Towards Practical Sleepy BFT

Dahlia Malkhi
Chainlink Labs
San Francisco, CA, USA
dahliamalkhi@gmail.com

Atsuki Momose*
University of Illinois at Urbana-Champaign
Urbana, IL, USA
atsuki.momose@gmail.com

Ling Ren
University of Illinois at Urbana-Champaign
Urbana, IL, USA
renling@illinois.edu

ABSTRACT

Bitcoin's longest-chain protocol pioneered consensus under dynamic participation, also known as sleepy consensus, where nodes do not need to be permanently active. However, existing solutions for sleepy consensus still face two major issues, which we address in this work. First, existing sleepy consensus protocols have high latency (either asymptotically or concretely). We tackle this problem and achieve 4Δ latency (Δ is the bound on network delay) in the best case, which is comparable to classic BFT protocols without dynamic participation support. Second, existing protocols have to assume that the set of corrupt participants remains fixed throughout the lifetime of the protocol due to a problem we call *costless simulation*. We resolve this problem and support growing participation of corrupt nodes. Our new protocol also offers several other important advantages, including support for arbitrary fluctuation of honest participation as well as an efficient recovery mechanism for new active nodes.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**.

KEYWORDS

BFT Protocols; Blockchain; Dynamic Participation; Sleepy Model

ACM Reference Format:
Dahlia Malkhi, Atsuki Momose, and Ling Ren. 2023. Towards Practical Sleepy BFT. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3576915.3623073

1 INTRODUCTION

Byzantine fault-tolerant (BFT) consensus, a decade-old problem in distributed computing and cryptography, allows a group of nodes to reach an agreement in the presence of corrupted nodes [25, 33]. Traditional consensus research has mainly focused on the static participation model where all honest nodes remain active throughout the execution [9, 23]. The celebrated Bitcoin protocol [29] pioneered consensus in a dynamic participation model, enabling nodes to switch between active and inactive states spontaneously without any prior notice. Furthermore, participants do not need to know how many other participants are currently active in the system. This dynamic and unknown participation model was later formalized as the *sleepy model* [32]. The sleepy model allows an arbitrary subset of n_t nodes out of a total of N eligible participants to be active at any given time t. The status of active/inactive can be determined arbitrarily by an adversary, making the participation dynamic and unknown.

Inspired by Bitcoin's *longest-chain* protocol, there have been many recent proposals employing the longest-chain paradigm for the sleepy model [12, 14, 32]. However, all of these protocols face two major problems, which we highlight in this work.

Problem 1: Latency. A notable drawback of the longest-chain paradigm has been its long latency. The latency of Nakamoto's longest-chain protocol depends on several factors, including the security parameter and the actual level of participation [31, 34]. Substantial effort has been made to remove these dependencies [6, 20, 22, 26], culminating in the work of Momose-Ren [28] that achieves constant latency. However, despite being asymptotically optimal, the concrete latency of Momose-Ren is still quite large. Specifically, its latency is at least 16Δ[1] (where Δ is the bound on network delay). This is much slower than classic BFT protocols operating under the static participation model, which can make decisions within Δ time [3].

Our first result is to address this issue and achieve concretely small latency in the sleepy model. Specifically,

Theorem 1.1 (informal). *Assuming a verifiable random function (VRF) and public-key infrastructure (PKI), there exists an atomic broadcast protocol with (best-case) 4Δ latency in the sleepy model where up to $f_t < n_t/2$ corrupt nodes are active at any given time t.*

Following prior works, we focus on the atomic broadcast problem [11], i.e., achieving consensus on a *linearizable* log.

The core ingredient of our protocol is a new construction in the classic view-based approach to BFT atomic broadcast. We construct each view from the composition of a graded proposal election (GPE) and a sequence of graded agreements (GA). We observe that most classic construction puts the decision at the end of each view after sequential invocations of GA, and this introduces a large latency. We instead push back most of the tasks done by the sequential GAs and make a decision earlier after minimum steps (in GPE). This results in a significant improvement in the best-case latency.

*Lead author.

[1] Assuming perfectly synchronized clocks for lockstep execution. This latency will further increase under the assumption of a bounded clock skew.

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11…$15.00
https://doi.org/10.1145/3576915.3623073

Problem 2: Costless simulation. Besides latency, another major limitation of previous sleepy consensus protocols (without proof-of-work) do not allow dynamic participation of corrupt nodes. The Bitcoin protocol allows both honest and corrupt nodes to fluctuate dynamically as long as there is an honest majority. However, once we remove the computationally expensive proof-of-work, we lose the crucial property that computational effort is not reusable. Because of this, the original sleepy model [32] by Pass-Shi assumes stable participation of corrupt nodes. To elaborate, at each point in time t, they allow a maximum of $f_t = O(n)$ active corrupt nodes where n represents the minimum count of active nodes throughout the entire execution. In other words, even if the overall (honest plus corrupt) participation level fluctuates tremendously throughout the execution, the count of corrupt participants must always be bounded by the minimum participation level rather than the current level. This assumption is hard to justify in practice. Suppose only dozens of nodes were active in the beginning, but a million nodes are active a few years later when a system attains widespread recognition. Even at that later time, the number of corrupt nodes must be limited to a few dozen out of the one million nodes!

This problem arises due to an attack known as *costless simulation* [15]. To elaborate, when a corrupt node becomes active, it can pretend to have always been active in the past. It can fabricate messages that were supposed to be sent when it was not active in an attempt to alter the consensus results in the past. Our protocol tackles this problem and accommodates growing corrupt participation proportional to the active participation level (formalized in Section 2).

Other advantages. Along the way, we also offer several other advantages elaborated below.

- We introduce a novel technique to eliminate the assumption of *eventual stable participation*, a requirement for ensuring liveness in Momose-Ren. Intuitively, their protocol assumes that eventually, a large fraction of active nodes stays active for a certain period of time to make progress. In contrast, our protocol advances consistently even under arbitrary churn in active participants, offering guarantees akin to those of longest-chain protocols.
- The original sleepy model by Pass-Shi assumes that nodes upon waking up receive all past messages including those sent during their sleep, which is impractical. Momose-Ren addresses this issue by introducing a concrete recovery mechanism for newly active nodes to retrieve only essential messages from other active nodes. However, in Momose-Ren, nodes are required to recover messages from the past $\Omega(\kappa)$ rounds (besides the log contents) where κ is a security parameter. Moreover, the recovery protocol introduces additional overhead to the main protocol, resulting in an increased latency of at least 19Δ. In contrast, our recovery protocol mandates nodes to recover messages from only the constant number of past rounds (in fact less than a dozen). This protocol also avoids introducing any additional latency.

Organization. The rest of this paper is organized as follows. After defining the model and some primitives in Section 2, we provide the overview of our protocol in Section 3. We present our graded agreement (GA) protocol in Section 4 graded proposal election protocol (GPE) in Section 5. Then, building on the GA and GPE protocols, we present our atomic broadcast protocol in Section 6. Finally, we review some related works in Section 8 and conclude this paper with some future works in Section 9.

2 MODEL AND DEFINITIONS

We consider a system comprising a total of N nodes communicating over a synchronous network. Note that network synchrony is necessary for consensus in the sleepy model [32]. Δ represents the bound on communication delay. For simplicity, we assume the existence of a perfectly synchronous clock, meaning nodes share access to a common global clock. We can extend our results to accommodate a model with bounded clock skew by applying the round transformation technique in [28] (with a minor increase in latency). We assume the communication channel is *unauthenticated*, implying that the origin of any message is unknown to nodes. Let κ denote the security parameter. We assume an adaptive adversary that can corrupt nodes anytime during an execution. Corrupt nodes exhibit arbitrary behavior under the control of an adversary. Any non-corrupt node is said to be *honest* and behave as instructed by the protocol.

The sleepy model. Our protocol operates in an extended sleepy model that accommodates the dynamic participation of corrupt nodes. Let us begin by briefly reviewing the original sleepy model introduced by Pass-Shi [32]. In this model, nodes exist in one of two states: *awake* or *asleep*. Awake nodes actively engage in the execution, while asleep nodes neither execute any code nor send/receive any message. The count of awake nodes at any given time t is represented as $0 < n_t \leq N$. At each time point, the status of each node can change at the adversary's control without any prior notice. Regarding the message delivery, the assumption is that if an honest node p is awake at time t, then p must have received all messages sent to it by other honest nodes prior to time $t - \Delta$. However, as pointed out in [28], this message delivery assumption is not realistic. It essentially assumes all past messages are magically buffered until the recipient comes back awake. We will eliminate this assumption in Section 6.3 where we introduce our recovery mechanism.

Dynamic participation of corrupt nodes. Now let us delve into the dynamic participation of corrupt nodes and clarify the difference between the original sleepy model and our extended version. The original sleepy model, while allowing arbitrary churn among honest nodes, imposes a strong restriction on the dynamic participation of corrupt nodes. Precisely, the count of active corrupt nodes is capped at $n/2$ where n is the minimum count of active nodes throughout the entire execution, essentially disallowing any fluctuation in the corrupt node's participation. This stems from the costless simulation problem, wherein corrupt nodes can fabricate past messages during their inactive period.

We address part of this issue and manage to allow corrupt nodes' participation to grow proportionally to the current overall participation level. Formally, we measure corrupt nodes' participation in the following way. Let $\mathcal{F}_t$ be the set of corrupt nodes awake at time t, and define

$$f(t, T_f, T_b) = \Big| \bigcup_{t - T_f \leq \tau \leq t + T_b} \mathcal{F}_\tau \Big|.$$

We say an execution is admissible in the (T_f, T_b, α)-sleepy model if for all $t \geq 0$

$$f(t, T_f, T_b) < \alpha n_t.$$

In other words, a corrupt node is counted as an active corrupt node for an extra T_f time forward and an extra T_b time backward beyond the time interval it is actually active. This essentially acknowledges that the protocol cannot effectively defeat costless simulation within that duration other than considering the corrupt node active in that duration. On the other hand, any simulation outside of this time frame must be tolerated by the protocol.

