On the Communication Complexity of Sleepy Consensus

Qiang Tang
The University of Sydney
qiang.tang@sydney.edu.au

Yuchen Ye
The University of Sydney
yuchenye19@gmail.com

Abstract

Sleepy consensus allows parties to join and leave execution arbitrarily, which is a fundamental requirement for large-scale distributed systems. Classic longest-chain protocols, such as Bitcoin and its variants, achieve consensus under this model but suffer from inherent long latency. In contrast, recent protocols that build upon the classic view-based BFT paradigm can achieve constant expected latency and short best-case latency under optimal resilience, but they often incur high communication cost. We observe that the high communication overhead stems from the *time-shifted quorums*, a technique that makes quorum certificates transferable under dynamic participation. The technique relies on extensive message forwarding to reconcile parties' inconsistent local views, and thus incurs a cubic communication cost unavoidably.

In this work, we tackle the problem by proposing a novel way to transfer certificates. Building on this, we construct a Byzantine Agreement (BA) protocol secure against the state-of-the-art *growing adversary* model. Our BA protocol achieves optimal resilience, constant expected round complexity, and an expected communication complexity of $O(nNL + nN\kappa + nN \log N)$, where n is the maximum number of awake parties throughout the execution, N is the total number of eligible parties, L is the input length, and κ is the security parameter. We also present an efficient recovery mechanism for our BA, incurring only $O(N\kappa + nL)$ bits per recovering party. Then we extend our BA to an Atomic Broadcast (ABC) protocol that achieves optimal resilience, constant expected latency, and an expected amortized communication complexity of $O(nNL + nN\kappa + nN \log N)$ per input value. The recovery mechanism for our ABC incurs $O(N\kappa + n\ell L + n\ell\kappa)$ bits per recovering party, where ℓ is the number of views that the party has slept for. Last but not least, we establish communication lower bounds of $\Omega(N^2 L)$ for sleepy BA and ABC. The result shows that our BA and ABC are communication-optimal when L is sufficiently large (i.e., when $L = \Omega(\kappa + \log N)$), and highlights a fundamental limitation of communication efficiency in the sleepy model.

1 Introduction

Byzantine fault-tolerant (BFT) consensus [LSP82] is one of the most fundamental problems in distributed computing and cryptography. It allows a set of parties to reach agreement in the presence of arbitrary failures. Classic BFT protocols have long operated under a static and known participation model, assuming a fixed, mutually known set of parties that stay active throughout the execution [CL99, PSL80, DS83, BT85]. The advent of Bitcoin [Nak08] showed that consensus is possible under dynamic participation, where parties can join and leave the execution arbitrarily and without prior notice. This setting was later formalized by Pass and Shi [PS17] in the *sleepy model*.

Inspired by Bitcoin [Nak08], early protocols in the sleepy model mainly adopted the longest-chain paradigm [PS17, DPS19, BGK+18]. While achieving dynamic participation, these longest-chain protocols suffer from inherently long latency. Blocks can only be decided after a sufficient number of subsequent blocks are generated, tying the latency heavily to the security parameter κ and the actual participation level γ [Ren19, PSS17]. Follow-up works [BKT+19, FGKR18, LT20, DNTT24, GLR21] have sought to reduce this latency, but they typically only manage to eliminate the dependency on either κ or γ, not both.

To overcome this latency bottleneck, Momose and Ren [MR22] revisited the classical view-based BFT paradigm, and adapted it to the sleepy model. Their protocol achieves constant expected latency, but requires periods of stable participation to guarantee liveness. Furthermore, the original sleepy model assumes that a newly joined party can receive all messages sent to it during its sleep, once it (re)joins the execution. To remove this impractical message delivery assumption, Momose and Ren proposed the *recovery model* where a party misses messages sent to it during its sleep and recovers its state by querying the awake parties when it (re)joins. Building on Momose and Ren's foundation, Malkhi, Momose, and Ren [MMR23] achieved consensus under the *growing adversary* model, where the number of byzantine parties can grow with the participation level. Subsequent works [MMR23, ET25, DSTZ25, MMR22, SHT+25] have further studied how to eliminate the stable participation assumptions, improve expected latency and censorship resilience, etc.

Despite these significant improvements in latency and security, existing view-based sleepy BFT protocols continue to suffer from high communication costs, leaving an important bottleneck in efficiency and scalability that has not been systematically studied. We observe that recent protocols heavily rely on the technique of *time-shifted quorum* [MR22] to make quorum certificates transferable under dynamic participation. At a high level, time-shifted quorums reconcile parties' inconsistent local views (caused by different perceived participation levels across time) by letting parties forward every received message throughout the execution, which incurs a *cubic* communication cost unavoidably. In contrast, classical BFT protocols under the static participation model can achieve consensus with quadratic communication complexity [KK06, ACD+19, AMN+20, ADD+19]. This gap raises a natural question:

Is the high communication cost inherent in sleepy consensus, or can we achieve the communication efficiency of classical BFT protocols while maintaining optimal resilience and constant latency?

In this work, we answer these questions. We tackle the communication bottleneck via a new way of transferring quorum certificates. By eliminating the expensive message forwarding, we reduce communication while preserving optimal resilience and constant latency. Nevertheless, we also identify a fundamental gap between sleepy and classic consensus. We prove that certain overheads are unavoidable due to the uncertainty of dynamic participation, thereby establishing new communication lower bounds for sleepy consensus. Specifically, we make the following contributions:

- **Byzantine Agreement:** We construct a communication-efficient sleepy BA protocol secure against the growing adversary model. Our protocol achieves optimal resilience and expected constant round complexity, with expected communication complexity $O(nNL+nN\kappa+nN\log N)$, where n is the maximum number of awake parties throughout the execution, N is the total number of parties, L is the input length, and κ is the security parameter (denoting the signature size).
- **Recovery Mechanism for Byzantine Agreement:** We introduce an efficient recovery mechanism that allows our BA protocol to operate in the recovery model. The recovery mechanism

incurs only $O(N\kappa + nL)$ bits per recovering party.

- **Atomic Broadcast:** We extend our BA protocol to ABC. Our ABC achieves optimal resilience, expected constant latency, and an expected amortized communication complexity of $O(nNL + nN\kappa + nN \log N)$ per input value. In the recovery model, the recovery mechanism incurs $O(N\kappa + n\ell L + n\ell\kappa)$ bits per recovering party, where ℓ is the number of views that the party has slept for.
- **Communication Lower Bounds:** We establish new communication lower bounds of $\Omega(N^2 L)$ for BA and ABC in both the sleepy and recovery models with adaptive sleepiness.[1] This contrasts with classical BFT protocols, which can achieve an optimal communication complexity of $O(NL + N^2\kappa)$, and highlights a fundamental limitation of communication efficiency in the sleepy model. Notably, we prove that our new lower bounds hold even in the absence of byzantine corruptions. The result shows that our BA and ABC protocols are communication-optimal when the input length is sufficiently large (i.e., when $L = \Omega(\kappa + \log N)$).

We provide a comprehensive comparison of our BA, ABC, and recovery mechanism with existing literature in Tables 1, 2, and 3. Detailed discussion of related work is in Section 9.

2 Model and Definitions

This paper studies the Byzantine Agreement and Atomic Broadcast problems under the model of [MMR23], which extends the original sleepy model [PS17]. We consider a system of N parties communicating over a synchronous network with a known maximum delay Δ.[2] Parties have access to a perfectly synchronous clock.[3] We consider an adaptive adversary capable of corrupting parties at any time throughout the execution. Corrupted parties are byzantine and behave arbitrarily. Non-byzantine parties are honest and execute the protocol faithfully.

Sleepy Model. In the *sleepy model*, parties are either *awake* (executing the protocol) or *asleep* (taking no actions).[4] The adversary can change each party's status arbitrarily without prior notice (i.e., adaptive sleepiness). For message delivery, we assume that if a party p is awake at time t, then it has received all messages sent to it by honest parties by time $t - \Delta$.

Recovery Model. In the *recovery model* [MR22], it is not guaranteed that an honest party that newly joins at time t has received all messages sent to it by honest parties by time $t - \Delta$. The newly joined party instead executes a recovery mechanism, in which it queries the awake parties about the correct history. The party is counted as neither awake nor asleep, but *recovering*, until the recovery mechanism completes [MR22, MMR23].

Adversary. The original sleepy model [PS17] assumes a static bound on the number of byzantine parties relative to the minimum participation level throughout the execution. In this work, we adopt the more permissive *growing adversary* model [MMR23], where the number of byzantine parties is bounded only by the current participation level. More specifically, let f_t and n_t denote the number

[1] For the lower-bound result, the adversary can always wake up all the eligible parties, in which case $n = N$, yielding the stated $\Omega(N^2 L)$ bound. For the upper-bound analysis, we keep n explicit to state the communication complexity more precisely.

[2] Synchrony and a known bound Δ are necessary for achieving consensus under dynamic participation [PS17].

[3] The assumption of a perfectly synchronous clock can be relaxed to bounded clock skew via round transformation [MR22].

[4] It is necessary to assume that a party enters the sleep state knowingly and takes no actions while asleep [MR22].

Table 1: Comparison of Sleepy Byzantine Agreement protocols that is $\frac{1}{2}$-*optimal resilient*, can tolerate *adaptive sleepiness*, and has *constant latency*. Resil. refers to the maximum fraction of corrupted parties that the protocol can tolerate, Corr. refers to the corruption type (either static or adaptive), Grow. refers to growing adversary, $\mathrm{CC}_{\mathrm{BA}}$ refers to the communication complexity, RC refers to the round complexity, Recov. refers to the availability of recovery mechanism, S.P. refers to the requirement of stable participation, and D-Sync. refers to deterministic synchronization (defined in Section 2.1). n is the maximum number of awake parties throughout the execution, N is the total number of parties, L is the input size, and κ is the signature size.

Paper	**Resil.**	**Corr.**	**Grow.**	$\mathbf{CC_{BA}}$	**RC**	**Recov.**	**No S.P.**	**D-Sync.**
[MMR23]*	1/2	adaptive	✓	$O(n^2NL + n^2N\kappa)^\dagger$	–	✓	✓	×
[ET25]	1/2	static	×	$O(n^2NL + n^2N\kappa)$	$O(1)$	×	✓	✓
This paper	1/2	adaptive	✓	$O(nNL + nN\kappa + nN\log N)^\ddagger$	$O(1)$	✓	✓	✓

Table 2: Comparison of Sleepy Atomic Broadcast protocols that is $\frac{1}{2}$-*optimal resilient*, can tolerate *adaptive sleepiness*, and has *constant latency*, where $\mathrm{CC}_{\mathrm{ABC}}$ refers to the communication complexity per input value, and $\mathrm{L}_{\mathrm{Best}}$ and $\mathrm{L}_{\mathrm{Exp}}$ refer to the latency in the best and expected cases respectively.

Paper	**Resil.**	**Corr.**	**Grow.**	$\mathbf{CC_{ABC}}$	$\mathbf{L_{Best}}$	$\mathbf{L_{Exp}}$	**Recov.**	**No S.P.**	**D-Sync.**
[MR22]	1/2	adaptive	×	$O(nNL + n^2N\kappa)$	16Δ	$O(\Delta)$	✓	×	×
[MMR23]	1/2	adaptive	✓	$O(n^2NL + n^2N\kappa)^\dagger$	4Δ	$O(\Delta)$	✓	✓	×
[DSTZ25]	1/2	adaptive	✓	$O(n^2NL + n^2N\kappa)$	6Δ	$O(\Delta)$	×	×	×
[ET25]	1/2	static	×	$O(n^3NL + n^3N\kappa + n^2N^2)^\S$	$5\Delta^\P$	$O(\Delta)$	×	✓	✓
This paper	1/2	adaptive	✓	$O(nNL + nN\kappa + nN\log N)^\ddagger$	5Δ	$O(\Delta)$	✓	✓	✓

Table 3: Comparison of Atomic Broadcast protocols that is $\frac{1}{2}$-*optimal resilient*, can tolerate *adaptive sleepiness*, and has *constant latency* in the recovery model, where $\mathrm{CC}_{\mathrm{Recov}}$ and $\mathrm{L}_{\mathrm{Recov}}$ refer to the total number of bits and rounds required for a party to recover, and ℓ is the number of views that the recovering party has slept for.

Paper	**Resil.**	**Corr.**	**Grow.**	$\mathbf{CC_{ABC}}$	$\mathbf{CC_{Recov}}$	$\mathbf{L_{Recov}}$
[MR22]	1/2	adaptive	×	$O(nNL + n^2N\kappa)$	$O(N\kappa + n\ell L + n^2L + n^2\kappa)$	2Δ
[MMR23]	1/2	adaptive	✓	$O(n^2NL + n^2N\kappa)^\dagger$	$O(N\kappa + n\ell L + n^2L + n^2\kappa)$	2Δ
This paper	1/2	adaptive	✓	$O(nNL + nN\kappa + nN\log N)^\ddagger$	$O(N\kappa + n\ell L + n\ell\kappa)$	2Δ

∗ Since MMR [MMR23] constructed an ABC protocol instead of BA, we compare our BA with a single view of their ABC protocol here.

