# Constant Latency in Sleepy Consensus

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

Dynamic participation support is an important feature of Bitcoin's longest-chain protocol and its variants. But these protocols suffer from long latency as a fundamental trade-off. Specifically, the latency depends at least on the following two factors: 1) the desired security level of the protocol, and 2) the actual participation level of the network. Classic BFT protocols, on the other hand, can achieve constant latency but cannot make progress under dynamic participation. In this work, we present a protocol that simultaneously supports dynamic participation and achieves constant latency. Our core technique is to extend the classic BFT approach from static quorum size to *dynamic quorum size*, i.e., according to the current participation level, while preserving important properties of static quorum. We also present a recovery mechanism for rejoining nodes that is efficient in terms of both communication and storage. Our experimental evaluation shows our protocol has much lower latency than a longest-chain protocol, especially when there is a sudden decrease of participation.

## CCS CONCEPTS

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

## KEYWORDS

BFT Protocols; Blockchain; Dynamic Participation; Sleepy Model

**ACM Reference Format:**
Atsuki Momose and Ling Ren. 2022. Constant Latency in Sleepy Consensus. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3548606.3559347

## 1 INTRODUCTION

Byzantine fault-tolerant (BFT) consensus [24], the underlying technology of blockchains [28], allows parties to reach consensus in the presence of malicious parties who behave arbitrarily. Classic BFT consensus protocols [10, 24, 32] assume a static and known participation model, i.e., all participants know each other in advance and all participants stay active in the protocol all the time. A central innovation of Nakamoto's Bitcoin protocol [28] is the support for dynamic participation, i.e., participants are not known beforehand and they can leave or join the system at will at any time.

The Bitcoin protocol invented an elegant "longest-chain" paradigm, where winners of random proof-of-work lotteries extend the longest chain of blocks in their views. Subsequent works extend the Bitcoin's longest-chain paradigm to proof-of-stake to avoid the expensive proof-of-work [4, 13, 14, 31]. Naturally, these protocols inherit the Bitcoin's support for dynamic participation, but also its fundamental drawback of long latency. More specifically, their latency is at least $\Omega(\frac{\kappa\Delta}{\gamma})$ where $\kappa$ is a security parameter, $\Delta$ is the network delay bound, and $\gamma \leq 1$ is the fraction of actual participants (compared to the anticipated level of participation). The dependence on the security parameter is due to the $k$-confirmation rule. A block is decided only after $k$ subsequent blocks are generated, where $k$ depends linearly on the desired security level. The block interval of longest-chain protocols must be noticeably larger than $\Delta$ [14, 30]. The block interval further increases if the actual participation level suddenly drops, reducing the total hash rate or active stakeholders (which is why the latency is inverse proportional to $\gamma$).

On the other hand, classic BFT protocols can achieve (expected or amortized) latency of $O(\Delta)$ [1, 19]. This motivates the following natural question:

*Can we design BFT consensus protocols that simultaneously support dynamic participation and achieve $O(\Delta)$ latency?*

In order to answer this question, we must formalize the consensus problem and the model of dynamic participation. We will focus on the BFT atomic broadcast problem [8, 12], where parties agree on a ledger (a linearizable log). As for the model, our starting point will be the *sleepy model* of Pass and Shi [31] (and we will further relax the model later). The sleepy model assumes that the number of honest participants can fluctuate at the adversary's control, with the constraint that there are more honest parties than Byzantine parties at any time.

The above question has recently been studied and partly addressed from two different angles. Goyal et al. [20] achieves $O(\kappa\Delta)$ latency by building on Algorand [11, 19]; In other words, the latency of their protocol no longer depends on the participation level but still depends on the security parameter. From another angle, Prism [5], Parallel Chains [18], and TaiJi [25] remove the dependence on the security parameter $\kappa$ but the dependence on participation level remains, i.e., $\Omega(\frac{\Delta}{\gamma})$ latency.

In this work, we answer the above question in the affirmative by resolving both of these trade-offs. Specifically we show the following result:

Theorem 1.1 (Informal). *Assuming a verifiable random function (VRF) and public-key infrastructure (PKI), there exists a BFT atomic broadcast protocol in the sleepy model that tolerates minority*

*The work was done while the author was at SECOM CO., LTD.


*CCS '22, November 7–11, 2022, Los Angeles, CA, USA*

ACM ISBN 978-1-4503-9450-5/22/11…$15.00
https://doi.org/10.1145/3548606.3559347

*faults and has an expected $O(\Delta)$ latency during periods of stable participation.*

We proceed to elaborate more on our result and key technical challenges below.

**Adopting classic BFT approach.** Our protocol follows the classic BFT approach. We first design a quorum-based graded agreement (GA) protocol [1, 17, 22, 27], and build an atomic broadcast protocol using GA. Adapting the quorum-based approach to the sleepy model brings several challenges. First, the quorum threshold in a classic protocol is set based on the total number of parties, which is fixed and known in advance. Obviously, such a static quorum threshold does not work in the sleepy model since the participation level is unknown and can fluctuate over the execution. This problem was also pointed out by Pass and Shi [31]: *"... the problem then becomes how to set the (quorum) threshold, as the protocol is not aware of how many players are actually awake."* Partly due to this technical difficulty, they and most prior works had to follow the longest-chain paradigm and inherit its long latency.

This is also the technical difficulty this paper sets out to address: we would like to adapt classic quorum-based approach to the sleepy model. Our first natural idea is to use a *dynamic quorum* where the quorum size is defined based on each party's "perceived" participation level. To elaborate, participants announce that they are active. If a party perceives that $m$ parties are currently participating in the protocol, then it locally considers the quorum size to be $\lfloor m/2 \rfloor + 1$, i.e., majority of its perceived participants.

However, this brings another challenge. In classic BFT protocols, a quorum of votes (with digital signatures) on a value, often called a *quorum certificate*, is *transferable*. In other words, a quorum certificate recognized by one party is also recognized as a quorum certificate by all other parties at all times. This trivial fact for static quorum does not hold for dynamic quorums, because the perceived participation level may differ across parties. This can happen because malicious parties may announce themselves to some honest parties but not others. To make the matters worse, even if all participants currently recognize a quorum certificate, they may stop recognizing it if more malicious parties announce themselves later to raise the perceived participation level and hence the required quorum size. In particular, newly joined parties cannot tell whether or not the quorum certificate was ever valid in the past. The main technical contribution of this paper is to restore this transferablility property with dynamic quorums.

**Efficient recovery.** The original sleepy model assumes that a party, upon rejoining the protocol, immediately receives all messages sent to it during its sleep. While simple in theory, this is an unrealistic assumption because it essentially requires each party to keep track of and re-transmit every single message it has ever sent or received. To avoid this issue, we design an efficient recovery mechanism for rejoining nodes where only ledger contents and recent protocol messages (independent of the length of its sleep) need to be re-transmitted. More specifically, our protocol periodically identifies points in the execution where parties can "forget" older messages. This also avoids the impractical storage requirement for past messages in the original sleepy model.

**Experimental evaluation.** To demonstrate the improvement of our protocol, we implement and evaluate our basic protocol and compare with the longest-chain protocol in [31]. Our experiments show the expected results. Specifically, the longest-chain protocol suffers from long latency even when all parties are active, due to the dependence on the security parameter $\kappa$. Its latency further deteriorates when the participation level suddenly drops due to the dependence on $\gamma$. In contrast, our protocol has low latency, except during periods of wild fluctuation.

**Additional advantage and assumption.** We remark on one more advantage and one (reasonable) extra assumption of our protocol.

Our protocol tolerates minority faults, which in the sleepy model means that the number of malicious parties $f$ is at most $n/2$ where $n$ is the minimum number of active participants over the course of the protocol execution. This $f < n/2$ condition has been shown to be necessary in the sleepy model [31]. This is another advantage of our protocol over existing dynamic participation protocols, all of which tolerate only $f \leq (1/2 - \epsilon)n$ where $\epsilon$ is a positive constant.

Our protocol makes progress only in periods of "stable participation", i.e., periods during which the participants do not change too wildly (formalized in Section 3). We believe this is a reasonable assumption in practice since it seems highly unlikely that the participation level fluctuates wildly all the time. Note that our protocol maintains safety even during arbitrarily wildly fluctuating periods and will start making progress again whenever the participation stabilize.

**Summary of results.** In summary, we have the following results in this work:

(1) We present a BFT atomic broadcast protocol in the sleepy model with expected $O(\Delta)$ latency (Section 5). Our protocol is built from a graded agreement (GA) protocol (Section 4) using a quorum-based design.
(2) We relax the sleepy model and present an efficient recovery mechanism for our protocol (Section 6).
(3) We experimentally evaluated our protocol and compared with the longest-chain protocol (Section 7).

## 2 RELATED WORKS

Classic BFT protocols assume static participation where all parties always participate. Nakamoto's longest-chain proof-of-work paradigm is the first BFT protocol to support dynamic participation. Recent works extend the longest-chain paradigm to proof-of-stake and inherit the long latency drawback [4, 13, 14, 31].

**Multi-chain protocols.** Multi-chain approach [5, 18, 25] has been studied for removing the dependence on the security parameter $\kappa$. They speed up the confirmation of blocks (or transaction) by running multiple longest-chains in parallel. Since the block interval of each longest-chain is subject to the total hash rate (or active stakeholders) their latency still depend on the active participation level $\gamma$. Prism and TaiJi still use the energy-inefficient proof-of-work approach.

**Goyal et al.** Goyal et al. [20] also takes the classic quorum-based approach to achieve $O(\kappa\Delta)$ latency in the sleepy model. It adopts Algorand's approach of sampling a committee of size $\Omega(\kappa)$ at every step [11, 19]. Since the committee size is fixed in advance, their

protocol requires there to be at least $n = \Omega(\kappa)$ participants at any time. On the other hand, our protocol works with any $n > 0$ participants. For the concrete latency, their protocol costs $18\kappa\Delta$, which is much worse (even for low security level, e.g., $\kappa = 10$), compared to $37\Delta$ of our protocol.

**Unknown participation model.** Another recent work [23] studies Byzantine agreement in the *unknown participation* model where the number of participants $n$ is unknown to the parties but remains fixed over time. They considers an unauthenticated setting and tolerate only $f < n/3$ faults. They also present protocols with dynamic participation support. However, their model is still stronger than the sleepy model. In their model, an honest party, when it goes offline, can announce its absence to the network. On the other hand, in the sleepy model, honest parties go offline without giving any advance notice.

**Crash fault tolerance.** If there are only crash faults and no Byzantine faults, supporting dynamic participation becomes much easier. In this case, parties will announce themselves honestly to the network, so a protocol can correctly observe the current participation level and does not run into the transferability problem. There exist prior works on crash fault tolerant dynamic atomic broadcast [6, 7]. These protocols also take the quorum-based approach.