For example, the original sleepy model can be described as the $(\infty, \infty, 1/2)$-sleepy model, and protocols in this model essentially are not tolerant to any backward or forward simulations. Bitcoin works in the $(0, 0, 1/2)$-sleepy model because any simulation is costly due to the non-reusable property of proof-of-works.

Our protocol is designed to operate in the $(\infty, T_b, 1/2)$-sleepy model with $T_b = O(\Delta)$. In other words, we are still unable to tolerate forward simulation because a corrupt node can simply give its secret key to the adversary before going to sleep. However, we prevent backward simulation for the most part. This allows the number of active corrupt nodes to grow proportionally to the overall participation, albeit with a slight delay of $T_b = O(\Delta)$. Further insight into these parameters will be provided in Section 3.3.

Atomic broadcast. An atomic broadcast protocol [11] allows nodes to agree on a linearizable log. Specifically, nodes input a finite set of values and decide on a growing sequence of values $[x_0, x_1, x_2, ...]$ called a *log*. The protocol provides the following guarantees:

(1) *Safety.* If two honest nodes decide logs $[x_0, .., x_j]$ and $[x'_0, .., x'_{j'}]$, then $x_i = x'_i$ for all $i \leq \min\{j, j'\}$.
(2) *Liveness.* If an awake honest node inputs a value x at time t, then there is a time $t' \geq t$ s.t. all awake honest nodes at any time after t' decide a log containing x.

Here, we do not specify what the values are. It might be from a finite class depending on the application built on top of the atomic broadcast.

Latency of atomic broadcast. We define latency as the time needed for a value input by an honest node to get decided. Namely, suppose an honest node inputs a value x at time t, and an honest node decides a log that includes the value x for the first time at time t'. In this context, the latency for deciding the value x is $t - t'$. This paper primarily focuses on the *best-case* latency, representing the shortest possible latency, typically when all nodes behave honestly.

Cryptographic assumptions. We make use of digital signatures with a public-key infrastructure (PKI). We use $\langle \mu \rangle_p$ to denote a message μ signed by node p. We assume a cryptographic hash function denoted $\mathsf{H}(\cdot)$. We also assume a verifiable random function (VRF). A node p with its secret key can evaluate $(\rho, \pi) \leftarrow \mathsf{VRF}_p(\mu)$ on any input μ. The output is a deterministic pseudorandom value ρ along with a proof π. Using π and the public key of node p, anyone can verify whether ρ is a correct evaluation of VRF_p on input μ.

2.1 Definitions and Primitives

We define some primitives and notions we will use in our protocol.

Blocks. As commonly done in recent BFT protocols, we employ the concept of *block*. In our protocol, a batch of values are grouped into a block. Each block contains a hash reference pointing to another block, forming a hash chain. The last block in the chain (i.e., without a hash reference) is called *genesis block* and is denoted as $B_0 = (\perp, \perp, 0)$. The *height* of a block represents its position in the chain, measured as the distance from the genesis block. The block of height k is formatted as

$$B_k := (b_k, \mathsf{H}(B_{k-1}), v)$$

where b_k is the batch of values in this block and $\mathsf{H}(B_{k-1})$ is the hash reference to the preceding block B_{k-1}. Any block B_k uniquely defines a chain $B_0 \dots B_k$, and hence a unique log. We say a block B_k *extends* B_l $(k \geq l)$ if $B_k = B_l$ or B_l is the ancestor of B_k in the chain (i.e., there is a path from B_k to B_l). We say two blocks B_k and B_l *conflict* with each other if neither of them extends the other. The last element v is an integer called *view* number. Intuitively, the view number identifies when the block was created (we will elaborate more later). We say the block B_k is of view v and use the notation $\mathsf{view}(B)$ to denote the view of block B. We say the block B_k is *valid* if the preceding block B_{k-1} is valid and is of view $v' < v$. In other words, the view numbers in any valid chain must be strictly increasing.

Graded agreement (GA). We use a primitive called graded agreement (GA), which is also used in Momose-Ren [28] and is similar to *gradecast* [23]. Each node takes as input a block B and outputs a set of blocks along with *grades*. More specifically, at the end of the protocol, each node outputs a set of pairs (B, g) of a block B and a grade bit $g \in \{0, 1\}$, subject to the following constraints:

- *Graded delivery.* If an honest node outputs $(B, 1)$, then all honest nodes output $(B, *)$.
- *Integrity.* If an honest node outputs $(B, *)$, then at least an honest node has input B' extending B.
- *Validity.* Let B be the highest block that every honest node's input extends. Then, all honest nodes output $(B, 1)$.

Note that the standard GA (also adopted in [28]) is defined for values, but we extend it to chained blocks. We also note that we do not have any *consistency* guarantee for outputs. In other words, nodes (even a single node) can output multiple conflicting blocks.

Graded proposal election (GPE). We introduce a primitive called *graded proposal election*, which resembles the composition of a leader/proposal election and a graded agreement. In GPE, nodes propose their own blocks B and elect a single block with grades. At the end of the protocol, each node outputs a *single* pair (B, g) of a block B (or $B = \perp$) and a grade bit $g \in \{0, 1\}$ with the following constraints:

- *Consistency.* If two honest nodes output $(B, *)$ and $(B', *)$ for $B, B' \neq \perp$, then $B = B'$.
- *Graded delivery.* If an honest node outputs $(B, 1)$, then all honest nodes output $(B, *)$.
- *Validity.* With a probability of more than $1/2$, all honest nodes output $(B, 1)$ where B is inputted by an honest node.
- *Integrity.* If an honest node outputs $(B, *)$, then the block B is *permissible* for at least an honest node.

Here, the criterion for a block to be considered *permissible* for a node is defined externally. It is important to note that there is a case that a block is permissible for one node but not for others.

Intuitively, with a probability of more than $1/2$, all honest and awake nodes will output the same honest node's input with grade 1. For the remaining less than $1/2$ probability, GPE still guarantees consistency in the sense at most one proposal is output, albeit not by all honest and awake nodes since some of them may output $\bot$. Furthermore, the block must pass an external safety check (be permissible) by at least one honest node, which helps eliminate unsafe proposals from corrupt nodes.

As mentioned, GPE resembles and can be implemented with, a composition of a leader election and a GA. However, we will directly implement a GPE that is more efficient.

3 OVERVIEW

In this section, we present an overview of this work to elaborate on the technical details.

3.1 View-based BFT with Early Decision

At a high level, we follow the classic view-by-view construction that is employed by most mainstream BFT protocols [1, 2, 8, 9, 18, 23, 36] as well as the latest sleepy consensus of Momose-Ren [28]. This paradigm is useful in achieving expected constant round latency. Specifically, the protocol progresses through a series of *views*, each possessing a fixed duration wherein one block is decided. View-based protocols in general (including non-sleepy protocols) involve (often implicitly) sequential invocations of a graded agreement (or a primitive with similar guarantees) and decide a block when all of the GAs from the initial to the final succeed. However, this approach brings a notable latency overhead, especially in the sleepy model, as each GA takes a few more rounds. For example, Momose-Ren involves five consecutive GAs, resulting in a latency of 16Δ at the minimum. To resolve this bottleneck, we introduce a new construction of each view. The high-level idea is that we can push back most of the tasks done by the sequential GAs to make a decision earlier. Concretely, we observe that we can instantiate a view from a composition of a GPE and two sequential GAs as outlined in Figure 1. The GPE performs the minimum task to make a safe decision within the view, and the latter two GAs resolve all other works to maintain safety and liveness across all views. This way, our protocol can decide on a block immediately after the GPE in the best case, taking 4Δ. We elaborate more on how our protocol maintains safety and liveness below.

Each view starts with a graded proposal election (GPE), and a grade-1 output from GPE is decided. Again, the crucial role of the GPE is to converge on a unique proposal. The consistency of GPE ensures that two distinct blocks cannot be decided simultaneously in the same view, thereby guaranteeing safety within a view. To maintain safety across views, we want to make all nodes *lock* on the decided block and discard any block conflicting with the lock in the subsequent views. To this end, the subsequent GAs resolve which block has possibly been decided by other nodes. Specifically, grade-0 output from GPE is handed over to the GAs, and grade-1 output from the second GA is locked. The graded delivery of GPE says, that if one node decides on a grade-1 output from GPE, then all other nodes at least output the same block with grade-0 from GPE. Thus, they input the block to the GAs. The validity of GA makes sure all nodes output this block with grade 1 and thus lock on this decided block. Lastly, any blocks conflicting with the locked block are deemed impermissible during the GPE and are discarded.

Each node in view v runs the following steps if it is awake. Let GA'_v and GA_v be the two graded agreements for view v.

GPE. Input to GPE a block B extending *candidate*: the highest grade-0 output from GA_{v-1}. A block is considered *permissible* within GPE if it extends *lock*: the highest grade-1 output from GA_{v-1}.

Decide. If GPE outputs $(B, 1)$, decide on B.

GA1. Each node inputs to GA'_v the output B from GPE (with any grade) if $B \neq \bot$, otherwise input *lock*.

GA2. Each node inputs to GA_v the highest block B s.t. GA'_v has outputted B with grade $g = 1$ and has not output (with either grade) any block conflicting with B.

lock, candidate are initialized to the genesis block B_0.

Figure 1: Summary of each view of our atomic broadcast protocol (simplified).

Now we also need to ensure liveness when some nodes lock on a block. It is important that other nodes extend this locked block in their proposals in later views; otherwise, honest nodes might discard an honest node's proposal. To this end, a grade-0 output from the second GA is set to *candidate*, and each node in the next view proposes a block extending the candidate. The graded delivery of GA makes sure that when some nodes lock on a block (by outputting from GA with grade 1), all other nodes at least output the same block with grade 0 from GA, so they will always set the locked block (or its descendant) as their candidates.

So far, we have only mentioned the role of the second GA. In fact, the second GA plays the primary role in maintaining safety and liveness across views. However, one missing aspect in the above is that a single GA can output conflicting blocks (recall that GA does not guarantee consistency). Therefore, a single GA does not guarantee that nodes lock on a unique block. The goal of the first GA is to prevent conflicting outputs from the second GA. We will provide further details in Section 6.