† MMR [MMR23] mentioned that the communication complexity can be reduced to $O(nNL+n^2N\kappa)$ by reaching consensus on hashes of blocks and transmitting the blocks separately, but further details are not provided.

‡ Our communication complexity can be reduced to $O(nNL + nN\kappa + nN\log n)$ by adopting DTS schemes that have constant-size aggregated signatures, such as [GJM+24] and [BGJ+23]. These schemes, however, often require additional assumptions, e.g., [GJM+24] is only proven secure in the Algebraic Group Model (AGM), and [BGJ+23] relies on SNARKs. The $\log n$ factor is due to multicasting the group sizes.

§ ET [ET25] constructed ABC using aggregate signatures with bitmaps of size $O(\kappa) + N$. It states that when $N = O(\kappa)$, the signature size becomes $O(\kappa)$, and the communication complexity reduces to $O(n^3NL + n^3N\kappa)$.

¶ In ET [ET25], ABC starts DACS at $t = 0$, DACS starts BA at $t = \Delta$, and BA starts CGA at $t = 0$. Since CGA takes 4 rounds, the overall latency is 5Δ in best cases.

of byzantine parties and awake parties at time t respectively. We assume that $f_t < n_t/2$ for every time t. A halted honest party is still counted towards the awake parties.

Our protocol tolerates *backward simulation* with $T_b = O(\Delta)$, meaning the number of byzantine parties can grow dynamically with the participation level, subject to a short time lag of $T_b = O(\Delta)$. In this paper, we set $T_b = 4\Delta$, which is shorter than the 11Δ lag in [MMR23]. Notably, as pointed out by [MMR23], sleepy consensus protocols cannot tolerate *forward simulation*, meaning that an awake byzantine party cannot go to sleep. Otherwise, it may leak its private key to other byzantine parties, allowing the adversary to violate the honest majority assumption.

Cryptographic Assumptions. We assume a public-key infrastructure (PKI). A message m signed by node p is denoted as $\langle m \rangle_p$. We assume Verifiable Random Functions (VRF). $\mathsf{VRF.Eval}$ outputs a deterministic pseudorandom value ρ and a proof π using a node's secret key sk and any input x. $\mathsf{VRF.Vfy}$ verifies whether ρ is a correct evaluation on x using the node's public key pk and proof π.

2.1 Preliminary and Definitions

Dynamic Threshold Signature (DTS). Let nodes $\{P_1, P_2, .., P_n\}$ be a set of n signers. A DTS scheme [SES+16, SS94, QT23, GJM+24, BGJ+23] supports efficient flexible signature aggregation, without the need to fix the aggregation threshold during system setup like conventional threshold signature. It consists of the following algorithms:

- $\mathsf{DTS.Setup}(1^\lambda)$. It takes as input the security parameter λ. It generates the system public parameters pp. Each node P_i obtains $(pk_1, .., pk_n, sk_i)$, where pk_j is a public key share of P_j and sk_i is a secret key share of P_i.
- $\mathsf{DTS.ParSign}(\mathsf{msg}, sk_i) \to \sigma_i$. It generates a partial signature σ_i for a message msg using the secret key share sk_i of P_i.
- $\mathsf{DTS.ParVfy}(\mathsf{msg}, pk_i, \sigma_i) \to 0/1$. It verifies whether a partial signature σ_i is valid for a message msg w.r.t. pk_i.
- $\mathsf{DTS.Combine}(\mathsf{msg}, \{(j, \sigma_j)\}_{j \in S}) \to (t, \Sigma)$, where $|S| = t$. It combines a total of t partial signatures (j, σ_j) for msg, and generates a final signature Σ.
- $\mathsf{DTS.Vfy}(\mathsf{msg}, t, \Sigma) \to 0/1$. It verifies whether the final signature Σ is valid for a message msg under the threshold t.

A DTS scheme is correct, if the following conditions are satisfied:

- A partial signature generated by an honest node is always valid. Formally, let sk_i and pk_i be the secret key share and the public key share of an honest node P_i. For any message msg,

$$\Pr[\mathsf{DTS.ParVfy}(\mathsf{msg}, pk_i, \mathsf{DTS.ParSign}(\mathsf{msg}, sk_i)) = 1] = 1$$

- Combining any number of valid partial signatures for the same message yields a valid final signature for that message. Formally, for any set $\{(j, \sigma_j)\}_{j \in S}$ such that $\mathsf{DTS.ParVfy}(\mathsf{msg}, pk_j, \sigma_j) = 1$ for every $j \in S$,

$$\Pr[\mathsf{DTS.Vfy}(\mathsf{msg}, \mathsf{DTS.Combine}(\mathsf{msg}, \{(j, \sigma_j)\}_{j \in S})) = 1] = 1$$

The unforgeability of DTS can be captured by the following security game. Consider an adversary $\mathcal{A}$ with bounded computational power, who is allowed to corrupt up to $t-1$ nodes. When $\mathcal{A}$ corrupts an honest node P_i, it gains access to the corresponding secret key share sk_i. Additionally, $\mathcal{A}$ can query a signing oracle. Given an input (msg, sk_i), the oracle returns the partial signature σ_i on msg. The objective of $\mathcal{A}$ is to generate a valid dynamic threshold signature Σ^* for a message msg^*, while ensuring that the number of corrupted nodes $\mathcal{C}$ plus the number of queried partial signatures for msg^*, denoted $\mathcal{Q}_{\mathsf{msg}^*}$, remains strictly less than the threshold t, i.e., $|\mathcal{C}| + |\mathcal{Q}_{\mathsf{msg}^*}| < t$. A DTS scheme is considered unforgeable if, for any such adversary $\mathcal{A}$, the probability of successfully generating a valid signature-message pair $(\Sigma^*, \mathsf{msg}^*)$ is negligible. we assume that the DTS scheme is correct and unforgeable.

In this paper, we adopt predicate aggregate signature (PAS) [QT23] as our DTS scheme, as it supports transparent setup. Since we only combine partial signatures for the same value, the aggregated signature size is $O(\log N)$ [QT23].

Weak Graded Agreement (WGA). In WGA (a weaker version of graded agreement [MR22, MMR23]), each party that is awake at the start of the protocol takes as input an L-bit $v \in \{0,1\}^L$. Each party outputs (x, g), a value x and a grade bit $g \in \{0, 1\}$. WGA satisfies:

- **Graded Delivery.** If an honest party outputs $(x, 1)$, then every honest party that produces an output outputs $(x, *)$.
- **Validity.** If all the honest parties that are awake at the start of the protocol input x, then every honest party that produces an output outputs $(x, 1)$.
- **Termination.** There exists a time $T \leq T_b$ such that every honest party that is awake at any time $t \in [T, T_b]$ outputs. Moreover, every honest party that is awake at any time $t \geq T$ halts.

Byzantine Agreement (BA). In sleepy BA [ET25], each party that is awake at the start of the protocol takes as input an L-bit $v \in \{0,1\}^L$. Each party outputs a value x, subject to the following constraints:

- **Consistency.** If two honest parties output x and x', then $x = x'$.
- **Validity.** If all the honest parties that are awake at the start of the protocol input x, then every honest party that produces an output outputs x.
- **Termination.** There exists a time T such that every honest party that is awake at any time $t \geq T$ outputs and halts.

BA in the recovery model. This definition matches Byzantine Agreement, except for the termination property:

- **Termination.** There exists a time T such that every honest party that is awake at any time $t \geq T$ outputs.

Atomic Broadcast (ABC). In ABC [CASD95, ET25], each party receives values from the environment and outputs a growing sequence of values $[x_0, x_1, x_2, \dots]$ called a *log*, subject to the following constraints:

- **Safety.** If two honest parties output logs $[x_0, .., x_j]$ and $[x'_0, .., x'_{j'}]$, then $x_i = x'_i$ for all $i \leq \min\{j, j'\}$.
- **Liveness.** If all the awake honest parties receive a value x from the environment at time t, then there exists a time $T \geq t$ such that every log outputted by awake honest parties at any time after T contains x.

Deterministic Synchronization. Once an honest node outputs a value, all the other honest nodes also produce outputs within constant time [ET25].

Stable Participation. A set of parties is assumed to be awake during a certain period of time [MR22, DSTZ25].

3 Technical Overview

In this section, we begin by identifying the fundamental challenges in achieving certificate transferability in the sleepy model and reviewing the existing design method, the time-shifted quorum [MR22]. To address the high communication cost caused by message forwarding in time-shifted quorums, we introduce WGA, a primitive that enables certificate transfer without message forwarding. Then we transition from WGA to BA. We use a framework similar to Efron-Tas [ET25] as a starting point, and tackle the new challenges raised by the growing adversary model, as Efron-Tas [ET25] assumed a static adversary. We then try to remove the impractical message delivery assumption. We show that the recovery model is not strictly weaker than the sleepy model and highlight the trade-offs. Then we present our recovery mechanism designed for newly joined parties. After that, we show how we extend our BA to ABC. Lastly, we outline the inherent difficulty of sleepy dispersal, and explore the communication lower bounds of sleepy BA and ABC inspired by it.

3.1 Challenges and Existing Design Method

Challenges in achieving certificate transfer in the sleepy model. The view-based paradigm of classic BFT protocols has an advantage in achieving expected constant latency. However, these protocols often rely on some threshold-based mechanisms to ensure safety in each view. In classical settings, the threshold is fixed (e.g., $f + 1$, where f is the maximum number of byzantine parties), but in the sleepy model, the number of awake byzantine parties is unknown and fluctuating, making static threshold unusable.

A natural adaptation is to employ a *dynamic quorum* formed based on locally "perceived" participation level. However, as pointed out by Pass-Shi [PS17] and Momose-Ren [MR22], dynamic quorums fail to guarantee *certificate transferability*. Specifically, in the classical setting, a quorum certificate of size $f+1$ is accepted by all honest parties. In the sleepy model, nonetheless, the adversary can manipulate perceived participation by selectively revealing their presence. Consequently, honest parties may estimate different quorum sizes, and a certificate formed under a lower perceived participation level may be rejected by parties observing a higher level. Worse still, even if there is no adversary, a certificate may become invalid when the participation level fluctuates.

Existing design method: Time-shifted quorum [MR22]. To address the non-transferability issue of dynamic quorums, Momose-Ren proposed the *time-shifted quorum* mechanism, which they

then used to construct *graded agreement*. Let M_p and $E_p(v)$ denote the number of awake parties, and messages supporting v, respectively, as perceived by an honest party p. Graded agreement ensures that if p recognizes a certificate for a value v (i.e., $E_p(v) > M_p/2$) with grade 1, then every other honest party q must also observe a certificate for the same v (i.e., $E_q(v) > M_q/2$) with grade at least 0.

Here $E_p(v)$, M_p, $E_q(v)$ and M_q are perceived at different times. Security hinges on maintaining an inclusion property between the views of honest parties across time. To achieve this, the protocol enforces a *continuous forwarding* mechanism, where parties continuously forward all newly received messages to all parties to "align" their local views. More specifically, $E_p(v)$ is perceived Δ time earlier than $E_q(v)$. Since p forwards all messages, q receives all messages supporting v seen by p, such that $E_q(v) \geq E_p(v)$. Similarly, M_p is perceived Δ time later than M_q, thus $M_p \geq M_q$. Together, if p observes $E_p(v) > M_p/2$ (and outputs v with grade 1), these relationships yield the critical invariant $E_q(v) \geq E_p(v) > M_p/2 \geq M_q/2$, which guarantees that every honest party q has $E_q(v) > M_q/2$ (and output v with grade ≥ 0), regardless of fluctuations in participation. In this way, the quorum becomes transferable.

3.2 Weak Graded Agreement

Reason for high communication cost: Forwarding. While Momose and Ren [MR22]'s time-shifted quorum resolves the transferability issue, it requires the parties to continuously forward all newly received messages to synchronize their views. This results in the high communication cost of $O(n^2NL)$, which is impractical for scalable consensus. A naive optimization might be employing a *Dynamic Threshold Signatures (DTS)* of size $O(\log N)$ to compress messages supporting the same value. However, this fails because in [MR22], a party P needs to count the total number of messages sent by *distinct* parties multiple rounds ago. In [MR22], this is achieved by parties forwarding every message received, so P can perceive all messages that the previous honest parties received. However, P could not figure out the number of distinct parties now when applying DTS, as the signers of multiple DTSs may *overlap*, and the sender information would have been lost if the parties sent P DTSs (otherwise communication blown up again).

