

Optimal Byzantine Atomic Broadcast with Message Drops

Dalin Zheng^{1,2}, Yuchen Ye³, Xiaolin Gui², Qiang Tang³, and Zhenliang Lu^{1✉}

¹ City University of Hong Kong, Hong Kong, China
dalizheng2-c@my.cityu.edu.hk, zhenliang.lu@cityu.edu.hk

² Xi'an Jiaotong University, Xi'an, China
xlgui@mail.xjtu.edu.cn

³ The University of Sydney, Sydney, Australia
yuchenye19@gmail.com, qiang.tang@sydney.edu.au

Abstract. Recent works have studied Byzantine agreement (BA) in the mixed-fault model with resilience threshold $n > 2t + r + s$, where n is the number of parties, t is the number of Byzantine parties, r is the number of receive-faulty parties, and s is the number of send-faulty parties. In particular, Feng et al. (ASIACRYPT 2025) proposed a BA protocol with optimal resilience, expected asymptotically quadratic communication complexity, and expected constant round complexity. However, existing BA protocols guarantee a meaningful output v only when all parties hold the same input v . Such a guarantee is insufficient for efficient construction of atomic broadcast protocols, leaving efficient atomic broadcast in the mixed-fault setting as an open problem.

In this paper, we present the first atomic broadcast protocol (UABC) with optimal resilience in the mixed-fault model. Our protocol achieves communication complexity $\mathcal{O}(n\ell + n^2\lambda^2)$ when $t = \Theta(n)$, where ℓ is the input size and λ is the security parameter. This complexity is asymptotically optimal for moderately large ℓ , namely when $\ell \geq n\lambda^2$. At the core of our design is an undead verifiable information dispersal (UVID) protocol, which allows an input to be efficiently dispersed and later reconstructed in the mixed-fault setting. Leveraging UVID together with the undead multi-valued consensus (ASIACRYPT 2025), we further construct a new primitive called weak undead multi-valued validated Byzantine agreement (wUMVBA) and its optimized variant (Opt-wUMVBA). Unlike standard multi-valued validated Byzantine agreement, wUMVBA and Opt-wUMVBA guarantee that the output is either a meaningful value satisfying the validity predicate or $\perp$. Moreover, the probability of outputting $\perp$ is at most $(t + s)/n$. Finally, by leveraging Opt-wUMVBA, we obtain our optimal atomic broadcast protocol for the mixed-fault setting.

Keywords: Atomic Broadcast · Omission Fault · Byzantine Agreement

1 Introduction

Atomic broadcast (ABC), also known as total-order broadcast, is a fundamental primitive in blockchain systems and distributed computing. It guarantees that

all honest parties deliver the same sequence of values in the same order [10, 15]. Informally, ABC must simultaneously guarantee two fundamental properties: *Safety* and *Liveness*. Safety requires that all honest parties eventually agree on the same sequence of transactions, while Liveness requires that every transaction proposed by an honest party is eventually delivered by all honest parties.

Classical Byzantine fault-tolerant protocols assume reliable communication among honest parties [2, 11, 14, 16, 20, 22, 24, 26, 29, 33, 35, 43, 47]. However, this assumption is often unrealistic in modern wide-area deployments, where messages may be lost due to network failures, congestion, or adversarial interference with communication infrastructure. To capture such behaviors, these works [1, 21, 23, 27, 30–32, 37, 39, 42, 48] consider the mixed-fault model, which combines Byzantine faults with communication omission faults. In this model, a send-faulty party, also called a *ghost*, faithfully follows the protocol but may have its outgoing messages suppressed, while a receive-faulty party, also called a *zombie*, faithfully follows the protocol but may fail to receive incoming messages. Byzantine parties remain fully adversarial.

The first optimal undead multi-valued Byzantine agreement [21]. Most recently, Feng et al. [21] and Loss et al. [31] considered the overlapping mixed-fault setting, where a party may simultaneously experience both send-omission and receive-omission faults and is counted toward both fault thresholds. Under this model, the optimal resilience condition is known to be $n > 2t + r + s$, where t denotes the number of Byzantine parties, s the number of send-faulty parties, and r the number of receive-faulty parties. In particular, Feng et al. [21] obtained an optimally resilient undead multi-valued consensus (UMVC) protocol with expected quadratic communication complexity and expected constant round complexity. Their UMVC protocol guarantees a meaningful output when all non-receive-faulty honest parties hold the same input value, while omission-faulty parties may use either this value or $\perp$ as input (see Definition 1). Furthermore, the UMVC construction incurs communication complexity $\mathcal{O}(n\ell + n^2\lambda^2)$ when $t = \Theta(n)$, where ℓ is the input size.