Comparison with PBFT. We can get more intuition by drawing some analogy to classic view-based BFT designs. The *decide-lock* relation and the *lock-candidate* relation that we employ are in fact two pillars of classic view-based BFT protocols [9, 36]. In more detail, if a block is decided, then all (or supermajority) other nodes must lock on the block to safeguard it from conflicting decisions in later views. For liveness, if a block is locked, all (or supermajority) other nodes must recognize the block as the candidate of their future proposals. We observe that our construction is somewhat similar to the classic PBFT-style construction with a main path for decision followed by a view-change sub-protocol to resolve conflict across views. The GPE and GA can be viewed as the main path and the view-change, respectively.

3.2 Graded Agreement without Stable Participation Requirement

Another key technical contribution is a new construction of graded agreement (GA) summarized in Figure 2. Our GA protocol builds on the GA protocol introduced by Momose-Ren but eliminates their reliance on the *eventual stable participation* assumption. For ease of exposition, let us consider a GA on binary values, i.e., $B \in \{0, 1\}$, instead of blocks.

Time-shifted quorum [28]. Our starting point is the *time-shifted quorum* idea introduced by Momose-Ren. Let us briefly review the original time-shifted quorum construction as a warm-up. First note that in the classic static participation model, achieving the graded delivery guarantee is trivial: forwarding a predefined quorum of votes is sufficient. When a node receives a quorum of votes (to obtain a grade-1 output), the node forwards these votes to all other nodes. All other nodes receive the quorum of votes one round later and output with grade $g = 0$. In the sleepy model, however, the quorum threshold (e.g., "majority") is not predefined but rather depends on the "perceived" participation level of each node. The above quorum forwarding approach obviously breaks down because a quorum of votes is no longer *transferable.* In other words, a set of votes may be accepted as a quorum by one node but may not meet the quorum threshold for another node. To address this challenge, Momose-Ren introduced the following time-shifted quorum technique.

Nodes send their inputs with "echo" messages at time $t = 0$.

- Let $E_1(B)$ and $E_2(B)$ denote the counts of "echo" messages for each $B \in \{0, 1\}$ received by time $t = \Delta$ and $t = 2\Delta$, respectively.
- Let E_2^* and E_3^* denote the count of "echo" messages (i.e., perceived participation level) received by time $t = 2\Delta$ and $t = 3\Delta$, respectively.

These counts are maintained locally by each node (if awake at the specified times), and nodes forward all received "echo" messages to all other nodes.

If a node p observes $E_1(B) > E_3^*/2$, it outputs B with grade 1. Since all "echo" messages are forwarded, any node q at time $t = 2\Delta$ receives at least the same number of "echo" messages for B as p. Similarly, node q at $t = 2\Delta$ cannot observe a higher participation level than what p observes at time $t = 3\Delta$. Thus, q must satisfy $E_2(B) > E_2^*/2$, leading it to send a "vote" message for B at time $t = 2\Delta$. This process causes all honest nodes awake at time $t \geq 3\Delta$ to recognize a majority "vote" for B, leading them to carry B as grade-0 output.

Removing the stability requirement. An observant reader may have noticed that the protocol described above imposes a constraint on nodes' churn. A node relies on the values of $E_1(B)$ and E_3^* counted at distinct points in time to output with grade-1. Therefore, the node must be active at both of these time points to make progress. This is why Momose-Ren assumes the participation level becomes *eventually stable* to ensure liveness.

Our protocol sidesteps this assumption through the following novel technique. Instead of directly utilizing the value of $E_1(B)$ counted at time $t = \Delta$, a node awake at $t = 3\Delta$ obtains an estimation from the values reported by those who were awake at time $t = \Delta$.

Input. Each node awake at time $t = 0$ multicasts the input value B through a message $\langle$"echo", $B\rangle$.

Report tally. Each node awake at time $t = \Delta$ multicasts the following value for each $B \in \{0, 1\}$.

- $E_1(B)$ is the # of $\langle$"echo", $B\rangle$

Vote. Each node awake at time $t = 2\Delta$ computes the following values.

- E_2^* is the # of $\langle$"echo", $*\rangle$
- $E_2(B)$ is the # of $\langle$"echo", $B\rangle$ for each $B \in \{0, 1\}$

If $E_2(B) > E_2^*/2$, then the node multicasts $\langle$"vote", $B\rangle$.

Output. Each node awake at time $t \geq 3\Delta$ compute the following values.

- E_3^* be the # of $\langle$"echo", $*\rangle$
- $\overline{E}_1(B)$ is the median of all $E_1(B)$ received for each $B \in \{0, 1\}$

If $\overline{E}_1(B) > E_3^*/2$, then output $(B, 1)$. Similarly, compute the following values.

- V_3^* is the # of $\langle$"vote", $*\rangle$
- $V_3(B)$ is the # of $\langle$"vote", $b\rangle$ for each $B \in \{0, 1\}$

If $V_3(B) > V_3^*/2$, then output $(B, 0)$.

Figure 2: Summary of our GA. For simplicity, we present an agreement on a binary value $B \in \{0, 1\}$.

To ensure a robust estimation, we take the median of the reported values. Since we have an honest majority at any time, the estimated value is both upper and lower bounded by values reported by honest nodes. Thus, the time-shifted quorum argument still holds without the eventual stable participation assumption.

3.3 Tolerating Backward Simulation with Stateless Algorithm

The challenge to tolerating *costless simulation* attacks lies in how to convince a newly awake node of the correct execution history. In the sleepy model, an honest node that just woke up has no idea what happened during its sleep. In particular, it cannot distinguish messages that were truly sent/received earlier in the execution from messages that corrupt nodes fabricate and claim to have been sent/received at those moments. To give a more concrete example, consider a proof-of-stake longest-chain protocol. Suppose a newly awake node receives two chains. One was built over the last ten years using the voting powers of honest nodes active at each point in time. Another is recently put together by corrupt nodes who only became active a few hours ago but claimed to have been building this chain over the entire decade. The newly awake node cannot tell the honestly generated chain from the simulated corrupt chain. The Momose-Ren protocol faces a similar challenge in spirit (despite not being based on longest chains). At each time, their protocol determines the next move based on the history of graded agreements. Recall that graded agreement decides the output once the number of votes reaches the threshold. Because corrupt nodes can fabricate votes in the past, they can inflate the threshold and convince a newly awake node of a fake output. This is also why all previous protocols have to assume $T_b = \infty$, or equivalently, disallow the growing participation of corrupt nodes. Without this assumption, a newly active corrupt node at time t can pretend to have been active all the way back when it was not counted in the corruption budget and help undermine the protocol's safety.

Making the protocol stateless. To tackle this issue, our approach is to make the protocol *stateless*. Note that each view v of our protocol (summarized in Figure 1) updates crucial variables (namely, candidate and lock) based on the result of GA from the immediate last view, occurring at most $T_b = O(\Delta)$ time earlier. Consequently, our protocol can ignore any message from the ancient past, including those fabricated by corrupt nodes. The key to a stateless protocol is to ensure that each node always inputs a non-empty value to GA even when the GPE results in failure. Specifically, each node provides its lock as input when GPE produces an output of $\perp$. This guarantees that GA in each view always yields an output (which could potentially match the result of the previous GA or even the genesis block). As a result, we can safely discard the outputs of all past GAs except the most recent one.

The stateless nature of our protocol also brings benefits to the efficiency of our recovery mechanism. When a new active node joins, it only needs to retrieve messages from the most recent view (along with any missing blocks).

Impossibility of fluctuating corruption. We note that our protocol does not tolerate fluctuating corruption. To elaborate, while our protocol allows active corrupt nodes to increase over time, it does not allow the set of corrupt participants to shrink, as implied by $T_f = \infty$. This limitation stems from the simple fact that corrupt nodes can hand off their secret keys to the adversary (or other corrupt nodes) before going inactive. Then, the adversary can use their keys to sign any future messages on their behalf as if those corrupt nodes never went inactive. In other words, a corrupt node can simulate indefinitely *forward*. Unfortunately, this issue is inherent in a model without constraints on adversary computational power such as those imposed by proof-of-work (Section 7).

4 GRADED AGREEMENT

This section presents a graded agreement (GA) protocol with 3Δ latency. Our protocol builds on the time-shifted quorum technique of Momose-Ren [28] but eliminates the *eventual stable participation* assumption with the "median trick" explained in Section 3. Our protocol is described in Algorithm 1.

The protocol runs up to time $t = T_b$ (c.f., Section 2) since the beginning of the execution. The specific value of T_b will be given when we present our atomic broadcast protocol in Section 6. We also note that GA defined in this paper can output multiple pairs of (B, g). Therefore, we denote outputs as the set of outputs. Now we proceed to provide a detailed explanation of our protocol below, mainly focusing on how to extend the binary-valued GA in Section 3 to support chained blocks.

Tally echo and report. At time $t = 0$, awake nodes multicast their input blocks through "echo" messages. At time $t = \Delta$, each awake node tallies "echo" messages received for each block and reports these tallies. Notably, even nodes that input non-conflicting blocks might input different blocks within the same chain. This implies we have to count "echo" for a block B as an implicit "echo" for all ancestors of B. Namely, for each block B, a node counts the number of "echo" messages received from distinct nodes for some block B' that extends B. This count is denoted as $E(B)$ and is reported via a "tally" message. Moreover, each node also forwards all "echo" messages counted in $E(B)$. To avoid sending an unbounded number of tallies (especially in cases where a corrupt node sends arbitrarily many "echo"), a node sends the "tally" message only if $E(B) > E^*/2$ where E^* is the total number of distinct nodes who send "echo" messages (for any block). This ensures at least one honest node must have sent "echo" for B (or its descendant) when an honest node reports a tally for B. If there is no tally to report, a node sends a "tally" for $\perp$ just to announce itself to other nodes.

Vote. At time $t = 2\Delta$, each awake node tallies "echo" messages in the same manner as above and sends a "vote" message for a block B that has a majority of "echo", namely satisfying $E(B) > E^*/2$. If there is no such block, then send "vote" for $\perp$. Additionally, if it has received an "echo" message (for any block) from any node q, it forwards the "echo" message if it has not done so already (line 22). This ensures that all awake nodes after time $t = 3\Delta$ will possess higher (or at least the same) quorum thresholds (i.e., E^*), a critical aspect for the time-shifted quorum argument.