Certificate transfer without forwarding via cheaper WGA. Recall that Malkhi, Momose, and Ren [MMR23] observed that if a party receives a set of values from the network (and every honest party sends a value), the *median* value received by that party is lower and upper-bounded by values sent by honest parties.

We extend the insight to the thresholds (τ) of DTS, establishing a stronger property. Specifically, if a party observes that a set of signatures for a value x constitutes a majority, and importantly, that their thresholds are all greater than or equal to the local median, we can infer that at least one honest party must have multicast a DTS for x with a threshold ($\tau \geq$ median), and thus every honest party will receive it within Δ time.

With this insight, we construct our weak graded agreement (WGA) primitive. Our WGA is weak in a sense that, unlike graded agreement in Momose-Ren [MR22] that ensures integrity, or consistent graded agreement in Efron-Tas [ET25] that ensures integrity and consistency, it gives up both, and guarantees only validity, termination, and graded delivery required for our consensus.

The protocol relies on a recursive guarantee across rounds (from $t = 2\Delta$ back to $t = 0$). For a party to output a value x with grade 1 at time 2Δ, it must satisfy the following two conditions:

1. The party receives a majority of DTS for x with thresholds $\tau \geq$ median m_1. This implies that at

least one honest party must have multicast a DTS for x with $\tau_h \geq m_1$ at time $t = \Delta$. Crucially, our protocol is designed such that an honest party only multicasts DTS with $\tau_h = \text{median } m_0$ at $t = \Delta$ if it had observed a majority of DTS for x with $\tau \geq m_0$. Recursively applying this logic implies that at least one honest party must have multicast a DTS for x with $\tau'_h \geq m_0$ at $t = 0$. Consequently, this guarantees that every honest party receives a DTS for x with $\tau'_h \geq m_0 = \tau_h \geq m_1$ by $t = \Delta$.

2. The party receives no DTS for any conflicting value $x' \neq x$ with a threshold $\tau \geq \text{median } m_1$.

Thus, it ensures that at $t = \Delta$, every honest party has DTS for x with $\tau \geq m_1$, and no party has DTS for $x' \neq x$ with $\tau \geq m_1$. As a result, if every honest party multicasts its DTS with the maximum threshold at $t = \Delta$, all the honest parties will observe the same majority at $t = 2\Delta$, *restoring certificate transferability without the need to forward the entire history.* This method only requires the parties to multicast values along with DTS, which incurs a communication cost of $O(nNL + nN\kappa + nN \log N)$. Further details are provided in Section 4.

3.3 From WGA to BA

Now with a weaker graded agreement, we will still lift it to BA.

Starting point: A similar framework to Efron-Tas [ET25]. Building upon our WGA primitive, we construct a BA protocol following a similar approach as [ET25]. The protocol proceeds in iterations, where each iteration consists of two sequential invocations of WGA followed by a fallback leader election mechanism. In the first WGA instance, parties attempt to commit to a value; if successful (outputting with grade 1), they lock in the decision. The second WGA instance serves to propagate this decision to ensure consistency. Finally, if no agreement is reached by the end of the WGA instances, a leader is elected via a Verifiable Random Function (VRF) to propose a value for the next iteration. Since the leader is chosen uniformly, the probability that the elected leader is honest is at least $\frac{1}{2}$, which ensures termination after an expected constant number of iterations. Since each WGA primitive incurs $O(nNL + nN\kappa + nN \log N)$ bits, the expected communication complexity is then $O(nNL+nN\kappa+nN \log N)$, which asymptotically reduces the $O(n^2NL+n^2N\kappa)$ complexity of Efron-Tas [ET25].

Note that although our WGA is weaker than the consistent graded agreement primitive in Efron-Tas as it does not ensure integrity and consistency, several new challenges arise.

New challenge: Growing adversaries after honest termination. Unlike Efron-Tas [ET25] which considers a model with static adversary, our employment of a *growing adversary* model introduces a critical vulnerability when combined with the requirement of *termination* property of sleepy BA. Typically, the termination property requires honest parties to halt execution after outputting a value. However, as honest parties reach consensus and leave the network, the number of awake honest parties decreases but the number of awake byzantine parties remains, which can lead to a dangerous scenario where the honest majority assumption is broken. Even worse, under a growing adversary, newly joined byzantine parties can simulate past presence, effectively claiming that they were awake a long time ago. Thus for an honest party (re)joining the execution after the "tipping point" when the first honest party halts, the honest majority assumption does not hold even for the period preceding the "tipping point".

Our observation: Time-relative safety invariant. A natural idea to counter the growing adversary is to rely on the cumulative invariant that the number of "honest parties ever awake"

is always greater than the number of "byzantine parties ever awake". In this way, newly joined byzantine parties pretending to be awake in the past are tolerated. However, some of the honest parties may have gone offline before they reach consensus with other parties, so they have no valid information to help newly joined parties decide their value. Thus, the invariant is insufficient to convince the parties (re)joining the execution after the "tipping point" of the correct history.

Let T denote the time when the *first* honest party halts (the "tipping point"). To resolve the issue, we observe a stronger invariant relative to this specific time: for any time $t \leq T$ and a time $t' \geq t$, the number of "honest parties ever awake during $[t, t']$" is strictly greater than the number of "byzantine parties ever awake before or at time t'". We derive the statement as follows: By $T_f = \infty$, for any time t', the number of "byzantine parties ever awake before or at time t'" equals the current byzantine population at t'. By basic honest-majority assumption at any moment, the current byzantine population at t' is always smaller than the current honest population at t'. Since $t' \geq t$, the current honest population at t' is always smaller than the number of "honest parties ever awake during $[t, t']$".

History reconstruction via time-relative safety invariant. This observation dictates our strategy for newly joined parties. Upon (re)joining the execution, a party does not blindly trust the current awake set (which may be compromised after T). Instead, it performs a historical validation by iteratively scanning discrete timestamps $t' = 0, \Delta, 2\Delta, \ldots$ The party checks if there exists a majority of messages timestamped *at or after* t' that support a consistent value x'. If such a majority is identified, the party immediately outputs x', multicasts its own supporting message for x' to assist future joiners, and halts. This allows the latecomers to decide consistently based on the reliable majority that existed in the past.

The correctness of this strategy relies on a specific protocol guarantee that all the honest parties awake at T reach consensus and multicast messages supporting the agreed value at T. We achieve this by letting parties that have outputted earlier multicast supporting messages for the value in every iteration before they halt. Since our sleepy BA protocol satisfies *deterministic synchronization*, this happens at most twice for each party in total, so the communication complexity does not blowup. Further details are included in Section 5. Moreover, although the exact value of T is unknown, by scanning from $t' = 0$ upwards, a party is guaranteed to eventually find a valid majority at $t' = T$ at the latest, ensuring termination.

3.4 Eliminating Message Delivery Assumption

In the original sleepy model [PS17], it is assumed that if a party p is awake at time t, then it must have received all messages sent to it by honest parties by time $t - \Delta$. However, as pointed out by Momose-Ren [MR22], this message delivery assumption is impractical, as it is unknown how the history messages can be magically buffered when the newly joined party was asleep. Motivated by this, the recovery model was proposed [MR22].

Sleepy model vs. recovery model: Trade-offs. To solve BA in the recovery model, we adopt an efficient recovery mechanism similar to [MR22, MMR23], where newly joined parties explicitly request missing data from currently awake parties, rather than relying on the history flood. While this eliminates the message delivery assumption, it implicitly requires a period of *stable participation* during the recovery. In this period, the *recovering* party is not considered awake until the recovery mechanism completes; meanwhile, the number of awake honest parties must exceed the number of byzantine parties in order to convince the recovering party of the correct history. This effectively

limits the adversary's ability to disrupt consensus through rapid participation fluctuations.

Therefore, we need to redefine the *termination* property of BA, transitioning it from the sleepy model to the recovery model. In the original sleepy model, termination means that there exists a time T such that every honest party that is awake at some time $t \geq T$ outputs and halts. In the recovery model, however, these parties cannot halt, otherwise, the honest majority assumption will be broken for newly joined parties trying to recover. Therefore, instead of halting, the parties transition into a *passive state*, where the parties cease generating new consensus messages but continue to respond to recovery queries.

Recovery mechanism for newly joined parties. With our protocol's stateless design, where step execution depends exclusively on the messages received within Δ time (as detailed in Section 6), parties (re)joining the execution before or at T can seamlessly join the execution without full state reconstruction. Consequently, the recovery mechanism is reserved primarily for parties (re)joining the execution *after* T. These newly joined parties do not need to replay history; instead, they simply query the network for their outputs, so they can adopt the majority's decision. This approach significantly reduces the communication overhead, since the network does not need to send messages to the newly joined parties to convince them of the correct history.

To prevent an adversary from exploiting the recovery mechanism, specifically by constantly spamming recovery queries, we introduce a counter-based limitation. Honest parties cap the number of recovery responses sent to peers. Specifically, a party p sends recovery responses to a party at most once per BA iteration, and no more than twice in total after p outputs. Due to the deterministic synchronization of our protocol, at least one response sent after p outputs is generated at or after T. And if a party sends the recovery query at or after T for the first time, it will always observe a majority and output. Thus, a two-response limit is sufficient for correctness while effectively curbing potential DoS attacks. As a result, the recovery mechanism incurs a total of $O(N\kappa + nL + n\kappa) = O(N\kappa + nL)$ bits for a party to recover, where $O(N\kappa)$ is incurred by the recovering party multicasting recovery queries, and $O(nL+n\kappa)$ is incurred by awake parties sending recovery responses to it.

3.5 From BA to ABC

While Efron-Tas [ET25] extends BA to ABC via a Dynamically Available Common Subset (DACS), their construction incurs substantial communication overhead in exchange for censorship resistance. Although the same approach could be applied here, our focus is on minimizing communication complexity.

Therefore, we adopt the classical paradigm of invoking BA instances periodically. Specifically, our ABC protocol proceeds in views. In each view v, a leader is elected using a Verifiable Random Function (VRF) and proposes a block of values. This block serves as the input for the v-th BA instance. To ensure total ordering, parties buffer the outputs of these BA instances as they may be executed parallelly. The values in the k-th block are delivered only after all preceding blocks (1 to $k-1$) have been delivered. As shown in Section 7, the expected number of concurrently active BA instances is constant. Thus, the expected amortized communication complexity of our ABC is $O(nNL + nN\kappa + nN \log N)$ per input value.

Regarding the recovery mechanism in the recovery model, multicasting the recovery queries incurs $O(N\kappa)$ bits. Then for a party that has slept for ℓ views, it will need to recover its status in ℓ BAs. So, the awake parties send a total of $O(\ell(nL + n\kappa))$ bits of recovery responses, and the total

communication cost is $O(N\kappa + n\ell L + n\ell\kappa)$ bits.

3.6 Communication Lower Bound

Inherent difficulty of sleepy dispersal. A natural strategy to further improve the communication complexity of consensus protocols is to decouple large payloads from the agreement process, typically by running consensus algorithms on hashes while dispersing the full data via erasure coding. We notice that this is difficult in the sleepy model. We cannot figure out how to disperse n messages of size L with $O(NL)$ bits using erasure codes, because if sufficiently many (e.g., all but one) parties holding the shares sleep before they multicast their shares, the remaining shares are insufficient to reconstruct the value.

Inspired by this, we establish communication lower bounds of $\Omega(N^2L)$ for BA and ABC in the sleepy and recovery models, where N is the total number of parties, and L is the input size. We show that the lower bound holds even in the absence of byzantine faults, as long as the sleepiness is adaptive[5]. To prove this, we first define a simplified problem called the *Dissemination Task*, establish its lower bound, and show that it reduces to BA and ABC, so the lower bound also applies to them. The goal of the dissemination task is simply to ensure that a value held by at least one awake party is eventually output by all parties. We guarantee that at least one party holding the value is awake at the start of the execution and has time to multicast once to all parties before it becomes asleep. The core difficulty arises from the combination of *adaptive sleepiness* and *initial indistinguishability*. Consider a scenario where a party p holds the input value. Due to the adaptive nature of the environment, the adversary could force all other currently awake parties to sleep immediately after the protocol starts. To guarantee that the value survives and reaches future (re)joining parties, p acts as a "lone sender" and has no choice but to transmit the full value of length L to all N parties before it sleeps, resulting in a communication cost of $\Omega(NL)$.

The lower bound follows from the fact that parties cannot distinguish among global states at the start of the execution, which is the only point they are guaranteed to be awake. Even if multiple parties are awake and hold the input initially, each individual party cannot determine whether it is effectively the only party that is awake (the "lone sender" scenario). Consequently, to ensure safety against the worst-case adaptive schedule, *every* party holding the input must conservatively act as if they are the only one responsible for transmitting the value. As a result, they all incur $\Omega(NL)$ bits. Since there are up to $O(N)$ such senders, the total communication complexity is $\Omega(N^2L)$. The formal proof is provided in Section 8.