**Asynchronous fallback.** Synchronous communication is necessary in the sleepy model [31]. To resolve this problem, ebb-and-flow [29] and checkpointed longest-chain [33] study ways to balance dynamic participation and partition tolerance. Their protocols output two ledgers: a longest-chain ledger and a BFT ledger. The longest-chain ledger supports dynamic participation while the BFT ledger tolerates asynchrony but does not support dynamic participation. Our protocol can replace the longest-chain ledger to improve the latency of their protocols.

## 3 MODEL AND DEFINITIONS

This paper considers a Byzantine fault-tolerant (BFT) atomic broadcast problem in the sleepy model [31]. We consider a system of $N$ total parties communicating over a synchronous network. Note that network synchrony is necessary [31] for the sleepy model. For simplicity, we use $\Delta$ to denote both bounds on communication delay and clock skew. An adversary is adaptive and can corrupt parties anytime during the protocol execution. Faulty parties are Byzantine and behave arbitrarily. A party that is not faulty throughout the execution is said to be honest and faithfully executes the protocol.

**Loosely synchronized clocks.** We assume all parties have access to their own local clocks that differ by at most $\Delta$ and start from 0 at the beginning of the protocol execution. Without loss of generality, we can use the local time of the *first* party to start as a reference time, also called the *global time*. This way, when the global time is $t$, each party $p$'s local time is $\tau_p = t - \delta_p$ where $0 \le \delta_p \le \Delta$.

**The sleepy model.** Our protocols in Section 4 and 5 assumes the sleepy model introduced in [31]. A party is either *awake* or *asleep*. An awake party actively participates in the execution, while an asleep party does not execute any code or send/receive any message. The status of each party can change at the adversary's control, at any time, without any advance notice. In the real world, this means parties are allowed to leave the execution at will without notifying other parties. The number of awake parties at each global time $t$ is denoted $0 < n_t \le N$; out of these at most $f < n_t/2$ can be faulty. All faulty parties are awake all the time. The message delivery assumption is that if an honest party $p$ is awake at global time $t$, then $p$ has received all messages that were sent to it by honest parties by global time $t - \Delta$.

**Atomic broadcast.** Atomic broadcast allows parties to agree on a growing sequence of values $[x_0, x_1, x_2, \ldots]$ called a *log*. It provides the following guarantees:

(1) **Safety.** If two honest parties decide logs $[x_0, x_1, .., x_j]$ and $[x'_0, x'_1, .., x'_{j'}]$, then $x_i = x'_i$ for all $i \le \min(j, j')$.
(2) **Liveness.** If an honest party inputs a value $x$, then there exists a global time $t$ such that all honest parties awake at any global time $t' \ge t$ decide a log containing $x$.

The above safety condition is also called *total order* and the liveness condition is also called *censorship resistance* or *fairness* [9, 26]. Some previous works [9, 26] define another property called *agreement* that says "if an honest party decides a value $x$, all honest parties also decide $x$". This is implied by our safety and liveness, and is hence redundant.

**Eventually stable participation.** As mentioned, when the number of awake honest parties fluctuate wildly, our protocol maintains safety but cannot make progress. To ensure liveness, we need an additional assumption called *eventually stable participation* (inspired by the well-known eventual synchrony model [16]). We say a party is *insomniac* during $[t, t']$ if it stays awake at all time during $[t, t']$.

*Definition 3.1 ($T$-eventually stable participation).* There exists a global time $T_s \ge 0$ (unknown to the parties) such that for all $t \ge T_s$, more than $\alpha/2$ honest parties are insomniac during $[t, t+T]$ where $\alpha$ is the number of parties ever awake at some time during $[t, t+T]$.

Our protocol assumes $7\Delta$-eventually stable participation. Intuitively, this means after the stabilization time $T_s$, awake parties are replaced slowly such that any time window of length at least $T = 7\Delta$ has sufficiently many parties that stay awake throughout. We remark that requiring stable participation at all time after $T_s$ is for theoretical convenience, just like the well established eventual synchrony model [16]. In practice, the protocol will make progress in any sufficiently long period of stable participation. Also note that, since $T_s$ is unknown, a protocol's safety cannot depend on this assumption.

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

### 3.1 Sleepy Model with Recovery

The original sleepy model assumes that an asleep party, upon waking up (i.e., rejoining the network), immediately receives *all* past

messages that were sent to it during its sleep (subject to communication delay). While simplifying the model, this is an impractical assumption as discussed in Section 1.

In Section 6, we remove this unrealistic assumption from the sleepy model and replace it with a concrete and practical recovering mechanism. In our model, a party is in one of three statuses: awake, asleep, and *recovering*. When an asleep party rejoins the network, it becomes *recovering* and has a grace period of $\Gamma$ (discussed later) before it becomes awake. If an honest party $p$, awake or recovering, sends a message $x$ at global time $t$ to a party $q$ who is awake or recovering at all time during $[t, t + \Delta]$, then $q$ receives $x$ by global time $t + \Delta$. In particular, our relaxed model does not assume reliable transmission of messages sent to an asleep party. Any subset of these messages can be lost. In other words, we show that a recovering node does not have to receive *all* missed messages. Instead, we will give an explicit mechanism for a recovering party and show that retrieving *recent* missed messages is sufficient.

**Bound on recovery delay.** In theory, a grace period of $\Gamma = 2\Delta$ is sufficient for recovering: upon rejoining the execution, the recovering party $p$ multi-casts a recovery request, which is received within $\Delta$ by all awake honest parties, who respond with messages and data $p$ missed, which takes another $\Delta$. In practice, the recovery process may take longer if a lot of data needs to be transmitted. Therefore, in this paper, we treat $\Gamma \gg 2\Delta$ as an independent parameter. Each party can independently choose its own $\Gamma$ based on how much data it needs to catch up.

**Relation to crash-recovery.** Once we provide an explicit recovery mechanism, the sleepy/recovering process can be thought of as the *crash/recovery* process in the distributed computing literature [8]. And our protocol can be thought of as one that tolerates any number of crash-recovery faults plus minority Byzantine faults. In this paper, however, we will use the awake/asleep terminology following [31].

## 3.2 Additional Remark on the Sleepy Model

Here, we emphasize again the real-world scenario we hope to capture with the sleepy model: we think of an asleep party as a party who temporarily leaves the system (i.e., knowingly shut down their computers), rather than a party who experiences sporadic delay of the network. This leads to an important difference in the model. A model that captures sporadic delays may assume that a party experiencing long network delays is unaware of it and keeps participating in the protocol [21]. These protocols still assume a majority of parties are honest and have good networks at any time, so they do not attempt the dynamic participation problem.

In contrast, to support dynamic participation, it is *necessary* for us to assume that an asleep party *knowingly went to sleep* and does not take any actions during its sleep. We briefly prove the necessity of this assumption below.

**Proof sketch.** Suppose parties do not know whether they are awake or asleep. Consider a network of two groups of parties $P$ and $Q$. Parties in $P$ are asleep and parties in $Q$ are awake. No party is malicious. The adversary delays all messages between $P$ and $Q$ (this is possible because $P$ is asleep). Because the protocol works under any level of participation, and asleep parties do not know they are asleep, both $P$ and $Q$ need to eventually decide. With no communication between them, they will decide differently. Therefore, without the knowledge of awake/asleep, any protocol in the sleepy model will lose safety.

# 4 GRADED AGREEMENT

This section presents a graded agreement subroutine, which will be an important building block of our atomic broadcast protocol discussed later.

**Graded Agreement (GA).** In our graded agreement, each party has an input value (possibly empty $\perp$) and outputs (possibly multiple[1]) pairs of $(b, g)$ of value $b$ and grade bit $g \in \{0, 1\}$ providing the following guarantees for a certain time $T$.

(1) **Consistency.** If an honest party outputs $(b, 1)$, then every honest party awake at time $t \geq T$ outputs $(b, *)$.
(2) **Integrity.** If no honest party inputs a value $b$, no honest party outputs $(b, *)$.
(3) **Validity.** If all honest parties awake at the beginning (time 0) has the same input value $b$, then all honest parties that stay awake all the time outputs $(b, 1)$.

The validity property our protocol achieves will be a little more complex than the one above, but for ease of exposition, let us use the above simple version for the time being.

**In the classic model.** In the classic static participation model, one can easily come up with a quorum-based protocol with honest majority, i.e., $N = n = 2f + 1$, as follows: In round 1 (time 0), each party multi-casts a vote for its own input. $f + 1$ votes for the same value is often called a *quorum certificate* (or certificate for short) for that value. If a party receives a certificate for $b$ at the end of round 1 (time $\Delta$), it outputs the value $b$ with grade 1, and forward the certificate to all other parties. If it receives the certificate in round 2 or later (time $t > \Delta$), it outputs the value $b$ with grade 0. Note that certificate *transferrability* is crucial for consistency. An honest party who outputs $(b, 1)$ will forward the certificate, and all honest parties will also recognize the certificate and output $(b, 0)$.

## 4.1 Warmup: A Lockstep GA

To aid understanding, let us first construct a GA in the lockstep round model where parties have access to a common clock, i,e., every party's local time equals the global time. The lockstep protocol is given in Figure 1 and we explain the key challenges and our techniques below.

**Dynamic quorum.** Our first natural idea is to make the quorum threshold based on the "perceived" participation level, i.e., the number of parties that have announced themselves to the network. This ensures that the number of honest parties awake in the voting round can meet the quorum threshold. Specifically, in our protocol, each party multi-casts $\langle \text{awake}, 3 \rangle$ in round 3 (i.e., the voting round), and the quorum threshold is based on the number of $\langle \text{awake}, 3 \rangle$ received so far, denoted as $\mathcal{M}_3$.

However, as mentioned, dynamic quorums do not automatically provide certificate *transferability*; in other words, a certificate recognized by one party may not be recognized as a certificate by other

[1] Our GA is weaker than classic graded agreement because we allow multiple outputs.

Each party $p$ runs the following four-round algorithm.

(1) **Round 1** (time $t = 0$). Multi-cast $\langle \text{awake}, 1 \rangle_p$. If it has non-empty input $b_p \neq \perp$, multi-cast $\langle \text{echo}, b_p \rangle$.

(2) **Round 2** (time $t = \Delta$). Set $\overline{\mathcal{E}}(b)$ for any $b$ to be the number of parties from whom $p$ received $\langle \text{echo}, b \rangle$.

(3) **Round 3** (time $t = 2\Delta$). Multi-cast $\langle \text{awake}, 3 \rangle_p$. Set $\mathcal{E}(b)$ to be the number of parties from whom $p$ received $\langle \text{echo}, b \rangle$, and $\mathcal{M}_1$ to be the number of parties from whom $p$ received $\langle \text{awake}, 1 \rangle$. For all $b$, if $\mathcal{E}(b) > \mathcal{M}_1/2$, multi-cast $\langle \text{vote}, b \rangle_p$.