Output. At time $t = 3\Delta$ or later up to $t = T_b$, awake nodes decide outputs based on "tally" and "vote" messages. In order to compute potential grade-1 outputs, each node obtains a robust estimation of $E(B)$ tallied at time $t = 2\Delta$ from the "tally" messages. Specifically, for each block B, a node calculates the set $\mathcal{E}$ of reported tallies for B as follows (line 25-30):

(1) If the node has received from a node q a "tally" for a block B' extending B, then the reported tally $E(B')$ in the message is adopted for node q.
(2) Otherwise, for example, if the node has received from a node q a "tally" for a block conflicting with B, then node q is considered reporting $E(B) = 0$.

Next, the median $\overline{E}(B)$ from the set $\mathcal{E}$ is selected as the estimation. If the estimated tally meets the threshold, i.e., $\overline{E}(B) > E^*/2$, then the node outputs B with grade $g = 1$. Finally, the node computes a grade-0 output based on "vote" messages. If the count of "vote" messages for a block B or its descendants (referred to as $V(B)$) exceeds the majority of voters (denoted V^*), then block B is taken as an output with grade $g = 0$.

Time-shifted quorum. Let us quickly go over the time-shifted quorum argument. Suppose an honest node p has the estimated tally $e_p = \overline{E}(B)$. The estimated tally e_p is upper bounded by an honest node's tally e_r at time $t = \Delta$. This is because e_p is the median of all reported tally and there is always an honest majority. Now, suppose an honest node q awake at time $t = 2\Delta$ has tally $e_q = E(B)$. Since the node r has forwarded all "echo" counted to e_r, we have $e_r \leq e_q$. As we also have $e_p \leq e_r$, this leads to $e_p \leq e_q$. Similarly, given that node q forwards all "echo" messages, the quorum threshold $m_p/2 = E^*/2$ for node p is higher than (or at least the same as) the threshold $m_q/2$ observed by node q at time $t = 2\Delta$. Consequently, if $e_p > m_p/2$ (indicating p would consider B as a grade-1 output), we have $e_q > m_q/2$. Thus node q sends "vote" for B, resulting in

Algorithm 1 Graded Agreement – GA_{id}

```
Initialize outputs = $\emptyset$. Node $p$ executes the following algorithm at every time $0 \le t \le T_b$ after starting the protocol. Below, we assume every message binds to the protocol's id denoted id.
1:  if $t = 0$ then
2:      multicast $\langle$"echo", $B\rangle_p$ for the input block $B$.
3:  if $t = \Delta$ then
4:      $E^* \leftarrow$ # of nodes $q$ s.t. $p$ has received $\langle$"echo", $*\rangle_q$
5:      for all block $B$ do // examine blocks from a higher height
6:          $E(B) \leftarrow$ # of nodes $q$ s.t. $p$ has received $\langle$"echo", $B'\rangle_q$ for a block $B'$ extending $B$
7:          if $E(B) > E^*/2$ and $p$ has not sent $\langle$"tally", $B', e\rangle_p$ for $e \ge E(B)$ and a block $B'$ extending $B$ then
8:              multicast $\langle$"tally", $B, E(B)\rangle_p$
9:              forward all "echo" counted in $E(B)$
10:     if $p$ has not sent "tally" then
11:         multicast $\langle$"tally", $\perp, \perp\rangle_*$
12: if $t = 2\Delta$ then
13:     for all block $B$ do // examine blocks from a higher height.
14:         $E(B) \leftarrow$ # of nodes $q$ s.t. $p$ has received $\langle$"echo", $B'\rangle_q$ for a block $B'$ extending $B$
15:         $E^* \leftarrow$ # of nodes $q$ s.t. $p$ has received $\langle$"echo", $*\rangle_q$
16:         if $E(B) > E^*/2$ then
17:             if $p$ has not sent "vote" for a block $B'$ extending $B$ then
18:                 multicast $\langle$"vote", $B\rangle_p$
19:     if $p$ has not sent "vote" then
20:         multicast $\langle$"vote", $\perp\rangle_p$
21:     forward all $\langle$"echo", $*\rangle_q$ // only once per $q$
22: if $3\Delta \le t \le T_b$ then
23:     for all block $B$ do // examine blocks from a higher height.
24:         $\mathcal{E} \leftarrow \emptyset$
25:         for all node $q$ s.t. $p$ has received $\langle$"tally", $B', e\rangle_q$ do
26:             if $B'$ extends $B$ then
27:                 add $e$ to $\mathcal{E}$
28:             else
29:                 add 0 to $\mathcal{E}$
30:         $\overline{E}(B) \leftarrow$ median in $\mathcal{E}$
31:         $E^* \leftarrow$ # of nodes $q$ s.t. $p$ has received $\langle$"echo", $*\rangle_q$
32:         if $\overline{E}(B) > E^*/2$ then
33:             add $(B, 1)$ to outputs
34:     for all block $B$ do
35:         $V^* \leftarrow$ # of nodes $q$ s.t. $p$ has received $\langle$"vote", $*\rangle_q$
36:         $V(B) \leftarrow$ # of nodes $q$ s.t. $p$ has received $\langle$"vote", $B'\rangle_q$ for a block $B'$ extending $B$
37:         if $V(B) > V^*/2$ then
38:             add $(B, 0)$ to outputs
39: // line 25: If multiple "tally" exist, choose one with $B'$ extending $B$ with the largest $e$; if no such $B'$ exists, pick one arbitrary.
```

a majority vote for B, making all other nodes at least have B as grade-0 output, thus achieving graded delivery.

4.1 Correctness Proof

We prove the correctness of our GA protocol. Below, we use the notion of each node's *tally*. We define the "tally e of node q for a block B" as follows: 1) $e = e'$ if node q sent $\langle$"tally", $B', e'\rangle_q$ for any block B' extending B (if multiple such e' exists, then pick the largest one), and 2) $e = 0$ otherwise. In other words, each node's tally is the value counted to $\mathcal{E}$ (line 25–30).

Lemma 4.1. *For any block B, let e_p be the value of $\overline{E}(B)$ observed by an honest node p at time $t \ge 3\Delta$, and e_q be the value of $E(B)$ observed by an honest node q at time $t = 2\Delta$. Then, $e_p \le e_q$.*

Proof. Since the total number of corrupt nodes ever awake by time $t = T_b$ is less than half of the nodes awake at time $t = \Delta$, the median e_p in the set $\mathcal{E}$ of all tallies for B must be upper bounded by at least one honest node's tally e for B. Consider the case where $e > 0$ (the lemma is obvious if $e = 0$). That node must have forwarded all $\langle$"echo", $B\rangle_*$ counted in e, which were received by node q by time $t = 2\Delta$. Hence, we have $e \le e_q$, leading to $e_p \le e_q$. □

Lemma 4.2 (Graded consistency). *If an honest node outputs $(B, 1)$, then for all t where $3\Delta \le t \le T_b$, all honest nodes awake at time t output $(B, *)$.*

Proof. Suppose an honest node p outputs $(B, 1)$. Let e_p and m_p be the values of $\overline{E}(B)$ and E^*, respectively, observed by node p. We have that $e_p > m_p/2$. Let e_q and m_q be the values of $E(B)$ and E^*, respectively, observed by any honest node q at time $t = 2\Delta$. By Lemma 4.1, we have $e_p \le e_q$. Since the node q forwards all "echo" messages counted in m_q, we also have $m_q \le m_p$. Thus, we have $e_q > m_q/2$. So all honest nodes awake at time $t = 2\Delta$ must have sent $\langle$"vote", $B\rangle_*$. Therefore, for all t from 3Δ to T_b, all honest nodes awake at time t observe $V(B) > V^*/2$ and output $(B, 0)$. □

Lemma 4.3 (Integrity). *If an honest node outputs $(B, *)$, then at least an honest node has input B' extending B.*

Proof. Suppose all honest nodes awake at time $t = 0$ input blocks that do not extend B. Let e and m represent the values of $E(B)$ and E^*, respectively, observed by an honest node awake at time $t = 2\Delta$. The "echo" messages counted toward the value e are only from corrupt nodes, while "echo" messages from honest nodes awake at time $t = 0$ are counted to m. Given that the number of all corrupt nodes ever awake by time $t \le T_b$ is less than half of all honest nodes awake at time $t = 0$, we have $e < m/2$. As a result, none of the honest nodes awake at time $t = 2\Delta$ would send $\langle$"vote", $B'\rangle$ for any block B' extending B. Therefore, any honest node awake at any time $3\Delta \le t \le T_b$ observes $V(B) < V^*/2$, and thus would not output $(B, 0)$. By graded consistency (Lemma 4.2), nor would they output $(B, 1)$. □

Lemma 4.4 (Validity). *Let B be the highest block that every honest node's input extends. Then, for any $3\Delta \le t \le T_b$, all honest nodes awake at time t output $(B, 1)$.*

Proof. Let p be any honest node awake at time $3\Delta \le t \le T_b$, and let e_p and m_p be the values of $\overline{E}(B)$ and E^* observed by node p at time t. We observe that there exists an honest node q awake at time $t = \Delta$ and the tally e of q for B satisfies $e \le e_p$. This is because the number of all corrupt nodes ever awake by time $t = T_b$ is less than half of nodes awake at time $t = \Delta$ (i.e., an honest majority), and the median e_p in the set $\mathcal{E}$ of all tallies for B must be lower bounded by at least one honest node's tallies e for B. Now, if every honest node's (awake at time $t = 0$) input extends B, then all "echo" messages sent by these honest nodes are counted to e. Again since we have an honest majority, we have $e > m_p/2$, hence $e_p > m_p/2$. Therefore, node p outputs $(B, 1)$. □

5 GRADED PROPOSAL ELECTION

This section presents a graded proposal election (GPE) protocol with 4Δ latency. Intuitively, the GPE protocol is a combination of a VRF-based leader election with the GA protocol presented in Section 4. Our protocol is presented in Algorithm 2.