4 Weak Graded Agreement

In this section, we present our weak graded agreement (WGA) primitive, a weaker version of graded agreement [MR22, MMR23] and consistent graded agreement [ET25]. The protocol is shown in Alg. 1.

Input. At time $t = 0$, each party P_i multicasts its input v_i together with a partial signature $\mathsf{DTS.ParSign}(v_i, sk_i)$ in an "input" message.

[5]Here we adopt the definition of ABC given in Section 2.1, where liveness is required when all *awake* honest parties receive x. There is also a weaker variant of ABC in which liveness is required only when all honest parties receive x. Our lower bound proof can be extended to this weaker variant, but the extension relies on the presence of byzantine corruptions.

Algorithm 1 Weak Graded Agreement – $\mathsf{WGA}_{\mathsf{id}}$

Initialize v_i to P_i's input. After starting the protocol, P_i executes the following algorithm at every time $0 \leq t \leq T_b$. All messages are bound to the protocol's id. All invalid messages are ignored.

if $t = 0$:
  multicast $\langle \text{"input"}, v_i, \mathsf{DTS.ParSign}(v_i, sk_i) \rangle_i$

if $t = \Delta$:
  $C \leftarrow [v_j \mid P_i \text{ has received } \langle \text{"input"}, v_j, \sigma_j \rangle_j]$
  $u \leftarrow$ the most frequent value in C *// Pick one arbitrarily if there are multiple*
  $E(u) \leftarrow \{(j, \sigma_j) \mid P_i \text{ has received } \langle \text{"input"}, u, \sigma_j \rangle_j\}$
  $(\tau, \Sigma) \leftarrow \mathsf{DTS.Combine}(u, E(u))$
  multicast $\langle \text{"tally"}, i, u, \tau, \Sigma \rangle_i$

if $t = 2\Delta$:
  $(q, u_q, \tau_q, \Sigma_q) \leftarrow$ received "tally" msg with max τ
  multicast $\langle \text{"vote"}, q, u_q, \tau_q, \Sigma_q \rangle_i$

if $t = 3\Delta$:
  $(q, u_q, \tau_q, \Sigma_q) \leftarrow$ received "vote" msg with max τ
  $(p, u_p, \tau_p, \Sigma_p) \leftarrow$ received "vote" msg with max τ s.t. $u_p \neq u_q$
  // $(\perp, \perp, 0, \perp)$ if there are no "vote" msg s.t. $u_p \neq u_q$
  multicast $\langle \text{"echo1"}, q, u_q, \tau_q, \Sigma_q \rangle_i$
  multicast $\langle \text{"echo2"}, p, u_p, \tau_p, \Sigma_p \rangle_i$
  $\overline{\tau} \leftarrow \text{median}([\tau_k \mid P_i \text{ has received } \langle \text{"vote"}, k, u_k, \tau_k, \Sigma_k \rangle_j])$
  $E \leftarrow \{j \mid P_i \text{ has received } \langle \text{"vote"}, k, u_q, \tau_k, \Sigma_k \rangle_j \text{ s.t. } \tau_k \geq \overline{\tau}\}$
  $E^* \leftarrow \{j \mid P_i \text{ has received } \langle \text{"vote"}, *, *, *, * \rangle_j\}$
  if $\overline{\tau} > 0 \wedge |E| > |E^*|/2$:
    $(l, u_q, \tau_l, \Sigma_l) \leftarrow$ received "vote" msg with min τ by $j \in E$
    multicast $\langle \text{"bar"}, l, u_q, \tau_l, \Sigma_l \rangle_i$
  else:
    multicast $\langle \text{"bar"}, \perp, \perp, 0, \perp \rangle_i$

if $4\Delta \leq t \leq T_b$:
  $(*, q, u_q, \tau_q, \Sigma_q) \leftarrow$ received "echo" msg with max τ
  $\overline{\tau} \leftarrow \text{median}([\tau_k \mid P_i \text{ has received } \langle \text{"bar"}, k, u_k, \tau_k, \Sigma_k \rangle_j])$
  $E \leftarrow \{j \mid P_i \text{ has received } \langle \text{"bar"}, k, u_q, \tau_k, \Sigma_k \rangle_j \text{ s.t. } \tau_k \geq \overline{\tau}\}$
  $E^* \leftarrow \{j \mid P_i \text{ has received } \langle \text{"bar"}, *, *, *, * \rangle_j\}$
  if $\overline{\tau} > 0 \wedge |E| > |E^*|/2$:
    $(r, u_q, \tau_r, \Sigma_r) \leftarrow$ received "bar" msg with min τ by $j \in E$
    if $\nexists \langle \text{"echo*"}, *, u', \tau', * \rangle$ s.t. $(u' \neq u_q \wedge \tau' \geq \tau_r)$:
      output $(u_q, 1)$ and halt
  $C \leftarrow [u_k \mid P_i \text{ has received } \langle \text{"echo1"}, k, u_k, \tau_k, \Sigma_k \rangle_j]$
  $u \leftarrow$ the most frequent value in C
  output $(u, 0)$ and halt

Tally. At time $t = \Delta$, each party collects "input" messages to consolidate evidence. Specifically, it identifies the most frequent value u among the received valid inputs and aggregates the partial signatures into a DTS tuple (τ, Σ). It then multicasts this aggregated evidence in a "tally" message.

Vote. At time $t = 2\Delta$, each party inspects the received "tally" messages and selects the one containing the maximal threshold τ. The party then multicasts a "vote" message endorsing this candidate.

Echo and Bar. At time $t = 3\Delta$, parties process "vote" messages to filter certificates based on the median-based insight. First, to ensure conflicting evidence is propagated, each party forwards the "vote" messages with the highest thresholds and a conflicting message via "echo1" and "echo2" messages. To determine the robustness of the vote, the party computes $\overline{\tau}$ as the median of thresholds τ_k across all received "vote" messages. Let E be the set of senders whose votes satisfy $\tau_k \geq \overline{\tau}$, and E^* be the set of all distinct senders of "vote" messages. If the robust votes form a majority (i.e., $|E| > |E^*|/2$), the party multicasts a "bar" message containing the minimum threshold observed in E; otherwise, it reports $\perp$.

Output. At time $t \geq 4\Delta$, parties determine the final output based on "bar" and "echo" messages. A party computes the median $\overline{\tau}$ of thresholds in received "bar" messages. If the set of valid "bar" messages (where $\tau_k \geq \overline{\tau}$) constitutes a majority, the party adopts the candidate u_q associated with the minimum threshold in this set. Finally, the party evaluates a safety condition: if there exists no "echo" message for a conflicting value $u' \neq u_q$ with a competing threshold $\tau' \geq \tau_r$, it outputs u_q with grade 1. Otherwise, it outputs the most frequent value in "echo1" messages with grade 0.

4.1 Correctness Proof

Now we prove the correctness of Alg. 1. Note that if a party is awake at some time $4\Delta \leq t \leq T_b$, it outputs and halts. Moreover, if $t > T_b$, the party halts immediately, ensuring termination.

Lemma 4.1 (Graded Delivery)**.** *In Alg. 1, if an honest party outputs* $(x, 1)$*, then every honest party that produces an output outputs* $(x, *)$*.*

Proof. Assume that an honest party P_i outputs $(x, 1)$ at some time $4\Delta \leq t \leq T_b$. Then at that time, P_i must have received a set of "bar" messages for x from parties in E such that $|E| > |E^*|/2$ (line 30). Since all the honest parties multicast "bar" at $t = 3\Delta$, and the number of byzantine parties by time T_b is less than the number of honest parties at $t = 3\Delta$, we can deduce that at least one honest party P_h has multicast "bar" for x with $\tau \geq \tau_r$ (line 31).

Likewise, from P_h multicasting "bar" for x with $\tau \geq \tau_r$ at $t = 3\Delta$, we can deduce that at least one honest party has multicast "vote" for x with $\tau_v \geq \tau_l$ at $t = 2\Delta$. This means that all the awake honest parties will receive the "vote" message at $t = 3\Delta$. Apart from that, since P_h multicast "bar" with $\tau = \tau_l$ (line 22), we can deduce that $\tau_v \geq \tau_l \geq \tau_r$.

Moreover, since P_i outputs $(x, 1)$, we can see that P_i has not received any "echo" for $x' \neq x$ with $\tau \geq \tau_r$ (line 32). This means that no honest parties have received "vote" for $x' \neq x$ with $\tau \geq \tau_r$ at $t = 3\Delta$, because the "vote" message for $x' \neq x$ with max τ has been repacked into "echo" and multicast (lines 13–16), but P_i has not received one with $\tau \geq \tau_r$.

Since at $t = 3\Delta$, all the honest parties hold "vote" for x with $\tau_v \geq \tau_r$ and do not hold "vote" for $x' \neq x$ with $\tau \geq \tau_r$, then all the honest parties multicast "echo1" for x. Assume that an honest party outputs (y, g) at some time $4\Delta \leq t \leq T_b$. If $g = 1$, by following the same procedure, we can deduce that all the honest parties multicast "echo1" for y at $t = 3\Delta$, so $y = x$. If $g = 0$, since all

the honest parties multicast "echo1" for x, every honest party will have $u = x$ (line 35), and output $(u, 0)$. Combining these, all the honest parties either output $(x, 0)$ or $(x, 1)$. □

Lemma 4.2 (Validity)**.** *In Alg. 1, if all the honest parties that are awake at the start of the protocol input x, then every honest party that produces an output outputs $(x, 1)$.*

Proof. Assume that there are h_0 honest parties and f byzantine parties at $t = 0$ where $h_0 > f$. Then by time T_b, the number of awake byzantine parties is bounded by f.

At $t = 0$, at least h_0 parties multicast "input" for x. Since h_0 is more than half of the total number of parties awake at $t = 0$, all the honest parties have $u = x$, and multicast "tally" for x with $\tau \geq h_0$ at $t = \Delta$. Since there are up to f partial signatures for $x' \neq x$ and $f < h_0$, at $t = 2\Delta$, all the honest parties multicast "vote" for x with $\tau \geq h_0$, and there are no "vote" for $x' \neq x$ with $\tau \geq h_0$ even from byzantine parties. Therefore, at $t = 3\Delta$, all the honest parties multicast "echo1" for x with $\tau \geq h_0$, and all "echo" messages for $x' \neq x$ have $\tau < h_0$. Since every honest party always receives messages from all the awake honest parties, more than half of the "vote" messages received are sent by honest parties, so these messages have $u = x$ and $\tau \geq h_0$. This ensures $\overline{\tau} \geq h_0$, and $|E| > |E^*|/2$. As a result, all the honest parties multicast "bar" for x with $\tau \geq h_0$, and there are no valid "bar" for $x' \neq x$ with $\tau \geq h_0$ even from byzantine parties. Therefore, at $t = 4\Delta$, all the honest parties have $u_q = x$, $\overline{\tau} \geq h_0$, and $|E| > |E^*|/2$. Since there are no "echo" for $x' \neq x$ with $\tau \geq h_0$, all the honest parties output $(u_q = x, 1)$. □

4.2 Efficiency Analysis

Round Complexity. The protocol is deterministic and completes in 4 rounds. Any party that is awake at some time $t \geq 4\Delta$ outputs (or decides not to output) and halts immediately.

Communication Complexity. At $t = 0$, every awake party multicasts a value of size L, a partial signature of size $O(\kappa)$, and a message signature of size $O(\kappa)$ to all parties. In a network of N parties, this incurs $O(nNL + nN\kappa)$ bits. Then at $t = \Delta$, 2Δ, and 3Δ, every party multicasts a constant number of IDs of size $O(\log N)$, values of size $O(L)$, τ of size $O(\log n)$, Σ of size $O(\log N)$, and message signatures of size $O(\kappa)$, which incurs a total of $O(nNL + nN\kappa + nN \log N)$ bits. Thus, the overall communication is $O(nNL + nN\kappa + nN \log N)$.

5 Byzantine Agreement

We now present our sleepy BA protocol, whose details are given in Alg. 2.

Iterative Structure. The protocol proceeds in iterations. Each iteration spans 9Δ rounds and is with two sequential invocations of WGA followed by a leader election and value update phase.

First WGA and Voting. At time $t = 0$ of iteration itr, parties invoke the first WGA instance WGA_{2itr}, using their current local value v_i. At $t = 4\Delta$, parties obtain the result (x, g) from this instance. If the output grade is $g = 1$, it indicates a potential consensus. Consequently, the party decides to output x, but crucially, it defers halting until $t = 4\Delta$ of the *subsequent* iteration $(itr + 1)$. To persist this decision for future joiners, the party multicasts a "vote" message for x tagged with the current iteration. Otherwise (if $g = 0$), it votes for $\perp$. Immediately following this, the second instance WGA_{2itr+1} is invoked with input x.