The absence of optimal atomic broadcast in the mixed-fault setting. Despite these advances, atomic broadcast in the mixed-fault setting remains largely unexplored. Existing undead consensus protocols only guarantee meaningful outputs when all non-receive-faulty honest parties hold the same input value v , while receive-faulty parties may use either v or $\perp$ as input. This limitation makes it difficult to directly construct atomic broadcast in such mixed-fault settings. In particular, atomic broadcast requires honest parties to continuously output transactions while still guaranteeing that values proposed by honest parties are eventually delivered, let alone achieving optimal communication complexity. In the Byzantine-only setting, the standard approach for achieving efficient long-input agreement is to separate data dissemination from consensus. For example, Dumbo-MVBA [33] first disperses each large value and produces a compact cryptographic certificate. Consensus is then executed only on these compact certificates rather than directly on ℓ -bit values, after which parties reconstruct the agreed value through a retrieval procedure. This design paradigm

avoids repeatedly transmitting large inputs during agreement and has become a standard technique for achieving communication-efficient Byzantine agreement.

However, extending this paradigm to the mixed-fault setting is highly non-trivial. Omission faults fundamentally complicate data dispersal and recovery: some honest parties may fail to receive fragments, while others may be unable to disseminate them reliably to the network. As a result, it remains unclear how to simultaneously achieve atomic broadcast with optimal resilience and optimal communication complexity in the mixed-fault setting. This leaves open the problem of constructing efficient atomic broadcast in the mixed-fault setting.

1.1 Our Contributions

In this work, we address this problem by presenting the first atomic broadcast protocol that simultaneously achieves optimal resilience and optimal communication complexity in the mixed-fault setting. Our main contributions are summarized as follows:

1. We present the first undead atomic broadcast (UABC) protocol in the mixed-fault setting. Our protocol achieves optimal resilience $n > 2t + r + s$ and expected communication complexity $\mathcal{O}(n\ell + n^2\lambda^2)$ when $t = \Theta(n)$. For inputs of size $\ell \geq n\lambda^2$, this complexity is asymptotically optimal.
2. We also introduce the first undead verifiable information dispersal (UVID) in the synchronous mixed-fault setting. Similar to asynchronous verifiable information dispersal protocols [4, 9], UVID consists of two stages: dispersal and retrieval. The dispersal stage is further divided into two sub-phases: dissemination and amplification. Importantly, the amplification phase does not need to be executed immediately after dissemination and can be delayed to a later time. In the dissemination phase, it has communication complexity $\mathcal{O}(\ell + n\lambda)$ and message complexity $\mathcal{O}(n)$. In the amplification phase, the communication complexity is $\mathcal{O}(n\ell + n^2\lambda)$ and the message complexity is $\mathcal{O}(n^2)$. Finally, the retrieval phase incurs communication complexity $\mathcal{O}(n\ell + n^2\lambda)$ and message complexity $\mathcal{O}(n^2)$.
3. To achieve efficient atomic broadcast, we also construct the first weak multi-valued validated Byzantine agreement protocols (MVBA) in the mixed-fault setting, including wUMVBA and its optimized variant Opt-wUMVBA. The wUMVBA protocol achieves communication complexity $\mathcal{O}(n^2\ell + n^3\lambda + n^2\lambda^2)$ and message complexity $\mathcal{O}(n^3)$, while Opt-wUMVBA further improves the complexity to $\mathcal{O}(n\ell + n^2\lambda^2)$ communication and $\mathcal{O}(n^2)$ messages. Both protocols output either a common externally valid value or $\perp$, with probability $\Pr[\text{output} = \perp] \leq (t + s)/n$.

