\lfloor \frac{s}{2} \rfloor + 1$ of the $\{\mathsf{Bcast}_i\}_{i\in S^*}$ terminate with output v'. Moreover, P must have $\mathsf{ready} = \mathsf{true}$ and so $s \geq n-t_a$. Together, these imply that P has seen at least

$$\left\lfloor \frac{n-t_a}{2} \right\rfloor + 1 > \left\lfloor \frac{2t_s}{2} \right\rfloor + 1 > t_s$$

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

At any point during a party's execution of the protocol, let $S^* \stackrel{\text{def}}{=} \{i : \mathsf{BA}_i \text{ output } 1\}$ and let $s = |S^*|$. Define the following boolean conditions:

- $C_1(v)$: 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.
- $C_2(v)$: $\mathsf{ready} = \mathsf{true}$, all executions $\{\mathsf{BA}_i\}_{i\in[n]}$ have terminated, and a 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: $\mathsf{ready} = \mathsf{true}$, all executions $\{\mathsf{BA}_i\}_{i\in[n]}$ have terminated, and all executions $\{\mathsf{Bcast}_i\}_{i\in S^*}$ have terminated.

Each party initializes $\mathsf{ready} := \mathsf{false}$ and then does:

- For all i: run Bcast_i with P_i as the sender, where P_i uses input v_i.
- When Bcast_i terminates with output v_i' do: if execution of BA_i has not yet begun, run BA_i using input 1.
- When $s \geq n - t_a$, set $\mathsf{ready} := \mathsf{true}$ and run any executions $\{\mathsf{BA}_i\}_{i\in[n]}$ that have not yet begun, using input 0.
- **(Exit 1:)** If at any point $C_1(v)$ for some v, output $\{v\}$.
- **(Exit 2:)** If at any point $\neg C_1 \wedge C_2(v)$ for some v, output $\{v\}$.
- **(Exit 3:)** If at any point $\neg C_1 \wedge \neg C_2 \wedge C_3$, output $S := \{v_i'\}_{i\in S^*}$.

After outputting:

- Continue to participate in any still-running Bcast executions.
- Once $C_1 = \mathsf{true}$: Stop sending and receiving messages for any still-running BA executions.

Fig. 2. An ACS protocol, parameterized by t_s and t_a.

executions of $\{\mathsf{Bcast}_i\}$ terminate with output v'. At least one of those executions must correspond to an honest party. But then Lemma 3 implies that $v' = v$.

Exit 3: Assume an honest party P takes Exit 3. Then P must have $\mathsf{ready} = \mathsf{true}$ (and so $s \geq n - t_a$), must have seen all executions $\{\mathsf{BA}_i\}_{i\in[n]}$ terminate, and must also have seen all executions $\{\mathsf{Bcast}_i\}_{i\in S^*}$ terminate. Because

$$|S^*| = s \geq n - t_a > 2t_s,$$

a majority of the executions $\{\mathsf{Bcast}_i\}_{i\in S^*}$ that P has seen terminate must correspond to honest parties. Lemma 3 implies that all those executions must have resulted in output v. But then $C_2(v)$ must be true for P, and it would not have taken Exit 3. □

The following two lemmas prove that $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ is t_a-consistent and t_a-live. First, we show that if two honest parties each output a set, then those sets are equal. Then we show that all honest parties do indeed output a set.

Lemma 5. *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. Then if honest parties P_1 and P_2 output sets S_1 and S_2, respectively, in an execution of $\Pi_{\mathsf{ACS}}^{t_s,t_a}$, it holds that $S_1 = S_2$.*

Proof. We consider different cases based on the possible exits taken by the two honest parties, and show that in all cases their outputs agree.

Case 1: *Either P_1 or P_2 takes Exit 1.* Say P_1 takes Exit 1 and outputs $\{v_1\}$. (The case where P_2 takes Exit 1 is symmetric.) We consider different sub-cases:

- *P_2 takes Exit 1:* Say P_2 outputs $\{v_2\}$. Then P_1 and P_2 must have each seen at least $n - t_s$ executions of $\{\mathsf{Bcast}_i\}$ output v_1 and v_2, respectively. Since $t_s < n/2$, at least one of those executions must be the same. But then t_a-consistency of Bcast implies that $v_1 = v_2$.
- *P_2 takes Exit 2:* Say P_2 outputs $\{v_2\}$. For $C_2(v_2)$ to be satisfied, P_2 must have $s \geq n - t_a$, and must have seen at least
$$\left\lfloor \frac{s}{2} \right\rfloor + 1 \geq \left\lfloor \frac{n - t_a}{2} \right\rfloor + 1$$
executions of $\{\mathsf{Bcast}_i\}$ output v_2. As above, P_1 must have seen at least $n - t_s$ executions of $\{\mathsf{Bcast}_i\}$ output v_1. But since
$$(n - t_s) + \left\lfloor \frac{n - t_a}{2} \right\rfloor + 1 > n - t_s + \left\lfloor \frac{2t_s}{2} \right\rfloor + 1 > n,$$
at least one of those executions must be the same and so t_a-consistency of Bcast implies that $v_1 = v_2$.
- *P_2 takes Exit 3:* We claim this cannot occur. Indeed, if P_2 takes Exit 3 then P_2 must have $\mathsf{ready} = \mathsf{true}$ (and so $s \geq n - t_a$), and must have seen all executions $\{\mathsf{BA}_i\}_{i\in[n]}$ terminate and all executions $\{\mathsf{Bcast}_i\}_{i\in S^*}$ terminate. Because P_1 took Exit 1, P_1 must have seen at least $n - t_s$ executions $\{\mathsf{Bcast}_i\}_{i\in[n]}$ output v_1, and therefore (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 is therefore at least $(n - t_a) - t_s > t_s$, which is 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. But then $C_2(v_1)$ is true for P_2, and it would not take Exit 3.