Algorithm 2 Byzantine Agreement – $\mathsf{BA}_{\mathsf{id}}$

In each iteration itr, P_i executes the following algorithm at every time $0 \leq t \leq 9\Delta$ during iteration itr, and proceeds to the next iteration $itr+1$, until it halts. The first iteration is indexed as $itr = 1$. Initialize v_i to P_i's input. Initialize $h = 0$. Each party outputs only once. All invalid messages are ignored.

1: **if** $t = 0$:
2: start $\mathsf{WGA}_{\mathsf{2itr}}$ with input v_i
3: **if** $t = 4\Delta$:
4: $(x, g) \leftarrow$ the output of $\mathsf{WGA}_{\mathsf{2itr}}$
5: **if** $g = 1$:
6: output x and halt after $t = 4\Delta$ of iteration $itr + 1$
7: multicast $\langle$"vote", $itr, x\rangle_i$
8: **else**:
9: multicast $\langle$"vote", $itr, \perp\rangle_i$
10: $h = itr$
11: start $\mathsf{WGA}_{\mathsf{2itr+1}}$ with input x
12: **if** $t = 8\Delta$:
13: $(x, g) \leftarrow$ the output of $\mathsf{WGA}_{\mathsf{2itr+1}}$
14: multicast $\langle$"echo", $x, g\rangle_i$
15: $\rho_i, \pi_i \leftarrow \mathsf{VRF.Eval}(sk_i, itr)$
16: multicast $\langle$"elect", $i, \rho_i, \pi_i, x\rangle_i$
17: **if** $t = 9\Delta$:
18: $y \leftarrow x$ with most $\langle$"echo", $x, 1\rangle$ received
19: $E \leftarrow \{j \mid P_i$ has received $\langle$"echo", $y, 1\rangle_j\}$
20: $E^* \leftarrow \{j \mid P_i$ has received $\langle$"echo", $*, *\rangle_j\}$
21: **if** $|E| > |E^*|/2$:
22: $v_i \leftarrow y$
23: **else**:
24: $(*, *, *, q) \leftarrow$ received "elect" msg with max ρ
25: $v_i \leftarrow q$
26: **upon** (re)joining the execution: *// execute the following procedure and then lines 1–25*
27: **if** P_i has outputted:
28: halt
29: **if** $t \leq 4\Delta$: $u = itr - 1$ **else**: $u = itr$
30: **if** $4\Delta < t < 5\Delta$:
31: wait till $t = 5\Delta$
32: **for** $k = h + 1$ **to** u:
33: ignore equivocating "vote" msg
34: $E' \leftarrow [x \mid P_i$ has received $\langle$"vote", $k, x\rangle_j]$
35: $x' \leftarrow$ the most frequent value in E'
36: **if** $x' \neq \perp$ and $|\{j \mid P_i$ has received $\langle$"vote", $k, x'\rangle_j\}| > |E'|/2$:
37: multicast $\langle$"vote", $k, x'\rangle_i$
38: output x' and halt after $t = 4\Delta$ of iteration $itr + 1$
39: **break**
40: **else**:
41: multicast $\langle$"vote", $k, \perp\rangle_i$
42: $h = u$

Second WGA and Election. The second WGA instance serves to propagate the locked value. At $t = 8\Delta$, upon receiving the output (x, g) from WGA_{2itr+1}, parties multicast an "echo" message containing this tuple. Simultaneously, to prevent liveness deadlocks in case of disagreement, parties execute a leader election mechanism. Each party P_i evaluates a Verifiable Random Function (VRF) using its secret key and the iteration index to obtain a random value ρ_i and a proof π_i, which are then disseminated via an "elect" message.

Value Update. At $t = 9\Delta$, parties determine the input value v_i for the next iteration. First, a party checks for a strong quorum: if the set of senders of "echo" messages with grade 1 for a value y (denoted as E) constitutes a majority (i.e., $|E| > |E^*|/2$), the party adopts y. Failing this, the party falls back to the leader's proposal. It identifies the "elect" message with the highest VRF output ρ and adopts the associated value q as its new v_i.

Catch-up. A critical component handling the *growing adversary* is the catch-up procedure executed upon (re)joining. Since honest parties may halt after reaching consensus, late joiners cannot rely solely on active participation. Instead, they inspect the historical trace of "vote" messages. Specifically, a joining party iterates through past iterations (from its last known state h to the current iteration). For each iteration k, it counts "vote" messages. If it observes that a value x' (where $x' \neq \perp$) has received votes from a majority of distinct senders, it implies that honest parties have already committed to x'. In this scenario, the joining party immediately adopts x', outputs it, multicasts a "vote" to sustain the evidence, and schedules its own halt, thereby preserving safety.

5.1 Correctness Proof

Now we prove the correctness of Alg. 2.

Lemma 5.1. *In Alg. 2, the first honest party that produces an output outputs at $t = 4\Delta$ in the main loop.*

Proof. We observe that an honest party outputs either at $t = 4\Delta$ in the main loop or when it (re)joins the execution. Let P_i be the first honest party that outputs. We prove by contradiction. Assume that no parties output in the main loop before P_i outputs. We can deduce that no parties multicast "vote" for $x \neq \perp$ (line 7) before P_i outputs, otherwise they output before P_i. Also, no parties halt before P_i outputs because parties only halt after they output. Thus, at any time before P_i outputs, there are more awake honest parties than byzantine parties. These awake honest parties multicast "vote" for $\perp$ (line 9) at $t = 4\Delta$ of every iteration before P_i outputs.

When a party (re)joins the execution and we assume that it is the first party that (re)joins the execution, it receives all the previous "vote" messages. For each iteration, over half of the "vote" messages sent by honest parties are for $\perp$. As a result, the party also multicasts "vote" for $\perp$ for all iterations. And it is the same for all newly joined parties. Note that although the number of byzantine parties may grow, it is offset by the growth of honest parties. When an honest party (re)joins the execution, it can decide what "vote" messages to multicast solely relying on the messages it received, and it can resume the main loop immediately after. Although its "vote" messages may take up to a round to be delivered to others, byzantine parties are further delayed by T_b. Consequently, over half of the "vote" messages are always for $\perp$. This inductively proves that all the newly joined honest parties in the future will only multicast "vote" for $\perp$ when they (re)join the execution, so does P_i. However, P_i should be the first party that outputs. By contradiction, P_i outputs x at $t = 4\Delta$ in the main loop. □

Lemma 5.2 (Consistency). *In Alg. 2, if two honest parties output x and x', then $x = x'$.*

Proof. Let P_i be the first honest party that outputs, and let it output x. By Lemma 5.1, P_i outputs at $t = 4\Delta$ of some iteration itr in the main loop. Then it should halt after $t = 4\Delta$ of iteration $itr+1$ (line 6). Since P_i outputs x, P_i obtains $(x, 1)$ from $\mathsf{WGA}_{\mathsf{2itr}}$ of iteration itr (line 4). By Lemma 4.1, the other honest parties obtain $(x, *)$. Some parties may output x if they obtain $(x, 1)$. Then all the honest parties input x into $\mathsf{WGA}_{\mathsf{2itr+1}}$. By Lemma 4.2, all the honest parties obtain $(x, 1)$ at $t = 8\Delta$. Then they multicast "echo" for $(x, 1)$ and "elect" for x. Since there are more awake honest parties than byzantine parties, at $t = 9\Delta$, all the honest parties will have $y = x$ and $v = x$. Then in the next iteration $itr + 1$, they start $\mathsf{WGA}_{\mathsf{2itr+2}}$ with input x at $t = 0$. By Lemma 4.2, all the awake honest parties obtain $(x, 1)$ at $t = 4\Delta$, and thus output x if they have not.

Now we consider the parties that (re)join the execution during the execution (lines 26–42). The parties process the "vote" messages in the order of iterations (line 32). First, if a party P_j (re)joins the execution before P_i outputs, as we have shown above, P_j will only multicast "vote" for $\perp$, and then resume the main loop.

Second, if P_j (re)joins the execution after P_i outputs, but before or at $t = 4\Delta$ of iteration $itr+1$, P_j will still multicast "vote" for $\perp$ for each iteration before itr. Also, by Lemma 4.1, P_j receives "vote" of iteration itr for either x or $\perp$ from the honest parties. Note that since P_i is the first honest party that outputs, no honest parties halt before or at $t = 4\Delta$ of iteration $itr + 1$. Thus, P_j can receive "vote" of iteration itr from all the honest parties that are ever awake at or after $t = 4\Delta$ of iteration itr. Since P_j needs over half of the "vote" messages to be for x' to multicast $x' \neq \perp$ (lines 36–37), in which at least one is from an honest party, it multicasts "vote" of iteration itr for either x or $\perp$. If P_j does not output x and multicasts "vote" for $\perp$, it will resume the main loop and output x at $t = 4\Delta$ of iteration $itr + 1$, unless it becomes asleep before that.

Third, if P_j (re)joins the execution after $t = 4\Delta$ of iteration $itr + 1$, it first processes "vote" messages of iterations from 1 to itr. As shown above, it may output x, otherwise it will need to process "vote" of iteration $itr + 1$. Also as shown above, no honest parties halt before or at $t = 4\Delta$ of iteration $itr + 1$, and all the honest parties multicast "vote" of iteration $itr + 1$ for x. Let P_k be the first party that (re)joins the execution after $t = 4\Delta$ of iteration $itr+1$. Note that P_k receives all the "vote" messages sent at $t = 4\Delta$ of iteration $itr + 1$, because it waits till $t = 5\Delta$ if $4\Delta < t < 5\Delta$ (line 31). Thus, it multicasts "vote" of iteration $itr + 1$ for x and outputs x. Then for the next newly joined party, all the honest parties that are ever awake at or after $t = 4\Delta$ of iteration $itr + 1$ multicast "vote" of iteration $itr + 1$ for x. The number of these parties is always more than the number of byzantine parties. Thus, all newly joined parties will multicast "vote" of iteration $itr + 1$ for x, and output x. □

Lemma 5.3 (Validity). *In Alg. 2, if all the honest parties that are awake at the start of the protocol input x, then every honest party that produces an output outputs x.*

Proof. At $t = 0$ of iteration 1, all the honest parties awake start WGA_0 with input x. By Lemma 4.2, all the honest parties obtain $(x, 1)$ at $t = 4\Delta$, and thus output x. By Lemma 5.1, the first honest party that outputs always outputs at $t = 4\Delta$ in the main loop. By Lemma 5.2, if two honest parties output, they output the same value. Thus, we can deduce that all the newly joined parties in the future that produce outputs will also output x. □

Lemma 5.4 (Termination). *In Alg. 2, there exists a time T s.t. every honest party that is awake at some time $t \geq T$ outputs and halts.*

Proof. In the proof of Lemma 5.2, we have shown that once the first honest party outputs, all the honest parties awake at or after $t = 4\Delta$ of the next iteration eventually output and halt. Thus, here we only need to prove that at least one honest party outputs.

We first show that before any honest parties output or halt, with probability over $\frac{1}{2}$ for each iteration, all the awake honest parties hold the same v after $t = 9\Delta$. If an honest party satisfies $|E| > |E^*|/2$ and holds $v = y$ (lines 21–22), then at least one honest party has multicast $\langle$"echo", $y, 1\rangle$ at $t = 8\Delta$. By Lemma 4.1, all the honest parties obtain $(y, *)$ from WGA at $t = 8\Delta$ and multicast "echo" and "elect" only for y. With probability over $\frac{1}{2}$, an honest party's VRF value will be the highest at $t = 9\Delta$, and then all the honest parties will hold $v = y$. If no honest parties satisfy $|E| > |E^*|/2$, then with probability over $\frac{1}{2}$ that the highest VRF value is from an honest party, all the honest parties hold the same $v = q$ multicast by the party with highest VRF value in its "elect" message.

Since all the awake honest parties hold the same v after $t = 9\Delta$, they start WGA with the same input v at $t = 0$ of the next iteration. By Lemma 4.2, the awake honest parties at $t = 4\Delta$ obtain $(v, 1)$ from WGA, and thus output v. As proven above, the newly joined parties will also output v once they (re)join the execution.

Furthermore, since the number of iterations needed for all awake honest parties to hold the same v after $t = 9\Delta$ follows a geometric distribution with a success probability of $\frac{1}{2}$ in each iteration, the probability that k iterations are needed is bounded by 2^{-k}, which ensures termination with overwhelming probability. □

5.2 Efficiency Analysis

Round Complexity. In the proof of Lemma 5.4, we have shown that the success probability of each iteration is at least $\frac{1}{2}$. Thus, the expected number of iterations to get an honest leader is 2. The protocol then requires the parties to execute one additional iteration before the parties halt. Note that any honest party that (re)joins the execution after this can decide its output solely relying on the previous messages, locally. Therefore, the total expected number of iterations is constant. Since each iteration takes constant number of rounds, the overall expected round complexity is $O(1)$.