(4) **After Round 3** (time $t \geq 3\Delta$). Update $\mathcal{M}_1$, the number of parties from whom $p$ received $\langle \text{awake}, 1 \rangle$. Set $\mathcal{M}_3$ to be the number of parties from whom $p$ received $\langle \text{awake}, 3 \rangle$. Let $\mathcal{V}(b)$ be the number of parties from whom $p$ received $\langle \text{vote}, b \rangle$. If $p$ was awake in round 2 and $\overline{\mathcal{E}}(b) > \mathcal{M}_1/2$, output $(b, 1)$. If $\mathcal{V}(b) > \mathcal{M}_3/2$, output $(b, 0)$.

At any time, forward any echo and awake message received.

**Figure 1: Warmup: Lockstep GA in the sleepy model**

parties. This is because, in the sleepy model, perceived participation level, and hence quorum threshold, can differ across parties, if Byzantine parties announce themselves to some parties but not others or withhold their announcements and release them later to newly joined parties to change the quorum threshold.

**Transferablility of dynamic quorum certificates.** To restore transferability of certificates, we need parties to collect votes from a majority of the *actual* participation level (i.e., the number of awake parties in the voting round) instead of the perceived participation level. Since the perceived participation level of any party does not exceed the actual participation level, a certificate consisting of votes from a majority of the actual participation level will be recognized by all parties at all times, even those who joined later. Therefore, our idea is to make sure all honest and awake parties (which is a majority of actual participation) vote, using the following technique.

**Time-shifted quorum.** We introduce a new technique called *time-shifted quorum*. In round 1, each party sends an echo message for its input value along with an awake message for round 1. Each party tracks the number of echo for any value $b$, denoted $\mathcal{E}(b)$, and the perceived participation level of round 1, denoted $\mathcal{M}_1$. Note that these two variables may change over time. In round 3, if a party observes a majority of echo for a value $b$, i.e., $\mathcal{E}(b) > \mathcal{M}_1/2$, it votes for $b$. Finally, after round 3, if the number of echo received by time $\Delta$ still meets the majority, i.e., $\overline{\mathcal{E}}(b) > \mathcal{M}_1/2$, a party can be assured that all honest parties awake in round 3 voted for $b$ (we will prove this next), and it can output $(b, 1)$.

Suppose an honest party $p$ outputs $(b, 1)$ after round 3 (say round $i$). Then, $p$ must have been awake in round 2 and fixed $e_p := \overline{\mathcal{E}}(b)$ to be the number of parties from whom it has received $\langle \text{echo}, b \rangle$ by then. Let $q$ be any honest party that is awake in round 3, and $e_q := \mathcal{E}(b)$ be the number of parties from whom it has received $\langle \text{echo}, b \rangle$ by then. Since $p$ forwards all messages and $q$ (awake in round 3) receives them, $e_p \leq e_q$. Let $m_q$ be the number of parties from whom $q$ has received $\langle \text{awake}, 1 \rangle$ by round 3, and $m_p$ be the number of parties from whom $p$ has received $\langle \text{awake}, 1 \rangle$ by round $i$. Since $q$ forwards all messages and $p$ (awake in round $i$) receives them, $m_q \leq m_p$. Since $p$ outputs $(b, 1)$, we have $e_p > m_p/2$. Since $e_p \leq e_q$ and $m_q \leq m_p$, we have $e_q > m_q/2$. Therefore, all honest parties awake in round 3 multi-cast $\langle \text{vote}, 2 \rangle$ along with $\langle \text{awake}, 3 \rangle$. Since there are always more awake honest parties than faulty parties, all honest parties awake after round 3 observe $\mathcal{V}(b) > \mathcal{M}_3/2$ (i.e., the certificate is transferable) and output $(b, *)$ (ensuring consistency).

## 4.2 Our GA Protocol under Loosely Synchronized Clocks

We now present the non-lockstep version of our GA protocol in Algorithm 1, where parties have loosely synchronized clocks that can differ by up to $\Delta$.

The basic idea is to reserve $2\Delta$ time for each round. Then, even when two parties' local clocks differ by $\Delta$, a message sent in round $r$ always arrives at the recipient by the end of round $r$ from the recipient's perspective. This simple trick would be sufficient to transform a lockstep protocol to loosely synchronized clocks in the classic model.

However, the sleepy model brings a new challenge. Note that our protocol instructs a party to send a message at a particular (local) time $t$. If a party is asleep at its local time $t$, it would not send this message. In particular, this may cause a party to skip sending its vote even when it observes a majority echo. Our time-shifted quorum argument would then break down.

We solve this problem with the following simple modification: we give a party a $\Delta$ time window to send a message. Suppose the lockstep protocol instructs a party to send a message at its local time $t$. Then, in the modified protocol, the party sends the message anytime during $[t, t + \Delta]$. In other words, if a party is asleep at local time $t$ but wakes up before $t + \Delta$, the party sends the message when it wakes up. This way, all honest parties awake at global time $t + \Delta$ will send the message, because their local clocks must be within $[t, t + \Delta]$ at that time.

Line 1 and 8 are applying the above trick. For example, in the voting step, a party votes during its local time $[4\Delta, 5\Delta]$. Thus, all honest parties awake at global time $5\Delta$ will vote (if the condition is met), which is enough to form a transferable certificate.

**Potential for a generic transformation.** We remark that Pass and Shi [31] mentioned very briefly (without proof) a generic transformation from lockstep to loosely synchronized clocks in the sleepy model. But they did not describe how to handle the issue described above, so it is unclear how to apply their transformation to our protocol. Moreover, even if there were a way to apply their transformation, our solution is more efficient; their transformation makes every single round $3\Delta$. It is of independent interest whether our technique constitutes a generic transformation. As of now, we only prove it works for our specific protocol.

**Conflict of messages.** In Figure 1, we let each party forward all echo messages. This can incur unbounded communication, because a faulty party can send echo for infinitely many different values. To avoid this problem, we define *conflict* between messages. Specifically, $\langle \text{echo}, b \rangle_p$ conflicts with $\langle \text{echo}, b' \rangle_p$ for any $b' \neq b$. Similarly, $\langle \text{vote}, b \rangle_p$ conflicts with $\langle \text{vote}, b' \rangle_p$. Our protocol only forwards a

**Algorithm 1** Graded agreement in the sleepy model.

At the beginning of the execution, initialize outputs = $\emptyset$.
Party $p$ executes the following algorithm at every local time $\tau \geq 0$.

*// corresponds to round 1*
1: **if** $0 \leq \tau \leq \Delta$ **then** do the following once
2: multi-cast $\langle \text{awake}, 1 \rangle_p$
3: **if** $p$ has an input value $b_p \neq \perp$ **then**
4: multi-cast $\langle \text{echo}, b_p \rangle_p$

*// corresponds to round 2*
5: **if** $\tau = 2\Delta$ **then**
6: **for all** $b \neq \perp$ such that $p$ has received $\langle \text{echo}, b \rangle$ **do**
7: $\overline{\mathcal{E}}(b) \leftarrow |\{q \mid p \text{ has received } \langle \text{echo}, b \rangle_q \text{ without } \textit{conflict}\}|$

*// corresponds to round 3*
8: **if** $4\Delta \leq \tau \leq 5\Delta$ **then** do the following once
9: multi-cast $\langle \text{awake}, 3 \rangle_p$
10: **for all** $b \neq \perp$ such that $p$ has received $\langle \text{echo}, b \rangle$ **do**
11: $\mathcal{E}(b) \leftarrow |\{q \mid p \text{ has received } \langle \text{echo}, b \rangle_q\}|$
12: $\mathcal{M}_1 \leftarrow |\{q \mid p \text{ has received } \langle \text{awake}, 1 \rangle_q\}|$
13: **for all** $b$ such that $\mathcal{E}(b) > m_1/2$ **do**
14: multi-cast $\langle \text{vote}, b \rangle_r$

*// corresponds to after round 3*
15: **if** $\tau \geq 7\Delta$ **then**
16: $\mathcal{M}_1 \leftarrow |\{q \mid p \text{ has received } \langle \text{awake}, 1 \rangle_q\}|$
17: **if** $p$ was awake at time $2\Delta$ **then**
18: **for all** $b$ such that $\overline{\mathcal{E}}(b) > \mathcal{M}_1/2$ **do**
19: outputs $\leftarrow$ outputs $\cup \{(b, 1)\}$
20: $\mathcal{V}(b) \leftarrow |\{q \mid p \text{ has received } \langle \text{vote}, b \rangle_q\}|$
21: $\mathcal{M}_3 \leftarrow |\{q \mid p \text{ has received } \langle \text{awake}, 3 \rangle_q\}|$
22: **for all** $b$ such that $\mathcal{V}(b) > \mathcal{M}_3/2$ **do**
23: outputs $\leftarrow$ outputs $\cup \{(b, 0)\}$
24: multi-cast all messages (without *conflict*) received but not yet forwarded

message if no other message conflicts with it (line 24). Also note that a party counts $\langle \text{echo}, b \rangle_q$ into $\overline{\mathcal{E}}(b)$ only if no other message conflicts with it (line 7). This makes sure every counted echo is always forwarded to all other parties, which is important for the time-shifted quorum argument.

## 4.3 Correctness of the Protocol

We prove consistency, integrity and validity of Algorithm 1. Since our GA protocol runs during the execution of our atomic broadcast protocol, when we refer to global/local time, it is the relative time from the beginning of the GA protocol. More specifically, suppose a GA protocol is supposed to start at time $T$; when we say "global/local time $t$", we mean global/local time $T + t$ (in this Section 4.3). Note that $t$ can be negative.

Lemma 4.1 (Consistency). *If an honest party outputs a pair $(b, 1)$, all honest parties awake at their local time $\tau \geq 7\Delta$ output $(b, *)$.*

Proof. Suppose an honest party $p$ outputs a pair $(b, 1)$ at a global time $t$, then $t \geq 7\Delta$ and $p$ must have been awake at its local time $2\Delta$. Let $H$ be the set of honest parties awake at global time $5\Delta$. All parties in $H$ must execute lines 10–15, because their local time must be $[4\Delta, 5\Delta]$ at global time $5\Delta$. Suppose a party $q \in H$ executes lines 9–14 at global time $t' \in [4\Delta, 5\Delta]$.

Let $m_q$ be the number of parties from whom $q$ received $\langle \text{awake}, 1 \rangle$ by global time $t'$, and $m_p$ be the number of parties from whom $p$ received $\langle \text{awake}, 1 \rangle$ by global time $t$. Since $q$ forwards all awake messages received from other parties and $p$ receives them by time $t > t' + \Delta$, we have $m_q \leq m_p$.

Similarly, let $e_p$ be the number of parties from whom $p$ received $\langle \text{echo}, b \rangle$ by its local time $2\Delta$, and $e_q$ be the number of parties from whom $q$ received $\langle \text{echo}, b \rangle$ by global time $t'$. Since $p$ forwards every $\langle \text{echo}, b \rangle$ message that is counted in $\overline{\mathcal{E}}(b)$ by its local time $2\Delta$ and $q$ (awake at $t'$) receives these, we have $e_q \leq e_p$.