Input. The protocol starts with a VRF-based leader election. In this process, each node p sends its own input block along with the VRF evaluation on the protocol's ID in an "input" message. For ease of presentation, we define the *winning input* to be the input message with the highest VRF value. Given that each node may receive a distinct set of messages, the winning input is defined individually for each node. Specifically, a message $\langle$"input"$, B, \rho, \pi\rangle_q$ is considered a winning input by a node p if both of the following conditions are satisfied:

(1) p has not received any $\langle$"input"$, *, \rho', \pi'\rangle_r$ with $\rho' > \rho$.
(2) p has not received any $\langle$"input"$, B', \rho, \pi\rangle_q$ for $B' \neq B$.

In simpler terms, if the VRF of node q's input is the highest among all received inputs and p has not received any equivocating input from node q, then node p considers node q's input as the winning input. We also refer to the corresponding block B as the *winning block*. Since we have an honest majority, there is a probability of at least $1/2$ that a VRF from an honest node will be the highest, resulting in its input being the winner.

GA on the winning input. The rest of the algorithm (from time $t = \Delta$ to $t = 4\Delta$) can be viewed as achieving graded agreement on a winning input. At time $t = \Delta$, awake nodes send "echo" messages for the winning blocks they have received. At time $t = 2\Delta$, nodes tally "echo" for the winning block and report their tallies. At time $t = 3\Delta$, if the count of "echo" for a winning block meets the majority, then the block is voted. Finally, at time $t = 4\Delta$, each node calculates the median of the reported tallies, and if it meets the majority, the node outputs the block with grade $g = 1$. If there are majority votes for a block, it becomes a grade-0 output.

The key distinction from the GA in Section 4 is that each node performs every action exclusively on the winning input/block. This helps achieve *consistency* of the GPE, a property that is not mandated by GA. More concretely, each node votes only for the winning block, and all blocks voted by honest nodes are echoed at least Δ time before. So it is impossible for two different blocks to get majority votes. Additionally, each node forwards the winning input to all other nodes. Similarly, if no winning input is present, indicating the highest VRF holder is equivocating, the equivocating inputs are propagated to all nodes. This ensures that if an honest node possesses a grade-1 output (at time $t = 4\Delta$), all honest nodes awake at time $t = 3\Delta$ have unanimously identified the same input as the winner, i.e., there exists no input with a higher VRF and no equivocating input. This makes sure the time-shifted quorum argument holds.

Another crucial distinction from GA is that each node sends "echo" only for a *permissible* block (line 8). Due to the honest majority, a block must be echoed by at least one honest node to become the GPE output. This guarantees the integrity of GPE. We reiterate that the condition for a block to be permissible is externally defined, which will be specified in Section 6.

Remark on the validity. We note that, during the tallying of "echo" or "vote" for a block, a node only considers the messages regarding the specific block and excludes blocks extending that block. This distinction arises from the validity requirement. Our GA must produce grade-1 output for the highest common input (i.e., the block that all honest nodes' inputs extend). In contrast, the validity of GPE requires the algorithm to have grade-1 output for an honest node's input (with probability $1/2$). Therefore, honest nodes will not echo different blocks when the winning input is from an honest node, which is why we do not need to count indirect echoes/votes.

5.1 Correctness Proof

We prove the correctness of our GPE protocol. As in the proof for GA, we employ the notion of each node's *tally*. We define the "tally e of node q for a block B" as follows: 1) $e = e'$ if node q sent $\langle$"tally"$, B, e'\rangle_q$, and 2) $e = 0$ otherwise.

Lemma 5.1 (Graded delivery). *If an honest node outputs $(B, 1)$, then all honest nodes output $(B, *)$.*

Proof. Suppose an honest node p outputs $(B, 1)$. Let e_p and m_p denote the values of $\overline{E}(B)$ and E^*, respectively, observed by node p at time $t = 4\Delta$. We have that $e_p > m_p/2$. We observe that there exists an honest node r awake at time $t = 2\Delta$ and the tally e for B satisfies $e_p \leq e$. Let q be any honest node awake at time $t = 3\Delta$, and e_q and m_q be the values of $\overline{E}(B)$ and E^*, respectively, observed by node q. Since node q forwards $\langle$"echo"$, *\rangle_s$ for every node s counted to m_q, we have $m_q \leq m_p$. We further observe that node r has B as the winning block; otherwise, it would have forwarded its winning input (or equivocating inputs), and node p would not have considered B as the winning block. So, r must have forwarded all "echo" for the winning block B, leading to $e \leq e_q$ and hence $e_p \leq e_q$. Given that $m_q \leq m_p$, it follows that $e_q > m_q/2$. Consequently, node q sends $\langle$"vote"$, B\rangle_q$. Therefore, any honest node awake at time $t = 4\Delta$ should observe $V(B) > V^*/2$ and output $(B, *)$. □

Lemma 5.2 (Consistency). *If two honest nodes output $(B, *)$ and $(B', *)$, respectively, then $B = B'$.*

Proof. Suppose for the sake of contradiction two honest nodes output $(B, *)$ and $(B', *)$, respectively, with $B \neq B'$. These two nodes must have independently observed local conditions $V(B) > V^*/2$ and $V(B') > V^*/2$, implying that both B and B' receive votes from honest nodes. Let p and q denote the nodes who voted for B and B', respectively. By definition, node p must have observed $E(B) > E^*/2$ at time $t = 3\Delta$, indicating that at least an honest node sent $\langle$"echo"$, B\rangle$. Similarly, node q, voting for B', must have witnessed $E(B') > E^*/2$, indicating an honest node must have sent $\langle$"echo"$, B'\rangle$. However, this scenario implies that both p and q have received the corresponding "input" messages for B and B', one of which is not the actual winning input. Given that nodes would not vote for non-winning blocks, this contradicts both B and B' are voted. □

Algorithm 2 Graded Proposal Election – $\mathsf{GPE}_{\mathsf{id}}$

```
Node $p$ executes the following algorithm at every time $0 \le t \le 4\Delta$ after
staring the protocol. Let id be the protocol's id.

if $t = 0$ then
    $\rho, \pi \leftarrow \mathsf{VRF}_p(\mathsf{id})$
    multicast $\langle$"input", $B, \rho, \pi\rangle_p$ for the input $B$.

if $t = \Delta$ then
    if there exists a winning input then
        Let $\langle$"input", $B, \rho, \pi\rangle_L$ be the winning input
        forward the winning input
        multicast $\langle$"echo", $B\rangle_p$ if $B$ is permissible
    else
        forward the equivocating inputs
        multicast $\langle$"echo", $\perp\rangle_p$

if $t = 2\Delta$ then
    if there exists a winning input then
        Let $\langle$"input", $B, \rho, \pi\rangle_L$ be the winning input
        forward the winning input (if not yet).
        $E(B) \leftarrow$ # of nodes $q$ s.t. $p$ has received $\langle$"echo", $B\rangle_q$
        forward all $\langle$"echo", $B\rangle_*$
        multicast $\langle$"tally", $B, E(B)\rangle_*$
    else
        forward the equivocating inputs (if not yet)
        multicast $\langle$"tally", $\perp, \perp\rangle_*$

if $t = 3\Delta$ then
    if there exists a winning input then
        Let $\langle$"input", $B, \rho, \pi\rangle_L$ be the winning input
        forward the winning input (if not yet)
        forward all $\langle$"echo", $*\rangle_q$ (once per node $q$)
        $E(B) \leftarrow$ # of nodes $q$ s.t. $p$ has received $\langle$"echo", $B\rangle_q$
        $E^* \leftarrow$ # of nodes $q$ s.t. $p$ has received $\langle$"echo", $*\rangle_q$
        if $E(B) > E^*/2$ then
            multicast $\langle$"vote", $B\rangle_p$
        else
            multicast $\langle$"vote", $\perp\rangle_p$
    else
        forward the equivocating inputs (if not yet)
        multicast $\langle$"vote", $\perp\rangle_p$

if $t = 4\Delta$ then
    if there exists a winning input then
        Let $\langle$"input", $B, \rho, \pi\rangle_L$ be the winning input
        $\mathcal{E} \leftarrow \emptyset$
        for all node $q$ s.t. $p$ has received $\langle$"tally", $B', e\rangle_q$ received do
            if $B' = B$ then
                add $e$ to $\mathcal{E}$
            else
                add 0 to $\mathcal{E}$
        $\overline{E}(B) \leftarrow$ the median in $\mathcal{E}$
        $E^* \leftarrow$ # of nodes $q$ s.t. $p$ has received $\langle$"echo", $*\rangle_q$
        if $\overline{E}(B) > E^*/2$ then
            output $(B, 1)$
    for all block $B$ do
        $V^* \leftarrow$ # of nodes $q$ s.t. $p$ has received $\langle$"vote", $*\rangle_q$
        $V(B) \leftarrow$ # of nodes $q$ s.t. $p$ has received $\langle$"vote", $B\rangle_q$
        if $V(B) > V^*/2$ and $p$ has not outputted yet then
            output $(B, 0)$
```

Here, we note that the validity we prove below assumes that every honest node inputs a permissible block; otherwise, the validity would be trivially false. It is worth noting that this assumption will be justified by our atomic broadcast (c.f., Section 6).

Lemma 5.3 (Validity). *With probability more than* $1/2$, *all honest nodes output* $(B, 1)$ *for a block* B *that honest node inputs.*

Proof. Let p be an honest node awake at time $t = 4\Delta$, and let e_p and m_p be the values of $\overline{E}(B)$ and E^*, respectively, observed by node p. We observe that there exists an honest node r awake at time $t = 2\Delta$ whose tally e_r for B satisfies $e_r \le e_p$. Given that the number of corrupt nodes ever awake by time $t = T_b$ is less than half of all honest nodes awake at time $t = 0$, with probability $\alpha > 1/2$, an honest node's VRF will be the highest, making its input B the winning block. Consequently, all honest nodes awake at time $t = \Delta$ will send $\langle$"echo", $B\rangle$. This leads to $e_r > m_p/2$. Given that $e_r \le e_p$, it follows that $e_p > m_p/2$. As a result, node p outputs $(B, 1)$. □

Lemma 5.4 (Integrity). *If an honest node outputs* $(B, *)$*, then the block* B *is* permissible *for at least an honest node.*

Proof. As we have observed, when an honest node outputs a block B, it indicates that at least one honest node has sent "echo" message for B. This means the block is deemed permissible by the honest node that sent the "echo". □

6 ATOMIC BROADCAST

This section presents an atomic broadcast protocol with 4Δ latency in the best case, building on the GA protocol (in Section 4) and the GPE protocol (in Section 5). Our protocol achieves safety and liveness in the $O(\infty, T_b, 1/2)$-sleepy model with $T_b = 11\Delta$.