Case 2: *Neither P_1 nor P_2 takes Exit 1.* We consider two sub-cases:

- *P_1 and P_2 both take Exit 2.* Say P_1 outputs $\{v_1\}$ and P_2 outputs $\{v_2\}$. Both P_1 and P_2 must have seen all $\{\mathsf{BA}_i\}$ terminate; by t_a-consistency of BA they must therefore hold the same S^*. Since $C_2(v_1)$ holds for P_1, it 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 $v_1 = v_2$.
- *Either P_1 or P_2 takes Exit 3.* Say P_1 takes Exit 3. (The case where P_2 takes Exit 3 is symmetric.) As above, P_1 and P_2 agree on S^* (this holds regardless of whether P_2 takes Exit 2 or Exit 3). Since C_3 holds for P_1 but C_2 does not, P_1 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_2 also does not see any value being output by a majority of those executions, and so will not take Exit 2. Since P_2 instead must take Exit 3, it must have seen all executions $\{\mathsf{Bcast}_i\}_{i\in S^*}$ terminate; t_a-consistency of Bcast then implies that P_2 outputs the same set as P_1. □

Lemma 6. *Fix t_a, t_s with $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ has t_a-set quality.*

Proof. Consider some honest party P. We again consider the various possibilities. Say P takes Exit 1 and outputs $S = \{v\}$. Then P has seen at least $n - t_s$ executions $\{\mathsf{Bcast}_i\}$ terminate with output v. Of these, at least $n - t_s - t_a > t_s \geq t_a$ must correspond to honest parties. By Lemma 3, those honest parties all had input v. This means that S contains the inputs of at least $t_a + 1$ honest parties.

Alternatively, say P takes Exit 2 or Exit 3 and outputs a set S. Then P holds $\mathsf{ready} = \mathsf{true}$, and so $|S^*| \geq n - t_a$. At least

$$n - 2 \cdot t_a > \max\{(n - t_a)/2,\ t_a\}$$

of the indices in S^* correspond to honest parties, and by Lemma 3 for each of those parties the corresponding output value v_i' that P holds is equal to that party's input. Thus, regardless of whether P takes Exit 2 (and S contains the majority value output by $\{\mathsf{Bcast}_i\}_{i \in S^*}$) or Exit 3 (and S contains every value output by $\{\mathsf{Bcast}_i\}_{i \in S^*}$), the set S output by P contains the inputs of at least $t_a + 1$ honest parties. □

Lemma 7. *Fix t_a, t_s with $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ is t_a-live.*

Proof. Assume at most t_a parties are corrupted during an execution of $\Pi_{\mathsf{ACS}}^{t_s,t_a}$. We consider two cases: either some honest party takes Exit 1 during this execution, or no honest party ever takes Exit 1 during this execution. In the first case, the first honest party to take Exit 1 must have seen at least $n - t_s$ executions $\{\mathsf{Bcast}_i\}_{i \in [n]}$ output with the same value v. Hence, t_a-consistency of Bcast implies that all other honest parties will eventually see at least those $n - t_s$ executions output v, and will output (if they have not already output via another exit).

In the second case, no honest party ever takes Exit 1. We argue that eventually all honest parties will set $\mathsf{ready} = \mathsf{true}$ and output. If no honest party ever takes Exit 1, then all honest parties continue to participate in any still-running BA executions indefinitely. At all times t' such that no honest party has yet set $\mathsf{ready} = \mathsf{true}$, for all $\{\mathsf{BA}_i\}_{i \in [n]}$, each honest party has either input 1 or not yet provided input. Such executions are indistinguishable from an execution in which all honest parties have input 1, but some messages have been delayed, and therefore t_a-validity of BA implies that (so long as no honest parties input 0) these executions eventually output 1. There are now two possibilities: either it continues to be true that no honest parties input 0 to any BA execution, or some honest party inputs 0 to some BA execution. In the first case, t_a-validity of BA implies that at least $n - t_a$ executions eventually output 1 for all honest parties, and therefore all honest parties set $\mathsf{ready} = \mathsf{true}$. In the latter case, some honest party must have already set $\mathsf{ready} = \mathsf{true}$ as a result of seeing at least $n - t_a$ BA executions output 1. Therefore, by t_a-consistency of BA, all honest parties will eventually see at least $n - t_a$ BA executions output 1, and therefore all honest parties set $\mathsf{ready} = \mathsf{true}$. Once $\mathsf{ready} = \mathsf{true}$, each honest party will output as soon as all $\{\mathsf{Bcast}_i\}_{i \in S^*}$ output. Each Bcast_i such that $i \in S^*$ is guaranteed to eventually output for the following reason: either the sender is honest, and Bcast_i terminates by Lemma 3, or the sender is dishonest, but by t_a-validity of BA, at least one honest party must have input 1 to BA_i as a result of seeing Bcast_i terminate, and therefore eventually everyone sees Bcast_i terminate due to t_a-consistency of Bcast.

Lemma 8. *Fix t_a, t_s with $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. If BA is t_a-terminating, then $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ has bounded communication complexity under both of the following conditions:*

1. *At most t_a parties are corrupted.*
2. *At most t_s parties are corrupted and all honest parties input the same value v.*

Proof. Because Bcast has bounded communication complexity, it remains to show that all honest parties eventually stop participating in all BA executions, either because they terminate or because they set $C_1 = \mathsf{true}$ and stop participating in any still-running executions. We consider each condition separately.

Case 1: *At most t_a parties are corrupted.* We must show that either all BA executions will eventually terminate, or all honest parties eventually set $C_1 = \mathsf{true}$ and stop participating in any still-running BA executions.

Assume no honest parties take Exit 1 during an execution of BA. Then all honest parties continue to participate in all BA executions, and so bounded complexity follows from t_a-termination of BA.

Now assume some party takes Exit 1 during an execution of $\Pi_{\mathsf{ACS}}^{t_s,t_a}$. That party must have seen at least $n - t_s$ executions $\{\mathsf{Bcast}_i\}_{i\in[n]}$ output with the same value. By t_a-consistency of Bcast, all honest parties eventually see those executions output with the same value, and thus can set $C_1 = \mathsf{true}$ and stop participating in any still-running BA executions.

Case 2: *At most t_s parties are corrupted and all honest parties input the same value v.* Because all honest parties input the same value v, t_s-validity of Bcast implies that all honest parties will eventually set $C_1 = \mathsf{true}$ and thus stop participating in any still-running BA executions.