Since $p$ outputs $(b, 1)$ at global time $t$, we have $e_p > m_p/2$. Since $e_q \leq e_p$ and $m_p \leq m_q$, we have $e_q > m_q/2$. Thus, all parties in $H$ must have sent $\langle \text{vote}, b \rangle$ along with $\langle \text{awake}, 3 \rangle$ by global time $5\Delta$. Here, some honest parties may wake up after global time $5\Delta$ and send $\langle \text{awake}, 3 \rangle$ until global time $6\Delta$ due to clock offsets. But by the same argument that we made for $H$, they also send $\langle \text{vote}, b \rangle$. Since there are less than $|H|$ faulty parties (at all time), for any $t'' \geq 7\Delta$, any honest party $r$ awake at global time $t''$ observes $\mathcal{V}(b) > \mathcal{M}_3/2$, and hence $r$ outputs $(b, *)$ after its local time reaches $7\Delta$. □

Lemma 4.2 (Integrity). *If no honest party inputs $b$, no honest party outputs $(b, *)$.*

Proof. Let $H$ be the set of honest parties awake at global time $\Delta$. Then, all parties in $H$ must multi-cast $\langle \text{awake}, 1 \rangle$ by global time $\Delta$ because their local time must be $[0, \Delta]$ at global time $\Delta$. Since no honest party sends $\langle \text{echo}, b \rangle$ and there are less than $|H|$ faulty parties (at all time), for any $t \geq 4\Delta$, any honest party awake at global time $t$ must observe $\mathcal{E}(b) < \mathcal{M}_1/2$. Therefore, no honest party sends $\langle \text{vote}, b \rangle$. Let $H'$ be the set of honest parties awake at global time $5\Delta$. All parties in $H'$ must multi-cast $\langle \text{awake}, 3 \rangle$ by time $5\Delta$ because their local time must be $[4\Delta, 5\Delta]$ at global time $5\Delta$. Since no honest party sends $\langle \text{vote}, b \rangle$ and there are less than $|H'|$ faulty parties (at all time), for any $t' \geq 6\Delta$, any honest party awake at time $t'$ must observe $\mathcal{V}(b) < \mathcal{M}_2/2$. Therefore, no honest party outputs $(b, *)$. □

Our protocol achieves the following validity property which is a little more complex than what is defined at the beginning of this section. Recall that $T_s$ is the stabilization time after which the participation is always stable, and a party is said to be *insomniac* during $[t, t']$ if it stays awake at all time during $[t, t']$ (as defined in Section 3).

Lemma 4.3 (Validity). *Suppose the protocol starts at global time $T \geq T_s + 5\Delta$. If every honest party insomniac during global time $[-5\Delta, 2\Delta]$ has the same input $b$, then every honest party insomniac during global time $[2\Delta, 9\Delta]$ outputs $(b, 1)$ and does not output $(b', *)$ for any $b' \neq b$.*

Proof. Let $P$ be the set of insomniac honest parties during global time $[-5\Delta, 2\Delta]$ that have the same input $b$. All parties in $P$ must multi-cast $\langle \text{echo}, b \rangle$ along with $\langle \text{awake}, 1 \rangle$ by global time $\Delta$. Let $Q$ be the set of insomniac honest parties during $[2\Delta, 9\Delta]$, and $\alpha$ be the number of parties that are ever awake sometime during $[0, 2\Delta]$. Due to the $7\Delta$-eventual stable participation assumption, we have $|P| > \alpha/2$. Therefore, all parties in $Q$ must observe $\overline{\mathcal{E}}(b) > \mathcal{M}_1/2$ at their local time $7\Delta$. Hence, every party in $Q$ outputs $(b, 1)$. Moreover, at their local time $4\Delta$, they must observe $\mathcal{E}(b') < \mathcal{M}_1/2$ for any

**Algorithm 2** Atomic broadcast

At the beginning of the execution, initialize variables notarized = $\{(\perp, 0)\}$, lock = 0, candidate = $(\perp, 0)$, $\bar{v} = 0$.
In each view $v$, party $p$ executes the following algorithm at every local time $0 \le \tau \le 37\Delta$ w.r.t view $v$, and enter the next view $v+1$.

1: **if** $0 \le \tau \le \Delta$ **then** do the following once
2: $h \leftarrow$ candidate.value, $b \leftarrow$ new block
3: $(\rho, \pi) \leftarrow \mathsf{VRF}_p.\mathsf{eval}(v)$
4: multi-cast $\langle \text{propose}, b, h, v, \rho, \pi \rangle_p$

5: **if** $2\Delta \le \tau \le 3\Delta$ **then** do the following once
6: Let $L$ be the party from whom $p$ received the largest valid VRF
7: Let $\langle \text{propose}, b, h, v, \rho, \pi \rangle_L$ be the proposal from $L$
8: **if** $\exists u \ge$ lock such that $(h, u) \in$ notarized **then**
9: multi-cast $\langle \text{block}, b, h, v \rangle_p$
10: start $\mathsf{GA}_{v,1}$ with input $H(h||b)$
11: **else**
12: start $\mathsf{GA}_{v,1}$ with input $\perp$

13: **if** $9\Delta \le \tau \le 10\Delta$ **then** do the following once
14: **if** $\mathsf{GA}_{v,1}$ outputs a pair $(h, 1)$ and no $(h', *)$ for $h' \ne h$ **then**
15: start $\mathsf{GA}_{v,2}$ with input $h$
16: **else**
17: start $\mathsf{GA}_{v,2}$ with input $\perp$

18: **for** $i \in \{3, 4, 5\}$ **do**
19: **if** $(7i-5)\Delta \le \tau \le (7i-4)\Delta$ **then** do the following once
20: **if** $\mathsf{GA}_{v,i-1}$ outputs a pair $(h, 1)$ **then**
21: start $\mathsf{GA}_{v,i}$ with input $h$
22: **else**
23: start $\mathsf{GA}_{v,i}$ with input $\perp$

*// The following events may occur after view v has ended*
24: **upon** $\mathsf{GA}_{v,2}$ outputs $(h, *)$ for $v \ge \bar{v}$
25: notarized $\leftarrow$ notarized $\cup \{(h, v)\}$
26: **upon** $\mathsf{GA}_{v,3}$ outputs $(h, *)$ for $v \ge \bar{v}$
27: **if** $v >$ candidate.view **then**
28: candidate $\leftarrow (h, v)$
29: **upon** $\mathsf{GA}_{v,4}$ outputs $(h, *)$ for $v \ge \bar{v}$
30: lock $\leftarrow \max(\text{lock}, v)$
31: **upon** $\mathsf{GA}_{v,5}$ outputs $(h, *)$ for $v \ge \bar{v}$
32: decide a log $\Lambda$ identified by the hash $h$
33: $\bar{v} \leftarrow v$

$b' \ne b$. Hence, no party in $Q$ sends $\langle \text{vote}, b' \rangle$. Let $\alpha'$ be the number of parties ever awake sometime during global time $[2\Delta, 9\Delta]$. Due to $7\Delta$-eventual stable participation, we have $|Q| > \alpha'/2$. Therefore, every honest party awake at its local time $7\Delta$ or later must observe $\mathcal{V}(b') < \mathcal{M}_3/2$, and will not output $(b', *)$. □

## 5 ATOMIC BROADCAST IN SLEEPY MODEL

This section presents an atomic broadcast protocol (Algorithm 2) building on the graded agreement protocol presented in the previous section.

**View-based construction.** Our protocol follows the standard view-by-view paradigm. Each view lasts for $37\Delta$ time. So view $v$ starts at local time $37(v-1)\Delta$ and ends at local time $37v\Delta$. (The first view is view 1.) For ease of exposition, we will use relative time with respect to a view, i.e., local time $\tau$ of view $v$ means local time $37(v-1)\Delta + \tau$. Each view consists of roughly two phases. In the first phase (line 1–4), each party proposes a new block (e.g., a batch of transactions) along with a VRF lottery. In the second phase (line 5–23), a proposal from an elected leader is decided through a series of graded agreement (GA) instances.

**Blocks and chaining.** A set of values (parties' inputs) are batched into a block $b$ (with values inside a block totally ordered). A log is then naturally represented by a list of blocks $\Lambda = [b_1, b_2, ..]$. Each log $\Lambda$ is uniquely identified by a hash $\overline{H}(\Lambda)$ defined as follows. An empty log [] is defined to have hash value $\perp$. The hash of a log $\Lambda' = \Lambda||b$ is defined recursively as $\overline{H}(\Lambda') = H(\overline{H}(\Lambda)||b)$.

**Propose and leader election.** A proposal (line 4) consists of a new block $b$ and a hash $h$ of a log $\Lambda$ (which has been already disseminated in the previous views). This proposes the new log $\Lambda||b$. The proposal also includes a VRF evaluation on the current view number, which acts as a leader election lottery. At the beginning of the second phase (line 6), each party considers the leader of this view to be the party from whom it has received a proposal (of the current view) with the largest VRF. Here, an adversary cannot guess the VRF evaluations of honest parties before they send their proposals. Therefore, an adversary cannot launch a targeted attack on the elected leader (i.e., corrupt or make the leader asleep). Since there are more awake honest parties than faulty parties, with probability at least 1/2, all honest parties recognize the same honest party as the leader.

**Deciding phase.** However, with constant probability, the leader election can be unsuccessful, i.e., honest parties elect different leaders and proposals. The second phase (line 5–23) therefore determines whether the locally elected leader's proposal can be decided or not. A proposal of an elected leader $L$ is processed through a series of five graded agreements denoted $\mathsf{GA}_{v,i}$ for $1 \le i \le 5$. The proposed log, represented by a pair $(b, h)$, has a hash $H(h||b)$, which is input to first GA (i.e., $\mathsf{GA}_{v,1}$ on line 10). After that, output of $\mathsf{GA}_{v,i}$ (for $1 \le i \le 4$) with grade 1 is passed into $\mathsf{GA}_{v,i+1}$ as input (line 14–15, 20–21). Intuitively, the deciding phase is further separated into four sub-phases "notary-candidate-lock-decide", which helps maintain safety and liveness of our protocol. This construction is inspired by the the three phases of HotStuff [34] (sometimes called "key-lock-commit" [2, 3]). The latter three phases have essentially the same role as the three phases of HotStuff. We have another phase called "notary" that is straightforward in classic protocols but becomes non-trivial in the sleepy model. We elaborate on each phase below.