Communication Complexity. In each iteration, the parties call WGA twice, which incurs $O(nNL+nN\kappa+nN\log N)$ bits. Besides that, the parties multicast "vote", "echo", "elect" messages, which incur $O(nNL+nN\kappa)$ bits. In particular, each party maintains a variable h which prevents the party from multicasting "vote" of same iterations multiple times. In other words, a party multicasts a "vote" at most once per iteration on average. As a result, since the expected number of iterations is constant, the overall expected communication complexity is $O(nNL + nN\kappa + nN\log N)$.

6 Byzantine Agreement with Recovery

In this section, we present our BA protocol in the recovery model. Our protocol is presented in Alg. 3.

Normal Execution. The protocol integrates the recovery mechanism into the standard BA flow. Parties execute the standard steps at $t = 0, 8\Delta,$ and 9Δ as in Algorithm 2. A critical divergence occurs at $t = 4\Delta$. When WGA_{2itr} outputs a value x with grade 1, the party updates its local persistent state res to x (if not already set). If the party is actively participating (not in a recovery phase), it outputs x immediately and schedules its halt for the subsequent iteration $(itr + 1)$. This

Algorithm 3 Byzantine Agreement with Recovery – $\mathsf{BA}_{\mathsf{id}}$

In each iteration itr, P_i executes the following algorithm at every time $0 \le t \le 9\Delta$ during iteration itr, and proceeds to the next iteration $itr + 1$. The first iteration is indexed as $itr = 1$. Initialize v_i to P_i's input. Initialize $res = \perp$, $hq = 0$, and $hr[k] = sr[k] = 0$ for $k \in [N]$. Each party outputs only once. When a party halts, it stops executing lines 1–10 but still responds to "query" messages. All invalid messages are ignored.

1: **if** $t = 0, 8\Delta$, or 9Δ:
2: . . . same as Alg. 2. . .
3: **if** $t = 4\Delta$:
4: $(x, g) \leftarrow$ the output of $\mathsf{WGA}_{\mathsf{2itr}}$
5: **if** $g = 1$:
6: **if** $res = \perp$:
7: $res \leftarrow x$
8: **if** P_i is not recovering:
9: output x and halt after $t = 4\Delta$ of iteration $itr + 1$
10: start $\mathsf{WGA}_{\mathsf{2itr}+1}$ with input x
11: **upon** (re)joining the execution: *// execute the following procedure and lines 1–10 concurrently*
12: **if** P_i has outputted:
13: **break**
// query at most once per interval between $t = 4\Delta$ of two consecutive iterations
14: **if** $t < 4\Delta$: $u = itr - 1$ **else**: $u = itr$
15: **if** $hq = u$: **break**
16: $hq = u$
17: multicast $\langle$"query"$\rangle_i$
18: wait for 2Δ
19: ignore equivocating "recover" msg
20: $E' \leftarrow [x \mid P_i$ has received $\langle$"recover", $x\rangle_j]$
21: $x' \leftarrow$ the most frequent value in E'
22: **if** $|\{j \mid P_i$ has received $\langle$"recover", $x'\rangle_j\}| > |E'|/2$:
23: $res \leftarrow x'$
24: **if** $res \neq \perp$:
25: output res and halt after $t = 4\Delta$ of iteration $itr + 1$
26: **upon** receiving $\langle$"query"$\rangle_j$:
// respond to a party at most once per interval between $t = 4\Delta$ of two consecutive iterations
27: **if** $t < 4\Delta$: $u = itr - 1$ **else**: $u = itr$
28: **if** $hr[j] = u$: **break**
29: $hr[j] = u$
// respond to a party with $res \neq \perp$ at most twice in total
30: **if** $sr[j] = 2$: **break**
31: **if** $res \neq \perp$:
32: $sr[j] \leftarrow sr[j] + 1$
33: send $\langle$"recover", $res\rangle_i$ to P_j

delayed halting ensures the party remains responsive to queries from late joiners for a sufficient duration.

Recovery Mechanism. Parties (re)joining the execution after the honest parties halt rely on a query-based mechanism to catch up. Upon joining, a party initiates a "query" to solicit the decided value from the network. To avoid network congestion, this query is rate-limited: a party multicasts a "query" message at most once per iteration interval (specifically, between $t = 4\Delta$ of two consecutive iterations).

After multicasting a query, the recovering party waits for a duration of 2Δ to collect responses. It processes the received "recover" messages to identify a majority consensus. Specifically, if a value x' appears in "recover" messages from a majority of the distinct respondents (relative to the set of received messages), the party adopts x' as its result res. Once res is established—either through this recovery process or direct WGA participation—the party outputs the value and terminates its participation after the designated delay.

To prevent Denial-of-Service attacks where adversaries might spam queries to exhaust honest resources, the protocol enforces strict response limits via local counters (hr and sr). A party responds to a specific peer's "query" at most once per iteration. Furthermore, a party limits the number of times it sends a valid result ($res \neq \perp$) to any single peer to at most twice throughout the entire execution. This constant limit suffices because of deterministic synchronization, as outlined below.

6.1 Correctness Proof

Now we prove the correctness of Alg. 3.

Lemma 6.1. *In Algorithm 3, the first honest party that produces an output outputs at $t = 4\Delta$ in the main loop.*

Proof. We observe that an honest party outputs either at $t = 4\Delta$ in the main loop or when it (re)joins the execution. Let P_i be the first honest party that outputs. We prove by contradiction. Assume that no parties output before P_i outputs. To output x, over half of the "recover" messages P_i receives are for $x \neq \perp$ (line 22), which means at least one is sent by an honest party P_j.

If P_j sends "recover" for x to P_i, it means P_j has $res = x$ when it receives "query" from P_i, and thus P_j is not P_i. Note that res gets updated only in lines 7 and 23. If P_j was not recovering, then P_j updated res to some value x' in line 7. In that case, it outputted x' in line 9, which contradicts the assumption that P_i is the first honest party that outputs. If P_j was recovering, it might update res to some value x' in line 7 or line 23. But in either case, at the end of the recovery process, it satisfies $res \neq \perp$ in line 24 and outputs x' in line 25, which also contradicts the assumption. Therefore, P_i outputs x at $t = 4\Delta$ in the main loop. □

Lemma 6.2 (Consistency)**.** *In Alg. 3, if two honest parties output x and x', then $x = x'$.*

Proof. Let P_i be the first honest party that outputs, and let it output x. By Lemma 6.1, P_i outputs at $t = 4\Delta$ of some iteration itr in the main loop. By following the proof of Lemma 5.2, we can see that some parties output at $t = 4\Delta$ of some iteration itr, and all the other honest parties that are awake at $t = 4\Delta$ of iteration $itr + 1$ output x at $t = 4\Delta$ of iteration $itr + 1$.

Now we consider the parties that recover through the recovery mechanism. We prove that, if a party outputs in the recovery mechanism, it outputs a value that an honest party has outputted.

Therefore, every party that outputs outputs the same x. First, if a party outputs in the recovery mechanism, it outputs res (line 25). Note that res gets updated in lines 7 and 23, because the recovering parties run the main loop and the recovery mechanism concurrently. Since P_i outputs at $t = 4\Delta$ of iteration itr in the main loop, res cannot be updated in line 7 before iteration itr. If res is updated in line 7 at $t = 4\Delta$ of iteration itr or later, res will only be x as we have shown above. To update res to x' in line 23, among all the "recover" messages that the recovering party receives, a majority needs to be for x', in which at least one is from an honest party. Since an honest party sends "recover" with $res \neq \perp$ if and only if it is awake and it outputs res, we can deduce that if a party outputs in the recovery mechanism, it always outputs a value that an honest party has outputted. □

Lemma 6.3 (Validity)**.** *In Alg. 3, if all the honest parties that are awake at the start of the protocol input x, then every honest party that produces an output outputs x.*

Proof. As we have shown in the proof of Lemma 5.3, if all the honest parties that are awake at the start of the protocol input x, all the honest parties awake at $t = 4\Delta$ of the first iteration obtain $(x, 1)$ from WGA_0, and thus output x. By Lemma 6.1, the first honest party that outputs always outputs at $t = 4\Delta$ in the main loop. By Lemma 6.2, if two honest parties output, they output the same value. Thus, we can deduce that all the newly joined parties in the future that produce outputs will also output x. □

Lemma 6.4 (Termination)**.** *In Alg. 3, there exists a time T such that every honest party that is awake at some time $t \geq T$ outputs.*

Proof. Let P_i be the first honest party that outputs, and it outputs x in iteration itr. We have shown that all the honest parties awake at $t = 4\Delta$ of iteration $itr + 1$ output x at $t = 4\Delta$ of either iteration itr or iteration $itr + 1$. For parties that are recovering at $t = 4\Delta$ of iteration $itr + 1$, since the recovering parties run the main loop and the recovery mechanism concurrently, they also update their res in the main loop (line 7). Then at the end of the recovery mechanism, no matter if the parties have updated their res elsewhere, $res \neq \perp$, thus the parties output res. By Lemma 6.2, $res = x$.

We now prove that every honest party that (re)joins the execution after $t = 4\Delta$ of iteration $itr + 1$ also produces an output. Let P_j be a party that runs the recovery mechanism after $t = 4\Delta$ of iteration $itr + 1$ for the first time. Note that an honest party responds to a party's query at most once per interval between $t = 4\Delta$ of two consecutive iterations (lines 27–29). Since the first honest party that outputs outputs at $t = 4\Delta$ of iteration itr, any honest party P_k has sent "recover" message for a non-$\perp$ value to P_j at most once before $t = 4\Delta$ of iteration $itr + 1$, i.e., $sr[k] \leq 1$. So P_k will not refuse to send $recover$ to P_j (line 30). Thus, P_j receives "recover" messages for x from all the awake honest parties. Given that even if a party halts, it still responds to "query" messages, and there are always more awake honest parties than byzantine parties at any time during the execution, P_j satisfies the majority condition in line 22 with $x' = x$ and outputs x in line 25. □

6.2 Efficiency Analysis

Round Complexity. Similar to Alg. 2, the expected round complexity is $O(1)$. In addition, when a party (re)joins the execution, it uses a round to multicast "query" messages and a round to wait for "recover" messages. Thus, the recovery mechanism takes a total of 2 rounds.

Communication Complexity. Similar to Alg. 2, the expected communication complexity of the main loop is $O(nNL + nN\kappa + nN \log N)$. In addition, when a party (re)joins the execution, it multicasts "query" together with its signature, which incurs $O(N\kappa)$ bits. Every awake honest party responds with the "recover" message which contains a value of size $O(L)$ and a signature. Thus, the recovery mechanism incurs $O(N\kappa + nL + n\kappa) = O(N\kappa + nL)$ bits for each party to recover.

Moreover, notice that an honest party responds to a party's query at most once per interval between $t = 4\Delta$ of two consecutive iterations and responds to a party's query with a non-$\perp$ value of length L at most twice in total. This prevents the byzantine parties from constantly multicasting "query" messages to blowup the communication complexity of recovery mechanism.

7 Atomic Broadcast

In this section, we present our ABC protocol which is secure in both the sleepy and the recovery models depending on which BA it uses. Our protocol is presented in Alg. 4.

Algorithm 4 Atomic Broadcast – $\mathsf{ABC}_{\mathsf{id}}$

In each view v, P_i executes the following algorithm at every time $0 \leq t \leq 4\Delta$ during view v, and proceeds to the next view $v + 1$. The first view is indexed as $v = 1$. Initialize $b[k] = \perp$ for $k \geq 1$. All invalid messages are ignored.

if $t = 0$:
  $B_i \leftarrow$ a new block that contains all undecided values
  $\rho_i, \pi_i \leftarrow \mathsf{VRF.Eval}(sk_i, v)$
  multicast $\langle$"elect", $\rho_i, \pi_i, B_i\rangle_i$

if $t = \Delta$:
  $(\rho_q, \pi_q, B_q) \leftarrow$ received "elect" msg with max ρ
  start BA_v with input B_q

upon any BA_w outputs:
  $b[w] \leftarrow$ the output of BA_w
  $u \leftarrow$ the view of the last block decided by P_i *// 0 if none*
  for $k = u + 1$ **to** v:
    if $b[k] \neq \perp$:
      sort values in $b[k]$ deterministically
      decide values in $b[k]$ that are not yet decided
    else:
      break

upon (re)joining the execution:
  for $k = 1$ **to** v:
    resume BA_k

View-Based Proposal. To achieve atomic broadcast with low communication overhead, we adopt the paradigm of invoking BA instances periodically. The protocol operates in a sequence of views. At the start of each view v (time $t = 0$), every party P_i constructs a candidate block B_i containing all transactions or values known to it but not yet decided. To select a canonical proposal for this view, parties execute a leader election via a Verifiable Random Function (VRF). Each party computes a VRF proof π_i and a priority value ρ_i using its secret key and the view index, then multicasts an "elect" message containing these credentials with its proposed block B_i.