1.2 Technical Overview

Although dispersal followed by agreement has become a standard paradigm, extending this approach to the mixed-fault setting is highly non-trivial. In the Byzantine-only setting, a threshold certificate on a vector commitment typically provides two guarantees simultaneously: first, the commitment is bound to a

unique valid value, and second, sufficiently many honest parties store fragments that later allow the value to be reconstructed. Under omission faults, however, these guarantees no longer hold automatically. Moreover, efficiently leveraging such a certificate to construct atomic broadcast introduces additional difficulties. This gives rise to the following two main issues.

Issue 1: Binding a certificate to a unique valid value and guaranteeing later retrieval. In the mixed-fault setting, a Byzantine sender may distribute coded fragments only to send-faulty honest parties or to receive-faulty honest parties that may later be unable to provide valid fragments during retrieval, while still obtaining a valid threshold signature. Consequently, obtaining a threshold signature on a vector commitment is insufficient to guarantee that the commitment corresponds to a unique externally valid value that has been dispersed to sufficiently many non-send-faulty and non-receive-faulty honest parties to ensure later retrieval. To achieve these guarantees, we let the sender first distribute coded fragments to all parties. Subsequently, every party forwards its received fragment to all other parties and attempts to reconstruct the value. This step is inspired by reliable broadcast protocols [7, 9], whose goal is to ensure that all non-receive-faulty honest parties reconstruct the same value. After reconstruction, parties verify the reconstructed value and only then provide threshold signature shares. The threshold is set to $t + 1$, ensuring that once one honest party contributes a valid signature share, the corresponding value has already been successfully dispersed and all non-receive-faulty honest parties have received the corresponding fragments required for later retrieval.

However, ensuring successful dispersal alone is not sufficient. If all honest parties hold the same valid certificate, we must further ensure that they eventually output the same value v , while receive-faulty honest parties (zombies) may output either v or $\perp$. At the same time, the protocol requires honest parties to participate in each component within a synchronized time window. This is difficult to guarantee during the retrieval phase because the underlying undead multi-valued consensus (UMVC) protocol [21] does not guarantee that all honest parties terminate simultaneously. Hence, we let each party wait for an additional $(k + 1)\Delta$ time before outputting, where Δ denotes the upper bound on network delay. The key observation is that if the earliest honest party outputs from UMVC at time t , then the latest honest party outputs by time $t + k\Delta$. Therefore, after waiting for $(k + 1)\Delta$, the earliest honest party is guaranteed to have received sufficiently many fragments to reconstruct the value.

Issue 2: Constructing efficient atomic broadcast. The existing undead consensus protocol UMVC [21] is insufficient for directly constructing atomic broadcast with our target communication complexity. First, as discussed earlier, UMVC only guarantees meaningful outputs when all non-receive-faulty honest parties hold the same input value. A direct approach would have every party broadcast its input to all other parties, elect a leader, and then use the value received from the elected leader as input to UMVC. However, broadcasting all ℓ -bit input values in this manner already incurs $\mathcal{O}(n^2\ell)$ communication. To address this limitation, we construct weak undead multi-valued validated Byzantine agree-

ment, denoted **wUMVBA**, together with its optimized variant **Opt-wUMVBA**. The term “weak” reflects that, unlike standard multi-valued validated Byzantine agreement [8, 33], our protocol does not guarantee that every execution outputs a value satisfying the predefined validity predicate. Instead, the protocol may output $\perp$. However, the probability of outputting $\perp$ is at most $(t + s)/n$, which is sufficient for constructing atomic broadcast. The optimized protocol **Opt-wUMVBA** achieves communication complexity $\mathcal{O}(n\ell + n^2\lambda^2)$ for large input sizes. Therefore, repeatedly executing **Opt-wUMVBA** instances naturally leads to an efficient atomic broadcast construction.

A further difficulty arises when repeatedly executing **Opt-wUMVBA** instances to construct atomic broadcast. In particular, **Opt-wUMVBA** relies on **UMVC** as a building block, while **UMVC** [21] does not guarantee that all honest parties terminate in the same synchronous round. In contrast, the dispersal protocol requires honest parties to participate within the same synchronized time window. Therefore, if the next dispersal instance starts only after the previous retrieval procedure terminates locally, honest parties may enter the next dispersal phase in different rounds. To overcome this difficulty, we start the next **Opt-wUMVBA** instance immediately after the undead common coin outputs, rather than waiting for the current instance to terminate completely. This ensures that all honest parties enter the next dispersal phase within the same synchronized time window, preventing the termination gap from carrying over to the next instance.