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

Proof. Lemmas 5 and 7 together prove t_a-consistency. The theorem follows from Lemmas 4 and 6. □

5 A Network-Agnostic SMR Protocol

In this section, we show our main result: an SMR protocol that is t_s-secure in a synchronous network and t_a-secure in an asynchronous network. We begin in Section 5.1 by constructing a useful sub-protocol for what we call *block agreement*. We use this to construct an SMR protocol in Section 5.2.

5.1 Block Agreement

Throughout this section, we assume a synchronous network. We use $\langle m \rangle_i$ as a shorthand for (i, m, σ), where σ is a valid signature on message m signed using P_i's secret key.

We define here a notion we call *block agreement*, and show a block-agreement protocol secure against any $t < n/2$ corrupted parties. The structure of our protocol is inspired by the *synod protocol* of Abraham et al. [1]. Block agreement is a form of agreement where (1) in addition to an input, parties provide signatures (in a particular format) on those inputs, and (2) a stronger notion of validity is required. Specifically, consider pairs consisting of a block B along with a set Σ of signed buffers $\langle \mathsf{buf}_j \rangle_j$. We say a pair (B, Σ) is *valid* if:

- Σ contains signed buffers from strictly more than $n/2$ distinct parties.
- For each $\langle \mathsf{buf}_j \rangle_j \in \Sigma$, $\mathsf{buf}_j \subseteq B$. (Note each buffer can be represented as a bit-vector of length $|B|$.)

Definition 5 (Block agreement). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i begins holding input (B_i, Σ_i) and parties terminate upon generating output.*

- **Validity:** *Π is t-valid if whenever at most t of the parties are corrupted, then every honest party that outputs, outputs a t-valid pair.*

– **Termination:** *Π is t-*terminating *if the following holds when at most t of the parties are corrupted: every honest party outputs and terminates with probability $1 - 2^{-\kappa}$.*
– **Consistency:** *Π is t-*consistent *if the following holds when at most t of the parties are corrupted: if every honest party inputs a t-valid pair, there is a (B, Σ) such that every honest party outputs (B, Σ).*

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

We construct a block-agreement protocol in a modular fashion. We begin by defining a subprotocol $\Pi^{P^*}_{\mathsf{Propose}}$ (see Figure 3) in which a designated party P^* serves as a *proposer*. A tuple (k, B, Σ, C) is called a *k-vote on* (B, Σ) if (B, Σ) is valid and either:

– $k = 0$, or
– $k > 0$ and C is a set of valid signatures from a majority of the parties on messages of the form $(\mathsf{Commit}, k', B, \Sigma)$ with $k' \geq k$ (where possibly different k' can be used in different messages).

When the exact value of k is unimportant, we simply refer to the tuple as a *vote*. A message of the form $\mathsf{status} = \langle \mathsf{Status}, k, B, \Sigma, C \rangle_i$ is a *correctly formed* Status *message (from party P_i)* if (k, B, Σ, C) is a vote. A message $\langle \mathsf{Propose}, \mathsf{status}_1, \ldots \rangle_*$ is a *correctly formed* Propose *message* if it contains correctly formed Status messages from a majority of the parties.

Protocol $\Pi^{P^*}_{\mathsf{Propose}}$

We describe the protocol from the point of view of a party P_i with input a vote (k, B, Σ, C). Let $t = \lceil (n+1)/2 \rceil$.

1. At time 0, send $\mathsf{status}_i := \langle \mathsf{Status}, k, B, \Sigma, C \rangle_i$ to P^*.
2. At time Δ, if P^* has received at least $s \geq t$ correctly formed Status messages $\mathsf{status}_1, \ldots, \mathsf{status}_t$ (from distinct parties), then P^* sets
$$m := (\mathsf{Propose}, \mathsf{status}_1, \ldots, \mathsf{status}_s),$$
and sends $\langle m \rangle_*$ to all parties.
3. At time 2Δ, if a correctly formed Propose message $\langle m \rangle_*$ has been received from P^*, then send $\langle m \rangle_*$ to all parties. Otherwise, output $\perp$.
4. At time 3Δ, let $\langle m \rangle_*^j$ be the correctly formed Propose message received from P_j (if any). If there exists j such that $\langle m \rangle_*^j \neq \langle m \rangle_*$, output $\perp$. Otherwise, let $\mathsf{status}_{\mathsf{max}} = \langle \mathsf{Status}, k', B', \Sigma', C' \rangle$ be the status message in $\langle m \rangle_*$ with maximal k' (picking the lowest index in case of ties). Output (B', Σ').

Fig. 3. A protocol $\Pi^{P^*}_{\mathsf{Propose}}$ with designated proposer P^*.

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

Lemma 9. *If honest parties P_i and P_j output $(B_i, \Sigma_i), (B_j, \Sigma_j) \neq \perp$, respectively, in an execution of $\Pi^{P^*}_{\mathsf{Propose}}$, then $(B_i, \Sigma_i) = (B_j, \Sigma_j)$.*

Proof. If P_i outputs $(B_i, \Sigma_i) \neq \perp$, then P_i must have received a correctly formed Propose message $\langle m \rangle_*$ by time 2Δ that would cause it to output (B_i, Σ_i). That message is forwarded by P_i to P_j, and hence P_j either outputs $\perp$ (if it detects an inconsistency) or the same value (B_i, Σ_i).

Assume less than half the parties are corrupted. We show that if there is some (B, Σ) such that the input of each honest party P_i is a vote of the form (k_i, B, Σ, C_i), and no honest party ever receives a vote (k', B', Σ', C') with $k' \geq \min_i\{k_i\}$ and $(B', \Sigma') \neq (B, \Sigma)$, then the only value an honest party can output is (B, Σ).

Lemma 10. *Assume fewer than $n/2$ parties are corrupted, and that the input of each honest party P_i to $\Pi^{P^*}_{\mathsf{Propose}}$ is a k_i-vote on (B, Σ). If no honest party ever receives a k'-vote on $(B', \Sigma') \neq (B, \Sigma)$ with $k' \geq \min_i\{k_i\}$, then every honest party outputs either (B, Σ) or $\perp$.*

Proof. Consider an honest party P who does not output $\perp$. That party must have received a correctly formed $\mathsf{Propose}$ message $\langle m \rangle_*$ from P^*, which in turn must contain a correctly formed Status message from at least one honest party P_i. That Status message contains a vote (k_i, B, Σ, C_i) and, under the assumptions of the lemma, any other vote (k', B', Σ', C') contained in $\langle m \rangle_*$ with $k' \geq k_i$ has $(B', \Sigma') = (B, \Sigma)$. It follows that P outputs (B, Σ).

