Analyzing the Real-World Security of the Algorand Blockchain

Fabrice Benhamouda
Algorand Foundation
New York, NY, United States
fabrice.benhamouda@gmail.com

Erica Blum*
Reed College
Portland, OR, United States
ericablum@reed.edu

Jonathan Katz
University of Maryland
College Park, MD, United States
jkatz2@gmail.com

Derek Leung
Massachusetts Institute of Technology
Cambridge, MA, United States
dtl@csail.mit.edu

Julian Loss
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
loss@cispa.de

Tal Rabin
University of Pennsylvania
Philadelphia, PA, United States
talr@seas.upenn.edu

ABSTRACT

The Algorand consensus protocol is interesting both in theory and in practice. On the theoretical side, to achieve adaptive security, it introduces the novel idea of *player replaceability*, where each step of the protocol is executed by a different randomly selected *committee* whose members remain secret until they send their first and only message. The protocol provides consistency under arbitrary network conditions and liveness under intermittent network partitions. On the practical side, the protocol is used to secure the Algorand cryptocurrency, whose total value is approximately $850M at the time of writing.

The Algorand protocol in use differs substantially from the protocols described in the published literature on Algorand. Despite its significance, it lacks a formal analysis. In this work, we describe and analyze the Algorand consensus protocol as deployed today in Algorand's ecosystem. We show that the overall protocol framework is sound by characterizing network conditions and parameter settings under which the protocol can be proven secure.

KEYWORDS

State Machine Replication, Asynchrony, Blockchain

1 INTRODUCTION

The Algorand proof-of-stake consensus protocol used at the heart of the Algorand blockchain is interesting in theory and in practice. From a theoretical point of view, the protocol attains security under adaptive corruption of parties by introducing the novel technique of *player replaceability*. Randomly selected *committees* of parties are assigned to execute different steps of the protocol, and a party's membership on any particular committee is secret until that party executes the corresponding step by sending a message. The protocol aims to ensure strong security guarantees in real-world network conditions: informally, it seeks to provide consistency in an asynchronous network and liveness given intermittent network partitions so long as there are sufficiently long periods of synchrony. From a practical point of view, the Algorand cryptocurrency, whose total value is roughly 850 million USD at the time of writing, relies on the protocol for security.

Understanding the security of the Algorand protocol is thus an important goal. Although analyses of early versions of the Algorand protocol have previously appeared in the literature [3–5, 9, 10], the Algorand protocol in use is very different from all the variants that have been previously described or analyzed. To the best of our knowledge, there is no formal analysis of the Algorand protocol that is actually deployed or anything similar to it (§1.3). Moreover, even the existing security proofs for earlier versions of the protocol do not prove all the properties that are claimed. For example, Chen and Micali [5] give a proof sketch for a simplified version of the protocol (without player replaceability) in a synchronous network, but they do not analyze the full version of the protocol in a network with intermittent partitions.

1.1 Our Contribution

In collaboration and consultation with researchers and engineers involved in the implementation and deployment of the Algorand blockchain, and using the technical specification made available by Algorand [1] as a primary source, we present a formal specification of the Algorand consensus protocol *as deployed*.[1] We analyze the protocol in a novel network model intended to accurately capture concerns of the protocol designers—in particular, recovery from intermittent network outages.

Our analysis establishes a variety of results regarding the protocol's security. Our positive results include a proof that the overall protocol framework is sound. We also describe how the parameters of the protocol (e.g., committee sizes and quorum thresholds) can be set to achieve any desired level of security.

On the other hand, some quantitative security bounds we obtain are relatively weak. For instance, we fail to establish a strong security bound for the protocol's liveness under partial synchrony. In particular, although we are able to define a set of good events that are sufficient for liveness, we fail to prove a meaningful lower bound on the probability that those good events occur. We leave as open questions whether our bounds can be improved and whether the deployed protocol's parameters should be adjusted to improve security margins.

1.2 Technical Overview

1.2.1 Overview of the Protocol. In the protocol, various committees play different roles in reaching agreement. This section focuses on

*Portions of this work were performed while at University of Maryland.

[1] The Algorand ecosystem allows for self-governance, meaning that a sufficiently large set of parties can decide to enact arbitrary changes to the protocol in the future; our analysis deals with the protocol as it is deployed at the time of writing.

the high-level protocol flow. Further details (e.g., how committees are elected, specific types of committees, etc.) follow in Section 2.3.

To achieve consensus, the parties repeatedly execute a graded-consensus protocol which we call GC (as in prior work [8, 12, 14]). GC itself invokes a sub-protocol which we call Vote. In Vote, parties on a committee vote for a value proposed by a randomly selected leader. (Honest parties may disagree about the leader if either the network is asynchronous or the leader is corrupted.) When a party receives a sufficient number of concurring votes (a "quorum") on a proposed value, it terminates Vote.

In the remainder of GC, the parties try to reach agreement on the output of Vote. Different committees play different roles in this process. Some committees ensure fast convergence in a partially synchronous network, and others enable recovery from intermittent network faults in a "synchronous enough" network. A decision is reached when parties receive a quorum from any committee.

1.2.2 Challenges. Defining an appropriate model in which to analyze the protocol is challenging. First, the Algorand engineers designed the protocol with real-world network conditions in mind, rather than any particular theoretical model. Second, they designed the protocol to simultaneously achieve different security guarantees under different network conditions. Thus, we analyze the protocol in multiple network models—some of which are not entirely standard—that we believe best represent the engineers' intentions.

Since we are analyzing an existing protocol, we lack the freedom to modify or simplify the protocol in order to facilitate the analysis. The protocol's complexity stems from its attempt to achieve security in multiple different network models simultaneously. Specifically, its recovery mechanisms try to account for both partial synchrony and for intermittent network partitions: the protocol seeks to both guarantee liveness and achieve good efficiency under real-world network conditions. Managing these competing requirements gives the protocol two characteristics that complicate analysis and prevent direct application of techniques in prior work:

(1) The protocol has multiple termination paths by which parties can decide on their output. Having multiple paths complicates the proof of consistency; in particular, the path intended to deal with partial synchrony and the one intended to address an intermittently synchronous network have subtle interactions.
(2) The protocol involves committees of widely varying sizes. For example, the smallest committee contains 500 parties in expectation, while the largest has 6000. The goal of the protocol designers here was to optimize over a trade-off: smaller committees improve efficiency but are more likely to contain too many corrupted parties. Thus, committees with strict consistency requirements are larger than those with loose consistency (but strict liveness) requirements.

1.2.3 Our Techniques. To prove security of the protocol, we introduce several proof techniques that may be of independent interest. For modularity, we identify a set of "bad events" and analyze security of the protocol in the absence of bad events; we then focus on upper-bounding the probabilities of those bad events. This allows us to cleanly separate a higher-level analysis of the protocol from lower-level probability calculations. It also enables us to give concrete bounds on the probabilities of certain security violations.

In analyzing the probabilities of the bad events, we must account for both adaptive corruptions and adversarially controlled message delivery. To cope with this complexity, we introduce a combinatorial game we call the *red-blue game*. We analyze an adversary's maximum possible advantage in that abstract game and then map the game to particular bad events in the protocol. We remark that while other works studying consensus also rely on combinatorial analyses, we are not aware of any such results that apply to a protocol with multiple termination paths as in our setting. Our analysis is also more complex since we need to take into account properties relating to *pairs* of committees (of different sizes).

1.3 Related Work

1.3.1 Prior Versions of Algorand. The Algorand protocol was introduced in a series of foundational works [4, 5, 10]. In addition to player replaceability, the Algorand protocol is generally credited with pioneering the use of *cryptographic sortition*, i.e., using verifiable random functions [13] to elect committee members from a large pool of participants. Following this series, there have been several subsequent alterations to the Algorand protocol [3, 9].

In this work, we analyze the Algorand protocol as it is currently deployed, as specified by Algorand Foundation [1]. The deployed version of the protocol differs from versions considered in prior work. Some prior work considered earlier versions of the protocol that used fewer committees than the current version [4, 5, 10]; other work did not consider committees at all but instead assumed all parties participate [3]. Another feature that distinguishes the deployed algorithm from versions analyzed in prior work is that it is designed to achieve security under different network conditions. This is not fully realized in earlier versions of the protocol.

1.3.2 Adaptive Security for Committee-Based Protocols. Subsequent work introduced other adaptively secure consensus protocols relying on player replaceability, committees, and/or VRFs [2, 6, 15]. Many of these protocols are simpler than the Algorand protocol we are analyzing, as they were designed expressly to enable a clean proof of security; since we are analyzing a protocol that is already deployed, we do not have that luxury. Thus, the proof techniques used in any of these prior works are insufficient for our goal.

2 MODEL AND PRELIMINARIES

We let N denote the number of parties running the protocol. A "party" in our context represents a unit of currency (namely, one microAlgo) staked by a user who wishes to participate in the protocol; hence, multiple parties might correspond to a single physical entity. We assume $N \geq 10^{12}$, which is a conservative lower bound: from genesis to July 2023, the Algorand system has had $N \geq 10^{15}$.

2.1 Network Model

We consider three network models: an *asynchronous* model, a modified *partially synchronous* model, and a model we call *synchronous-enough*. In all of these models, we assume that any pair of parties can communicate via a point-to-point channel and that a PKI is in place so that parties agree on each others' public keys. We also

assume that each party has a local clock and that parties' clocks run at the same rate; however, their clocks need not be synchronized.

The asynchronous model is the standard one: all messages may be delayed arbitrarily by an adversary, so long as any message sent by an honest party is eventually delivered.

For the partially synchronous model, we consider the *unknown GST* flavor, in which message delays are unbounded until an unknown global stabilization time GST [7], and after which the network *stabilizes* and messages are delivered in a synchronous fashion with (known) bounded delay. As a strengthening of the usual definition, we assume two delay parameters, with "small" messages delivered in time δ and "large" messages delivered in time Λ. (For our purposes, a message is large if it contains a block and small otherwise.) As a theoretical weakening of the usual definition, some of our results require an upper bound on GST. While in theory this is equivalent to the synchronous model with delay GST + Λ, the required bound on GST can be on the order of the lifetime of the universe and so is immaterial in practice.

The "synchronous-enough" model is a new model we introduce that is intended to capture real-world networks that mostly operate in a synchronous fashion but may occasionally experience large delays. Formally, in the $(\beta, \Gamma, \delta, \Lambda)$-synchronous-enough model the network experiences *good* intervals and *bad* intervals. During good intervals, the network is synchronous; small messages are delivered within time δ and large messages are delivered within time Λ. During bad intervals, the network is asynchronous and messages can be delayed arbitrarily. However, a bad interval can last for time at most β, and any bad interval must be followed by a good interval of length at least $\Gamma \geq \Lambda$. The parameters $\beta, \Gamma, \delta, \Lambda$ are known to the honest parties, but the honest parties do not know whether the network is bad or good at any given time. (The adversary, however, is aware of the state of the network.)

Although one can view this model as a standard synchronous model with delay bounded by $\beta + \Lambda$, working in this model has the potential to facilitate quantitatively stronger liveness claims for the top-level SMR protocol. In particular, we imagine β to be on the order of hours or days, in which case standard synchronous protocols would be extremely slow even if the network is usually fast. By comparison, the Algorand protocol is slow to terminate in the worst case (due to its exponential backoff), but when the network is good for long periods of time, it can terminate executions of BA in time much less than β (see e.g. Lemma A.11).