**Notary.** A log (or its hash $h$) is first *notarized* in its proposed view $v$ when $\mathsf{GA}_{v,2}$ outputs its hash (with either grade 0 or 1). This is maintained by a set notarized of pairs of notarized value $h$ and its view $v$ (line 24–25). Intuitively, the notary of a log confirms the uniqueness of the log in the view, i.e., there is no other log notarized in the view, which ensures safety within a view. To this end, an output $h$ of the first $\mathsf{GA}_{v,1}$ is passed to the second $\mathsf{GA}_{v,2}$ only if there is no other output $h' \ne h$ with either grade 0 or 1 (line 14). Recall that our GA definition allows multiple outputs. However, consistency of $\mathsf{GA}_{v,1}$ allows honest parties to detect inconsistent outputs and prevent multiple notaries generated in the same view.

**Candidate.** Each party keeps track of the highest view $v$ such that $\mathsf{GA}_{v,3}$ outputs a hash $h$, and maintains in a pair candidate of

the hash $h$ and the view $v$ (line 26–28). The hash and view can be accessed by candidate.value and candidate.view, respectively.

**Lock.** Each party always *lock* on the highest view $v$ such that $\mathsf{GA}_{v,4}$ outputs a hash $h$ and its corresponding notary for $h$ (line 29–30). Moreover, a proposal $(b, h)$ of a leader is input to $\mathsf{GA}_{v,1}$ only if the party observes a notary $(h, u) \in$ notarized of view $u < v$ higher than the locking view (the safety guard condition on line 8). If an honest party decides a log of hash $h$ in a view $v$, all honest parties will lock on the view $v$ and $h$ is only a notarized value in the view (due to consistency and integrity of GA). By the safety guard condition above, in the next view $v + 1$, only proposals extending $h$ can be decided, which ensures safety across views. Looking back, candidate always pass the safety guard condition. This is because if an honest party lock on a view $v$, all honest parties will update candidate in the view (due to consistency and integrity of GA). Therefore, a proposal from an honest leader will always be decided, which ensures liveness.

**Decide.** Finally, if $\mathsf{GA}_{v,5}$ outputs a hash, the party decides a log identified by the hash (line 31–32). Here, parties can always obtain the corresponding log because all blocks in the decided log are forwarded in their views (line 9).

These variables may be updated after the corresponding view (i.e., view $v$ where $\mathsf{GA}_{v,*}$ trigger updates) has ended. Here, these variables are not useful for lower views than a decided view because these are always updated at the end of the decided view. Therefore, our protocol keeps track of the highest decided view $\bar{v}$ (line 33), and update variables only for view $v \geq \bar{v}$ (line 24,26,29,31).

## 5.1 Correctness of the protocol.

We prove safety and liveness of Algorithm 2.

Lemma 5.1. *For all $i \in \{2, 3, 4, 5\}$ and $v$, if $\mathsf{GA}_{v,i}$ of an honest party outputs $(b, *)$, then for all honest party awake at its local time $\tau \geq (7i - 5)\Delta$ of view $v$, its $\mathsf{GA}_{v,i-1}$ outputs $(b, *)$.*

Proof. If $\mathsf{GA}_{v,i}$ of an honest party outputs $(b, *)$, then by the integrity of GA, at least an honest party (say $p$) must have input $b$ to $\mathsf{GA}_{v,i}$. This implies $\mathsf{GA}_{v,i-1}$ of the party $p$ must have output $(b, 1)$. By the consistency of GA, for all honest party awake at its local time $\tau \geq (7i - 5)\Delta$ of view $v$ (i.e., at least $7\Delta$ after $\mathsf{GA}_{v,i-1}$ starts), its $\mathsf{GA}_{v,i-1}$ outputs $(b, *)$. □

Based on the above lemma, we prove the following lemma that shows (i) every decided log (its hash) is notarized, and (ii) once a log is decided, no conflicting log can be notarized ever after. Here, we say two logs $\Lambda$ and $\Lambda'$ *conflict* with each other if neither is a prefix of the other.

Lemma 5.2. *If an honest party $p$ decides a log $\Lambda$ in view $v$, then (i) $p$ observes $(H(\Lambda), v) \in$ notarized at that time, and (ii) for any $u \geq v$ and log $\Lambda'$ that conflicts with $\Lambda$, no honest party observes $(H(\Lambda'), u) \in$ notarized at any time.*

Proof. If an honest party $p$ decides a log $\Lambda$ of view $v$ at global time $t$, then $\mathsf{GA}_{v,5}$ must have output $(h, *)$ where $h := \overline{H}(\Lambda)$, and $t \geq 37\Delta$ of view $v$. By Lemma 5.1, for all honest parties awake at their local time $\tau \geq 30\Delta$ of view $v$, and hence for all honest parties awake at global time $t$, $\mathsf{GA}_{v,4}$ must have output $(h, *)$. Applying the same logic inductively, for all honest parties awake at global time $t$ (which must include $p$), $\mathsf{GA}_{v,2}$ must have output $(h, *)$. Hence $p$ must have observed $(h, v) \in$ notarized at global time $t$; (i) is proven.

Next we prove (ii) by induction. We first prove the base case of $u = v$. Applying Lemma 5.1 one more time, for all honest parties awake at their local time $\tau \geq 9\Delta$ of view $v$, $\mathsf{GA}_{v,1}$ must have output $(h, *)$, and hence no honest party could have input $h' \neq h$ to $\mathsf{GA}_{v,2}$. Due to the integrity of GA, no honest party outputs $(h', *)$ from $\mathsf{GA}_{v,2}$ and hence no honest party observes $(h', v) \in$ notarized.

We have shown in the proof of (i) that for all honest parties awake at their local time $\tau' \geq 30\Delta$ of view $v$, $\mathsf{GA}_{v,4}$ must have output $(h, *)$. Hence, they must have set lock $\geq v$. Therefore, none of them could have observed $(h', w) \in$ notarized with $w \geq$ lock and $h' \neq h$. Then, no honest party could have input $h'' := \overline{H}(\Lambda')$ for any log conflicting log $\Lambda'$ to $\mathsf{GA}_{v+1,1}$. By the integrity of GA, no honest party outputs $(h'', *)$ from $\mathsf{GA}_{v+1,1}$. By the same logic, no honest party outputs $(h'', *)$ from $\mathsf{GA}_{v+1,2}$. Therefore, no honest party could have observed $(h'', v + 1) \in$ notarized at any time. This completes the inductive step, and proves that for all $u \geq v$, no honest party observes $(\overline{H}(\Lambda'), u) \in$ notarized at any time. □

We can then show the safety of our protocol.

Lemma 5.3 (Safety). *Honest parties do not decide two different values at the same log height.*

Proof. Honest parties decide two different values at the same log height implies they decide two conflicting logs. Suppose for the sake of contradiction that honest parties $p$ and $q$ decide two conflicting logs $\Lambda$ and $\Lambda'$ in view $v$ and $u$, respectively. Without loss of generality, we assume $v \leq u$. By (i) of Lemma 5.2, $(\overline{H}(\Lambda), v) \in$ notarized for the party $p$ and $(\overline{H}(\Lambda'), u) \in$ notarized for the party $q$. However, this contradicts the (ii) of Lemma 5.2. □

Before proving the liveness, we first show the following lemma that honest party can always obtain a log that corresponds to a decided hash (i.e., output of the fifth GA).

Lemma 5.4. *If $\mathsf{GA}_{v,5}$ of an honest party outputs $(h, *)$, then all honest parties awake at their local time $\tau \geq 37\Delta$ of view $v$ observe a log $\Lambda$ such that $\overline{H}(\Lambda) = h$.*

Proof. If $\mathsf{GA}_{v,5}$ of an honest party outputs $(h, *)$, then at least an honest party $p$ must have input $h$ to $\mathsf{GA}_{v,1}$. Then $p$ must have multi-cast $\langle \text{block}, b_k, h_{k-1}, v \rangle$ where $h = H(h_{k-1}||b_k)$, which implies $p$ must have observed $(h_{k-1}, u) \in$ notarized for a view $u < v$ during local time $[2\Delta, 3\Delta]$ of view $v$. Then, at least an honest party $q$ must have input $h_{k-1}$ to $\mathsf{GA}_{u,1}$. Then $q$ must have multi-cast $\langle \text{block}, b_{k-1}, h_{k-2}, u \rangle$ where $H(h_{k-2}||b_{k-1}) = h_{k-1}$, which implies $q$ must have observed $(h_{k-2}, w) \in$ notarized during local time $[2\Delta, 3\Delta]$ of view $u$. Applying this logic repeatedly, there is a log $\Lambda := [b_1, b_2, .., b_k]$ such that for all $i \in [1, k]$, at least an honest party must have multi-cast $\langle \text{block}, b_i, h_{i-1}, * \rangle$ where $h_{i-1} = \overline{H}([b_1, b_2, ..b_{i-1}])$. Therefore, all honest parties awake at their local time $\tau \geq 37\Delta$ of view $v$ observe a log $\Lambda$ identified by the hash $h$. □

Next we show the following lemma which intuitively says that every honest party's proposal passes the safety check on line 8.

Lemma 5.5. *Let $h$ be the* candidate.value *of an honest party $p$ awake at some global time $t \in [0, \Delta]$ of view $v$. Then, any honest party $q$ awake at any global time $t' \in [2\Delta, 3\Delta]$ of view $v$ observes $(h, u) \in$ notarized for some $u \geq$ lock.*

Proof. We consider two cases: $h$ is $\perp$ or not. If $h = \perp$, it is clear that the party $q$ observes lock = 0 (and $(\perp, 0) \in$ notarized from initialization); Otherwise, for some view $w < v$, $\mathsf{GA}_{w,4}$ of $q$ must have output some value by global time $t'$ of view $v$. Then, by Lemma 5.1, $\mathsf{GA}_{w,3}$ of $p$ must have output some value by global time $t$ of view $v$, and $h$ could not have been $\perp$.

If $h \neq \perp$, there exists a view $u < v$ such that $\mathsf{GA}_{u,3}$ of $p$ must have output $(h, *)$ by global time $t$ of view $v$. Then, by Lemma 5.1, $\mathsf{GA}_{u,2}$ of $q$ must have output $(h, *)$ by global time $t'$ of view $v$. Therefore, the party $q$ must have observed $(h, u) \in$ notarized at global time $t'$. Let $w > 0$ be the value of lock that $q$ observes at global time $t'$ (if $w = 0$, the lemma is obvious), then $\mathsf{GA}_{w,4}$ of $q$ must have output $(h', *)$ for a value $h'$ by time $t'$ of view $v$. Then, by Lemma 5.1, $\mathsf{GA}_{w,3}$ of $p$ must have output $(h', *)$ by time $t$ of view $v$. Since $p$ always updates candidate based on the output of $\mathsf{GA}_{*,3}$ of the highest view, $u \geq w$. □

Now we show the liveness of our protocol.