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

Lemma 11. *Assume fewer than $n/2$ parties are corrupted. If every honest party's input to $\Pi^{P^*}_{\mathsf{Propose}}$ is a vote and P^* is honest, then every honest party outputs the same valid $(B, \Sigma) \neq \perp$.*

Proof. Since every honest party's input is a vote, the honest P^* will receive at least $\lceil (n+1)/2 \rceil$ correctly formed Status messages, and so sends a correctly formed $\mathsf{Propose}$ message to all honest parties. Since P^* is honest, this is the only correctly formed $\mathsf{Propose}$ message the honest parties will receive, and so all honest parties will output the same valid $(B, \Sigma) \neq \perp$.

We now present a protocol Π^k_{GC} that uses $\Pi^{P^*}_{\mathsf{Propose}}$ to achieve a form of graded consensus on a valid pair (B, Σ). (See Figure 4.) As in the protocol of Abraham et al. [1], we rely on an atomic leader-election mechanism Leader with the following properties: On input k from a majority of parties, Leader chooses a uniform leader $\ell \in \{1, \ldots, n\}$ and sends (k, ℓ) 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 k before the adversary can learn the identity of ℓ. A leader-election mechanism tolerating any $t < n/2$ faults can be realized (in the synchronous model with a PKI) based on general assumptions [14]; it can also be realized more efficiently using a threshold unique signature scheme.

Below, we refer to a message $\langle \mathsf{Commit}, k, B, \Sigma \rangle_i$ as a *correctly formed* Commit *message* (*from* P_i *on* (B, Σ)) if (B, Σ) is valid. We refer to a message $(\mathsf{Notify}, k, B, \Sigma, C)$ as a *correctly formed* Notify *message on* (B, Σ) if (B, Σ) is valid and C is a set of valid signatures on $(\mathsf{Commit}, k, B, \Sigma)$ from more than $n/2$ parties; in that case, C is called a *k-certificate for* (B, Σ).

For an output $((B, \Sigma, C), g)$, we refer to g as the *grade* and (B, Σ, C) as the *output*. When a party's output is (B, Σ, C), we may also say that its output is a k-certificate for (B, Σ).

Lemma 12. *Assume fewer than $n/2$ parties are corrupted, and that the input of each honest party P_i to Π^k_{GC} is a k_i-vote on (B, Σ). If no honest party ever receives a k'-vote on $(B', \Sigma') \neq (B, \Sigma)$ with $k' \geq \min_i\{k_i\}$ in step 1 of Π^k_{GC}, then (1) no honest party sends a Commit message on $(B', \Sigma') \neq (B, \Sigma)$ and (2) any honest party who outputs a nonzero grade outputs a k-certificate for (B, Σ).*

Proof. By Lemma 10, every honest party outputs either (B, Σ) or $\perp$ in every execution of Π_{Propose} in step 1. It follows that no honest party P_i sends a Commit message on $(B', \Sigma') \neq (B, \Sigma)$, proving the first part of the lemma. Since less than half the parties are corrupted, this means an honest

Protocol Π^k_{GC}

We describe the protocol from the point of view of a party P_i with input a vote (k', B, Σ, C'). Let $t = \lceil (n+1)/2 \rceil$.

1. At time 0, run parallel executions of $\Pi^{P_1}_{\mathsf{Propose}}, \ldots, \Pi^{P_n}_{\mathsf{Propose}}$, each using input (k', B, Σ, C'). Let (B_j, Σ_j) be the output from the jth protocol.
2. At time 3Δ, call $\mathsf{Leader}(k)$ to obtain the response ℓ. If $(B_\ell, \Sigma_\ell) \neq \bot$, send $\langle \mathsf{Commit}, k, B_\ell, \Sigma_\ell \rangle_i$ to every party.
3. At time 4Δ, if at least t correctly formed Commit messages $\langle \mathsf{Commit}, k, B_\ell, \Sigma_\ell \rangle_j$ from distinct parties have been received, then form a k-certificate C for (B_ℓ, Σ_ℓ), send $m := (\mathsf{Notify}, k, B_\ell, \Sigma_\ell, C)$ to every party, output $((B_\ell, \Sigma_\ell, C), 2)$, and terminate.
4. At time 5Δ, if a correctly formed Notify message $(\mathsf{Notify}, k, B, \Sigma, C)$ has been received, output $((B, \Sigma, C), 1)$ and terminate. (If there is more than one such message, choose arbitrarily.) Otherwise, output $(\bot, 0)$ and terminate.

Fig. 4. A graded block-consensus protocol Π^k_{GC}, parameterized by k.

party will receive fewer than $\lceil (n+1)/2 \rceil$ correctly formed Commit messages on anything other than (B, Σ); it follows that if an honest party outputs grade $g = 2$ then that party outputs (B, Σ, C) with C a k-certificate for (B, Σ).

Arguing similarly, no honest party will receive a correctly formed Notify message on anything other than (B, Σ). Hence any honest party that outputs grade 1 outputs (B, Σ, C) with C a k-certificate for (B, Σ).

Lemma 13. *Assume fewer than $n/2$ parties are corrupted. If an honest party outputs (B, Σ, C) with a nonzero grade in an execution of Π^k_{GC}, then no honest party sends a Commit message on $(B', \Sigma') \neq (B, \Sigma)$.*

Proof. Say an honest party outputs (B, Σ, C) with a nonzero grade. That party must have received a correctly formed Notify message on (B, Σ). Since that Notify message includes a k-certificate C with signatures from more than half the parties, at least one honest party P must have sent a Commit message on (B, Σ). This means that P must have received (B, Σ) as its output from $\Pi^{P_\ell}_{\mathsf{Propose}}$. By Lemma 9, this means the output of any other honest party from $\Pi^{P_\ell}_{\mathsf{Propose}}$ is either (B, Σ) or $\bot$. The lemma follows.