2.2 Adversary Model

We assume that at most $\alpha \cdot N$ parties can be adaptively corrupted by an adversary. (We generally leave α as a variable but note that the Algorand protocol is designed to tolerate $\alpha = 0.2$.) Parties who are not corrupted are called *honest*. Once a party is corrupted, the adversary can freely coordinate its actions for the duration of the protocol. Since parties represent units of stake rather than individual users, this is equivalent to assuming that the adversary controls at most an α fraction of the total stake at any time. The exact distribution of stake across users is irrelevant as long as this assumption holds.

All the network models we consider prohibit *after-the-fact message removal*: if the adversary corrupts a party P after P sends a message, the adversary cannot prevent that message from being delivered. (This avoids the lower bound of Abraham et al. [2] on the communication complexity of consensus protocols tolerating after-the-fact message removal.)

2.3 Committee Election

In the deployed version of the protocol, parties are elected to committees using verifiable random functions (VRFs). Our analysis omits the implementation details and instead considers an idealized *committee-election mechanism* that randomly and independently selects committees as needed.

We assume that each party is independently selected to be on the committee with probability E/N, where E is a parameter determining the expected size of the committee. As part of this mechanism, the party receives a proof of its membership on the committee. Furthermore, we require that the mechanism does not reveal to the adversary whether an honest party is on a particular committee until the party actively proves its membership—this is key to achieving adaptive security.

In the Algorand protocol, each honest committee member sends only a single message on behalf of that committee. This message serves as both a proof of membership on the committee and a vote for some decision. We also assume that corruption of an honest party P *after* P proves its membership should not allow the adversary to send any other votes on P's behalf (as a member of that committee). This can be achieved, for example, by using forward secure signatures and having party P update its secret key immediately after sending its message as part of the committee.[2]

In the rest of the paper, we handle committee membership abstractly, implicitly assuming that parties locally determine when they are on a committee and that they verify committee membership of other parties when determining validity of votes.

2.4 Security Definitions

The high-level goal of the Algorand protocol is to implement *state machine replication* (SMR), by which a group of parties collectively agree on a sequence of blocks. Algorand achieves SMR using sequential executions of a *Byzantine agreement* (BA) protocol to agree on one block at a time. For our purposes the syntax of blocks is irrelevant, and we can consider BA as an interactive protocol that allows a set of parties, each with some initial input v_i, to agree on a common output v. The goal is to achieve certain notions of *consistency* (also called *safety*) and *liveness*. Consistency, roughly, means all honest parties agree on their output, and liveness means all honest parties eventually output something. Formally:

Definition 2.1. Let BA be an N-party interactive protocol, where each party P_i has an input value v_i and outputs a value v_i'. We say that BA is secure if it achieves the following:

- (Consistency) If honest parties P_i, P_j output values v_i', v_j', respectively, then $v_i' = v_j'$.
- (Liveness) Every honest party eventually terminates with some output.

[2] In the deployed protocol, parties update their keys only after a given iteration, not after each message they send. This is likely an optimization reflecting the fact that adaptive corruption in the real world is not instantaneous. In our formal analysis we assume parties update their keys after each message they send.

Algorand aims for consistency under arbitrary message delays (i.e., in an asynchronous network), and liveness assuming either a partially synchronous or a synchronous-enough network.

The real-world protocol also achieves a notion of *external validity*, meaning that only "valid" values are output by honest parties. We do not explicitly consider this notion since (1) we do not want to define what "valid" means in the context of the protocol, and (2) honest parties can achieve external validity by simply ignoring invalid values.

3 THE ALGORAND PROTOCOL

The Algorand SMR protocol (i.e., blockchain) uses sequential invocations of a core BA protocol to agree on blocks. Consistency of the blockchain follows readily from consistency of the BA protocol, and (under mild synchrony assumptions, such as those used here) liveness for the blockchain follows from liveness of the BA protocol. We therefore restrict our attention to the core BA protocol.

The pseudocode is presented in a modular form: the top-level BA protocol consists of iterations of the graded consensus protocol GC (Algorithm 2). Adopting the term used by the protocol designers, we call these iterations *periods*. The pth invocation of GC is denoted by GC_p. Periods are local rather than global, and honest parties may be participating in different periods at any given time. In each period, GC_p in turn calls a dedicated subprotocol Vote_p (Algorithm 3). Each of these building blocks is explained in greater detail below.

3.1 Committees, Votes, and Quorums

The Algorand protocol defines seven types of committees: propose, soft, cert, next, late, down, and redo. Each committee serves a different function and has a different (expected) size, given in Table 1. These committees are resampled in each period, so that the sets of parties on different committees are independent. For example, the set of parties in late committee of period $p = 1$ is independent of the set of parties on the late committee for $p = 2$. Furthermore, there are 250 independent next committees per period; we include an index k in the subscript to distinguish between them. As a convenient shorthand, we often refer to these committees by the abbreviations $\mathcal{PV}$, $\mathcal{SV}$, etc., as shown in Table 1.

Name	Notation	Expected size	Quorum size
p-propose	$\mathcal{PV}_p$	20	n/a
p-soft-vote	$\mathcal{SV}_p$	2990	2267
p-cert-vote	$\mathcal{CV}_p$	1500	1112
p-next$_k$-vote	$\mathcal{NV}_{p,k}$	5000	3838
p-late-vote	$\mathcal{LV}_p$	500	320
p-redo-vote	$\mathcal{RV}_p$	2400	1768
p-down-vote	$\mathcal{DV}_p$	6000	4560

Table 1: Expected sizes and quorum sizes of Algorand committees.

Committee members send messages in support of proposed values; we refer to the messages sent by the soft committee as soft-*votes*, and likewise for other committees. (The exception is the propose committee, whose messages we call *proposals*.) To refer to a specific period, we may write "p-soft-vote" as shorthand for a soft-vote sent in period p (and so on for the other committees).

Sufficiently many matching votes from members of a particular committee are called a *quorum*. Each committee has a *quorum size* that determines the number of votes needed to form a quorum (Table 1). For example, a p-cert-*quorum for* v consists of 1112 p-cert-votes on v by distinct members of $\mathcal{CV}_p$.

At a high level, each committee plays a unique role in fulfilling one of two goals. The primary goal is to reach agreement on a proposed value; failing that, the secondary goal is to advance to a new period and try again. The propose, soft, and cert committees are most directly responsible for the primary goal, with the remaining committees supporting the secondary goal. Specifically, members of the propose committee propose a value to all parties at the start of each period. The soft committee votes in favor of proposals they receive. The cert committee "certifies" the decision of the soft committee, and agreement is reached with a cert-quorum.

If a proposal is not certified before a fixed deadline, the remaining committees ($\mathcal{NV}_{p,k}$, $\mathcal{LV}$, $\mathcal{DV}$, and $\mathcal{RV}$) seek to safely initiate a new period. Informally, each of these committees is responsible for raising a distinct "flag" related to the status of the current period, which has different implications for the next attempt. A late-quorum indicates that a soft-quorum was formed on a proposed value after the deadline. A redo-quorum indicates that enough parties support a particular value to warrant keeping it for the next period, even if there was no soft-quorum for that value. A down-quorum indicates that the primary path made insufficient progress. Depending on which "flag" is raised, parties will either retain a previously proposed value or start from scratch with a new proposal. The next$_k$-committees are the most flexible in that each member can advocate for retrying or starting from scratch as it sees fit. (To avoid spamming the network, next$_k$ voting takes place in exponentially-increasing intervals, whereas the other committees vote almost immediately once the condition they are waiting for is met.) A party begins the next iteration as soon as it receives any one of the above quorums.

Intuitively, having multiple "paths" to agreement is beneficial because the protocol can proceed as soon as any of the paths completes. However, it also introduces a complication: if *more than one* of the paths succeeds, it is critical that each path arrives at the same result. Carefully accounting for interactions between these paths—and ensuring consistency for the overall protocol—is one of the most technically challenging aspects of our analysis.

3.2 Technical Details

The BA protocol (cf. Algorithm 1) sequentially invokes a sub-protocol for graded consensus (GC) until a termination condition is satisfied. Each party P_i begins the BA protocol holding an input $v_i \neq \perp$, which is (the hash of) a block of transactions.

At the outset of the pth period, each party P_i holds an input of the form (v_i, v, b_i), where v_i is that party's original input to the BA protocol, v (informally) represents a value on which partial agreement may have been reached, and $b_i \in \{0, 1\}$ represents the party's confidence in that partial agreement. (Initially in GC_1, each honest party P_i uses input $(v_i, \perp, 0)$.) During execution of the

Algorithm 1: The Algorand BA protocol, from the perspective of party P_i with input $v_i \neq \perp$.

throughout:
  on receiving a p-quorum ($p < 2^{64}$) for any committee, forward it to all other parties
$\mathrm{GC}_1(v_i, \perp, 0)$
on *terminating* GC_p *with output* (v, g)**:**
  if $g = 2$ **then**
    output v and terminate
  else
    if $g = 1$ **then** $\mathrm{GC}_{p+1}(v_i, v, 1)$;
    else $\mathrm{GC}_{p+1}(v_i, \perp, 0)$;

protocol, each party P_i maintains a local variable b that is initialized to b_i but is set to 0 if, at any point during execution of GC_p, that party observes evidence that agreement has not been reached. Each P_i also maintains a local timer (denoted clock_i) that is reset to 0 at the start of GC_p and incremented in sync with its local clock.

In the GC protocol (cf. Algorithm 2), the parties' primary goal is to reach agreement on a proposed value; this is the purpose of the Vote_p subprotocol described in Algorithm 3. If Vote_p fails to reach agreement in a timely fashion, the parties' secondary goal is to collectively move on to the next period to try again.

At the start of Vote_p, each member of $\mathcal{PV}_p$ proposes its own input to all other parties. At time 2δ, each member of $\mathcal{SV}_p$ votes as follows: if $b = 1$, or P_i has not received any valid p-proposals, then P_i simply p-soft-votes for v. Otherwise, it identifies[3] a p-leader from among the parties in $\mathcal{PV}_p$ from whom it received a proposal, and it p-soft-votes for the leader's proposal. Whenever any party receives a quorum of p-soft-votes for some value v', it outputs v' and terminates Vote_p. We remark that when Vote_p is run in an asynchronous network, it may never terminate (for example, if honest parties start the protocol at very different times) or may terminate after an unbounded amount of time.

Returning to the execution of GC_p, if a member of $\mathcal{CV}_p$ terminates Vote_p with output $v' \neq \perp$ before time $\max\{4\delta, \Lambda\}$, it p-cert-votes for v'. After that, at a sequence of 250 instants (indexed by $k = 1, \ldots, 250$) whose separation increases exponentially, members of $\mathcal{NV}_{p,k}$ vote for different values depending on whether Vote_p has terminated and their current value of b. Each party sends its vote at some instant within a fixed "(next-)voting round." More precisely, each committee member P_i sends its next_k-vote at time $T_k + r$, where T_k is fixed by the protocol, and r is an offset sampled uniformly from the interval $[0, 2^k\delta]$. (The offset is a heuristic optimization for avoiding network congestion.) We define a subroutine wakeup to represent this; i.e., when $k = 1$, $\mathrm{wakeup}(k)$ returns $\max\{4\delta, \Lambda\}$; for $2 \leq k \leq 250$, $\mathrm{wakeup}(k)$ samples a uniform offset $r \in [0, 2^k\delta]$ and returns $\max\{4\delta, \Lambda\} + 2^k\delta + r$.

In parallel to the next_k-vote committees, members of $\mathcal{LV}_p$, $\mathcal{RV}_p$, and $\mathcal{DV}_p$ also send votes. At regular intervals of time λ_f, they check to see if some condition is satisfied. (The delay λ_f is an adjustable parameter; in practice, using a low value for λ_f gives better latency if the network is fast but adds unnecessary work if the network is slow.) As with the next_k-vote committees, parties determine their vote based on whether or not they have observed Vote_p terminate and their current value of b.