Lemma 5.6 (Liveness). *If an honest party inputs a value $x$, then there exists a time $t$ such that all honest parties awake at global time $t$ decides a log that contains $x$.*

Proof. Let $v$ be a view after the stabilization time $T_s$. Let $P$ be the set of insomniac honest parties during $[0, 2\Delta]$ of view $v$. Then, all parties in $P$ multi-cast their own proposals by global time $\Delta$ of view $v$. Let $\alpha$ be the number of parties awake at some time during $[0, 2\Delta]$. Due to $7\Delta$-eventual stable participation, $|P| > \alpha/2$. Therefore, with probability more than $1/2$, all honest parties awake at their local time $2\Delta$ of view $v$ observe the same honest leader $L$ of view $v$ and its proposal.

Suppose the above good event happens, i.e., an honest leader $L$ is elected at view $v$. Let $\langle \text{propose}, b, \bar{h}, v, \rho, \pi \rangle_L$ be the proposal of the leader $L$. Let $Q$ be the set of insomniac honest parties during $[2\Delta, 4\Delta]$ of view $v$. Then, all parties in $Q$ input $h = H(\bar{h}||b)$ to $\mathsf{GA}_{v,1}$ at their local time $2\Delta$ of view $v$ by Lemma 5.5. Let $S$ be the set of insomniac honest parties during $[4\Delta, 11\Delta]$ of view $v$. By the validity of GA, all parties in $S$ output $(h, 1)$ at their local time $9\Delta$ of view $v$ and do not output $(h', *)$ for any $h' \neq h$. Then, all parties in $S$ input $h$ to $\mathsf{GA}_{v,2}$ at their local time $9\Delta$ of view $v$. Let $\alpha'$ be the number of parties awake at some time during $[9, 11\Delta]$ of view $v$. Due to $7\Delta$-eventual stable participation, $|S| > \alpha'/2$. Applying the validity of GA inductively, for some honest parties, $\mathsf{GA}_{v,5}$ outputs $(h, 1)$. By the consistency of GA, for all honest parties awake at their local time $\tau \leq 37\Delta$ of view $v$, $\mathsf{GA}_{v,5}$ must output $(h, *)$. By Lemma 5.4, they all obtain and decide a log $\Lambda$ such that $\overline{H}(\Lambda) = h$.

Since the above good event happens in each view with probability $> 1/2$, it must happens eventually and repeatedly (except with negligibly small probability). Therefore, the honest party's input $x$ will eventually be included in a decided log. □

Finally, our protocol achieves an expected $O(\Delta)$ latency under stable participation as formalized below.

**Algorithm 3** Atomic broadcast with efficient recovery

At the beginning of the execution, initialize variables notarized = $\{(\perp, 0)\}$, lock = 0, candidate = $(\perp, 0)$, $\bar{v} = 0$, $\underline{v} = 0$.
In each view $v$, party $p$ executes the following algorithm at every local time $0 \leq \tau \leq 44\Delta$ w.r.t view $v$, and enter the next view $v + 1$.

1: *// same as line 1–33 of Algorithm 2 except a sixth GA step is added to keep track of the critical view $\underline{v}$*

⋮

18: **for** $i \in \{3, 4, 5, \mathbf{6}\}$ **do**

⋮

*// update the critical view*
34: **upon** $\mathsf{GA}_{v,6}$ outputs $(h, *)$ for $v \geq \underline{v}$
35: $\quad \underline{v} \leftarrow v$

*// respond to recovery request*
36: **upon** receiving $\langle \text{recover}, u \rangle_q$
37: $\quad \Lambda_q \leftarrow$ the log of view $u$ that $p$ decided
38: $\quad \Lambda_p \leftarrow$ the log of view $\underline{v}$ that $p$ decided
39: $\quad$ **for all** $b \in \Lambda_p \setminus \Lambda_q$ **do**
40: $\quad\quad$ send to $q$ the corresponding $\langle \text{block}, b, *, * \rangle$
41: $\quad$ **for all** $w \geq \underline{v}$ **do**
42: $\quad\quad$ send to $q$ all messages of view $w$ (without any *conflict*).

*// for recovering party $p$*
43: Upon joining the execution, $p$ multi-casts $\langle \text{recover}, \bar{v} \rangle_p$, wait for $\Gamma$, and resume the execution of lines 1–42

Theorem 5.7 (Latency). *If an honest party inputs a value $x$ at global time $t \geq T_s$, then there exists a time $t' = t + O(\Delta)$ such that all honest parties awake at global time $t'$ decide a log that contains $x$ in expectation.*

Proof. The honest party's input $x$ is disseminated to all honest parties by $\Delta$, and will be included in their proposals. At least in $O(1)$ views (in expectation) after that, an honest party's proposal is decided. Since each view takes $O(\Delta)$ time, the input $x$ is incorporated into a decided log within $O(\Delta)$. □

## 6 ATOMIC BROADCAST WITH PRACTICAL RECOVERY

This section augments our atomic broadcast protocol from the previous section with an efficient recovery mechanism, where newly joined parties recover only necessary information of past views. The augmented protocol is described in Algorithm 3.

Our basic protocol (Algorithm 2), described for the original sleepy model, requires each party to send $O(l(nL + \kappa n))$ amount of information to help a recovering party $q$ that slept for $l$ views where $L$ is the size of a block and $\kappa$ is the digital signature size. The $O(nL)$ term is due to sending all proposals in each view, and the $O(\kappa n)$ term is due to GA messages. Our augmented protocol reduces the recovery data size to (expected) $O(lL + nL + \kappa n)$ under stable participation. Intuitively, each party only needs to send messages of recent views (of size $O(nL + \kappa n)$), plus the decided log contents (of size $O(lL)$) during the recovering party's sleep.

Similarly, the augmented protocol saves storage. In our basic protocol, each party must store *all* past messages. In the augmented

protocol, parties only store messages of recent views (plus log contents).

**Identifying the critical view.** Reducing the $O(lnL)$ term to $O(lL)$ is not hard because non-decided proposals in those views are not useful. The recovering party $q$ sends a recovery request with its latest decided view $\bar{v}$ (line 43), and each party just needs to send the decided log contents of view $\bar{v}$ or higher (line 37–40).

Reducing from $O(\kappa ln)$ to $O(\kappa n)$, on the other hand, is challenging since missing GA messages in these views can lead to incorrect updates to state variables (i.e., notary, candidate, lock, and decide), which is the key to the safety and liveness of our atomic broadcast. To this end, each party $p$ needs to identify the oldest view $\underline{v}$ of which it needs to send GA messages to the recovering party $q$. Recall that, in Algorithm 2, only GAs of the highest decided view $\bar{v}$ or later trigger variable updates. Therefore, $q$ only requires messages of view $v \geq \bar{v}$ where $\bar{v}$ is the highest view of which $q$ will decide a log immediately after the recovery is completed. However, $p$ does not know the value of $\bar{v}$ of $q$, which can be different from $\bar{v}$ of $p$ if they receive different messages. To solve this problem, we add another GA step, i.e., $\mathsf{GA}_{v,6}$. If $\mathsf{GA}_{v,6}$ outputs some value, parties update $\underline{v}$ with view $v$ (line 34–35). If an honest party outputs some value from $\mathsf{GA}_{v,6}$, then all honest parties must output the value from $\mathsf{GA}_{v,5}$ (by consistency and integrity of GA). Therefore, the recovering party $q$ always sets $\bar{v}$ to be at least $\underline{v}$ that $p$ observes, and $p$ needs to send only messages of view $\underline{v}$ or higher (line 41–42). Since $\underline{v}$ is updated every constant views (in expectation), parties send messages of only the recent few views (plus log contents) to recovering parties.

**Conflict of messages.** Since faulty parties can send propose and block messages for infinitely many different blocks in the same view, we define *conflict* of these messages to avoid unbounded communication in the recovery phase. Specifically, two different propose (or block) messages of the same view $v$ signed by the same party conflict with each other.

## 6.1 Correctness of the Protocol

We prove safety and liveness of Algorithm 3. We say a party is *up-to-date* with respect to view $v$ at (global or local) time $t$ if the party has received all messages of view $v$ (without conflict) sent (or forward) by time $t - \Delta$ by all honest parties.

**Proof sketch.** The proof has two main parts. First, Lemma 6.1 says, all honest parties awake in view $v$ or $v+1$ are up-to-date w.r.t. view $v$. Then, the time-shifted quorum argument and the security of GA still hold just as in the basic protocol. Next, Lemma 6.3 says that parties can recover all messages of view $\bar{v}$ or higher. This helps prove that parties can correctly update their states for safety (Lemma 6.4) and for liveness (Lemma 6.6).

Lemma 6.1 (Recovery for current view). *Any honest party awake at a global time within a view $v$ or $v+1$ is up-to-date w.r.t view $v$ at that time.*

Proof. Suppose an honest party $p_1$ is awake at global time $t_1$ within a view $v$ or $v+1$. Let $t_1'$ be the last time when $p_1$ became awake. (If $p_1$ was awake all the time, the lemma is obvious.) Then, $p_1$ must have rejoined the execution at global time $t_1' - \Gamma$, and its recover message must have been received by an honest party $p_2$ that is awake at global time $t_2 \leq t_1' - \Gamma + \Delta$. By the end of view $v+1$, $p_2$ (in fact, any party) must have $\underline{v} \leq v$, so by time $t_2$, $p_2$ must have sent $p_1$ all messages of view $v$. These messages will be received by $p_1$ by global time $t_1'$ (since $\Gamma \gg 2\Delta$). Applying this logic repeatedly, we can define a list of parties $[p_1, p_2, .., p_i]$ where the last party $p_i$ stays awake from the beginning of view $v$ to global time $t_i$. Then, $p_i$ must be up-to-date w.r.t view $v$ at global time $t_i$. Moreover, if $p_k$ is up-to-date at global time $t_k$, then $p_{k-1}$ is also up-to-date at global time $t_{k-1}$ (because $p_k$ sends to $p_{k-1}$ all messages of view $v$ received by global time $t_k$). Therefore, by induction, by global time $t_1$, $p_1$ must receive all messages of view $v$ and hence be up-to-date w.r.t view $v$. □

The consistency and integrity of $\mathsf{GA}_{v,*}$ just need minor modifications. They now apply to parties that are up-to-date w.r.t. view $v$. The statement of validity stays the same as in Lemma 4.3.

(1) *Consistency.* If an honest party $p$ **up-to-date w.r.t view** $v$ outputs $(b, 1)$, all honest parties awake and **up-to-date w.r.t view** $v$ at their local time $\tau \geq 7\Delta$ output $(b, *)$.
(2) *Integrity.* If no honest party inputs $b$, then no honest party **up-to-date w.r.t view** $v$ outputs $(b, *)$.

The proofs are almost the same as Lemma 4.1, 4.2, and 4.3, once we argue that parties involved in the proofs (e.g., parties $p$ and $q$ in the time-shifted quorum argument) are up-to-date w.r.t view $v$, due to Lemma 6.1.