Lemma 14. *Assume fewer than $n/2$ parties are corrupted. If an honest party outputs (B, Σ, C) with grade 2 in an execution of Π^k_{GC}, then every honest party outputs a k-certificate on (B, Σ) with a nonzero grade.*

Proof. Say an honest party P outputs (B, Σ, C) with a grade of 2. By Lemma 13, this means no honest party sent a correctly formed Commit message on $(B', \Sigma') \neq (B, \Sigma)$; it is thus impossible for any honest party to output $(B', \Sigma') \neq (B, \Sigma)$ with a nonzero grade. Since P sends a correctly formed Notify message on (B, Σ) to all honest parties, every honest party will output (B, Σ) with a nonzero grade.

Lemma 15. *Assume fewer than $n/2$ parties are corrupted. Then with probability at least $1/2$ every honest party outputs a k-certificate on the same valid (B, Σ) with a grade of 2.*

Proof. The leader ℓ chosen in step 2 was honest in step 1 with probability at least $1/2$. We show that whenever this occurs, every honest party outputs grade 2. Agreement on a valid (B, Σ) follows from Lemma 14.

Assume ℓ was honest in step 1. Lemma 11 implies that every honest party holds the same valid $(B_\ell, \Sigma_\ell) \neq \perp$ in step 2, and so sends a correctly formed Commit message on (B_ℓ, Σ_ℓ). Since there are at least $\lceil (n+1)/2 \rceil$ honest parties, the lemma follows.

In Figure 5 we describe our block-agreement protocol $\Pi_{\mathsf{BLA}}^{t_s}$.

Protocol $\Pi_{\mathsf{BLA}}^{t_s}$

We describe the protocol from the point of view of a party P with input a valid pair (B, Σ).
Initialize $(k^*, B^*, \Sigma^*, C^*) := (0, B, \Sigma, \emptyset)$ and $k := 1$. While $k \leq \kappa$ do:

1. At time $(5k - 5) \cdot \Delta$, run Π_{GC}^{k} using input $(k^*, B^*, \Sigma^*, C^*)$ to obtain output $((B, \Sigma, C), g)$.
2. At time $5k \cdot \Delta$ do: If $g > 0$, set $(k^*, B^*, \Sigma^*, C^*) := (k, B, \Sigma, C)$. If $g = 2$, output (B, Σ). Increment k.

Fig. 5. A block-agreement protocol $\Pi_{\mathsf{BLA}}^{t_s}$.

Lemma 16. *If $t < n/2$, then $\Pi_{\mathsf{BLA}}^{t_s}$ is t-secure.*

Proof. Assume fewer than $n/2$ parties are corrupted. Let k be the first iteration in which some honest party outputs (B, Σ). We first show that in every subsequent iteration: (1) every honest party P_i uses as its input in step 1 a k_i-vote on (B, Σ); and (2) corrupted parties cannot construct a k'-vote on $(B', \Sigma') \neq (B, \Sigma)$ for any $k' \geq \min_i\{k_i\}$.

Say an honest party outputs (B, Σ) in iteration k. Then that party must have output a k-certificate for (B, Σ) in the execution of Π_{GC}^{k} in iteration k. By Lemma 14, this means every honest party output a k-certificate on (B, Σ) in the same execution of Π_{GC}^{k}, and so (1) holds in iteration $k + 1$. Moreover, Lemma 13 implies that no honest party sent a Commit message on $(B', \Sigma') \neq (B, \Sigma)$ in the execution of Π_{GC}^{k}, and so (2) also holds in iteration $k + 1$. Lemma 12 implies, inductively, that the stated properties continue to hold in every subsequent iteration.

It follows from Lemma 12 that any other honest party P who generates output in $\Pi_{\mathsf{BLA}}^{t_s}$ also outputs (B, Σ), regardless of whether they generate output in iteration k or a subsequent iteration.

Lemma 15 shows that in each iteration of $\Pi_{\mathsf{BLA}}^{t_s}$, with probability at least 1/2 all honest parties output some (the same) valid (B, Σ) in that iteration. Thus, after κ iterations all honest parties have generated output with probability at least $1 - 2^{-\kappa}$ (note that all parties terminate after κ iterations).

5.2 State Machine Replication

At a high level, the protocol proceeds as follows. The parties attempt to achieve agreement on a block for each slot j using the block agreement protocol $\Pi_{\mathsf{BLA}}^{t_s}$. If that protocol succeeds in reaching agreement, parties use its output B as input to the ACS protocol $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ from the previous section. This ensures that if the network is synchronous and at most t_s parties are corrupted, then all parties agree on their input B to $\Pi_{\mathsf{ACS}}^{t_s,t_a}$; hence, t_s-validity of $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ ensures that all parties output B. On the other hand, if $\Pi_{\mathsf{BLA}}^{t_s}$ fails to output in a certain amount of time, parties abandon it and instead attempt to reach agreement using the ACS protocol directly. Now, t_a-consistency and set quality of the ACS protocol ensure agreement (for at most t_a corruptions) even if the network happens to be asynchronous.

In our protocol for state machine replication, parties agree on an ordered sequence of blocks. The data included in each block is determined by the output of the ACS subprotocol. Parties begin agreement on each new block at regular intervals (thanks to their synchronized clocks). All parties begin each iteration at the same time as long as the network is synchronous, though different parties may finish each block at different times, and indeed some parties may continue participating in an earlier iteration while at the same time beginning to participate in a new iteration. On the other hand, if the network is asynchronous, parties might not start iterations at the same time and the protocol must continue to ensure secure progress. In the following, we show how this can be achieved by relying on the network-agnostic properties of $\Pi_{\mathsf{ACS}}^{t_s,t_a}$.

Protocol $\Pi_{\mathsf{SMR}}^{t_s,t_a}$

We describe the protocol from the point of view of a party P_i.