A party's output from GC_p includes both a value and a grade. The protocol has several termination conditions that can be triggered when different quorums of votes have been observed. If

[3]The precise mechanism by which this is done is irrelevant for our purposes, but we note that the leader is chosen deterministically based on the proposals received, so two honest parties who receive the same sets of proposals will identify the same leader. Honest parties who receive different sets of proposals may identify different leaders.

Algorithm 2: The GC protocol, from the perspective of party P_i with input (v_i, v, b_i).

$b := b_i$; set $\mathrm{clock}_i := 0$
throughout:
  On receiving a $(p-1)$-next_*-quorum for $\perp$ or a $(p-1)$-down-quorum for $\perp$, set $b := 0$
/* Step 1: */
starting at $\mathrm{clock}_i = 0$**:**
  $\mathrm{Vote}_p(v_i, v)$
/* Step 2: */
while $\mathrm{clock}_i \in (2\delta, \max\{4\lambda, \Lambda\})$, *if* $P_i \in \mathcal{CV}_p$ **do**
  on *terminating* Vote_p *with output* $v' \neq \perp$**:**
    p-cert-vote for v'
/* Step 3: */
for $k \in [1..250]$ **do**
  on *signal from* $\mathrm{wakeup}(k)$, *if* $P_i \in \mathcal{NV}_{p,k}$**:**
    if Vote_p *terminated with output* $v' \neq \perp$ **then**
      p-next_k-vote for v'; // 3a
    if $b = 1$ *and* Vote_p *either has not terminated or terminated with output* $\perp$ **then**
      p-next_k-vote for v; // 3b
    if $b = 0$ *and* Vote_p *either has not terminated or terminated with output* $\perp$ **then**
      p-next_k-vote for $\perp$; // 3c
/* Step 4: */
starting at $\mathrm{clock}_i = \max\{4\delta, \Lambda\}$, *repeat every* λ_f *time steps*:
  if $P_i \in \mathcal{LV}_p$ *and* Vote_p *terminated with output* $v' \neq \perp$ **then**
    p-late-vote for v'; // 4a
  if $P_i \in \mathcal{RV}_p$, $b = 1$, *and* Vote_p *either has not terminated or terminated with output* $\perp$ **then**
    p-redo-vote for v; // 4b
  if $P_i \in \mathcal{DV}_p$, $b = 0$, *and* P_i *either did not output in* Vote_p *or output* $\perp$ *in* Vote_p **then**
    p-down-vote for $\perp$; // 4c
/* Termination conditions: */
on *receiving one of the following quorums, output the value and grade indicated and terminate*:
  p'-cert-quorum ($p' < 2^{64}$) for $v' \neq \perp$: $(v', 2)$; // T1
  p-next_*-quorum for $v' \neq \perp$: $(v', 1)$; // T2
  p-late-quorum for $v' \neq \perp$: $(v', 1)$; // T3
  p-redo-quorum for $v' \neq \perp$: $(v', 1)$; // T4
  p-next_*-quorum for $\perp$: $(\perp, 0)$; // T5
  p-down-quorum for $\perp$: $(\perp, 0)$; // T6

Algorithm 3: The Vote protocol, from the perspective of party P_i with input $v_i \neq \perp$.

at $\text{clock}_i = 0$**:**
  if $P_i \in \mathcal{PV}_p$ **then**
    If $b = 0$, p-propose v_i; otherwise, p-propose v.
at $\text{clock}_i = 2\delta$**:**
  if $P_i \in \mathcal{SV}_p$ **then**
    if $b = 1$ *or* P_i *has not received any valid* p*-proposal* **then** p-soft-vote for v;
  else Identify a leader P_ℓ from among the valid proposals received and p-soft-vote for the value v_ℓ proposed by P_ℓ. ;
on *receiving a* p-soft-*quorum for some* v'**:**
  Output v' and terminate.

a party receives a p'-cert-quorum (for any value $p' < 2^{64}$, not necessarily equal to p) for some value v', then it outputs v' with grade 2; returning to the outer BA protocol, this means that party will terminate its execution with output v'. A party who receives a p-next$_k$-quorum (for any k), a p-late-quorum, or a p-redo-quorum for some value $v' \neq \perp$ will output v' with grade 1. Finally, if a party receives a p-next$_k$-quorum or a p-down-quorum for $\perp$, then it outputs $\perp$ with a grade of 0.

4 GOOD EXECUTIONS

Throughout, we denote by $\star\mathcal{V}_p^H$ the set of parties on a committee $\star\mathcal{V}_p$ who remain honest until they have sent their message for that committee and define $\star\mathcal{V}_p^C = \star\mathcal{V}_p \setminus \star\mathcal{V}_p^H$ to be the set of parties on that committee who are corrupted at any point prior to that.

Our security proofs assume certain *good* events happen during the course of an execution (and conversely, that certain *bad* events do not happen).The first type of good event, denoted with the prefix **A**, relates to the number of corrupted parties on a particular committee. We say a committee is *safe* if the number of honest parties on the committee plus twice the number of corrupted parties on the committee is less than twice the quorum threshold; when a committee is safe, it is not possible for there to be two conflicting quorums for that committee. We say a committee is *valid* if the number of corrupted parties on the committee is less than the quorum threshold; note that validity is implied by safety. The **A**-events relate to safety and validity of different committees; we bound the probabilities of such events in Section 6.1.

The second type of event, denoted with the prefix **B**, relates to the number of honest parties on a given committee. We say a committee is *live* if the number of honest parties on that committee is at least the quorum threshold. Note that if a committee is not live then the protocol may become deadlocked even if there are no corrupted parties at all, and conversely if a committee is live then honest parties can potentially generate a quorum even if all corrupted parties on the committee abort. We bound the probabilities of these events in Section 6.2.

The last type of event, denoted with the prefix **C**, relates to inconsistencies between different committees. The bounds for these events are the most technically challenging; they are analyzed using our red-blue game in Section 6.4.

Definition 4.1. Define the following events, parameterized by an iteration $p \leq 2^{64}$:

A.1 $|\mathcal{SV}_p^H| + 2 \cdot |\mathcal{SV}_p^C| < 2 \cdot 2267$.

A.2 $|\mathcal{CV}_p^C| < 1112$.

A.3 For all $1 \leq k \leq 250$, $|\mathcal{NV}_{p,k}^C| < 3838$.

A.4 $|\mathcal{LV}_p^C| < 320$.

A.5 $|\mathcal{RV}_p^C| < 1768$.

A.6 $|\mathcal{DV}_p^C| < 4560$.

B.1 There exists $p' \in \{p, \ldots, p+60\}$ such that all of the following hold:

B.1.1 $|\mathcal{PV}_{p'}^H| \neq 0$ and the p'-leader is honest.

B.1.2 $|\mathcal{SV}_{p'}^H| \geq 2267$.

B.1.3 $|\mathcal{CV}_{p'}^H| \geq 1112$.

B.2 For all $p' \in \{p, \ldots, p+60\}$ and $k \in \{1, \ldots, 235\}$, there are distinct $k_1, k_2, k_3 \in \{k, \ldots, k+15\}$ with $|\mathcal{NV}_{p',k_i}^H| \geq 3838$ for $i \in \{1, 2, 3\}$.

B.3 $|\mathcal{LV}_p^H| \geq 320$.

B.4 $|\mathcal{RV}_p^H| \geq 1768$.

B.5 $|\mathcal{DV}_p^H| \geq 4570$.

C.1 If an honest party receives a p-cert-quorum for $v \neq \perp$ then, for all k, no honest party receives a p-next$_k$-quorum for $v' \neq v$.

C.2 If an honest party receives a p-cert-quorum for $v \neq \perp$ then no honest party receives a p-down-quorum for $\perp$.

C.3 If an honest party receives a p-soft-quorum for v then, for all k, no honest party receives a p-next$_k$-quorum for $v' \notin \{v, \perp\}$.

C.4 If an honest party receives a p-soft-quorum for v then no honest party receives a p-redo-quorum for $v' \neq v$.

An execution of the protocol is *secure* if all events with prefixes **A** and **C** occur for all $p < 2^{64}$. A secure execution is *intermittently fair* if events **B.1** and **B.2** occur for all p and *eventually fair* if events **B.1**, **B.3**–**B.5** occur for all p.

In Section 5, we prove that consistency holds for the Algorand BA protocol in any secure execution and that liveness holds in any intermittently fair or eventually fair execution. Following that, we give bounds on the probabilities that an execution is secure and intermittently fair, and make partial progress towards a bound on the probability that an execution is eventually fair.

5 SECURITY ANALYSIS

This section summarizes the results of our security analysis. Informally, we show:

- In a secure execution, the Algorand protocol satisfies consistency in an asynchronous network.

- In an eventually fair execution, the Algorand protocol satisfies liveness in a partially synchronous network.
- In an intermittently fair execution, the Algorand protocol satisfies liveness in a synchronous-enough network.

5.1 Consistency

We show that, assuming a secure execution, the Algorand protocol satisfies consistency in an asynchronous network. In such a network, an adversary can prevent quorums from propagating for arbitrarily long periods of time. As such, parties' knowledge of prior votes may be incomplete when they cast their own votes. The main challenge is to show that despite this, the parties cannot make inconsistent decisions.

Theorem 5.1 (Consistency). *Assume a secure execution of BA in an asynchronous network. If honest parties P, P' output v, v', respectively, then $v = v'$.*

The proof is given in Appendix A.1.

5.2 Liveness

We present two liveness results, showing that the protocol makes progress in either a partially synchronous network or a synchronous-enough network. The high-level intuition for both results is the same. First, we show that even if the parties get "stuck" in some instance of GC_p due to a period of asynchrony, they will become "unstuck" and advance to a new iteration once the network becomes synchronous—either after GST (in the partially synchronous model) or upon entering a good period (in the synchronous-enough model). Once that happens, the protocol terminates within a small number of iterations (in expectation). The main difference between the two results is which committees are used to advance the iteration of GC. In the partially synchronous model, advancing relies on the late-, down-, or redo-vote committees. In the synchronous-enough model, advancing relies on the next_k-vote committees. We present both results to illustrate how both sets of committees contribute to liveness under different network conditions.

To state our results we introduce some notation. Recall that $\mathsf{wakeup}(k)$ determines when a party on the p-next_k-vote committee tries to vote. We define a related function $\mathsf{wakeup}_{\mathsf{min}}(k)$ indicating the *earliest* possible time a party on the p-next_k-vote committee can vote. Specifically,

$$\mathsf{wakeup}_{\mathsf{min}}(k) = \begin{cases} \max\{4\delta, \Lambda\} & k = 1 \\ \max\{4\delta, \Lambda\} + 2^k\delta & k \in \{2, \ldots, 250\}. \end{cases}$$

Theorem 5.2. *Assume an eventually fair execution of BA in a partially synchronous network, and* $\mathsf{GST} \leq \mathsf{wakeup}_{\mathsf{min}}(230)$. *Then all honest parties terminate BA by time* $\mathsf{GST} + 60 \cdot \Delta + 4\delta$, *where* $\Delta = \max\{4\delta, \Lambda\} + 5\delta + 5\lambda_f$.

Theorem 5.3. *Assume an intermittently fair execution of BA in a $(\beta, \Gamma, \delta, \Lambda)$-synchronous enough network, let*

$$\begin{aligned} T^* \quad = \quad & \mathsf{wakeup}_{\mathsf{min}}(\log_2(\beta - \max\{4\delta, \Lambda\}))/\delta + 16) \\ & + 60 \cdot (\mathsf{wakeup}_{\mathsf{min}}(16) + \delta) + 6\delta, \end{aligned}$$

and assume $\Gamma \geq T^*$. If the network becomes good at time T, then all honest parties terminate BA by time $T + T^*$.