Lemma 6.2. *For all $i \in \{2, 3, 4, 5, 6\}$ and $v$, if $\mathsf{GA}_{v,i}$ outputs $(b, *)$ for an honest party up-to-date w.r.t view $v$ at global time $t \geq (7i+2)\Delta$, then for any honest party up-to-date w.r.t view $v$ at its local time $\tau \geq (7i-5)\Delta$ of view $v$, $\mathsf{GA}_{v,i-1}$ outputs $(b, *)$.*

Proof. If $\mathsf{GA}_{v,i}$ of an honest party up-to-date w.r.t view $v$ at global time $t \geq (7i+2)\Delta$ outputs $(b, *)$, then by the integrity of GA, at least an honest party (say $p$) must have input $b$ to $\mathsf{GA}_{v,i}$ at a global time $t'$ within view $v$. This implies $\mathsf{GA}_{v,i-1}$ of the party $p$ must have output $(b, 1)$. Since $p$ has been awake at global time $t'$ within view $v$, it must have been up-to-date w.r.t view $v$ by Lemma 6.1. By the consistency of GA, for any honest party up-to-date w.r.t view $v$ at its local time $\tau \geq (7i-5)\Delta$, its $\mathsf{GA}_{v,i-1}$ outputs $(b, *)$. □

Lemma 6.3 (Recovery across views). *If an honest party $p$ is awake and observes $\bar{v}$ at global time $t$, then $p$ is up-to-date w.r.t any view $v \geq \bar{v}$ at global time $t$.*

Proof. Let $t' \leq t$ be the last time the party $p$ became awake. (If $p$ is awake all the time, the lemma is obvious.) $p$ must have rejoined the execution at global time $t' - \Gamma$ and multi-cast recover, which must have been received by all honest parties awake at global time $t' - \Gamma + \Delta$ and they must have sent all messages of $\underline{v}$ and higher, and all of these messages must have reached $p$ by global time $t'$ (since $\Gamma \gg 2\Delta$). Let $q_1$ be an honest party awake at global time $t_1 = t' - \Gamma + \Delta$, and let $\underline{v}_1$ be the value of $\underline{v}$ it observes. Again, let $t_1' \leq t_1$ be the last time $q_1$ became awake. The party $q_1$'s recover message must have been received by some honest party $q_2$ awake at global time $t_2 = t_1' - \Gamma + \Delta$, who observes $\underline{v}_2$ as the value of $\underline{v}$. This way, we can define a list of parties $Q = [q_1, q_2, ..]$ and their corresponding views $V = [\underline{v}_1, \underline{v}_2, ..]$. Let $u$ be the highest view in $V$. Then, any party $q_i \in Q$ must have been up-to-date w.r.t view

$u$ at global time $t_i$. Therefore, $p$ must be up-to-date w.r.t view $u$ at global time $t$. Let $j$ be the index such that $\underline{v}_j = u$. Then, $\mathsf{GA}_{u,6}$ of $q_j$ must have output $(h, 1)$ for some $h$ by global time $t_j$. By Lemma 6.2, $\mathsf{GA}_{u,5}$ of $p$ must have output $(h, *)$ at global time $t$, and hence $p$ must observe $\bar{v} \geq u$ at global time $t$. Therefore, $p$ is up-to-date w.r.t any view $v \geq \bar{v}$ at global time $t$. □

Lemma 6.4. *If an honest party $p$ decides a log $\Lambda$ in view $v$, then (i) $p$ observes $(\overline{H}(\Lambda), v) \in$ notarized at that time, and (ii) For any $u \geq v$ and log $\Lambda'$ that conflicts with $\Lambda$, no honest party observes $(\overline{H}(\Lambda'), u) \in$ notarized at any time.*

Proof. If an honest party $p$ decides a log $\Lambda$ in view $v$ at global time $t$, then $\mathsf{GA}_{v,5}$ must have output $(h, *)$ where $h := \overline{H}(\Lambda)$, and $t \geq 37\Delta$ of view $v$. Since honest parties decide in view $\bar{v}$ or higher, $p$ must have been up-to-date w.r.t view $v$ at that time by Lemma 6.3. By Lemma 6.2, for all honest parties awake at their local time $30\Delta$ of view $v$ (who must have been up-to-date w.r.t view $v$), $\mathsf{GA}_{v,4}$ must have output $(h, *)$. Applying Lemma 6.2 inductively, for all honest parties up-to-date w.r.t view $v$ at global time $t$ (including $p$), $\mathsf{GA}_{v,2}$ must have output $(h, *)$. Hence $p$ observes $(h, v) \in$ notarized at global time $t$. (i) is proven.

Next we prove (ii). We first prove for $u = v$. Applying Lemma 6.2 inductively, for all honest parties awake at global time $[9\Delta, 10\Delta]$ of view $v$, $\mathsf{GA}_{v,1}$ must have output $(h, *)$, and hence no honest party could have input $h' \neq h$ to $\mathsf{GA}_{v,2}$. Due to the integrity of GA, for any $t' \geq 0$, no honest party up-to-date w.r.t view $v$ at global time $t'$ could have output $(h', *)$. Here, if an honest party adds some pair $(*, v)$ to notarized, the party must be up-to-date w.r.t view $v$ (by Lemma 6.3). Therefore, no honest party observes $(h', v) \in$ notarized at any time; (ii) is proven for $u = v$.

By Lemma 6.2, for all honest parties awake at global time $[2\Delta, 4\Delta]$ of view $v + 1$, $\mathsf{GA}_{v,4}$ must have output $(h, *)$, and hence they must have observed lock $\geq v$. Therefore, none of them could have observed $(h', w) \in$ notarized for $w \geq$ lock and $h' \neq h$. Then, no honest party could have input $h'' := \overline{H}(\Lambda')$ for any log $\Lambda'$ that conflicts with $\Lambda$ to $\mathsf{GA}_{v+1,1}$. By the integrity of GA, no honest party awake at global time $[9\Delta, 11\Delta]$ could have output $(h'', *)$ from $\mathsf{GA}_{v+1,1}$. By the same logic, no honest party up-to-date w.r.t view $v + 1$ could have output $(h'', *)$ from $\mathsf{GA}_{v+1,2}$. Therefore, no honest party observes $(h'', v + 1) \in$ notarized at any time. This completes the inductive step, and proves that for all $u \geq v$, no honest party observes $(\overline{H}(\Lambda'), u) \in$ notarized at any time. □

We can prove safety using Lemma 6.4 similar to the proof of Lemma 5.3 in Section 5.

Lemma 6.5. *If an honest party $p$ observes* candidate.view $= v$ *(or* lock $= v$*), then $p$ is up-to-date w.r.t any view $u \geq v$.*

Proof. Suppose for the sake of contradiction that $p$ is not up-to-date w.r.t view $v$ or higher. Then, $p$ must observes $\bar{v} > v$. Let $w$ be the view $p$ observes as $\bar{v}$. Then, $p$ must have output some value from $\mathsf{GA}_{w,5}$ by then. By Lemma 6.4, $p$ must have output the value from $\mathsf{GA}_{w,4}$ and $\mathsf{GA}_{w,3}$, and updated candidate.view and lock with $w > v$ by then. This contradicts that $p$ observes candidate.view $= v$. □

Lemma 6.6. *Let $h$ be the* candidate.value *of an honest party $p$ awake at global time $t \in [0, \Delta]$ of view $v$. Then, every honest party $q$ awake at global time $t' \in [2\Delta, 3\Delta]$ of view $v$ observes $(h, u) \in$* notarized *for some $u \geq$* lock.

Proof. We consider two cases: $h$ is $\perp$ or not.

**Case 1:** $h = \perp$**.** We will prove that party $q$ observes lock $= 0$ at global time $t'$ (we have $(0, 0) \in$ notarized from initialization). Suppose for the sake of contradiction that $q$ observes lock $\neq 0$ at global time $t'$. Then, there exists a view $u < v$ such that $q$ must have output some value from $\mathsf{GA}_{u,4}$ and updated lock based on the output. At that time, $q$ must have been up-to-date w.r.t view $u$ by Lemma 6.5. Also, by Lemma 6.5, $p$ must have been up-to-date w.r.t view $u$ at global time $t$, because $p$ observes candidate.view $= 0$ (because $h = \perp$). By Lemma 6.2, $p$ must have output some value from $\mathsf{GA}_{u,3}$ by global time $t$, which must have triggered updating candidate. This contradicts with $h = \perp$. Therefore, $q$ observes lock $= 0$ at global time $t'$, and we complete the proof for the first case.

**Case 2:** $h \neq \perp$**.** There exists a view $u < v$ in which $p$ outputs $(h, *)$ from $\mathsf{GA}_{u,3}$ and updates candidate based on this output. By Lemma 6.5, $p$ must have been up-to-date w.r.t view $u$ at that time. Let $u$ be the highest such view.

We can prove that $q$ must have been up-to-date w.r.t view $u$ at global time $t'$. Otherwise, $q$ must have observed $\bar{v} > u$ at global time $t'$ (by Lemma 6.3). This implies $q$ must have output some value from $\mathsf{GA}_{w,5}$ for some $w > u$ (when it is up-to-date w.r.t view $w$). Since $p$ observes candidate.view $= u$, it must have been up-to-date w.r.t view $w > u$ at global time $t$ (by Lemma 6.5). Therefore, $p$ must have output some value from $\mathsf{GA}_{w,3}$ (by Lemma 6.2), and updated candidate based on the output. However, this contradicts with $u$ being the highest such view.

Now, we have that $p$ outputs $(h, *)$ from $\mathsf{GA}_{u,3}$ when it is up-to-date w.r.t view $u$, and $q$ is also up-to-date w.r.t view $u$ at global time $t'$. By Lemma 6.2, $q$ must have output $(h, *)$ from $\mathsf{GA}_{u,2}$ by global time $t'$. Therefore, $q$ must have observed $(h, u) \in$ notarized at global time $t'$.

Finally, we show that $q$ observes $u \geq$ lock at global time $t'$. Let $s$ be the value of lock that $q$ observes at global time $t'$ (if lock $= 0$, the lemma is obvious), then $q$ must have output some value from $\mathsf{GA}_{s,4}$ (when it is up-to-date w.r.t. view $s$). By the same logic as above, $p$ must have been up-to-date w.r.t view $s$ at global time $t$. By Lemma 6.2, $p$ must have output some value from $\mathsf{GA}_{s,3}$ by global time $t$. Since $p$ always updates candidate based on the output of $\mathsf{GA}_{*,3}$ of the highest view, $u \geq s$. □

Finally, we can prove liveness based on Lemma 6.6 as in the basic protocol (Lemma 5.6).

## 7 EXPERIMENTAL EVALUATION

To demonstrate the low latency of our protocol, we implemented and evaluated our basic protocol and the longest-chain protocol in [31] under varying participation level. Our experiment was conducted with 100 parties with each party executed on an Amazon EC2 *t2.large* instance. We executed our protocol and the longest-chain protocol under the same schedule of participation level (shown in the bottom of Figure 2) generated as follows.

(1) **Stable participation.** At the beginning of the execution, 50 parties are awake, and until 1110 seconds (30 views for our

protocol), it randomly increases or decreases by one party every second.