To agree on $\mathsf{Blocks}_i[k]$, execute the following steps, starting at time $5\kappa\Delta \cdot (k-1)$:

1. Set $\mathsf{Epochs}_i[k] := 1$, and initialize $B := \emptyset, \Sigma := \emptyset$.
2. Send $\langle \mathsf{buf}_i \rangle_i$ to every party.
3. While $|\Sigma| \leq t_s$:
   - Denote as m_j the message $\langle \mathsf{buf}_j \rangle_j$ received from party P_j.
   - Set $B := B \cup \mathsf{buf}_j, \Sigma := \Sigma \cup \{m_j\}$
4. At time $5\kappa\Delta \cdot (k-1) + 1$, run $\Pi_{\mathsf{BLA}}^{t_s}$ on input (B, Σ). Let (B', Σ') denote the output or set $B' := \bot$ if there is no output at time $5\kappa\Delta k + 1$.
5. If $B' \neq \bot$, set $B^* := B'$; else set $B^* := B$. Run $\mathsf{BlockSet} \leftarrow \Pi_{\mathsf{ACS}}^{t_s,t_a}$ using input B^*.
6. Set $\mathsf{Blocks}_i[k] := \bigcup_{\hat{B} \in \mathsf{BlockSet}} \hat{B}$. Set $\mathsf{buf} := \mathsf{buf} \setminus \mathsf{Blocks}_i[k]$.

Fig. 6. A protocol for state machine replication.

Theorem 2 (Consistency). *Fix t_a, t_s with $t_a < n/3$ and $t_a + 2 \cdot t_s < n$. The following guarantee holds for any execution of $\Pi_{\mathsf{SMR}}^{t_s,t_a}$ for which either (1) at most t_s parties are dishonest and the network is synchronous, or (2) at most t_a parties are dishonest and the network is asynchronous: If two honest parties P_i and P_j output $\mathsf{Blocks}_i[k]$ and $\mathsf{Blocks}_j[k]$, respectively, in slot k, then $\mathsf{Blocks}_i[k] = \mathsf{Blocks}_j[k]$.*

Proof. We consider two cases. In the first, at most t_s parties are dishonest and the network is synchronous, in the second, at most t_a parties are dishonest and the network is asynchronous.

Case 1. In this case, by consistency and validity of $\Pi_{\mathsf{BLA}}^{t_s}$, all parties receive output (B, Σ) from the block-agreement subprotocol after time at most $5\kappa\Delta$, and furthermore, (B, Σ) must be t_s-valid. By t_s-validity of $\Pi_{\mathsf{ACS}}^{t_s}$ (Lemma 4), any two honest parties P_i and P_j receive output $\mathsf{BlockSet} = \{B\}$ from $\Pi_{\mathsf{ACS}}^{t_s}$. Therefore, both parties set $\mathsf{Blocks}_i[k] = \mathsf{Blocks}_j[k] = B$ and terminate the subprotocol for slot k.

Case 2. In this case, by t_a-consistency of $\Pi_{\mathsf{ACS}}^{t_s}$, we have that all honest parties agree on the same set $\mathsf{BlockSet}$. Hence, everybody outputs the same block for slot k and therefore in particular, $\mathsf{Blocks}_i[k] = \mathsf{Blocks}_j[k]$. □

Theorem 3 (Strong liveness). *Fix t_a, t_s with $t_a < n/3$ and $t_a + 2 \cdot t_s < n$. The following guarantee holds for any execution of $\Pi_{\mathsf{SMR}}^{t_s,t_a}$ for which either (1) at most t_s parties are dishonest and the network is synchronous, or (2) at most t_a parties are dishonest and the network is asynchronous:*

If a transaction tx *has been received by every honest party before entering epoch* k*, then for every honest party* P_i*,* $\mathsf{tx} \in \mathsf{Blocks}_i[k']$ *such that* $k' \leq k$*.*

Proof. We consider two cases. In the first, at most t_s parties are dishonest and the network is synchronous, in the second, at most t_a parties are dishonest and the network is asynchronous. Note that in either case, all parties eventually hold a set B of size $t_s + 1$, since there are at most $n - t_s > t_a + t_s > t_s$ corruptions in either case (since $t_a \leq t_s$).

Case 1. In this case, by consistency and validity of $\Pi_{\mathsf{BLA}}^{t_s}$, all parties receive output (B, Σ) from the block-agreement subprotocol after time at most $5\kappa\Delta$, and furthermore, (B, Σ) must be t_s-valid. By t_s-validity of $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ (Lemma 4), all parties receive output $\{B\}$ from the ACS subprotocol and hence output $\{B\}$ as the block for slot k. Since (B, Σ) is t_s-valid, it includes all the transactions held in some honest party P_j's buffer at the point in time that it entered epoch k. Hence, either tx was in P_j's buffer (and so B includes tx), or tx was *not* in P_j's buffer, which implies that P_j has already output a block $\mathsf{Blocks}_j[k']$ such that $k' < k$ and $\mathsf{tx} \in \mathsf{Blocks}_j[k']$ (and so by Theorem 2, all other honest parties eventually output that same block).

Case 2. In this case, note that honest parties only supply B^* such that either (B^*, Σ) or (B^*, Σ') is t_s-valid as input to $\Pi_{\mathsf{ACS}}^{t_s,t_a}$ (since they input either B or B'). By t_a-consistency and t_a-set quality of $\Pi_{\mathsf{ACS}}^{t_s,t_a}$, we have that all parties agree on a set $\mathsf{BlockSet}$ containing at least $t_a + 1$ honest blocks. In particular, all of these honest blocks contain the buffer buf of an honest party P_j at the point it time that it entered epoch k. Hence, either tx was in P_j's buffer (and so $\hat{B} \in \mathsf{BlockSet}$ includes tx), or tx was *not* in P_j's buffer, which implies that P_j has already output a block $\mathsf{Blocks}_j[k']$ such that $k' < k$ and $\mathsf{tx} \in \mathsf{Blocks}_j[k]$ (and so by Theorem 2, all other honest parties eventually output that same block). In either case, there exists an epoch $k' \leq k$ such that all honest blocks in $\mathsf{BlockSet}$ include tx, and it immediately follows that the union of blocks in $\mathsf{BlockSet}$ also contains tx. ◻