At time $t = \Delta$, parties determine the input for the consensus instance of the current view. Each party collects valid "elect" messages and identifies the proposal (B_q, ρ_q, π_q) with the highest VRF priority ρ. This selected block B_q is then used as the input to invoke the v-th instance of the BA protocol, denoted as BA_v.

Total Ordering. To ensure the total ordering property of atomic broadcast, parties employ a buffering and sequential delivery mechanism. The output of BA_w is stored in a local buffer $b[w]$. A party delivers blocks only when previous blocks have been delivered. Specifically, upon receiving an output from any instance BA_w, the party checks if there exists a gap in the sequence of decided blocks from the last delivered view u. It iterates from $u + 1$ upwards; for each consecutive view k where the decision $b[k]$ is available ($b[k] \neq \perp$), the party deterministically sorts the values within $b[k]$ and delivers any previously undecided values. The delivery process halts immediately upon encountering the first view with a missing decision, ensuring no gaps in the final total order.

7.1 Correctness Proof

Now we prove the correctness of Alg. 4.

Lemma 7.1 (Safety)**.** *In Alg. 4, if two honest parties output logs $[x_0, .., x_j]$ and $[x'_0, .., x'_{j'}]$, then $x_i = x'_i$ for all $i \leq \min\{j, j'\}$.*

Proof. In Alg. 4, the output log is constructed deterministically by sorting values contained in $b[1], b[2], \ldots$ and appending them. Specifically, lines 11–16 ensure that a party only processes values in block $b[k]$ if all the preceding blocks $b[1], b[2], \ldots, b[k-1]$ have been processed and decided.

Let P_i and P_j be two honest parties. Let $b_t[w]$ denote the block outputted by P_t in $\mathsf{BA_w}$. Assume that P_t has processed values in blocks from $b_t[1]$ to $b_t[v_t]$. We consider any view $w \in [\min(v_i, v_j)]$. By consistency of BA, since P_i outputs $b_i[w]$ and P_j outputs $b_j[w]$ from the same BA instance $\mathsf{BA_w}$, then $b_i[w] = b_j[w]$. Since both parties sort values within a block deterministically (line 13) and filter duplicates based on the history of the log (line 14), we can deduce that their output logs agree on all positions up to the length of the shorter log. □

Lemma 7.2. *In any view v, if there exists an honest party P_h whose VRF value ρ_h is the highest among all awake parties at $t = \Delta$ of view v, then all the honest parties will obtain B_h from $\mathsf{BA_v}$.*

Proof. P_h multicasts $\langle$"elect", $h, \rho_h, \pi_h, B_h\rangle_h$ at $t = 0$. Since it is honest, all the honest parties receive the message by $t = \Delta$ of view v. Since P_h has the global highest VRF value ρ, all the honest parties select $B_q = B_h$. Consequently, every honest party starts $\mathsf{BA_v}$ with the same input B_h. By validity and termination of BA, every honest party obtains B_h from $\mathsf{BA_v}$ eventually. □

Lemma 7.3 (Liveness)**.** *In Alg. 4, if an awake honest party inputs a value x at time t, then there exists a time $T \geq t$ such that every log outputted by awake honest parties at any time after T contains x.*

Proof. After an awake honest party inputs a value x, every awake honest party will include x in their block B at $t = 0$ of every future view until x is decided. Since there are always more awake honest parties than byzantine parties, in each view the probability that the highest VRF value at time $t = \Delta$ belongs to an honest party is greater than $\frac{1}{2}$. Thus, with probability 1, there exists a future view v' where an honest party P_h is elected leader with a global highest VRF value ρ_h in

view v'. Also, B_h constructed by P_h in view v' contains x. By Lemma 7.2, all the honest parties will obtain B_h from $\mathsf{BA}_{\mathsf{v}'}$ eventually. By termination of BA, for every view $w < v'$, BA_{w} eventually produces an output for all awake honest parties. Thus, values in $\mathsf{BA}_{\mathsf{v}'}$ including x will be processed and decided. □

7.2 Efficiency Analysis

Now we give the analysis of Alg. 4. We analyze the latency for a value w input by an honest party at time t_{in}.

Best-case Latency. The best-case latency is 5Δ. First, since all the parties behave honestly, w will be contained in every party's block at $t = 0$. No matter which party becomes the leader at $t = \Delta$, the parties will then start BA with the same input block. In BA, by validity of WGA, all the awake parties will output the same block at $t = 4\Delta$ of BA. Then the parties will process the block and output w which is contained in the block. Thus, the total latency is 5Δ.

Expected Latency. We now show that the expected latency is $O(\Delta)$. Let v_{in} be the view that time t_{in} is in. The value w might not be proposed if the leader of v_{in} is byzantine. Let $v_h \geq v_{in}$ be the index of the first view after w is inputted where the elected leader is honest. Let T_{v_h} denote the start time of view v_h. Since w is included in block B_{v_h} and thus decided at some time D_{v_h}, the total latency is:

$$L = D_{v_h} - t_{in} = (T_{v_h} - t_{in}) + (D_{v_h} - T_{v_h})$$

We first consider $T_{v_h} - t_{in}$, i.e., the delay incurred waiting for an honest leader. In each view, since there are strictly more awake honest parties than awake byzantine parties, the probability that the randomly elected leader (via VRF) is honest is at least $\frac{1}{2}$. The number of views to wait follows a geometric distribution with a success probability of over $\frac{1}{2}$, and thus its expectation is bounded by 2. Since each view lasts 4Δ, the expected waiting time is $O(\Delta)$.

We then consider $L' = D_{v_h} - T_{v_h}$, i.e., the time to decide block B_{v_h} relative to view v_h start time. Recall that values in B_{v_h} are processed only when all BA instances up to v_h have outputted. Let X_k be the random variable representing the time T such that every party awake at or after T outputs from BA_k, relative to the start of BA_k. Since BA_k starts at $t = \Delta$ of view k,

$$L' = D_{v_h} - T_{v_h} = \max_{1 \leq k \leq v_h} (T_k + \Delta + X_k) - T_{v_h} = \Delta + \max_{0 \leq j \leq v_h - 1} (X_{v_h - j} - 4\Delta j)$$

In the round complexity analysis, we have shown that BA has an exponential tail bound, i.e., $\Pr[X > (C+s)\Delta] \leq 2^{-cs}$ for constants C, c. Let $Y_j = X_{v_h - j} - 4\Delta j$ represent the residual execution time of a BA instance. The probability that any BA instance delays the current decision by time t is bounded by the sum of tail probabilities:

$$\Pr[\max_{0 \leq j \leq v_h - 1} Y_j > t] \leq \sum_{j=0}^{v_h - 1} \Pr[X_{v_h - j} > t + 4\Delta j] \leq \sum_{j=0}^{\infty} 2^{-c(t/\Delta + 4j - C)}$$

Since the series $\sum 2^{-4cj}$ converges, the tail of L' decays exponentially, thus $\mathbb{E}[L'] = \int_0^\infty \Pr[L' > t]\, dt = O(\Delta)$.

Combining both parts, the total expected latency is $\mathbb{E}[L] = \mathbb{E}[T_{v_h} - t_{in}] + \mathbb{E}[D_{v_h} - T_{v_h}] = O(\Delta) + O(\Delta) = O(\Delta)$.

Communication Complexity. We analyze the expected amortized communication complexity of our ABC per input value. We first calculate the communication complexity per view. The communication consists of the leader election phase and the execution of BA instances. In the leader election phase (lines 3–6), every awake party multicasts an "elect" message containing a block, a VRF value, a VRF proof, and a signature. Thus, n parties incur a total of $O(nNL + nN\kappa)$ bits.

For the execution of BA instances, multiple BA instances invoked in the previous views may run concurrently. For convenience, we say a BA instance is active if the parties have not reached a time T such that at or after T, every party can output immediately when they become awake. The expected total cost is the expected number of active instances multiplied by the communication cost of a single instance in a view. Let M_v be the number of BA instances active at view v. We can write $M_v = \sum_{k=0}^{v-1} I_k$, where I_k is the indicator variable that BA_{v-k} is active. Using the exponential tail bound derived in the latency analysis, i.e., $\Pr[\mathsf{BA} \text{ active duration} > 4\Delta k] \leq 2^{-ck}$ for a constant c, we have:

$$\mathbb{E}[M_v] = \sum_{k=0}^{v-1} \mathbb{E}[I_k] = \sum_{k=0}^{v-1} \Pr[\mathsf{BA}_{v-k} \text{ is active}] \leq \sum_{k=0}^{\infty} 2^{-ck}.$$

Since this geometric series converges to a constant, the expected number of concurrent instances is $O(1)$. Thus, the expected amortized communication cost is proportional to the cost of a single BA, i.e., $O(nNL + nN\kappa + nN \log N)$.

Combining both parts, the total expected amortized communication complexity is $O(nNL + nN\kappa + nN \log N)$ per view. Since we have shown that the expected latency of our ABC is constant, the expected amortized communication complexity of our ABC per input value is also $O(nNL + nN\kappa + nN \log N)$. Note that a value may be included in multiple blocks before it is decided. However, since the latency of ABC is constant, the value is only included in a constant number of blocks in expectation, so the total expected communication complexity remains unchanged.

If in the recovery model, regarding the recovery mechanism, the newly joined party multicasting the recovery queries first incurs $O(N\kappa)$ bits. Then if the party has slept for ℓ views, it will recover its states in $O(\ell)$ BAs. Therefore, the awake parties send a total of $O(\ell(nL + n\kappa))$ bits of recovery responses, and the total communication cost is $O(N\kappa + n\ell L + n\ell\kappa)$ bits per recovering party.

8 Communication Lower Bounds

In this section, we establish communication lower bounds for BA and ABC in the sleepy and recovery model. We show that even without byzantine faults, the uncertainty introduced by adaptive sleepiness forces a communication overhead of $\Omega(N^2 L)$.

We begin with the communication lower bound for BA:

Theorem 8.1. *Let Π be any protocol that solves byzantine agreement among N parties in the sleepy or recovery model with adaptive sleepiness. Even in the absence of byzantine corruptions, there exists an execution of Π where the total communication complexity is $\Omega(N^2 L)$, where L is the input length.*

To prove Theorem 8.1, we first define a simplified problem called the *dissemination task* and identify necessary conditions for solving it. We consider that all parties are honest. Since an adaptive adversary can always choose not to corrupt parties, any lower bound derived in this setting naturally applies to the byzantine setting.

Dissemination Task (DT). In DT, each party that is awake at the start of the protocol takes as an input $x \in \{m, \perp\}$, where $m \in \{0,1\}^L$. At least one party awake at the start of the protocol has input m. Each party outputs a value, subject to the following constraints:

- **Validity.** Every party that produces an output outputs m.
- **Termination.** There exists a time T such that every honest party that is awake at any time $t \geq T$ outputs.

Now we show that every BA protocol Π can be used as a black-box to construct a solution to DT via the following construction: let P_i be a party that is awake at the start of the protocol. If P_i has input m, it invokes Π with input m at the start of the protocol; if P_i has input $\perp$, it does not invoke Π at the start of the protocol, instead, it joins the execution of Π one round later. Since there are no byzantine parties, all the parties that invoke Π at the start of the protocol have input m, and there is at least one such party. By validity of Π, every party that produces an output outputs m. By termination of Π, every party that is awake after some time T outputs. Thus, the DT protocol is secure. Since DT reduces to Π with no additional communication overhead, the communication lower bound for DT also applies to Π. To prove the lower bound for DT, we first prove the following two lemmas:

Lemma 8.2 (Lone Sender Complexity)**.** *Consider an execution that only one party S holds input m and all the other awake parties hold $\perp$. S must send $\Omega(NL)$ bits at the start of the protocol to ensure all parties can eventually receive m.*

Proof. First, S has to transmit some information about m at the start of the protocol, otherwise, if S sleeps immediately after the start of the protocol, no other parties can get m. Assume for the sake of contradiction that S sends $o(NL)$ bits at the start of the protocol. By the pigeonhole principle, there exists at least one receiver party $R \neq S$ such that the number of bits it receives from S is less than L. Since the model supports adaptive sleepiness, the adversary can enforce a sleep schedule where all parties other than R are asleep immediately after the start of the protocol. In this case, R must reconstruct m solely based on the information it receives from S. Since m has L bits, and R receives fewer than L bits of information, R cannot recover m with certainty from an information-theoretic perspective [Sha48]. Therefore, S has to send $\Omega(L)$ bits to every other party, incurring $\Omega(NL)$ bits in total. □

Lemma 8.3 (Indistinguishability at the start)**.** *Consider a party P with input m that is awake at the start of the protocol. For P at the start of the protocol, it is indistinguishable if there exists another party with input m that is awake at the start of the protocol.*