Proofs of the above are given in Appendix A.2.

6 PROBABILITY BOUNDS

In Section 5, we proved security of the Algorand protocol assuming certain "good" events occur. We now prove lower bounds for the probabilities of those events under any adversary who can adaptively corrupt parties during execution of the protocol. Intuitively, the protocol is designed so that an adaptive adversary only learns information about whether an honest party is on some committee once it is too late for that information to be useful for the adversary. For example, the adversary can learn that some party P is on a committee when it sees P send a vote, but at that point P has already deleted the key signing the vote and so corruption of that party will not allow the adversary to send a conflicting vote. Alternately, the adversary may learn that some honest party P' is *not* on some committee by inferring that P' would have already voted if it were eligible to do so, but—since committee membership is decided independently for each party—this does not help the adversary determine which other parties are on that committee.

The reader can find code which computes the bounds in this paper at https://github.com/fabrice102/analyzing-algorand.

6.1 Safety/Validity Failures

Recall that a committee is *safe* if the number of honest parties on the committee plus twice the number of corrupted parties on the committee is less than twice the quorum threshold, and it is *valid* if the number of corrupted parties on the committee is less than the quorum threshold. We show that soft-vote committees are safe and all other committees (except $\mathcal{PV}$) are valid, with high probability.

To prove our bounds, we use the fact that an adaptive adversary has no advantage over a static adversary in terms of increasing $|\star\mathcal{V}_p^C|$ (with an idealized committee-election mechanism). This follows because an adaptive adversary learns any information about whether some honest party P is in $\star\mathcal{V}_p$ in only two cases: when P sends a vote (in which case it is too late for the adversary to gain any advantage from corrupting P), and when the adversary learns that P is *not* in $\star\mathcal{V}_p$ (in which case corrupting P is irrelevant). Thus, the bounds in this section can be derived from standard concentration inequalities using the bound on the fraction of corrupted parties.

Theorem 6.1. *Assume $N \geq 10^{12}$. For an adaptive adversary corrupting at most 20% of the parties, events* **A.1**–**A.6** *all occur in any given iteration except with probability at most* 2^{-128}.

The proof is in Appendix A.3.

6.2 Liveness Failures

Recall a committee is *live* if the number of honest parties on the committee is at least the quorum size. As in the previous section, an adaptive adversary has no advantage relative to a static adversary and we can bound the probabilities of liveness failures using standard concentration bounds.

Theorem 6.2. *Assume $N \geq 10^{12}$. For an adaptive adversary corrupting at most 20% of the parties, in any given iteration we have:*

- *Event* **B.1** *occurs except with probability at most* 2^{-135}.
- *Event* **B.2** *occurs except with probability at most* 2^{-94}.
- *Event* **B.3** *occurs except with probability at most* 2^{-16}.
- *Event* **B.4** *occurs except with probability at most* 2^{-12}.
- *Event* **B.5** *occurs except with probability at most* 2^{-12}.

The proof is in Appendix A.4.

6.3 The Red-Blue Game

In preparation for the last set of bounds, we introduce a combinatorial game we call the *red-blue game*. Intuitively, the game is designed to model an adversary attacking a simple protocol with only two committees (red and blue). The adversary's goal is to trick the honest committee members into making two contradictory decisions. This becomes our basis for analyzing interactions between committees in the real protocol, specifically for bounding events **C.1**–**C.4**. For now, we describe and analyze the game abstractly; Section 6.4 formally relates the game to the real protocol.

In the red-blue game, an adversary $\mathcal{A}$ interacts with two committees of different types, called *red* and *blue*. The adversary's goal is to convince the red committee to make one decision (call this "decision A") and the blue committee to make a contradictory decision (call this "decision B"). The adversary can influence honest parties' decisions by controlling which messages they receive (e.g., by forwarding a quorum for a certain value). It can also adaptively corrupt parties. However, the adversary cannot know whether a specific party P is on the red or blue committee until it corrupts P or P reacts in some way to a received message. Therefore, each time the adversary attempts to convince a red node to vote for decision A, it might inadvertantly cause a *blue* node to vote for decision A, which is counterproductive to its goal. (Parties will not vote more than once, so if the adversary causes a party to vote for the "wrong" decision, that party effectively becomes useless to the adversary.)

Fix parameters for the total number of parties N, expected number of red and blue parties $E_r, E_b \le N$, and red and blue thresholds T_r, T_b, with $1 \le T_r < E_r$ and $1 \le T_b < E_b$. Also, fix an upper bound on the fraction of corruptions $\alpha \in [0, 1]$, and define the number of corruptions $f = \alpha \cdot N$. The adversary is adaptive and can corrupt parties during the game (described below).

At the beginning of the game, vectors $\mathbf{v}_r, \mathbf{v}_b \in \{0,1\}^N$ are sampled, where each entry of $\mathbf{v}_r$ (resp., $\mathbf{v}_b$) is independently set to 1 with probability p_r (resp., p_b) and is set to 0 otherwise. Variables $\text{score}_r, \text{score}_b$ are initialized to 0. The adversary's ability to interact with the corrupted and honest parties is represented by a set of allowed queries, which have the following syntax and behavior:

(1) On *red query* (r, i) with $1 \le i \le N$, the adversary is given $\mathbf{v}_r[i]$, which is also added to score_r.
(2) On *blue query* (b, i) with $1 \le i \le N$, the adversary is given $\mathbf{v}_b[i]$, which is also added to score_b.
(3) On *corrupt query* $(\text{corrupt}, i)$, for $1 \le i \le N$, there are three cases:
   - If $\mathcal{A}$ had previously queried (r, i) it is given $\mathbf{v}_b[i]$, which is also added to score_b.
   - If $\mathcal{A}$ had previously queried (b, i) it is given $\mathbf{v}_r[i]$, which is also added to score_r.
   - Otherwise, $\mathcal{A}$ is given $\mathbf{v}_r[i]$ and $\mathbf{v}_b[i]$, which are added to score_r and score_b, respectively.

There are three restrictions on the behavior of $\mathcal{A}$: (1) $\mathcal{A}$ may make at most f corrupt queries overall, (2) for a given i, no additional queries can be made after a corrupt query, and (3) $\mathcal{A}$ cannot make both a red query and a blue query for a given i. (Figure 1 illustrates the allowed sequences of queries on an index i.) The game ends

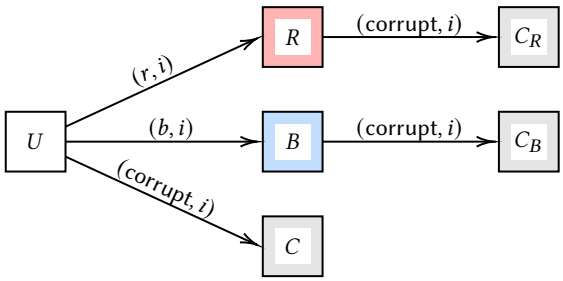

Figure 1: Allowed combinations of queries for a given index i, represented as state transitions. The adversary can issue at most f *corrupt* queries during a single game.

when there are no more legal queries for $\mathcal{A}$ to make. $\mathcal{A}$ *wins* if $\text{score}_r \ge T_r$ and $\text{score}_b \ge T_b$ at the end of the game.

Lemma 6.3. *Let* $\gamma \stackrel{\text{def}}{=} \frac{T_r}{E_r} + \frac{T_b}{E_b}$ *and assume* $1 + \alpha \le \gamma$. *Let* $\delta \stackrel{\text{def}}{=} (1+\alpha)/\gamma \le 1$. *Then* $\mathcal{A}$ *wins the above game with probability at most*

$$\min_{t \ge 0} e^{t(1+\alpha) - T_r \ln(1+t/E_r) - T_b \ln(1+t/E_b)}. \tag{1}$$

Proof. Let $p_r = E_r/N$, $p_b = E_b/N$, and $f = \alpha \cdot N$. We describe a modified version of the game that does not change the winning probability of the adversary. Now, at the beginning of the game we sample vectors $\mathbf{V}_r, \mathbf{V}_b \in \{0,1\}^N$, where each entry of $\mathbf{V}_r$ (resp., $\mathbf{V}_b$) is independently set to 1 with probability p_r (resp., p_b). We initialize $\text{ctr}_r, \text{ctr}_b$ to 0 and answer oracle queries of $\mathcal{A}$ as follows:

(1) On query (r, i), increment ctr_r. The adversary is given $\mathbf{V}_r[\text{ctr}_r]$, which is also added to score_r.
(2) On query (b, i), increment ctr_b. The adversary is given $\mathbf{V}_b[\text{ctr}_b]$, which is also added to score_b.
(3) On query $(\text{corrupt}, i)$, there are three cases:
   - If $\mathcal{A}$ had previously queried (r, i), increment ctr_b. The adversary is given $\mathbf{V}_b[\text{ctr}_b]$, which is also added to score_b.
   - If $\mathcal{A}$ had previously queried (b, i), increment ctr_r. The adversary is given $\mathbf{V}_r[\text{ctr}_r]$, which is also added to score_r.
   - Otherwise, increment ctr_r and ctr_b. The adversary is given $\mathbf{V}_r[\text{ctr}_r]$ and $\mathbf{V}_b[\text{ctr}_b]$, which are added to score_r and score_b, respectively.

The restrictions on $\mathcal{A}$ are the same as in the original game. Since the distribution on $\mathcal{A}$'s view in this modified game is identical to the distribution in the original game (as there is a one-to-one correspondence between vectors $\mathbf{v}_r, \mathbf{v}_b$ and "permuted" vectors $\mathbf{V}_r, \mathbf{V}_b$ that lead to the same view), $\mathcal{A}$'s winning probability is unchanged. In the remainder of the proof we analyze the modified game only.

For a given instance of the game, defined by specific values for $\mathbf{V}_r$ and $\mathbf{V}_b$, let i_r (resp., i_b) be the smallest value such that $\sum_{i=1}^{i_r} V_r[i] = T_r$ (resp., $\sum_{i=1}^{i_b} V_b[i] = T_b$). (If no such value exists, the adversary loses automatically.) Note that $\mathcal{A}$ can only possibly win a given instance of the game if $i_r + i_b \le N + f$. This follows since $\mathcal{A}$ wins the game only if, at the end of the game, $\text{ctr}_r \ge i_r$ and $\text{ctr}_b \ge i_b$. But it is clear from the rules of the game that $\text{ctr}_r, \text{ctr}_b \le N$ and $\text{ctr}_r + \text{ctr}_b \le 2f + (N - f) = N + f$.

Letting I_r, I_b be random variables corresponding to i_r, i_b, we can thus compute an upper bound on the winning probability of any

adversary by bounding the probability (over the choice of $\mathbf{V}_r, \mathbf{V}_b$) that $I_r + I_b \leq N + f$. Note that I_r is the sum of T_r geometric random variables, each with parameter p_r, and I_b is the sum of T_b geometric random variables, each with parameter p_b; the expectation of I_r is $T_r/p_r = N \cdot T_r/E_r$, and similarly $\mathbf{E}[I_b] = N \cdot T_b/E_b$.

Define $\mu = \mathbf{E}[I_r + I_b] = \gamma \cdot N$. We want to bound

$$\varepsilon \stackrel{\text{def}}{=} \Pr[\mathcal{A} \text{ wins}] \leq \Pr[I_r + I_b \leq N + f] = \Pr[I_r + I_b \leq \delta\mu].$$

Following the proof of [11, Theorem 3.1] up to Equation (3.4), we get:

$$\ln \varepsilon \leq t\delta\mu - T_r \ln(1 + t/p_r) - T_b \ln(1 + t/p_b),$$

for any $t \geq 0$. Rescaling t by a factor of N concludes the proof. □

6.4 Inconsistency Failures

We now use the red-blue game to prove lower bounds on the **C**-type events.