The protocol is described in Algorithm 3. It progresses through repeated *views*. Each view is identified by an integer $v > 0$ and takes 10Δ time. As mentioned in Section 3, each view consists of a GPE and two GAs. Based on the output from GPE and GA, nodes lock on a potentially decided block to safeguard it from future conflicting decisions. Nodes also determine the next proposal does do not conflict with locked blocks to ensure liveness. These values are maintained by the variables lock and candidate, initially set to the genesis block B_0 (which is considered a block of view $v = 0$).

Each view begins with a GPE. At time $t = 0$, each node inputs to GPE_v a block B that extends its current candidate. candidate is updated to the highest block that GA_{v-1} has output with grade 0. Within the GPE, a block is considered *permissible* if it extends lock. Again, this makes sure any block conflicting with a potentially decided block is precluded from the GPE output.

The output from GPE_v is passed to two consecutive GAs (first GA'_v and then GA_v). Let us call them the pre-GA and the main GA, respectively. The pre-GA GA'_v is to preclude conflicting outputs from the main GA. Specifically, a node inputs to GA_v a grade-1 output from GA'_v only if there is no other conflicting output from GA'_v. This makes sure honest nodes' inputs to GA_v are always non-conflicting, avoiding divergent locks and candidates among nodes.

A grade-1 output from GPE_v is decided immediately. When a node decides on a block, the node multicasts a "decide" message for the block to let other nodes (especially those who were not awake at $t = 4\Delta$) decide on the block. If a node receives "decide" messages for a block B (or its descendants) from a majority of all senders of "decide" messages, i.e., $D(B) > D^*/2$, the node also decides on the block B (line 27–31, 4–8).

Tolerating backward simulation. Each view in our protocol examines only messages from the immediate preceding view. To be more specific, all steps that depend on previous messages are summarized below:

(1) At time $t = 0$, each node computes its input to GPE_v based on the result of GA_{v-1} (i.e., candidate), which starts at time $t = 7\Delta$ of view $v - 1$. These messages are sent at most 3Δ earlier.
(2) At time $t = 4\Delta$, each node decides a block or computes its input to GA_v based on the result of GA_{v-1} (i.e., lock), which are derived from messages sent at most 7Δ earlier.
(3) At any time up to time $t = 5\Delta$ of view v, nodes decide blocks based on the "decide" messages sent at time $t = 4\Delta$ of view $v-1$, which are at most 11Δ earlier.

To sum up, attempts by corrupt nodes to fabricate messages of more than $T_b = 11\Delta$ time before have no impact on the execution of honest nodes.

6.1 Safety and Liveness Proofs

We prove the safety and liveness of our atomic broadcast protocol. We say a node *directly* decides a block B if the node has not decided any descendant of B by that moment. We first show below that locks are always non-conflicting in the same view.

Lemma 6.1. *Let p and q be honest nodes awake at time $t = 4\Delta$ of a view v, and let lock_p and lock_q be the value of lock observed by honest nodes p and q, respectively. Then, lock_p and lock_q do not conflict with each other.*

Proof. In each view, an honest node inputs to GA_v (the main GA) a block received from GA'_v (the pre GA) with grade $g = 1$. Moreover, the block should not conflict with any other outputs from GA'_v. The graded delivery ensures that other honest nodes deliver the block with grade $g = 0$, so they would not input any conflicting block to GA_v. Due to the integrity of GA, GA_v will output non-conflicting blocks. Thus, lock_p and lock_q do not conflict with each other. □

Lemma 6.2. *If a block B of view v is directly decided by an honest node, then at least an honest node has sent ⟨"decide", B, v⟩.*

Proof. Assume for the sake of contradiction that none of the honest nodes awake at time $t = 4\Delta$ in view v sends ⟨"decide", B, v⟩. Then, the block B will not be decided (either directly or indirectly) until time $t = 5\Delta$ of view $v + 1$. Thus, none of the honest nodes awake at time $t = 4\Delta$ in view $v + 1$ will send ⟨"decide", $B, v + 1$⟩. By induction, in all subsequent views $v' \geq v$, honest nodes will never send ⟨"decide", B, v'⟩. This contradicts that the block B was directly decided. □

Algorithm 3 Atomic Broadcast

```
    Variables are initialized as lock, candidate = $B_0$.
    In each view $v$, node $p$ executes the following algorithm at every time
    $0 \leq t \leq 10\Delta$ during view $v$, and enter the next view $v + 1$.
    // update variables
 1: candidate ← the highest block $B$ s.t. $\mathsf{GA}_{v-1}$ outputs $(B, *)$
 2: lock ← the highest block $B$ s.t. $\mathsf{GA}_{v-1}$ outputs $(B, 1)$
 3: if $t \leq 5\Delta$ then
 4:     $D^* \leftarrow$ # of nodes $q$ s.t. $p$ has received ⟨"decide", $*, v-1$⟩$_q$.
 5:     for all block $B$ do
 6:         $D(B) \leftarrow$ # of nodes $q$ s.t. $p$ has received ⟨"decide", $B', v-1$⟩$_q$
            for any block $B'$ extending $B$.
 7:         if $D(B) > D^*/2$ then
 8:             decide $B$ and all its ancestors
    // GPE invocation
 9: if $t = 0$ then
10:     $B \leftarrow (b, \mathsf{H}(B'), v)$ where $B'$ = candidate.
11:     start $\mathsf{GPE}_v$ with input $B$; within GPE, any block is considered *permissible* if it extends lock and view$(B) = v$.
    // pre GA invocation
12: if $t = 4\Delta$ then
13:     $B, g \leftarrow$ the output from $\mathsf{GPE}_v$.
14:     if $g = 1$ then
15:         decide $B$ and all its ancestors
16:         multicast ⟨"decide", $B, v$⟩$_p$
17:     else
18:         multicast ⟨"decide", $B', v$⟩$_p$ for the highest decided block $B'$.
19:     if $B \neq \perp$ then
20:         start $\mathsf{GA}'_v$ with input $B$
21:     else
22:         start $\mathsf{GA}'_v$ with input lock
    // main GA invocation
23: if $t = 7\Delta$ then
24:     $B \leftarrow$ the highest block s.t. $\mathsf{GA}'_v$ has output $(B, 1)$ but has not output
        $(B', *)$ for any $B'$ conflicting with $B$
25:     start $\mathsf{GA}_v$ with input $B$
    // decide
26: if $t \geq 5\Delta$ then
27:     $D^* \leftarrow$ # of nodes $q$ s.t. $p$ has received ⟨"decide", $*, v$⟩$_q$.
28:     for all block $B$ do
29:         $D(B) \leftarrow$ # of nodes $q$ s.t. $p$ has received ⟨"decide", $B', v$⟩$_q$ for
            any block $B'$ extending $B$.
30:         if $D(B) > D^*/2$ then
31:             decide $B$ and all its ancestors
```

Next, we show that a directly decided block will always be handed over to the immediately following GA, thereby ensuring its subsequent locking.

Lemma 6.3. *If an honest node sends ⟨"decide", B, v⟩ for a block B of view v, then all honest nodes awake at time $t = 7\Delta$ of view v input the block B to GA_v.*

Proof. Suppose an honest node p sends ⟨"decide", B, v⟩. This implies that node p must have received a block B of view v from GPE_v with grade $g = 1$. Let q be any honest node awake at time

$t = 4\Delta$. The graded delivery of GPE ensures that node q has received B from GPE_v (with any grade). So node q will input B to GA'_v (the pre-GA). Due to the validity of GA, the pre-GA will output B (or a block extending B) to all honest nodes awake at time $t = 7\Delta$, leading them to input B to GA_v (the main GA). □

Lemma 6.4 (Safety). *If two honest nodes decide B and B', then B does not conflict with B'.*

Proof. Suppose for the sake of contradiction two conflicting blocks are decided by honest nodes. This implies there are two conflicting blocks B and B' of view v and v', respectively, decided directly by honest nodes. We have that $v \neq v'$ due to the consistency of GPE. Without loss of generality, we assume $v < v'$. Based on Lemma 6.2 and 6.3, all honest nodes awake at time $t = 4\Delta$ of view v input blocks extending B into GA_v, leading to all honest nodes awake during view $v + 1$ locking on B (i.e., set lock to B or its descendants). Consequently, any conflicting block will be precluded from GPE/GA outputs during view v. By induction, in all subsequent views, all honest nodes keep inputting blocks extending B to GPE/GA. However, by the same argument, in view v', honest nodes must input blocks extending B' into $\mathsf{GA}_{v'}$. This contradicts that B and B' are conflicting. □

The above lemma directly implies *safety* as non-conflicting blocks B and B' represent consistent logs, i.e., one of them is a prefix of the other.

Lemma 6.5 (Liveness). *If an awake honest node inputs a value x at time t, then there is a time $t' \geq t$ s.t. all honest nodes awake after t' decide a log containing x.*

Proof. We first observe that if an honest node (say p) inputs a block B to GPE_v, then B extends lock observed by other honest nodes, indicating that B is permissible for all honest nodes. This is due to the graded delivery of GA. An honest node q sets to lock_q a grade-1 output from GA_{v-1}. This implies node p has received lock_q from GA_{v-1}, at least as grade-0 output, leading to node p setting it to p's candidate. Node p inputs a block extending candidate, so B must extend lock_q.

Now, if the honest node's input becomes the winning block, then all honest nodes awake at time $t = 4\Delta$ of the view decide the block B and send ⟨"decide", B, v⟩. So all honest nodes awake at any time from time $t = 5\Delta$ of view v to time $t = 5\Delta$ of view $v + 1$ decide the block B. By induction, all honest nodes awake at any time after $t = 5\Delta$ of view v decide B.