Proof. For P at the start of the protocol, an execution $\mathcal{E}_{many}$ where multiple parties have input m is indistinguishable from an execution $\mathcal{E}_{lone}$ where P is the only party with input m. This is because the local view of a party P at the start of the protocol (before any messages are sent or received) consists solely of its initial input and the protocol code. In both $\mathcal{E}_{many}$ and $\mathcal{E}_{lone}$, P's input is m. Since P has not yet received any messages from other parties, its local history is empty in both scenarios. Consequently, the local view of P is identical in both executions. □

We now combine Lemma 8.2 and Lemma 8.3 to prove the lower bound of DT. Consider an execution $\mathcal{E}^*$ where a set of parties $\mathcal{S}$ are awake at the start of the protocol with input m. The

remaining parties awake at the start of the protocol have input $\perp$. For every party $P_i \in \mathcal{S}$, by Lemma 8.3, P_i cannot distinguish execution $\mathcal{E}^*$ from an execution where P_i is the only party with input m. In that execution, by Lemma 8.2, P_i must send $\Omega(NL)$ bits at the start of the protocol to ensure all parties receive m eventually, which is required by termination of DT. Thus, P_i needs to adopt the same strategy in execution $\mathcal{E}^*$ and send $\Omega(NL)$ bits at the start of the protocol. Since there are up to $O(N)$ such parties in $\mathcal{S}$, the total communication is $\Omega(N^2L)$. Since we have shown that the lower bound of DT also applies to Π, the communication lower bound for Π is $\Omega(N^2L)$. □

We now establish the communication lower bound for ABC:

Theorem 8.4. *Let Π be any protocol that solves atomic broadcast among N parties in the sleepy or recovery model with adaptive sleepiness. Even in the absence of byzantine corruptions, there exists an execution of Π in which a value of length L is delivered, and the total communication complexity is $\Omega(N^2L)$.*

Proof. Like how we prove theorem 8.1, we show that every ABC protocol Π can be used as a black-box to construct a solution to DT via the following construction: let P_i be a party that is awake at the start of the protocol. If P_i has input m, it invokes Π with input m at the start of the protocol; if P_i has input $\perp$, it does not invoke Π at the start of the protocol, instead, it joins the execution of Π one round later. Since there are no byzantine parties, all the parties that invoke Π at the start of the protocol have input m, and there is at least one such party. By liveness of Π, every party that is awake after some time T outputs a log that contains m. Since there are no honest inputs other than m, by safety of Π, every honest party's log is either empty or only contains one value m. Thus, the DT protocol is secure. Since DT reduces to Π with no additional communication overhead, the communication lower bound for DT also applies to Π. The communication lower bound for Π is $\Omega(N^2L)$ per value. □

9 Related Work

Byzantine fault-tolerant (BFT) consensus [LSP82] is arguably one of the most fundamental problems in distributed computing and cryptography. It has been studied extensively for several decades, traditionally under the static and known participation model [LSP82, CL99, PSL80, DS83, BT85, FLP83]. The advent of Nakamoto's Bitcoin protocol [Nak08] showed that consensus is possible under dynamic participation [GKL15]. This dynamic and unknown participation setting was later formalized by Pass and Shi [PS17] as the sleepy model.

Longest-chain paradigm. Early work on sleepy consensus largely followed the longest-chain paradigm pioneered by Bitcoin. They replaced the computationally intensive proof-of-work paradigm with proof-of-stake, while retaining the same structural backbone [PS17, BGK+18, KRDO17, DPS19]. While these protocols achieve dynamic participation communication-efficiently, they suffer from long latency same as bitcoin inherently, especially when the honest participation level fluctuates. Concretely, the latency is $O(\frac{\kappa\Delta}{\gamma})$, where κ is the security parameter, γ is the actual participation level, and Δ is the upper bound of network delay.

This bottleneck sparked a sequence of work aiming to reduce the latency by eliminating dependence on κ and γ. In particular, several protocols remove the dependence on κ by running multiple longest chains in parallel [BKT+19, FGKR18, LT20], but the dependence on the active participation level γ remains. On the other hand, Goyal, Li, and Raizes [GLR21] manages to eliminate γ

but κ remains. Also, they adopted committee sampling techniques [GHM+17], and thus required a stable presence of $\Omega(\kappa)$ parties throughout the execution, which limits the tolerance to participation churn. More recent works [DNTT24, DZ24] can achieve constant latency in optimistic cases, i.e., when the honest participation level is high, but still has an expected latency relevant to κ or γ when the participation level is low.

Sleepy ABC with constant latency. To overcome this latency bottleneck, Momose and Ren [MR22] revisited the view-based structure of classical BFT protocols, and adapted it to the sleepy model. They restored certificate transferability with a new technique called time-shifted quorum. They were the first to achieve sleepy ABC with optimal resilience and constant expected latency. However, their protocol assumed eventually stable participation, and while asymptotically optimal, their protocol incurred relatively large constants. The follow-up work by Malkhi, Momose, and Ren [MMR23] removed the stable participation assumption, and achieved much shorter best-case and expected latency. They also supported growing adversary which allows the number of corrupted parties to grow as the overall participation level grows. Most recently, Yandamuri, Shrestha, Zanolini, and Nayak [YSZN26] proposed a protocol whose expected block-production interval adapts to the actual network delay, achieving one block every $O(\Delta_{ideal})$ time, where $\Delta_{ideal} < 2\delta$. The other subsequent works [ET25, DSTZ25, SHT+25] focused on further improving the expected latency and censorship resilience. We noticed that all these protocols rely on extensive message forwarding to reconcile parties' inconsistent local views, which incurs a communication cost of $\Omega(n^3)$. Alternatively, another work by Malkhi, Momose, and Ren [MMR22] achieved quadratic communication cost, but their protocol tolerates only $f < \frac{n}{3}$ byzantine parties. Apart from that, a work by Efron, Neu, and Pitassi [ENP25] proposed a new model called external adversary, in which byzantine parties cannot divulge information about their secret keys. With this restriction, fully-fluctuating participation becomes possible. They also constructed a protocol with optimal resilience and constant latency in the model.

Sleepy BA. The first BA protocol proven secure in the sleepy model was by Garay, Kiayias, and Leonardos [GKL15]. Their construction follows the longest-chain paradigm of bitcoin, so it inherently has a latency of $O(\frac{\kappa\Delta}{\gamma})$. Subsequently, Goyal, Li, and Raizes [GLR21] improved the latency to $O(\kappa\Delta)$. We noticed that while there are a range of works on sleepy ABC with constant latency, sleepy BA with constant latency has not been systematically studied, until recently, Efron and Tas [ET25] presented a BA protocol that achieves constant expected latency and cubic expected communication complexity. Their protocol also achieves deterministic synchronization, meaning that once an honest party outputs a value, the other honest parties also output values within constant rounds. Unfortunately, their work assumed static corruption and the adversary cannot grow.

Moreover, the work by Khanchandani and Wattenhofer [KW21] studied BA in a weaker model where the number of parties is unknown but remains fixed throughout the execution. Also, the honest parties can notify the network about their leave before they go to sleep. The work by Gafni and Losa [LG23] achieved sleepy BA with optimal resilience and expected constant latency, but they also assumed a weaker model where there are no backward simulation. Very recently, a work [ENRT25] provided a complete characterization of the optimal good-case latency for BA and BB in the synchronous sleepy model.

Acknowledgment. We would like to thank the anonymous reviewers at ACM CCS 2026 for their valuable comments. This work is supported in part by ARC Discovery Project (DP250101739),

research awards from the Stellar Development Foundation, and Input Output (iog.io) as part of the Cardano Vision 2026 research initiative.

References

[ACD+19] Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *PODC*, pages 317–326. ACM, 2019.

[ADD+19] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected O(1) rounds, expected $o(n^2)$ communication, and optimal resilience. In *Financial Cryptography*, volume 11598 of *Lecture Notes in Computer Science*, pages 320–334. Springer, 2019.

[AMN+20] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *SP*, pages 106–118. IEEE, 2020.

[BGJ+23] Leemon Baird, Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Threshold signatures in the multiverse. In *SP*, pages 1454–1470. IEEE, 2023.

[BGK+18] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *CCS*, pages 913–930. ACM, 2018.

[BKT+19] Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *CCS*, pages 585–602. ACM, 2019.

[BT85] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.

[CASD95] Flaviu Cristian, Houtan Aghili, H. Raymond Strong, and Danny Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. *Inf. Comput.*, 118(1):158–179, 1995.

[CL99] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, pages 173–186. USENIX Association, 1999.

[DNTT24] Francesco D'Amato, Joachim Neu, Ertem Nusret Tas, and David Tse. Goldfish: No more attacks on ethereum?! In *FC (1)*, volume 14744 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2024.

[DPS19] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography*, volume 11598 of *Lecture Notes in Computer Science*, pages 23–41. Springer, 2019.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.

[DSTZ25] Francesco D'Amato, Roberto Saltini, Thanh-Hai Tran, and Luca Zanolini. TOB-SVD: total-order broadcast with single-vote decisions in the sleepy model. In *ICDCS*, pages 1033–1043. IEEE, 2025.

[DZ24] Francesco D'Amato and Luca Zanolini. Recent latest message driven GHOST: balancing dynamic availability with asynchrony resilience. In *CSF*, pages 127–142. IEEE, 2024.

[ENP25] Yuval Efron, Joachim Neu, and Toniann Pitassi. Fully-fluctuating participation in sleepy consensus. In *AFT*, volume 354 of *LIPIcs*, pages 17:1–17:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[ENRT25] Yuval Efron, Joachim Neu, Ling Ren, and Ertem Nusret Tas. Optimal good-case latency for sleepy consensus. *IACR Cryptol. ePrint Arch.*, page 1856, 2025.

[ET25] Yuval Efron and Ertem Nusret Tas. Dynamically available common subset. *IACR Cryptol. ePrint Arch.*, page 16, 2025.

[FGKR18] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. *IACR Cryptol. ePrint Arch.*, page 1119, 2018.

[FLP83] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. In *PODS*, pages 1–7. ACM, 1983.

[GHM+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68. ACM, 2017.

[GJM+24] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hints: Threshold signatures with silent setup. In *SP*, pages 3034–3052. IEEE, 2024.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

[GLR21] Vipul Goyal, Hanjun Li, and Justin Raizes. Instant block confirmation in the sleepy model. In *Financial Cryptography (2)*, volume 12675 of *Lecture Notes in Computer Science*, pages 65–83. Springer, 2021.

[KK06] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 445–462. Springer, 2006.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO (1)*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388. Springer, 2017.

[KW21] Pankaj Khanchandani and Roger Wattenhofer. Byzantine agreement with unknown participants and failures. In *IPDPS*, pages 952–961. IEEE, 2021.

[LG23] Giuliano Losa and Eli Gafni. Consensus in the unknown-participation message-adversary model. *CoRR*, abs/2301.04817, 2023.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[LT20] Songze Li and David Tse. Taiji: Longest chain availability with BFT fast confirmation. *CoRR*, abs/2011.11097, 2020.

[MMR22] Dahlia Malkhi, Atsuki Momose, and Ling Ren. Byzantine consensus under fully fluctuating participation. *IACR Cryptol. ePrint Arch.*, page 1448, 2022.

[MMR23] Dahlia Malkhi, Atsuki Momose, and Ling Ren. Towards practical sleepy BFT. In *CCS*, pages 490–503. ACM, 2023.

[MR22] Atsuki Momose and Ling Ren. Constant latency in sleepy consensus. In *CCS*, pages 2295–2308. ACM, 2022.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[PS17] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *ASIACRYPT (2)*, volume 10625 of *Lecture Notes in Computer Science*, pages 380–409. Springer, 2017.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT (2)*, volume 10211 of *Lecture Notes in Computer Science*, pages 643–673, 2017.

[QT23] Tian Qiu and Qiang Tang. Predicate aggregate signatures and applications. In *ASIACRYPT (2)*, volume 14439 of *Lecture Notes in Computer Science*, pages 279–312. Springer, 2023.

[Ren19] Ling Ren. Analysis of nakamoto consensus. *IACR Cryptol. ePrint Arch.*, page 943, 2019.

[SES+16] Yusuke Sakai, Keita Emura, Jacob C. N. Schuldt, Goichiro Hanaoka, and Kazuo Ohta. Constructions of dynamic and non-dynamic threshold public-key encryption schemes with decryption consistency. *Theor. Comput. Sci.*, 630:95–116, 2016.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3):379–423, 1948.

[SHT+25] Hans Schmiedel, Runchao Han, Qiang Tang, Ron Steinfeld, and Jiangshan Yu. Constant latency and finality for dynamically available DAG. In *SP*, pages 1910–1927. IEEE, 2025.

[SS94] Hung-Min Sun and Shiuh-Pyng Shieh. On dynamic threshold schemes. *Inf. Process. Lett.*, 52(4):201–206, 1994.

[YSZN26] Sravya Yandamuri, Nibesh Shrestha, Luca Zanolini, and Kartik Nayak. Low-latency dynamically available total order broadcast. In *FC*. Springer, 2026.