A critical feature of the protocol (and hence the game) is that the adversary does not know the identities of honest committee members in advance. If the adversary *did* know the identities of committee members, it could easily bias the probability of **C**-type events. For example, say the adversary learned the identities of honest parties in $\mathcal{SV}$ and $\mathcal{CV}$. It is then trivial for the adversary to prevent event **C.1** from occurring: it simply delivers a soft-quorum to all the honest members of $\mathcal{CV}$, while delaying all messages to the honest members of $\mathcal{NV}_{p,k}$ until after they have next_k-voted for $\perp$.

Theorem 6.4. *For an adaptive adversary corrupting at most 20% of the parties, events* **C.1**–**C.4** *all occur in any given iteration except with probability at most* 2^{-120}.

Proof. We prove the bound for **C.1**; the other items follow by an identical argument.

Fix some iteration p and an index k. An attacker $\mathcal{A}$ can only possibly obtain a p-cert-vote for some value $v \neq \perp$ from an honest party $P_i \in \mathcal{CV}_p$ who outputs v in Vote_p by local time 4λ, or from a corrupted party $P_i \in \mathcal{CV}_p$. Similarly, $\mathcal{A}$ can only possibly obtain a p-next_k-vote for a value $v' \neq v$ from an honest party $P_i \in \mathcal{NV}_{p,k}$ who did *not* output v in Vote_p by local time 4λ, or from a corrupted party $P_i \in \mathcal{NV}_{p,k}$.

We can thus map an adversary $\mathcal{A}$ attacking the protocol to an adversary $\mathcal{A}'$ in the red-blue game as follows. Let N be the number of parties, let $E_r = 1500$, $T_r = 1112$ be the expected size and quorum threshold for $\mathcal{CV}$, and let $E_b = 5000$, $T_b = 3838$ be the expected size and quorum threshold for $\mathcal{NV}_{p,k}$. Take $\alpha = 0.2$, and note that $1 + \alpha \leq T_r/E_r + T_b/E_b$. Vector $\mathbf{v}_r$ (resp., $\mathbf{v}_b$) indicates which parties are in $\mathcal{CV}_p$ (resp., $\mathcal{NV}_{p,k}$). If $\mathcal{A}$ delivers messages to a party P_i that would cause P_i to output[4] v in Vote_p by local time 4λ, then $\mathcal{A}'$ makes oracle query (r, i). If $\mathcal{A}$ causes P_i to *not* output v in Vote_p (whether by delaying messages or by delivering messages that would cause P_i to output $v' \neq v$), then $\mathcal{A}'$ makes oracle query (b, i). If $\mathcal{A}$ corrupts P_i, then $\mathcal{A}'$ makes query $(\text{corrupt}, i)$. Note that $\mathcal{A}'$ makes at most $\alpha \cdot N$ corrupt queries and never queries both (r, i) and (b, i) for the same i. (Indeed, it is not possible for $\mathcal{A}$ to cause P_i to output v in Vote_p and then subsequently cause P_i to not output v in Vote_p, or vice versa.) Finally, if $\mathcal{A}$ obtains a p-cert-quorum for $v \neq \perp$ and a p-next_k-quorum for $v' \neq v$ then $\mathcal{A}'$ wins the red-blue game.

It thus follows from Lemma 6.3 (using SageMath to approximate the minimum) that if an honest party receives a p-cert-quorum for $v \neq \perp$ then no honest party receives a p-next_k-quorum for $v' \neq v$ except with probability at most 2^{-128}. Taking a union bound over all k implies that event **C.1** occurs except with probability at most $250 \cdot 2^{-128} \leq 2^{-120}$. The remaining events can be bounded similarly, showing that event **C.2** occurs except with probability at most 2^{-128}, event **C.3** occurs except with probability at most $250 \cdot 2^{-222} \leq 2^{-214}$, and event **C.4** occurs except with probability at most 2^{-129}. Taking a union bound over the probabilities that any of the four events fails to occur completes the proof. □

6.5 Summary

By applying a union bound over the appropriate events whose probabilities were analyzed in the previous sections, we can compute upper bounds on the probabilities that a given execution of BA fails to be secure or intermittently fair.

Corollary 6.5. *A given execution of the Algorand BA protocol is secure except with probability at most* 2^{-55} *and intermittently fair except with probability at most* 2^{-29}.

Proof. Combining Theorems 6.1 and 6.4, we see that an execution of GC is secure except with probability $2^{-120} + 2^{-128} < 2^{-119}$. Similarly, using Theorem 6.2, we see that an execution of GC is intermittently fair except with probability $2^{-135} + 2^{-94} < 2^{-93}$. Taking a union bound over all $p < 2^{64}$ gives the stated bounds. □

Meanwhile, our lower bounds on the probabilities of Events **B.3**-**B.5** are not sufficiently tight to yield a meaningful bound via a similar union bound (even over a much smaller number of iterations). Tightening these bounds without modifying the expected committee sizes for $\mathcal{LV}$, $\mathcal{RV}$, and $\mathcal{DV}$ is an open question; in the meantime, one could increase the expected committee sizes (and quorum thresholds) for $\mathcal{LV}$, $\mathcal{RV}$, and $\mathcal{DV}$ until the lower bounds for Events **B.3**-**B.5** are large enough for the union bound to yield an acceptable bound.

ACKNOWLEDGMENTS

We thank Georgios Vlachos for providing essential intuition for the formulation of the red-blue game, and for helpful discussions regarding the details of the protocol implementation.

Work of Erica Blum and Jonathan Katz was supported by a grant from the Algorand Foundation, LTD. Work of Derek Leung was supported by an NSF Graduate Research Fellowship under Grant No. 1745302.

REFERENCES

[1] Algorand blockchain features specification v. 1.0. Algorand Foundation Github Repository, 2019. https://github.com/algorandfoundation/specs/blob/master/overview/Algorand_v1_spec-2.pdf.

[2] Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. *Distributed Comput.*, 36(1):3–28, 2023.

[3] Jing Chen, Sergey Gorbunov, Silvio Micali, and Georgios Vlachos. Algorand agreement: Super fast and partition resilient byzantine agreement, 2018. Available at https://eprint.iacr.org/2018/377.

[4] It is not hard to see that the optimal strategy for $\mathcal{A}$ is to use some fixed v and that the value of v is irrelevant.

[4] Jing Chen and Silvio Micali. Algorand, 2017. Available at https://arxiv.org/abs/1607.01341.
[5] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.
[6] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Adv. in Cryptology—Eurocrypt 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Heidelberg, 2018.
[7] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. In *3rd ACM PODC*, pages 103–118. ACM, August 1984.
[8] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.
[9] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*, pages 51–68. ACM, 2017. Full version available at http://eprint.iacr.org/2017/454.
[10] Sergey Gorbunov and Silvio Micali. Democoin: A publicly verifiable and jointly serviced cryptocurrency, 2015. Available at https://eprint.iacr.org/2015/521.
[11] Svante Janson. Tail bounds for sums of geometric and exponential variables. *Statistics & Probability Letters*, 135:1–6, 2018. Available at https://arxiv.org/abs/1709.08157.
[12] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *J. Comput. Syst. Sci.*, 75(2):91–112, 2009.
[13] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 120–130. IEEE Computer Society, 1999.
[14] Silvio Micali and Vinod Vaikuntanathan. Optimal and player-replaceable consensus with an honest majority. Technical Report MIT-CSAIL-TR-2017-004, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2017.
[15] Atsuki Momose and Ling Ren. Constant latency in sleepy consensus. In *Proc. ACM Conference on Computer and Communications Security, CCS 2022*, pages 2295–2308. ACM, 2022.
[16] Sebastien Roch. *Modern Discrete Probability: An Essential Toolkit.*

A DEFERRED PROOFS

A.1 Proof of Consistency

Lemma A.1. *If event* **A.1** *occurs in iteration p and honest parties P_i, P_j terminate* Vote_p *with output v_i, v_j, respectively, then $v_i = v_j$.*

Proof. Since P_i output v_i from Vote_p, it must have received at least 2267 p-soft-votes on v_i; thus, at least $2267 - |\mathcal{SV}_p^C|$ parties in $\mathcal{SV}_p^H$ sent p-soft-votes for v_i. Similarly, at least $2267 - |\mathcal{SV}_p^C|$ parties in $\mathcal{SV}_p^H$ sent p-soft-votes for v_j. Since each party in $\mathcal{SV}_p^H$ sends a p-soft-vote for at most one value, $v_i \neq v_j$ would imply $|\mathcal{SV}_p^H| \geq 2 \cdot (2267 - |\mathcal{SV}_p^C|) = 4534 - 2 \cdot |\mathcal{SV}_p^C|$, contradicting the occurrence of event **A.1**. □

We now consider an iteration of GC_p.

Lemma A.2. *Let P be an honest party, and consider the following events: (E1) P receives a p-soft-quorum for v; (E2) P terminates* Vote_p *with output v; (E3) P sends a p-cert-vote for v. Then E3 $\Rightarrow$ E2 $\Rightarrow$ E1.*

Proof. Inspecting the protocol description, we see that an honest party who terminates Vote_p with output v must have received a quorum of p-soft-votes for v (so E2 $\Rightarrow$ E1), and an honest party who sends a p-cert-vote for v must have terminated Vote_p with output v (so E3 $\Rightarrow$ E2). □

Note that E1 does not imply E2 (as P might receive a p-soft-quorum for v after it has already terminated GC_p), nor does E2 imply E3 (since P might terminate Vote_p with output $v \neq \perp$ too late to send a p-cert-vote).

Corollary A.3. *In a secure execution, if an honest party terminates* Vote_p *with output v or sends a p-cert-vote for v, then no honest party receives a p-next$_k$-quorum (for any k) or a p-redo-quorum for $v' \neq v$.*

Proof. This follows from the preceding lemma and the fact that events **C.3** and **C.4** occur. □

Lemma A.4. *Assume events* **A.3**, **A.6**, **C.1**, *and* **C.2** *occur in some execution. If an honest party receives a p-cert-quorum for v, then:*

(1) For $p' \geq p$ and all k, no honest party receives a p'-next$_k$-quorum for $\perp$ or a p'-down-quorum for $\perp$.
(2) For $p' \geq p$, if an honest party terminates $\text{GC}_{p'}$ it does so with grade at least 1.
(3) For $p' > p$, no honest party ever sets b to 0 in $\text{GC}_{p'}$.