Theorem 4 (Completeness). *Fix* t_a, t_s *with* $t_a < n/3$ *and* $t_a + 2 \cdot t_s < n$*. The following guarantee holds for any execution of* $\Pi_{\mathsf{SMR}}^{t_s,t_a}$ *for which either (1) at most* t_s *parties are dishonest and the network is synchronous, or (2) at most* t_a *parties are dishonest and the network is asynchronous: for all* k*, any honest party* P_i *eventually outputs a block* $\mathsf{Blocks}_i[k]$ *for slot* k*.*

Proof. Regardless of whether the protocol is run in a synchronous network with at most t_s corruptions, or in an asynchronous network with at most t_a corruptions, the lemma follows by the liveness and termination properties of the sub-protocols used. ◻

6 A Lower Bound in the Proof-of-Work Setting

In this section we show that the parameters achieved by our protocol are optimal, even in the proof-of-work (PoW) setting. To do so, we first show that any (network-agnostic) SMR protocol can be used to construct a (network-agnostic) BA protocol with the same thresholds. This may be of independent interest. We then prove that the parameters we achieve for BA are optimal. This extends the analogous result by Blum et al. [4], who do not consider the PoW setting.

6.1 SMR Implies BA

In Figure 7 we show a BA protocol that relies on an SMR protocol Π_{SMR} as a subroutine. In the following, we once again use $\langle v_i \rangle_i$ as a shorthand for (i, m, σ), where $\sigma_i \leftarrow \mathsf{Sign}(sk_i, v_i)$.

Protocol Π_{BA}

We describe the protocol from the point of view of a party P_i with input v_i.

- Set $V_i := \emptyset$.
- Send $\langle v_i \rangle_i$ to every party. Denote as m_j the message $\langle v_j \rangle_j$ received from party P_j.
- Upon receiving m_j, set $\mathsf{buf}_i := \mathsf{buf}_i \cup \{m_j\}$.
- Begin to run Π_{SMR} at time Δ.
- Upon outputting a block $\mathsf{Blocks}_i[k]$ in Π_{SMR} do: For each m_j contained in $\mathsf{Blocks}_i[k]$ for which V_i does not already contain a value from party P_j, set $V_i := V_i \cup \{(v_j, j)\}$.
- If at any point during the execution $|V_i| \geq n - t_s$, then output the majority value among all values in V_i.

Fig. 7. A protocol for Byzantine agreement.

Lemma 17 (Validity). *Let $2t_s + t_a < n$. If Π_{SMR} is t-live in a synchronous (resp., asynchronous) network, then Π_{BA} is t-valid in a synchronous (resp., asynchronous) network.*

Proof. Suppose that all honest parties hold input v. We consider two cases. In the first, the network is synchronous and there are at most t_s corrupted parties. In the second, the network is asynchronous and there are at most t_a corrupted parties.

Case 1. Because the network is synchronous, all parties begin to run the protocol at the same time 0 and all signed values $\langle v \rangle$ sent by honest parties must be delivered to all honest parties (by the gossip network) within time Δ. Now, since all parties simultaneously start to run Π_{SMR} at time Δ, strong liveness of Π_{SMR} ensures that every honest party's value-signature pair must appear in the first block $\mathsf{Blocks}_i[1]$ for any honest party P_i. Hence, P_i holds V_i of size at least $n - t_s$ immediately after outputting the block $\mathsf{Blocks}_i[1]$. Since all honest parties started with input v, V_i must include this value at least $n - t_s \geq t_s + 1$ times. Since at most t_s parties are corrupted, the honest majority among values in V_i can only be v. Thus, all honest parties output v.

Case 2. In this case, if any party holds V_i of size at least $n - t_s$, it will always contain a (strict) majority of honest values, because $\lfloor \frac{n-t_s}{2} \rfloor > t_a$ and there are at most t_a corrupted parties. Therefore, any honest party that produces output, will output v (since all honest parties's input is v). Thus, it remains to show that all honest parties eventually hold V_i of size at least $n - t_s$. We know that the $n - t_s$ value-signature pairs sent by honest parties are eventually delivered to all honest parties (by the gossip network); therefore, for each of these honest value-signature messages $m_j = \langle v_j \rangle_j$, strong liveness of Π_{SMR} guarantees that all honest parties eventually output a block that includes m_j. Thus, every honest party P_i eventually gathers V_i of size at least $n - t_s$. ☐

Lemma 18 (Consistency). *Let $2t_s + t_a < n$. If Π_{SMR} is $t < t_s$-consistent in a synchronous (resp., $t < t_a$-consistent in an asynchronous) network, then Π_{BA} is t-consistent in a synchronous (resp., asynchronous) network. then $v_i = v_j$.*

Proof. The lemma follows directly. ☐

6.2 Optimality of our Thresholds

We first discuss how to extend our model to incorporate proofs of work (PoW). First, we assume a global clock that always runs at the same rate, regardless of the state of the system, i.e., regardless of whether we are in the synchronous or asynchronous setting. We stress that the global clock merely serves as a means of defining time, i.e., defining how often a PoW can be solved. In a synchronous network, we may simply identify the global clock with the clock of any honest party; in the asynchronous case there need be no correlation between the global clock and the clock of any party. Second, we assume a random oracle H (modelling a hash function) that each party can query at a bounded rate π over a given global time Δ in order to solve the hash puzzles that form the PoW used in the protocol (where Δ denotes the network delay in the synchronous setting). Our model captures the typical setting considered for PoW-based protocols as in, e.g., the Nakamoto consensus protocol, where computing a PoW amounts to finding a preimage x of the hash of the entire sequence of previously output slots. Our proof is similar to that of [4] and the classical result due to Toueg [31]. The main challenge that we have to overcome in comparison to the setting of [4] is the fact that work can not be reused by the adversary in its simulation with the two parts of the network partitions that it forms. Instead, it has to finish the simulation with both of those parts one after the other, resetting the clock to the initial time after completing the first part of simulation. All though this bound is a simple modification of the one given in [4], it clearly shows that PoWs are of no help unless some form of assumption on the clock rates of parties is made. We are now ready to state our lower bound.