(2) **Unstable participation.** During the next 1110 seconds (until 2220 seconds), the participation level is selected independently randomly from 1 to 100 parties every second.
(3) **High participation level.** The third period (next 1110 seconds) starts with 66 parties (two thirds) and randomly increases or decreases by one party every second, but never drops below 66 parties.
(4) **Low participation level.** Finally, the last period starts with 33 parties (1/3), and randomly increases or decreases by one party every second, but never exceeds 33 parties.

We set the synchrony bound $\Delta = 1$ second. For the longest-chain protocol, we used the block generation rate for each party $\lambda = 1/3700$ per seconds so that a block is generated at about the same interval as our protocol when all parties are awake. This provides the fault tolerance of $f/n \approx 0.49$ [31]. For the confirmation length, we use $k = 10$. This is quite low a security level; the probability of safety violation is at least 0.001 if an adversary controls more than 20% stake and simply launches a private mining attack [28].

Figure 2 shows the latency and log length over time. The log length at each time is the number of blocks decided so far. The latency at each time is the time it takes for the next proposed block to be decided. In other words, if $b$ is the first block proposed after time $t$ and $b$ is decided at time $t'$, then we plot the latency at time $t$ to be $t' - t$. The latency of the longest-chain protocol after 3097 seconds is not reported because after 3097 seconds takes, it takes too long to produce a block, and no block gets decided when we terminate the experiment. We give some analysis on the experimental results below.

**Dependency on the security parameter.** First, the latency of the longest-chain protocol is always longer than that of our protocol (labeled as Constant Sleepy), due to the dependence on the security parameter $\kappa$ in the longest-chain protocol: a block is decided only after $k = \Omega(\kappa)$ subsequent blocks are proposed. As a block is proposed at most every 37 seconds in expectation, the latency is at least a few hundred seconds (the best case is 239 seconds). This is why it took 750 seconds for the longest-chain protocol to decide its first block, compared to 74 seconds (2 views) in our protocol. The confirmation length $k$ must be much larger to get a higher security level, i.e., a smaller probability of safety violation, and the latency of the longest-chain protocol will be much longer.

**Dependency on the participation level.** The log in our protocol grows consistently in our protocol (except during the unstable participation period). In sharp contrast, in the longest-chain protocol, the growth speed of log length heavily depends on the participation level (due to the $\gamma$ term in the asymptotic latency). Remarkably, the log of the longest-chain protocol grew by 25 blocks in the high participation period (2220–3330 seconds) but by only 5 blocks in the low participation period (3330–4440 seconds). The average participation levels of these two periods are 87 and 27, respectively. For the same reason, the latency of the longest-chain protocol is at the minimum of 239 seconds around the beginning of the high participation period and increases as it gets closer to the low participation period. As mentioned, we could not report latency of the longest-chain protocol after 3097 seconds because it is taking too long to produce blocks during low participation, which is in itself a demonstration of its poor latency.

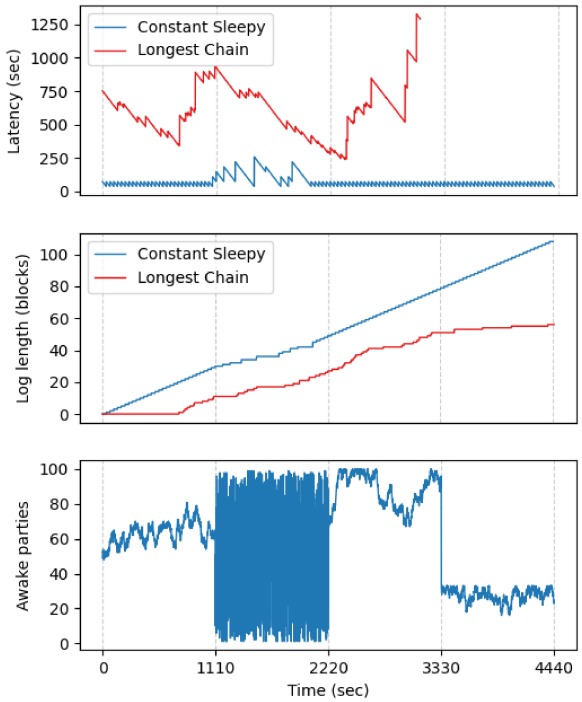


**Figure 2: Experimental result. The bottom figure shows the participation level over time, and the top and middle figures show the latency and log length, respectively, of both our protocol and the longest-chain protocol in [31].**

**Unstable participation period.** As mentioned, our protocol does not guarantee progress when the participation level fluctuates wildly. We indeed observe that during the unstable participation period (1110–2220 seconds), 17 out of 30 views failed to decide blocks. Nonetheless, our protocol decided blocks in some of these views. This is because the stability condition (Definition 3.1) is sufficient but not necessary; it may be further relaxed to weaker (but less clean, hence not adopted) forms. For example, our protocol can decide a block in view $v$ if a majority of awake parties at the beginning of $\mathsf{GA}_{v,i}$ are also awake at time $2\Delta$ of $\mathsf{GA}_{v,i-1}$ for all $1 < i \leq 5$. As we randomly generated the schedule of participation level, our protocol decided blocks in some lucky views that satisfy this condition. Our protocol would not make progress at all if an adversary carefully controls the participation level. In comparison, the longest-chain protocol grows its log length (16 blocks) almost proportionally to the average participation level (49 parties).

## 8 DISCUSSION

**Communication complexity.** Our protocols costs (excluding recovery cost) $O((\kappa\bar{n}^2 + L\bar{n})N)$ bits of communication in total per block, where $\bar{n} = \max_{t\geq 0}(n_t)$ is the maximum number of awake parties throughout the execution, $\kappa$ is the signature length, and $L$ is the size of a block. The $\kappa\bar{n}^2N$ term is due to nodes forwarding every echo that includes a hash of block, and the $L\bar{n}N$ term is due to each node sending a block once before the start of the first GA (line 9 of Algorithm 2). On the other hand, the longest-chain protocol (also excluding recovery cost) in [31] costs $O((\kappa\bar{n} + L\bar{n})N)$ bits of communication per block. Thus, our protocol's communication cost is higher than that of longest-chain protocol if the block size $L$ is small, but becomes comparable when the block size $L$ is big.

**On the necessity of randomization.** Our protocol uses randomized leader election. Randomization is necessary for low latency. It is well known that $\Omega(f)$ round is necessary for deterministic protocols [15], which applies to the sleepy model (since sleepy is strictly harder than the classic model).

Our protocol as well as all existing protocols in the sleepy model only achieve *almost-surely termination*, i.e., the probability of termination approaches 1 as the protocol keeps running but never becomes 1. This is inherent to leader-based protocols in the sleepy model, because the adversary always has a (decreasing but positive) chance to guess the leader and make the leader go to sleep. It is an interesting open question whether this is inherent or avoidable by leaderless protocols.

## 9 CONCLUSION AND FUTURE DIRECTIONS

We present a BFT atomic broadcast protocol that simultaneously supports dynamic participation and achieves expected $O(\Delta)$ latency. Our protocol follows the quorum-based design by extending the classic static quorums to dynamic quorums. To restore the transferability of quorum certificate, we introduce a new technique called time-shifted quorum. We also present an efficient recovery process for rejoining nodes.

Our protocol makes progress only during periods of stable participation. It is an interesting open question whether it is possible to make progress even during wildly fluctuating periods while achieving $O(\Delta)$ latency. A possible solution may be to adopt the multi-chain paradigm [5, 18, 25] without longest-chain protocols. The multi-chain paradigm takes a different approach from ours to remove the dependency on $\kappa$. It is interesting to see if we can further remove from it the dependence on $\gamma$. The remaining challenge is that existing multi-chain protocols either require proof-of-work [5, 25] or still depend on $\kappa$ to decide between conflicting transactions [5, 18].

## ACKNOWLEDGMENTS

We thank our shepherd Qiang Tang and the anonymous reviewers at ACM CCS 2022 for their helpful feedback. We also thank Joachim Neu, Ertem Nusret Tas, and David Tse, for valuable discussions. We also thank Keisuke Hasegawa and Masashi Sato for helping us with the experiment. 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, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Reaching consensus for asynchronous distributed key generation. *arXiv preprint arXiv:2102.09041* (2021).
[3] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically optimal validated asynchronous byzantine agreement. In *ACM Symposium on Principles of Distributed Computing (PODC)*. 337–346.
[4] 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.
[5] 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.
[6] Ziv Bar-Joseph, Idit Keidar, and Nancy Lynch. 2002. Early-delivery dynamic atomic broadcast. In *International Symposium on Distributed Computing (DISC)*. Springer, 1–16.
[7] Kenneth P Birman and Robbert Van Renesse. 1993. *Reliable distributed computing with the Isis toolkit*. IEEE Computer Society Press.
[8] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. 2011. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media.
[9] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference (CRYPTO)*. Springer, 524–541.
[10] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *3rd Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, 173–186.
[11] Jing Chen and Silvio Micali. 2016. Algorand. *arXiv preprint arXiv:1607.01341* (2016).
[12] 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.
[13] 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.
[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] Danny Dolev and H. Raymond Strong. 1983. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput.* 12, 4 (1983), 656–666.
[16] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the presence of partial synchrony. *J. ACM* 35, 2 (1988), 288–323.
[17] Paul Feldman and Silvio Micali. 1988. Optimal algorithms for Byzantine agreement. In *20th Annual ACM Symposium on Theory of Computing (STOC)*. 148–161.
[18] 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).
[19] 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.
[20] Vipul Goyal, Hanjun Li, and Justin Raizes. 2021. Instant Block Confirmation in the Sleepy Model. In *Financial Cryptography and Data Security (FC)*.
[21] Yue Guo, Rafael Pass, and Elaine Shi. 2019. Synchronous, with a chance of partition tolerance. In *Annual International Cryptology Conference (CRYPTO)*. Springer, 499–529.
[22] Jonathan Katz and Chiu-Yuen Koo. 2009. On expected constant-round protocols for byzantine agreement. *J. Comput. System Sci.* 75, 2 (2009), 91–112.
[23] Pankaj Khanchandani and Roger Wattenhofer. 2021. Byzantine Agreement with Unknown Participants and Failures. *arXiv preprint arXiv:2102.10442* (2021).
[24] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* 4, 3 (1982), 382–401.
[25] Songze Li and David Tse. 2020. TaiJi: Longest Chain Availability with BFT Fast Confirmation. *arXiv preprint arXiv:2011.11097* (2020).
[26] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 31–42.
[27] Atsuki Momose and Ling Ren. 2021. Optimal Communication Complexity of Authenticated Byzantine Agreement. In *International Symposium on Distributed Computing (DISC)*.
[28] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[29] Joachim Neu, Ertem Nusret Tas, and David Tse. 2021. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 446–465.

[30] 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.

[31] 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.

[32] 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.

[33] 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).

[34] 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.