Proof. We prove the lemma by induction on p'. When $p' = p$, the first claim follows directly from the occurrence of events **C.1** and **C.2**, and the second claim immediately follows from the first. For the inductive step, fix $p' > p$ and assume the first two claims hold for $p' - 1$. Then any honest party who runs $\text{GC}_{p'}$ does so using an input of the form $(v_i, v' \neq \perp, 1)$ and never sets $b = 0$ during execution of $\text{GC}_{p'}$ (thus proving the third claim). Therefore, no honest party will run steps 3c or 4c in an execution of $\text{GC}_{p'}$, and any honest party who runs step 3b in an execution of $\text{GC}_{p'}$ does so with $v' \neq \perp$. In other words, no honest party p'-next$_k$-votes for $\perp$ (for any k) or p'-down-votes for $\perp$. Since events **A.3** and **A.6** occur, there are not enough corrupted parties to form a p'-next$_k$-quorum (for any k) or a p'-down-quorum on their own and so the first and second claims hold. □

Lemma A.5. *Assume event* **A.1** *occurs in some iteration p. If there exists $v \neq \perp$ such that every honest party who runs GC_p uses input of the form $(\star, v, 1)$, and no honest party sets $b = 0$ while running GC_p, then for any honest party who terminates* Vote_p *with output v' it holds that $v' = v \neq \perp$.*

Proof. If an honest party output v' from Vote_p, it must have received a p-soft-quorum on v'. Occurrence of event **A.1** then implies that some honest party p-soft-voted for v'. Since every honest party who runs GC_p uses input $(\star, v, 1)$ and never sets $b = 0$ during execution of GC_p, it follows that any p-soft-vote sent by an honest party is for v. Thus $v' = v$. □

Lemma A.6. *In a secure execution, if an honest party receives a p-cert-quorum for v, then for all $p' > p$ any honest party who runs $\text{GC}_{p'}$ does so with input $(\star, v, 1)$.*

Proof. Lemma A.4 already shows that for all $p' > p$ no honest party terminates $\text{GC}_{p'-1}$ with grade 0. So any honest party who runs $\text{GC}_{p'}$ does so with input of the form $(\star, \star, 1)$. We prove by induction that the input is of the form $(\star, v, 1)$.

The base case is when $p' = p + 1$. Suppose an honest party P runs GC_{p+1} using input $(\star, v', 1)$. This implies P terminated GC_p with output $(v', 1)$ via termination condition T2, T3, or T4. In each case, we show $v' = v$.

- If P output $(v', 1)$ from GC_p due to condition T2, then for some k it must have received a p-next$_k$-quorum for v'. Occurrence of event **C.1** implies $v' = v$.

- If P output $(v', 1)$ from GC_p due to condition T3, it must have received a p-late-quorum for v'. Occurrence of event **A.4** implies that some honest party must have sent a p-late-vote for v', and hence must have output v' from Vote_p. Because there is a p-cert-quorum for v, however, the fact that event **A.2** occurred implies that some honest party must have p-cert-voted for v and thus must have output v from Vote_p. Lemma A.1 implies $v' = v$.
- If P output $(v', 1)$ from GC_p due to condition T4, it must have received a p-redo-quorum for v'. Because there is a p-cert-quorum for v, occurrence of event **A.2** implies that some honest party must have p-cert-voted for v and hence (by Corollary A.3) $v' = v$.

For the inductive step, assume any honest party who runs $\mathrm{GC}_{p'}$ does so with input of the form $(\star, v, 1)$; we show this also holds for $\mathrm{GC}_{p'+1}$. By Lemma A.4, any honest party who runs $\mathrm{GC}_{p'+1}$ must have terminated $\mathrm{GC}_{p'}$ with output $(v', 1)$ via termination condition T2, T3, or T4. As before, we show $v' = v$ in each case.

- If P output $(v', 1)$ from $\mathrm{GC}_{p'}$ due to condition T2, then for some k it must have received a p'-next_k-quorum for v'. Occurrence of event **A.3** implies that some honest party P must have p'-next_k-voted for v'. Since all honest parties hold $b = 1$ during execution of $\mathrm{GC}_{p'}$ (cf. Lemma A.4), P must have p'-next_k-voted for v' in step 3a or 3b. If it p'-next_k-voted for v' in step 3a, then Lemma A.5 implies $v' = v$. If P instead p'-next_k-voted for v' in step 3b, then v' is its original input and so again $v' = v$.
- If P output $(v', 1)$ from $\mathrm{GC}_{p'}$ due to condition T3, it must have received a p'-late-quorum for v'. Occurrence of event **A.4** implies some honest party must have sent a p'-late-vote for v' in step 4a. Lemma A.5 implies $v' = v$.
- If P output $(v', 1)$ from $\mathrm{GC}_{p'}$ due to condition T4, it must have received a p'-redo-quorum for v'. Occurrence of event **A.5** implies that some honest party must have sent a p'-redo-vote for v'. Since v' is that party's original input, we have $v' = v$.

This concludes the proof. □

Lemma A.7. *In a secure execution, if honest parties receive a p-*cert*-quorum for $v \neq \perp$ and a p'-*cert*-quorum for $v' \neq \perp$, then $v = v'$.*

Proof. Occurrence of event **A.2** implies that some honest party P must have p-cert-voted for v, and some honest party P' must have p'-cert-voted for v'. Lemma A.2 shows that P must have terminated Vote_p with output v, and P' must have terminated $\mathrm{Vote}_{p'}$ with output v'. If $p = p'$, Lemma A.1 implies $v = v'$. Otherwise, suppose $p' > p$. Lemma A.2 shows that P' must have received a p'-soft-quorum for v'. Occurrence of event **A.1** implies that some honest party must have p'-soft-voted for v'. But Lemma A.6 shows that any honest party that runs $\mathrm{GC}_{p'}$ does so with input of the form $(\star, v, 1)$, and moreover never sets b equal to 0 during execution of $\mathrm{GC}_{p'}$. This means that $v' = v$. □

Theorem 5.1 (Consistency). *Assume a secure execution of BA in an asynchronous network. If honest parties P, P' output v, v', respectively, then $v = v'$.*

Proof. P must have received a p-cert-quorum (for some p) for v; likewise, P' must have received a p'-cert-quorum (for some p') for v'. Lemma A.7 implies $v = v'$. □

A.2 Proofs of Liveness

We continue by proving several lemmas that will be useful in proving both of our main results.

Lemma A.8. *Consider any secure execution. For all p, if there is a value v such that every honest party who runs GC_p uses input of form $(\star, \perp, 0)$ or $(\star, v, 1)$, then any honest parties who terminate GC_p with grade $g = 1$ output the same value $v_p \neq \perp$.*

Proof. We first claim that if an honest party P terminates GC_p with output $(v_p, 1)$ then either (1) some honest party output v_p from Vote_p or (2) some honest party received a p-next_k-quorum (for some k) or a p-redo-quorum for v_p, and $v_p = v$. To see this, note that P must have terminated due to condition T2, T3, or T4. If it terminated via T2, it must have received a p-next_k-quorum (for some k) on v_p. By validity of $\mathcal{NV}_{p,k}$ (cf. event **A.3**), some honest party P' must have p-next_k-voted for v_p. Consider the sub-cases:

(1) If P' sent a p-next_k-vote for v_p in step 3a, then P' output v_p from Vote_p, as desired.
(2) If P' sent a p-next_k-vote for v_p in step 3b, then by our lemma's assumption $v_p \in \{v, \perp\}$. $v_p \neq \perp$, so $v_p = v$.
(3) P' did not p-next_k-vote for v_p in step 3c since $v_p \neq \perp$.

The case where P terminated via T3 is like the second sub-case above. Namely, if P terminated via T3, then it must have received a p-late-quorum for v_p. By validity of $\mathcal{LV}_p$ (cf. event **A.4**), some honest party P' must have p-late-voted for v_p. This implies that P' output v_p from Vote_p.

The case where P terminated via T4 is like the first sub-case above. Namely, if P terminated via T4, then it must have received a p-redo-quorum on v_p. By validity of $\mathcal{RV}_p$ (cf. event **A.5**), some honest party must have p-redo-voted on v_p. But then the assumption of the lemma implies that $v_p \in \{v, \perp\}$. Since $v_p \neq \perp$, we must have $v_p = v$.

Now, suppose honest parties P_i, P_j terminate GC_p with output $(v_i, 1)$ and $(v_j, 1)$, respectively, and apply the above claim to v_i and v_j. If honest parties output v_i, v_j from Vote_p, Lemma A.1 implies $v_i = v_j$. If an honest party output v_i from Vote_p and there is a p-next_k-quorum or p-redo-quorum on v_j (or vice versa), Corollary A.3 implies $v_i = v_j$. Finally, if $v_i = v$ and $v_j = v$ then $v_i = v_j$. □

Corollary A.9. *Consider any secure execution. Then for all p there is a value v_p such that every honest party who terminates GC_p with grade $g = 1$ outputs v_p, and every honest party who runs GC_{p+1} with input $(\star, v, 1)$ has $v = v_p$.*

Proof. We prove this by induction on p. All honest parties who run GC_1 use input $(\star, \perp, 0)$, so when $p = 1$ the claim follows from Lemma A.8. For the inductive step, assume the claim holds for GC_{p-1}. Then there is a value v_{p-1} such that all honest parties who terminate GC_{p-1} with grade $g < 2$ output either $(v_{p-1}, 1)$ or $(\star, 0)$. Consequently, any honest party who runs GC_p uses input of the form $(\star, \perp, 0)$ or $(\star, v_{p-1}, 1)$, and so the claim again follows from Lemma A.8. □

Lemma A.10. *Consider any secure execution. Suppose the network is good from time T to time $T+\delta$. Then:*

- *If some honest party P outputs v from Vote_p by time T, then by time $T+\delta$ each honest party either outputs v from Vote_p, advances to GC_{p+1}, or terminates BA.*
- *If some honest party P holds $b=0$ in GC_p by time T, then by time $T+\delta$ every honest party either holds $b=0$ in GC_p, advances to GC_{p+1}, or terminates BA.*

Proof. First note that because the network is good in the given interval, every honest party starts running GC_p by time $T+\delta$. (This follows since, by time T, honest party P must have seen quorums allowing it to terminate $GC_1, \ldots, \mathrm{GC}_{p-1}$; hence P forwards those quorums to all other parties by time T, and they are delivered to all honest parties by time $T+\delta$.)

If P outputs v from Vote_p by time T, then it sends a p-soft-quorum for v to all other parties by time T that is delivered by time $T+\delta$. If P' is some other honest party, there are three cases: (1) P' is still running Vote_p at time $T+\delta$; (2) P' has already output a value from Vote_p by time $T+\delta$; or (3) P' terminated Vote_p by time $T+\delta$ without outputting anything. In the first case, P' outputs v from Vote_p after receiving the p-soft-quorum from P. In the second case, consistency of Vote_p (cf. Lemma A.1) implies that P' already output v from Vote_p. The third case only occurs if P' has already advanced to GC_{p+1} or terminated BA. This proves the first claim.

For the second claim, regardless of whether P began running GC_p with $b=0$ or whether it set $b=0$ during execution of GC_p, it must have received a $(p-1)$-next_k-quorum for $\perp$ or a $(p-1)$-down-quorum for $\perp$ by time T that it forwards to all other parties; hence all other parties receive the appropriate quorum by time $T+\delta$. So if some honest party P' is still running GC_p at time $T+\delta$ it will also set $b=0$ (if it does not already hold $b=0$); otherwise, P' must have already advanced to GC_{p+1} or have terminated BA. □

We say *all honest parties begin running GC_p together at time T* if every honest party begins running GC_p at some time in the interval $[T, T+\delta]$. When p is irrelevant, we simply say that honest parties begin running *GC* together at time T.

Lemma A.11. *Suppose that during GC_p, event* **B.1** *occurs. Assume all honest parties begin running GC_p together at time T, and that the network is good from time T to time $T+5\delta$. Then all honest parties terminate BA by time $T+5\delta$.*

Proof. First note that if some honest party P terminates BA before time $T+4\delta$, then all honest parties will do so by time $T+5\delta$ (since P will forward the relevant cert-quorum to all other parties) and we are done. So, assume no honest party terminates BA before time $T+4\delta$. Since no honest party also sends a p-next-, p-late-, p-redo-, or p-down-vote before time $T+4\delta$, and all those committees are valid, every honest party is still running GC_p at time $T+4\delta$.

Since event **B.1.1** happens, the honest leader sends a p-proposal for some value v_ℓ by time $T+\delta$, which is delivered to all honest parties by time $T+2\delta$. At that time, all honest parties' local clocks (within GC_p) are at time at most 2δ, so all honest $P_i \in \mathcal{SV}_p$ will send a p-soft-vote for either the leader's value v_ℓ (if $b=0$) or its own value v_i (if $b=1$) by time $T+3\delta$.