The validity of GPE implies that such a view v eventually and repeatedly appears. All values input by honest nodes before this view will get included in block B (with input dissemination, i.e., honest nodes multicast their input values) and get decided. So all values input by honest nodes will eventually be decided. □

6.2 Analysis

We give the analysis of the latency and communication complexity of our protocol.

Latency. The best-case latency of our protocol is 4Δ as nodes can decide on a block immediately after GPE. This is far better than prior and concurrent works such as 16Δ of Momose-Ren [28] and 10Δ of Gafni-Losa [27]. The expected latency of our protocol is 14Δ, which is also better than prior and concurrent works: 32Δ of Momose-Ren and 20Δ of Gafni-Losa. Finally, the latency in the worst case (except with negligible probability) is $O(\kappa\Delta)$. This matches the lower bound [4].

Communication complexity. The expected communication complexity of our protocol described in Algorithm 3 is $O(Ln^3)$ per view, where L represents the block size. To elaborate, recall that honest nodes input to GA an output from GPE or blocks they are locking on. Since GPE guarantees consistency, there is no disagreement among honest nodes on the former. For the latter, honest nodes might have distinct locks. However, in the last few (constant in expectation) views, there is at least one "good" view where all honest nodes lock on the same block proposed by an honest node. Therefore, honest nodes collectively input at most constant number of blocks to GA. This implies each honest node sends $O(1)$ "vote/tally" messages and forwards $O(n)$ "echo" messages associated with these vote/tally, which means $O(n^3)$ messages are sent in total.

The communication complexity can be reduced to $O(Ln^2 + \kappa n^3)$ with a simple modification: each message contains only the hash of the block (of length $O(\kappa)$) and nodes transmit blocks separately (and only once per block). When dealing with sufficiently large blocks, i.e., $L = \Omega(\kappa n)$, it will be $O(Ln^2)$. This matches the cost of all existing sleepy consensus protocols including longest-chain protocols.

6.3 Efficient Recovery

We have assumed for simplicity that any message sent by an honest node at time t is received by the recipient awake at any time $t' \geq t + \Delta$ (Section 2). This is clearly an impractical assumption since it implies that messages must be magically buffered until the recipient comes back awake. In practice, we have to assume that the message will be lost if not received by the recipient within Δ time, and we must provide an explicit message recovery mechanism for newly awake nodes.

Let us begin by adjusting the model to accommodate message loss. We follow the *sleepy model with recovery* model introduced by Momose-Ren [28]. In addition to the *awake/asleep* statuses, we introduce a third status called *recovering*. When a node transitions from asleep to awake, it enters the recovering status. During this period, the node retrieves missing information from other awake nodes to catch up. The length of this grace period is denoted as $\Gamma \geq 2\Delta$. In theory, $\Gamma = 2\Delta$ suffices as a single round trip fetches all missing data. In practice, Γ will depend on how much data a node needs to retrieve. The message delivery assumption is that if an honest node p awake at time t sends a message, then the message will be received by the recipient q as long as q is recovering or awake at all times during $[t, t + \Delta]$. A node is treated as awake after completing the recovery process.

Now we present the concrete recovery sub-protocol of our atomic broadcast in Algorithm 4. When a node p joins the execution (i.e., as a new recovering node), it begins the process by querying other nodes with a "recovery" message with the hash of the highest block B it has ever decided (line 1–4). If p has not decided or has not even been awake, then the hash corresponds to the genesis block. Other nodes respond to the recovering node with the required

Algorithm 4 Recovery mechanism for Algorithm 3

```
Node $p$ executes the following algorithm.
// query other nodes
upon joining the execution
    $B \leftarrow$ the highest block that $p$ has ever decided
    multicast $\langle$"recover", $\mathsf{H}(B)\rangle_p$
    wait for $\Gamma$ and resume the execution of Algorithm 3
// respond to a recovering node
upon receiving $\langle$"recover", $h\rangle_q$
    if $p$ has decided a block $B$ s.t. $\mathsf{H}(B) = h$ then
        send to $q$ all decided blocks extending $B$
    Let $v$ be the current view.
    sends to $q$ all messages of view $v$ and $v - 1$.
```

information p might have missed. Specifically, each node sends p all decided blocks after B (i.e., missing log contents) as well as all messages of the current view v and the preceding view $v - 1$. Recall that each view of our main protocol (Algorithm 3) relies only on messages from the current view and the immediate last view. More concretely, it relies on the result from GA_{v-1} and "decide" messages from view $v - 1$. As a direct consequence, all messages from older views $v' < v - 1$ do not need to be recovered (except for the log contents).

Note that our recovery protocol is completely decoupled from the main protocol, so the proofs in Section 6.1 still hold.

7 IMPOSSIBILITY OF SUPPORTING FLUCTUATING CORRUPTION

This section shows the impossibility of consensus in the sleepy model with corrupt nodes' fluctuation. Namely, there is no atomic broadcast protocol against the standard (PPT) adversary in the (T_f, T_b, α)-sleepy model with any bounded T_f, T_b, and constant α.

Assumptions. For clarity, let us review the assumptions that are critical to the result.

(1) We assume a standard probabilistic polynomial-time (PPT) adversary that is allowed to perform any polynomial (in κ) amount of computation. This means we do not have any proof-of-work style assumption on relative computation power.

(2) The adversary can fully control any corrupt node once it becomes awake. This includes extracting the entire private state as well as deciding all the messages the node sends.

(3) We assume the communication channels between nodes are *unauthenticated*. When an honest node receives a message, the node cannot tell the origin of the message. It is worth emphasizing and clarifying that what we are assuming here is that there are no *innate* authenticated channels in the model. A protocol can choose to implement authenticated channels using digital signatures and PKI; but looking ahead, these *cryptographic* authenticated channels will be broken by an adversary who extracts private keys.

Under these assumptions, we can show the following result.

Theorem 7.1. *For any $T_f, T_b = \mathsf{poly}(\kappa)$ and $0 < \alpha < 1$, no atomic broadcast protocol exists in the (T_b, T_f, α)-sleepy model.*

Proof sketch. We give a sketch of the proof here and defer the full proof to Appendix A). Intuitively, our proof is based on an adversary performing a forward simulation attack. More concretely, consider a network of two sets of $1/\alpha$ nodes P and Q (assume for simplicity that $1/\alpha$ is an integer), and a node r. Nodes in P are honest and always awake. Since less than α fraction of awake nodes can be corrupt, we can have one corrupt node in each period of $T = T_f + T_b$ time. For $k \in [1, 1/\alpha]$, the adversary makes each node $q_k \in Q$ awake and then asleep immediately at time $t = (k - 1)T$ after extracting all its private states. Now, after time $t = T/\alpha$, the adversary holds all private states of Q and can simulate any execution using nodes in Q as if they were awake from the beginning. This essentially breaks the honest majority requirement [32] of the $(\infty, \infty, 1/2)$-sleepy model (the original sleepy model) and allows the adversary to convince an honest node r who wakes up after time T/α with the simulated execution, leading to an incorrect decision.

8 RELATED WORK

Byzantine consensus has been studied for several decades, with a primary focus on the static and known participation model [7, 9, 16, 17, 19, 25]. The emergence of the Bitcoin protocol [29] marked a turning point, which inspired a new area of research in Byzantine consensus that considers unknown and dynamic participation. This unknown and dynamic participation model was later formalized as the sleepy model [32]. Below, we review the related works in sleepy consensus research.

Latency of sleepy consensus. Early research on sleepy consensus naturally adopted Bitcoin's longest-chain paradigm. A number of works generalized the longest-chain paradigm by substituting the computationally intensive proof-of-work with proof-of-stake [5, 12, 24, 32]. However, one of the major drawbacks of the longest-chain protocol is its inherent long latency. In particular, the basic longest-chain protocol like Bitcoin has a latency of $\Omega(\frac{\kappa\Delta}{\gamma})$ where κ is the desired security level, γ is the active participation level (i.e., the fraction of active nodes compared to the total nodes), and Δ is the bound on network delay. Efforts have been made to eliminate some of the factors that contribute to this long latency. Prism [6], Parallel Chain [20], and Taiji [26] removed the dependency on κ using many parallel instances of longest chains, but maintained the dependency on γ. A recent work by D'Amato et al. [13] achieves $O(\Delta)$ latency under optimistic conditions where the participation level is high, but it inherits the long latency of a longest-chain protocol under low participation level.

Another line of work adapts the classic BFT paradigm from the traditional known and static participation model to the sleepy model. Goyal et al. [22] removes the dependency on γ by extending Algorand [21], but the dependency on κ remains. Furthermore, due to the use of a static quorum threshold, it places a constraint on honest nodes' fluctuation as it requires a steady presence of $\Omega(\kappa)$ awake honest nodes at all times.

The closest to our work is the work by Momose-Ren [28], which for the first time eliminates both of the above dependencies and achieves $O(\Delta)$ latency. The protocol is built on the classic view-based construction and each view consists of a VRF-based leader election and five consecutive invocations of GA. That protocol

	latency	
protocol	best-case	expected
longest-chain PoS [5, 12, 24, 32]	$O(\kappa\Delta/\gamma)$	
multi-chain [6, 20, 26]	$O(\Delta/\gamma)$	
Goyal et al. [22]	$O(\kappa\Delta)$	
Momose-Ren [28]	16Δ	32Δ
Gafni-Losa [27]	10Δ	20Δ
this work	4Δ	14Δ

Table 1: Latency of sleepy consensus. κ is the security level, γ is the active participation level, and Δ is the bound on network delay.

incurs a latency of at least 16Δ time. Moreover, their GA protocol requires *eventual stable participation* for liveness.

A concurrent and independent work by Gafni-Losa [27] also presents a sleepy consensus with $O(\Delta)$ expected latency with optimal corruption threshold. They also remove the eventual stable participation assumption. The best-case latency is 10Δ and the expected latency is 20Δ. In contrast, our protocol achieves 4Δ best-case latency and 14Δ expected latency. The protocol consists of sequential invocations of *Commit-Adopt* sub-protocol (similar to GA). We also note that their protocol is a single-shot and agreement-style protocol rather than an atomic broadcast. Furthermore, their adversary model is rather strong, abstracting away the possibility of a costless simulation attack.