Lemma 19. *Fix t_a, t_s, n with $t_a + 2t_s \geq n$. If an n-party SMR agreement protocol Π is t_s-strongly 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 SMR protocol Π. Partition the n parties into sets S_0, S_1, S_a where $|S_0| = |S_1| = t_s$ and $|S_a| = t_a$, and consider the following experiment:

- At global time 0, parties in S_b begin running Π with m_b in their buffers, where m_0, m_1 are drawn uniformly and independently at random from the set $\{0,1\}^\kappa$. All communication between parties in S_0 and parties in S_1 is blocked (but 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^0 begin running Π (at global time 0) with m_0 in their buffers, and communicate only with each other and parties in S_0. Parties in S_a^1 begin running Π (at global time 0) with m_1 in their buffers, and communicate only with each other and parties in S_1.

Consider an execution of Π at global time 0 in a synchronous network where parties in S_1 are corrupted and simply abort, and all remaining (honest) parties starting with m_0 in their buffers. The views of the honest parties in this execution are distributed identically to the views of $S_0 \cup S_a^0$ in the above experiment. In particular, t_s-strong liveness of Π implies that all parties in S_0 include m_0 in $\mathsf{Blocks}[0]$. Analogously, all parties in S_1 include m_1 in $\mathsf{Blocks}[0]$.

Next consider an execution of Π in an asynchronous network where parties in S_a are corrupted, and first run Π honestly with S_0 where all parties initially hold m_0 in their buffers and the parties in S_0 having their local clocks set to 0 at the start of the execution of Π. Once this execution of Π terminates, parties in S_a run Π again, this time using m_1 as the buffered value and interacting with S_1, where again the local clocks of parties in S_1 are set to time 0. Moreover, all communication between the (honest) 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 high probability, since a party in S_1 includes m_0 in $\mathsf{Blocks}[0]$ with probability $1 - \frac{1}{2^\kappa}$. □

Our lower bound holds when the model of asynchrony includes both unbounded message delay and arbitrary clock skew. It would be interesting to see what can be done in a PoW setting when the network is asynchronous but parties have synchronized clocks. We leave this as an open question for future work.

References

1. Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Efficient synchronous Byzantine consensus, 2017. Available at `https://eprint.iacr.org/2017/307`.
2. Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync HotStuff: Simple and practical synchronous state machine replication, 2019. Available at `http://eprint.iacr.org/2019/270`.
3. Zuzana Beerliová-Trubíniová, Martin Hirt, and Jesper Buus Nielsen. On the theoretical gap between synchronous and asynchronous MPC protocols. In *29th Annual ACM Symp. on Principles of Distributed Computing (PODC)*, pages 211–218. ACM Press, 2010.
4. Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In *14th Theory of Cryptography Conference—TCC 2019*, volume 11891 of *LNCS*. Springer, 2019. Available at `https://eprint.iacr.org/2019/692`.
5. Gabriel Bracha. An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In *3rd Annual ACM Symp. on Principles of Distributed Computing (PODC)*, pages 154–162. ACM Press, 1984.
6. Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Computer Systems*, 20(4):398–461, 2002.
7. Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. *The Computer Journal*, 49(1):82–96, 2006.
8. Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *12th Intl. Conference on Theory and Practice of Public Key Cryptography—PKC 2009*, volume 5443 of *LNCS*, pages 160–179. Springer, 2009.
9. Matthias Fitzi and Jesper Buus Nielsen. On the number of synchronous rounds sufficient for authenticated Byzantine agreement. In *23rd Intl. Symp. on Distributed Computing (DISC)*, volume 5805 of *LNCS*, pages 449–463. Springer, 2009.
10. Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In *30th Annual ACM Symp. on Principles of Distributed Computing (PODC)*, pages 179–186. ACM Press, 2011.
11. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology—Eurocrypt 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, 2015.
12. Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In *Advances in Cryptology—Crypto 2019, Part I*, volume 11692 of *LNCS*, pages 499–529. Springer, 2019.
13. Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In *Advances in Cryptology—Eurocrypt 2010*, volume 6110 of *LNCS*, pages 466–485. Springer, 2010.
14. Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for Byzantine agreement. *J. Computer and System Sciences*, 75(2):91–112, 2009.
15. Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. Zyzzyva: Speculative Byzantine fault tolerance. *ACM Trans. Computer Systems*, 27(4):7:1–7:39, 2009.
16. Klaus Kursawe. Optimistic Byzantine agreement. In *21st Symposium on Reliable Distributed Systems (SRDS)*, pages 262–267. IEEE Computer Society, 2002.
17. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7), 1978.
18. Leslie Lamport. The part-time parliament. Technical Report 49, DEC Systems Research Center, 1989.
19. Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine generals problem. *ACM Trans. Programming Language Systems*, 4(3):382–401, 1982.
20. Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolic. XFT: Practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 485–500. USENIX Association, 2016.
21. Chen-Da Liu-Zhang, Julian Loss, Tal Moran, Ueli Maurer, and Daniel Tschudi. Robust MPC: Asynchronous responsiveness yet synchronous security. Unpublished manuscript.
22. Julian Loss and Tal Moran. Combining asynchronous and synchronous Byzantine agreement: The best of both worlds, 2018. Available at `http://eprint.iacr.org/2018/235`.
23. Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible Byzantine fault tolerance. In *26th ACM Conf. on Computer and Communications Security (CCS)*, pages 1041–1053. ACM Press, 2019. Available at `https://arxiv.org/abs/1904.10067`.

24. Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In *23rd ACM Conf. on Computer and Communications Security (CCS)*, pages 31–42. ACM Press, 2016.
25. Rafael Pass, Lior Seeman, and Abhi 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.
26. Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *31st International Symposium on Distributed Computing (DISC)*, volume 91 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.
27. Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Advances in Cryptology—Eurocrypt 2018, Part II*, volume 10821 of *LNCS*, pages 3–33. Springer, 2018.
28. Arpita Patra and Divya Ravi. On the power of hybrid networks in multi-party computation. *IEEE Trans. Information Theory*, 64(6):4207–4227, 2018.
29. M. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.
30. Fred Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.
31. Sam Toueg. Randomized Byzantine agreements. In *3rd Annual ACM Symp. on Principles of Distributed Computing (PODC)*, pages 163–178. ACM Press, 1984.