Moreover, all these votes are for the same value. If the leader holds $b=0$ at the time it sends its proposal, then Lemma A.10 shows that all honest parties hold $b=0$ by time $T+2\delta$; since that is the earliest time any honest party can vote, all honest parties send a p-soft-vote for the leader's value. Otherwise, the leader must have used input $(\star, v_\ell, 1)$ to GC_p, and Corollary A.9 shows that any honest party who also holds $b=1$ at the time they send a p-soft-vote will vote for v_ℓ.

Thus, using the fact that $\mathcal{SV}_p$ is live (cf. event **B.1.2**), by time $T+4\delta$ each honest party receives a p-soft-quorum for v_ℓ and so terminates Vote_p with output v_ℓ by that time. (Lemma A.1 implies that no honest party can terminate Vote_p with a different output.) This means that each honest party in $\mathcal{CV}_p$ will send a p-cert-vote for v by time $T+4\delta$. Since $\mathcal{CV}$ is live (cf. event **B.1.3**), by time $T+5\delta$ every honest party receives a quorum of p-cert-votes and terminates BA. □

A.2.1 Liveness under partial synchrony. It is useful to define a function $\mathsf{Timer}(\cdot,\cdot)$ characterizing the timing of late-, redo-, and down-votes. Given a time T and a number c, $\mathsf{Timer}(T,c)$ is an upper bound on the time when all honest parties will advance to GC_{p+c}, where p is the highest iteration that any party is in at time T. Roughly, this quantity depends on how quickly a late-, redo-, or down-quorum forms in each new iteration, which in turn depends on when the parties first send these votes. Unlike next_k-votes, which follow an exponential backoff, these votes occur at fixed intervals. The exact timing depends on several protocol parameters: the network delay parameters δ, Λ (set by the model), and the "fast recovery" protocol parameter λ_f. Formally:

$$\mathsf{Timer}(T,c) = T + c \cdot (\max\{4\delta, \Lambda\} + 5\delta + 5\lambda_f).$$

With this notation in hand, we can begin our analysis, starting with an upper bound on the time to advance through a single execution of graded consensus.

Lemma A.12. *Assume an eventually-fair execution, and that the network is good from time T to time $\mathsf{Timer}(T,1)$. Then either all honest parties begin running a new iteration of GC together at some time $T' \le \mathsf{Timer}(T,1) - \delta$ or all honest parties terminate BA by time $\mathsf{Timer}(T,1)$.*

Proof. Let p be maximal such that some honest party is running GC_p at time T. The lemma holds if we can show that by time $T_0 = \mathsf{Timer}(T,1) - \delta$ some honest party terminates GC_p, or that by time $\mathsf{Timer}(T,1)$ all honest parties terminate GC_p. Since all honest parties start running GC_p by time $T+\delta$, that means we may assume all honest parties are still running GC_p at time T_0. Consider three possibilities:

Case 1: By time $T_1 = T_0 - 2\delta - 2\lambda_f$, some honest party terminates Vote_p with output $v \neq \perp$. By Lemma A.10, all honest parties terminate Vote_p with output v by time $T_1 + \delta \ge T + \delta + \max\{4\delta, \Lambda\}$. Each honest party on the p-late-vote committee will therefore send a p-late-vote for v by time $T_1 + \delta + 2\lambda_f = T_0 - \delta$, and liveness of the p-late-vote committee (cf. event **B.3**) then implies that all honest parties receive a p-late-quorum for v by time T_0. Thus, every honest party terminates GC_p by time $T_0 = \mathsf{Timer}(T,1) - \delta$.

Case 2: Case 1 does not hold, but some honest party has $b=0$ by time $T_2 = T_1 - \delta - 2\lambda_f$. Lemma A.10 shows that by

time $T_2 + \delta \geq T + \delta + \max\{4\delta, \Lambda\}$ all honest parties have $b = 0$. Each honest party on the p-down-vote committee will thus send a p-down-vote for $\perp$ by time $T_2 + \delta + 2\lambda_f = T_1$; liveness of the p-down-vote committee (cf. event **B.5**) implies that all honest parties receive a p-down-quorum for $\perp$ by time $T_1 + \delta \leq T_0$. Thus, every honest party terminates GC_p by time $T_0 = \mathsf{Timer}(T, 1) - \delta$.

Case 3: Both the above do not hold. In this case all honest parties on the p-redo-vote committee send a p-redo-vote by time $T + \delta + \max\{4\delta, \Lambda\} + \lambda_f \leq T_2$; Corollary A.9 shows that all those votes are on the same value v (since all honest parties hold $b = 1$ and hence used input $(\star, v, 1)$ to GC_p). Liveness of the p-redo-vote committee (cf. event **B.4**) implies that all honest parties receive a p-redo-quorum by time $T_2 + \delta \leq T_0$. Thus, every honest party terminates GC_p by time $T_0 = \mathsf{Timer}(T, 1) - \delta$. □

Corollary A.13. *Fix $T \geq \mathsf{GST}$, let p be the highest iteration any honest party is running at time T, and let $c > 0$. In an eventually-fair execution, by time $\mathsf{Timer}(T, c)$ either all honest parties terminate BA, or all honest parties begin running iteration $p + c$ of GC together, .*

Proof. This follows by induction on c, with the base case following directly from Lemma A.12 and the inductive case by recursive application of the lemma. □

Theorem 5.2. *Assume an eventually fair execution of BA in a partially synchronous network, and $\mathsf{GST} \leq \mathsf{wakeup}_{\mathsf{min}}(230)$. Then all honest parties terminate BA by time $\mathsf{GST} + 60 \cdot \Delta + 4\delta$, where $\Delta = \max\{4\delta, \Lambda\} + 5\delta + 5\lambda_f$.*

Proof. By Lemma A.13, either all honest parties terminate the BA protocol by time $\mathsf{Timer}(\mathsf{GST}, 60)$ (in which case we are done) or begin running GC_{p+60} together by time $\mathsf{Timer}(\mathsf{GST}, 60) - \delta$. In the latter case, Lemma A.11 shows that honest parties terminate BA by time $\mathsf{Timer}(\mathsf{GST}, 60) + 4\delta$. □

A.2.2 Liveness in the synchronous-enough model. We use the following notation related to voting rounds and timers: if an honest party P_i is running GC_p and $\mathsf{wakeup}_{\mathsf{min}}(k) \leq \mathsf{clock}_i < \mathsf{wakeup}_{\mathsf{min}}(k+1)$, we say P_i is "in voting round (p, k)" (or just "in voting round k" when p is clear from context). We define a natural notion of comparison for voting rounds, so that $(p, k) \leq (p', k')$ if either $p = p'$ and $k \leq k'$, or $p < p'$. In the same vein, at any time T, we refer to the maximum (resp., minimum) voting round of any honest party as the *highest (resp., lowest) voting round at time T*.

We first state a useful lemma showing that parties who take the same branch in Step 3 must vote for the same value:

Lemma A.14. *In an execution of GC_p, honest parties $P_i, P_j \in \mathcal{NV}_{p,k}$ that run the same branch (a, b, or c) of Step 3 also vote for the same value.*

Proof. If P_i, P_j are in (a), the claim follows from Lemma A.1. If they are in (b), it follows from Lemma A.9. Finally, if they are in (c), both parties vote for $\perp$. □

Lemma A.15. *Consider a intermittently-fair execution, and assume the network is synchronous-enough. Suppose the network becomes good at time T, and let (p, k) be the highest voting round of any honest party at time $T - \delta$. While the network remains good, for any $p \leq p' \leq p + 60$ and $3 \leq r \leq 15$, all honest parties begin a strictly higher iteration or terminate BA by time $T + \mathsf{wakeup}_{\mathsf{min}}(k + r + 1) + \delta$. Moreover, while the network is good, all honest parties begin running new iterations together.*

Proof. If some honest party advances to a new iteration or terminates BA, then all honest parties will terminate within time δ of the network becoming good, so consider a point in time such that no honest party has yet advanced or terminated BA.

By time $T + \delta$, either all honest parties are running GC_p, or some honest party is running a higher iteration. In the latter case, all honest parties catch up within time δ, and we are done. Otherwise, all honest parties are still running GC_p.

We have an intermittently-fair execution, so event **B.2** occurs, and consider some $k_3 > k_2 > k_1 \geq k$ for which $k_3 \leq k + 15$ and $|\mathcal{NV}^H_{p,k_\ell}| \geq 3838$ for all $\ell \in \{1, 2, 3\}$. By definition, GC_p begins after time $T + \delta$. Furthermore, by Lemma A.10, if any honest party has seen Vote_p output a value v' or has set $b = 0$, all honest parties have done the same by time $T + \delta$. Thus, if the adversary does nothing, or if all honest parties are in branch (a) and will not switch, then all honest members of the committee will be in the same branch of Step 3 at the time they vote; by Lemma A.14, they will all vote for the same value. In this case, a quorum forms and parties will advance within time δ of the end of the voting round.

The other case is that the adversary causes at least one honest party to switch to either branch (a) or (c) prior to voting. It is still possible for a quorum to form in voting round k_1 even if parties take different branches, but for simplicity assume no quorum forms and we proceed to voting round k_2. Consider three subcases.

(1) If all honest parties are in (a) by the start of voting round k_2, the adversary can no longer disrupt agreement, so we advance.
(2) If all honest parties are in (c) when they vote, they advance.
(3) If the parties are split across (a) and (c) when they vote in round k_2, assume for simplicity that no quorum forms in round k_2. Since $k_3 > k_2$, by the start of round k_3, all honest parties are in (a) and the adversary has no way to disrupt agreement, so the case proceeds as before.

Thus, for any $3 \leq r \leq 15$, the honest parties advance by $T + \mathsf{wakeup}_{\mathsf{min}}(k + r + 1) + \delta$ (i.e., within δ of the end of round $k+r$). □

If we consider all such values $p' \in \{p, \ldots p + 60\}$, we obtain the following corollary:

Corollary A.16. *Suppose there is a time T such that all honest parties start running iteration p of GC during the interval $[T, T + \delta]$. For any $3 \leq r \leq 15$, if the network remains good from time $T - \delta$ until at least time $T + L \cdot (\mathsf{wakeup}_{\mathsf{min}}(r + 1) + \delta)$ (with $L \leq 60$), then parties advance to at least iteration $p + L$ (or terminate BA) within that time.*

We now have all of the machinery needed to prove liveness in the synchronous-enough model (Theorem 5.3).

Proof. Let (p, k) denote the highest voting round at time $T - \delta$. The network has been bad for at most time β, so the largest timer any honest party has is β. This implies $k \leq \log_2(\beta - \max\{4\delta, \Lambda\})/\delta$. Lemma A.15 implies that, taking $r = 15$, the honest parties begin a new iteration GC_p together within time that is no more than $\mathsf{wakeup}_{\mathsf{min}}(\log_2(\beta - \max\{4\delta, \Lambda\})/\delta + 16) + \delta$.

Since we are in an intermittently-fair execution, there is an iteration $\mathsf{GC}_{p'}$ with $p \le p' \le p + 60$ where the conditions listed in event **B.1** have occurred. By Lemma A.11, all parties terminate 5δ after this iteration $\mathsf{GC}_{p'}$ begins. The latest point at which this happens (if we have not already terminated) must be $T + (\mathsf{wakeup}_{\mathsf{min}}(\log_2(\beta - \max\{4\delta, \Lambda\})/\delta + 16) + \delta) + 60 \cdot (\mathsf{wakeup}_{\mathsf{min}}(16) + \delta)$ by Corollary A.16 using $L = 60$ and $r = 15$; this follows since the network is good for all this time (i.e., it is less than $T + \Gamma$). □

A.3 Proof of Theorem 6.1

We begin by stating several probability lemmas we rely on.