Dynamic participation of corrupt nodes. Another drawback of the previous sleepy consensus protocols is that they do not support growing corrupt participation (proportional to the overall participation level) unless assuming proof-of-work (or other strong assumptions [12, 14, 15]) due to the *costless simulation* problem [15]. Our protocol removes this constraint and supports growing corruption. An interesting insight we learned is "stateless" algorithm is the key to supporting growing corruption. Namely, each protocol's step must depend only on the recent messages. If the protocol makes decisions based on long-past messages, it would be vulnerable to an adversary trying to fabricate past messages (i.e., the backward simulation). This shows another advantage of the classic BFT-like quorum-based design for sleepy consensus; the longest-chain paradigm is by design vulnerable to backward simulation as they depend on the whole mining results in the past.

A possible but orthogonal technique to defend against backward simulation is *key evolution* [10, 14]. Here, honest nodes constantly evolve their signing keys and erase their stale keys so that when they are corrupted, the adversary cannot access their old keys to simulate past messages on their behalf. This technique is effective in preventing backward simulation in the static and known participation model (e.g., in Algorand [10]). In the sleepy model with dynamic participation, however, an additional strong assumption is required for this technique to work: an adversary cannot corrupt a new active node before it completes evolving the key. Otherwise, an adversary could corrupt nodes immediately after they wake up, gain access to their original keys and sign their past messages.

Fallback under asynchrony. As mentioned in Section 2, the synchrony assumption is necessary for consensus in the sleepy model. Therefore, it is inherent that our protocol loses safety under asynchrony. Nonetheless, there have been a few proposals to balance dynamic participation and partition tolerance [30, 35]. These protocols involve running a partially synchronous checkpointing protocol on top of the underlying sleepy consensus, which is also applicable to our protocol.

9 CONCLUSION

This work presents an atomic broadcast protocol in the sleepy model with 4Δ latency in the best case, based on the new construction of a view-based protocol optimized for reducing best-case latency. Our protocol achieves liveness under wildly fluctuating honest participation with the new GA protocol that does not require stable honest nodes. The stateless nature of our atomic broadcast protocol allows us to achieve tolerance to growing corruption and also achieve efficient recovery for new active nodes.

We have shown the impossibility of supporting fluctuating corruption (both growing/shrinking) with the standard adversary that can extract all corrupt nodes' private state. However, the assumption is too strong in practice. For example, in the proof-of-stake protocols, it is highly unlikely that corrupt nodes hand off their secret keys to the adversary (or other corrupt nodes) at the risk of losing their entire stake. Supporting corrupt nodes' fluctuation in a weaker but more realistic model is an interesting future work.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers at ACM CCS 2023 for their helpful feedback. We also thank Lorenzo Alvisi, Ittay Eyal, Jacob Leshno, Kartik Nayak, Youer Pu, Jun Wan, for valuable discussions. This work is supported in part by NSF award 2143058.

REFERENCES

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

[2] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. 2020. Sync HotStuff: Simple and Practical Synchronous State Machine Replication. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 106–118.

[3] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. 2021. Good-case latency of byzantine broadcast: A complete categorization. In *ACM Symposium on Principles of Distributed Computing (PODC)*. 331–341. https://doi.org/10.1145/3465084.3467899

[4] Hagit Attiya and Keren Censor. 2008. Lower bounds for randomized consensus under a weak adversary. In *ACM Symposium on Principles of Distributed Computing (PODC)*. 315–324.

[5] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. 2018. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 913–930.

[6] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. 2019. Prism: Deconstructing the blockchain to approach physical limits. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 585–602.

[7] Gabriel Bracha and Sam Toueg. 1985. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)* 32, 4 (1985), 824–840.

[8] Ethan Buchman. 2016. *Tendermint: Byzantine fault tolerance in the age of blockchains.* Ph.D. Dissertation.

[9] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *3rd Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, 173–186.

[10] Jing Chen and Silvio Micali. 2016. Algorand. *arXiv preprint arXiv:1607.01341* (2016).

[11] Flaviu Cristian, Houtan Aghili, Ray Strong, and Danny Dolev. 1995. Atomic broadcast: From simple message diffusion to Byzantine agreement. *Information and Computation* 118, 1 (1995), 158–179.

[12] Phil Daian, Rafael Pass, and Elaine Shi. 2019. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography and Data Security (FC)*. Springer, 23–41.

[13] Francesco D'Amato, Joachim Neu, Ertem Nusret Tas, and David Tse. 2022. No More Attacks on Proof-of-Stake Ethereum? *arXiv preprint arXiv:2209.03255* (2022).

[14] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Springer, 66–98.

[15] Soubhik Deb, Sreeram Kannan, and David Tse. 2020. PoSAT: Proof-of-Work Availability and Unpredictability, without the Work. *arXiv preprint arXiv:2010.08154* (2020).

[16] Danny Dolev and H. Raymond Strong. 1983. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput.* 12, 4 (1983), 656–666.

[17] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the presence of partial synchrony. *J. ACM* 35, 2 (1988), 288–323.

[18] Paul Feldman and Silvio Micali. 1988. Optimal algorithms for Byzantine agreement. In *20th Annual ACM Symposium on Theory of Computing (STOC)*. 148–161.

[19] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)* 32, 2 (1985), 374–382.

[20] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Parallel Chains: Improving Throughput and Latency of Blockchain Protocols via Parallel Composition. *IACR Cryptology ePrint Archive, Report 2018/1119* (2018).

[21] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *26th Symposium on Operating Systems Principles (SOSP)*. 51–68.

[22] Vipul Goyal, Hanjun Li, and Justin Raizes. 2021. Instant Block Confirmation in the Sleepy Model. In *Financial Cryptography and Data Security (FC)*.

[23] Jonathan Katz and Chiu-Yuen Koo. 2009. On expected constant-round protocols for byzantine agreement. *J. Comput. System Sci.* 75, 2 (2009), 91–112.

[24] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference (CRYPTO)*. Springer, 357–388.

[25] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* 4, 3 (1982), 382–401.

[26] Songze Li and David Tse. 2020. TaiJi: Longest Chain Availability with BFT Fast Confirmation. *arXiv preprint arXiv:2011.11097* (2020).

[27] Giuliano Losa and Eli Gafni. 2023. Consensus in the Unknown-Participation Message-Adversary Model. *arXiv preprint arXiv:2301.04817* (2023).

[28] Atsuki Momose and Ling Ren. 2022. Constant latency in sleepy consensus. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2295–2308.

[29] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[30] Joachim Neu, Ertem Nusret Tas, and David Tse. 2020. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. *arXiv preprint arXiv:2009.04987* (2020).

[31] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Springer, 643–673.

[32] Rafael Pass and Elaine Shi. 2017. The sleepy model of consensus. In *Annual International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Springer, 380–409.

[33] Marshall Pease, Robert Shostak, and Leslie Lamport. 1980. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)* 27, 2 (1980), 228–234.

[34] Ling Ren. 2019. Analysis of Nakamoto Consensus. IACR Cryptology ePrint Archive, Report 2019/943. (2019).

[35] Suryanarayana Sankagiri, Xuechao Wang, Sreeram Kannan, and Pramod Viswanath. 2020. The Checkpointed Longest Chain: User-dependent Adaptivity and Finality. *arXiv preprint arXiv:2010.13711* (2020).

[36] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. Hotstuff: Bft consensus with linearity and responsiveness. In *ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 347–356.

A PROOF OF IMPOSSIBILITY OF SUPPORTING FLUCTUATING CORRUPTION

We give the proof of Theorem 7.1 below.

Theorem A.1 (Theorem 7.1 restated.). *For any $T_f, T_b = \mathsf{poly}(\kappa)$ and $0 < \alpha < 1$, there does not exists an atomic broadcast in the (T_b, T_f, α)-sleepy model.*

Proof. Let $\beta = 1/\alpha$. We assume β is an integer (the proof is easily extended to the case β is not an integer). Suppose there exists such a protocol. Let $T = T_f + T_b$. Let $\Gamma > \beta T$ be the smallest value s.t. the protocol satisfies Γ-Liveness. Let P and Q be two disjoint sets of β nodes, and $r \notin P \cup Q$ be a node. Consider the following two executions.

W1. For each $k \in [1, \beta]$, there is a unique corrupt node $p_k \in P$ that becomes awake and then asleep immediately at time $t = (k-1)T$. All nodes in Q are honest and always awake. At time $t = \Gamma$, a new honest node r becomes awake. The honest nodes Q input a set X of values. All corrupt nodes P do not send any message. Until time $t = \Gamma$, the adversary simulates an honest execution among the corrupt nodes P as if all nodes in P were always awake from time 0 to Γ and they had a set X' of input values s.t. $X \cap X' = \emptyset$. At time $t = \Gamma$, the adversary delivers all messages in the simulated execution to the new awake node r.

W2. The second execution is symmetric. For each $k \in [1, \beta]$, there is a unique corrupt node $q_k \in Q$ that becomes awake and then asleep immediately at time $t = (k-1)T$. All nodes in P are honest and always awake. At time $t = \Gamma$, a new honest node r becomes awake. The honest nodes P input a set X' of values. All corrupt nodes Q do not send any message. Until time $t = \Gamma$, the adversary simulates an honest execution among the corrupt nodes Q as if all nodes in Q were always awake from time 0 to Γ and they had a set X of input values s.t. $X \cap X' = \emptyset$. At time $t = \Gamma$, the adversary delivers all messages in the simulated execution to the new awake node r.

Let view_1 and view_2 be the random variables that describe the set of messages that r receives in W1 and W2, respectively. Obviously, the distribution of these two variables must be identical since they both consist of two separate honest executions among P with input X', and among Q with input X. So, the log decided by r must be identically distributed in both W1 and W2. However, due to Γ-liveness, r must decide a log containing X in W1, and in W2, r must decide a log containing X'. So the two distributions should be different; a contradiction. □