Definition A.17. Consider two random variables X and X'. We say that X *dominates* X' if and only if for any $t \in \mathbb{R}$ it holds that $\Pr[X \ge t] \ge \Pr[X' \ge t]$.

Let n be a positive integer and $0 \le p \le 1$. It is well known that when $\lambda = np$ is held constant and n is large enough, a Binomial random variable of parameter (n, p) can be approximated by a Poisson random variable of parameter λ, hence the following lemma.

Lemma A.18. *Let $n \in \mathbb{N}$ and $0 \le p \le 1$. Let X be a Binomial random variable of parameters (n, p), and let X' be a Poisson random variable of parameter $\lambda' \ge np + p$. Then X' dominates X.*

Proof. By [16, Example 4.2.5], a Poisson random variable of parameter λ' dominates a Poisson variable of parameter $\lambda'' \le \lambda'$. So we just need to prove the lemma for $\lambda' = np + p$.

Let $m = n+1$. X is a Binomial random variable of parameters $(m-1, \lambda'/m)$. Using [16, Example 4.2.8], we have that X' dominates X. □

We are ready for a lemma which will be useful for establishing a committee's safety.

Lemma A.19. *Let t, n_Y, n_Z be three positive integers and set $n = n_Y + n_Z$. Let $X_1, \ldots, X_n$ be independent Bernoulli random variables of parameter p. Let Y' and Z' be Poisson variables of parameters $\lambda'_Y \ge pn_Y + p$ and $\lambda'_Z \ge pn_Z + p$, respectively. Then:*

$$\Pr\left[\sum_{i=1}^{n_Y} 2X_i + \sum_{i=n_Y+1}^{n} X_i \ge t\right] \le \Pr[2Y' + Z' \ge t]. \tag{2}$$

Proof. Let $Y = \sum_{i=1}^{n_Y} X_i$ and $Z = \sum_{i=n_Y+1}^{n} X_i$. Those random variables are independent Binomial randomial variables of parameters (n_Y, p) and (n_Z, p) respectively.

Lemma A.18 shows that Y, Z are dominated by Y', Z', respectively. Using [16, Theorem 4.2.3] concludes the proof. □

To show a committee's safety, we rely on computer approximation: we use the fact we can choose the parameters of the lemma so that the right hand side of eq. (2) does not depend on N (as long as $N \ge 10^{12}$).

We next record the following result for establishing a committee's validity, which follows easily from a standard Chernoff bound:

Lemma A.20. *Assume there are N parties, at most $\alpha \cdot N$ of whom are corrupted, and that each party is selected to be on some committee with independent probability E/N. Then the probability that there are $(1 + \epsilon)\alpha E$ or more corrupted parties on the committee is at most $e^{-\epsilon^2 \alpha E/(2+\epsilon)}$.*

If Q is a quorum threshold, $p = E/N$ is a probability of selection, and α is a fraction of corrupted parties, then the corrupted parties do not form a quorum except with probability at most $e^{-(\alpha E - Q)^2/(\alpha E + Q)}$. Note that this loose bound is sufficient for us.

The individual Chernoff bounds for Theorem 6.1 are written out below; the theorem follows from a union bound.

(1) A soft-vote committee's expected size is $E = 2990$. Let $n_Y = \alpha E$, $n_Z = (1 - \alpha)E$, and $n = n_Y + n_Z = E$. Let $p = E/N$, the probability that each party is selected. Let $t = 2 \cdot 2267$. Let $\lambda'_Y = \alpha E + E/10^{12}$ and $\lambda'_Z = (1 - \alpha)E + E/10^{12}$. We apply lemma A.19 with the parameters above and get:

$$\Pr[|\mathcal{SV}_p^H| + 2|\mathcal{SV}_p^C| \ge 2 \cdot 2267] \le \Pr[2Y' + Z' \ge t],$$

where Y', Z' are independent Poison random variables of parameters λ'_Y, λ'_Z, respectively, which as stated above do not depend on N. Using SageMath, we get that the probability that $|\mathcal{SV}_p^H| + 2|\mathcal{SV}_p^C| \ge 2 \cdot 2267$ is at most $2^{-128.2}$.

(2) A cert-vote committee's expected size is $E = 1500$. By Lemma A.20, the probability that $|\mathcal{CV}_p^C| \ge 1112$ is at most 2^{-673}.

(3) A next_k-vote committee's expected size is $E = 5000$. Lemma A.20 shows that $|\mathcal{NV}_{p,k}^C| \ge 3838$ for any fixed k with probability at most 2^{-2401}. Applying a union bound over k gives $250 \cdot 2^{-2401} \le 2^{-2393}$.

(4) A late-vote committee's expected size is $E = 500$. By Lemma A.20, the probability that $|\mathcal{LV}_p^C| \ge 320$ is at most 2^{-166}.

(5) A redo-vote committee's expected size is $E = 2400$. By Lemma A.20, the probability that $|\mathcal{RV}_p^C| \ge 1768$ is at most 2^{-1064}.

(6) A down-vote committee's expected size is $E = 6000$. By Lemma A.20, the probability that $|\mathcal{DV}_p^C| \ge 4560$ is at most 2^{-2827}.

A.4 Proof of Theorem 6.2

Lemma A.21. *Let X be a Binomial random variable of parameters (n, p) with $p = \lambda/n$. Let Y be a Poisson random variable of parameter λ. Then for any $t \le \lambda$:*

$$\Pr[X \le t] \le \Pr[Y \le t] \cdot e^{(2\lambda t + t)/(2n)}.$$

When n is large enough $e^{(t^2+t)/(2n)}$ is essentially 1. The reason this lemma is useful is that $\Pr[Y \le t]$ is independent of n and thus we will be able to bound $\Pr[X \le t]$ easily for any large enough n, by estimating $\Pr[Y \le t]$.

Proof. We have:

$$\begin{aligned}\Pr[X \le t] &= \sum_{k=0}^{t} \binom{n}{k} p^k (1-p)^{n-k} \\ &= \sum_{k=0}^{t} \frac{\lambda^k}{k!} \left(\prod_{i=0}^{k-1} (1 - i/n)\right) (1 - \lambda/n)^{n-k}.\end{aligned}$$

Since $\ln(1+x) \le x$, we have:

$$\begin{aligned}
&\ln\left(\left(\prod_{i=0}^{k-1}(1-i/n)\right)(1-\lambda/n)^{n-k}\right)\\
&= \sum_{i=0}^{k-1}\ln(1-i/n) + (n-k)\ln(1-\lambda/n)\\
&\le -\sum_{i=0}^{k-1} i/n - (n-k)\lambda/n\\
&= -\frac{(k-1)k}{2n} - \lambda + \frac{k\lambda}{n}\\
&= -\lambda + \frac{(2\lambda-k+1)\cdot k}{2n} \le -\lambda + \frac{2\lambda t + t}{2n},
\end{aligned}$$

where the last inequality comes from $0 \le k \le t \le \lambda$.

Therefore, we have:

$$\Pr[X \le t] \le \sum_{k=0}^{t}\frac{\lambda^k}{k!}e^{-\lambda}\cdot e^{(2\lambda t+t)/(2n)},$$

which concludes the proof. □

Using Lemma A.21 with $t = T-1$ and $n = (1-\alpha)N$ gives:

Lemma A.22. *Assume $N \ge 10^{12}$, at most $\alpha \cdot N$ of whom are corrupted, and that committees are formed by choosing each party to be on the committee with independent probability E/N. Then the probability that the committee includes fewer than T honest parties is at most $\Pr[Y \le T-1]\cdot e^{(2\lambda(T-1)+T-1)/(2(1-\alpha)10^{12})}$, where Y is a Poisson random variable of parameter $\lambda = E$.*

The next theorem follows immediately from the above and estimating $\Pr[Y \le T-1]$ using SageMath.

Theorem A.23. *Assume $N \ge 10^{12}$. For an adaptive adversary corrupting at most 20% of the parties, the following hold for any p:*

- *The probability that $|\mathcal{PV}_p^H| < 1$ is at most 2^{-23}.*
- *The probability that $|\mathcal{SV}_p^H| < 2267$ is at most $2^{-7.7}$.*
- *The probability that $|\mathcal{CV}_p^H| < 1112$ is at most $2^{-7.7}$.*
- *For any k, the probability that $|\mathcal{NV}_{p,k}^H| < 3838$ is at most $2^{-7.7}$.*
- *The probability that $|\mathcal{LV}_p^H| < 320$ is at most 2^{-16}.*
- *The probability that $|\mathcal{RV}_p^H| < 1768$ is at most $2^{-12.2}$.*
- *The probability that $|\mathcal{DV}_p^H| < 4560$ is at most $2^{-12.1}$.*

Theorem A.23 takes care of all of the type-**B** events but **B.1** and **B.2**; we address these by combining the bounds seen so far.

Corollary A.24. *Consider some p, and let $c \in \mathbb{N}$ and $N \ge 10^{12}$.*

- *Except with probability at most $2^{-2.25c}$, $\exists p' \in \{p,\dots,p+c\}$ such that all of the following hold:*
  - *$|\mathcal{PV}_{p'}| > 0$ and the p-leader is honest.*
  - *$|\mathcal{SV}_{p'}^H| \ge 2267$.*
  - *$|\mathcal{CV}_{p'}^H| \ge 1112$.*
- *Fix integers $k \in [1..235], c' \ge 0$ s.t. $k + c' \le 250$. Except with probability at most $c \cdot \sum_{i=0}^{2}\binom{c'+1}{i}2^{-7.7(c'+1-i)}\cdot(1-2^{-7.7})^i$, there exist distinct $k_1,k_2,k_3 \in \{k,\dots,k+c'\}$, such that $|\mathcal{NV}_{p',k_\ell}^H| \ge 3838$ for all $p' \in \{p,\dots,p+c\}$ and $\ell \in \{1,2,3\}$.*

Proof. For the first part, we want to compute an upper bound on the probability that in a sequence of c iterations, there is no iteration where three good events all occur. The probability that an honest party is the leader in some iteration p is at least $(1-\alpha)$ times the probability that $|\mathcal{PV}_{p'}| > 0$, i.e., at least $(1-\alpha)(1-2^{-14.42})$. The probabilities for the other two events were computed above. Taking a union bound, the probability that any of the three events fails to occur in a particular iteration is at most $\rho := 1-(1-\alpha)(1-2^{-14.42}) + 2^{-4.76} + 2^{-4.78} < 2^{-2.25}$, and so the probability that at least one of the three good events does not occur in every $p' \in \{p,\dots,p+c\}$ is at most $2^{-2.25c}$.

For the second part, we seek the probability that for no more than two distinct $k \in \{k,\dots,k+c'\}$, $|\mathcal{NV}_{p',k}^H| < 3838$. This is simply the probability of at most 2 failures over $c'+1$ Bernoulli trials (representing committees) parameterized by a failure probability $|\mathcal{NV}_{p',k}^H| < 3838$, which is $2^{-7.7}$ by Theorem A.23. Taking a union bound over all c of these events gives the desired expression. □

By setting $c = 60$ and $c' = 15$, this allows us to conclude our proof of Theorem 6.2.

Theorem A.25. *Assume $N \ge 10^{12}$. For any iteration p and an adaptive adversary corrupting at most 20% of the parties:*

- *The probability that event **B.1** does not hold is $2^{-135.2}$.*
- *The probability that event **B.2** does not hold is $2^{-94.7}$.*