2 Related Work

Consensus in the mixed-fault model. Consensus in the synchronous setting has traditionally been studied under the assumption that all honest parties can communicate reliably with each other. However, this assumption is often unrealistic, as messages may be lost due to network disruptions, congestion, or adversarial interference with the communication infrastructure. To address this issue, early works initiated the study of agreement under processor and communication omission faults [27, 39]. Garay et al. [23] introduced a continuum of failure models and investigated agreement protocols tolerating both Byzantine and crash faults. Siu et al. [42] studied mixed processor and link failures and analyzed how omission faults affect the feasibility of Byzantine agreement. Koo [30] further considered secure computation under partial message loss, demonstrating that omission faults can be incorporated into cryptographic protocols at the cost of stronger resilience conditions. Then Hauser et al. [48] refined the omission-fault model by distinguishing between send-omission and receive-omission faults.

Recent works have provided sharper characterizations of agreement under mixed faults. Specifically, Eldefrawy et al. [19] studied the impact of send corruptions and established lower bounds showing that Byzantine agreement with authentication requires $n > 2t + s + r$, where t denotes the number of Byzantine parties, s denotes the number of send-omission faults, and r denotes the number of receive-omission faults. Loss et al. [32] matched this resilience bound in the non-overlapping mixed-fault setting where a party cannot be send-faulty and

receive-faulty at the same time, while Loss et al. [31] further extended the feasibility result to overlapping settings. More recently, Feng et al. [21] proposed an optimally resilient binary Byzantine agreement protocol achieving optimal expected quadratic communication complexity and expected constant round complexity. They also presented a multi-valued Byzantine agreement protocol with an expected communication complexity of $\mathcal{O}(n^2\ell/t + n^2\lambda^2)$, which is optimal when $t = \Theta(n)$. Kamp et al. [28] further reduced the round complexity while preserving quadratic communication complexity, at the cost of weaker resilience.

Multi-valued validated Byzantine agreement. Multi-valued validated Byzantine agreement (MVBA) enables parties to agree on a value satisfying a predefined global function. The notion was first introduced by Cachin et al. [8] and later improved by Abraham et al. [3], who achieved quadratic communication complexity in the Byzantine setting. Dumbo-MVBA [33] further popularized the now-standard dispersal-then-retrieval architecture, in which parties first disperse large inputs and then run consensus only on compact metadata, such as vector commitments, rather than on the actual input values. After consensus terminates, parties execute a retrieval protocol to reconstruct the agreed value.

Under the mixed-fault setting, there are currently no existing MVBA protocols. Existing works [21, 31, 32] only proposed undead consensus algorithms, which only guarantee meaningful outputs when all the inputs are identical (except that in the undead binary consensus protocol of Feng et al. [21], send-faulty and receive-faulty parties may also input $\perp$). In this work, we propose a weak MVBA protocol component in the mixed-fault setting. Our protocol guarantees that the output is either a meaningful value v satisfying the validity predicate or $\perp$. Furthermore, the probability of outputting $\perp$ is at most $(t + s)/n$.

Atomic broadcast. Atomic broadcast has been extensively studied under various network settings, including synchronous, partially synchronous, and asynchronous models [2, 5, 6, 11–13, 22, 25, 26, 29, 40, 43, 44, 46, 47]. State-of-the-art atomic broadcast protocols can achieve optimal communication complexity together with constant or expected constant round complexity. However, most existing works focus only on the Byzantine setting. In contrast, a related line of work studies atomic broadcast under dynamic participation. In the sleepy model, parties may temporarily go offline and later return. Pass and Shi [38] initiated the study of consensus in this model. Subsequent works [17, 18, 34, 36, 41, 45] further explored how to remove stable participation assumptions, improve latency and communication complexity, and achieve censorship resilience. However, regarding the mixed-fault setting, to the best of our knowledge, no atomic broadcast protocols have been proposed in the mixed-fault model considered in this paper.

3 System Model

In this work, we adopt the model of [21, 31]. We work in a synchronous network model. A message sent by an honest party at time t is guaranteed to be delivered to its recipient by time $t + \Delta$, where Δ denotes the upper bound on communication delay. There are a total of n parties $\mathcal{P} = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$, which are pairwise

connected through an authenticated channel. We assume a trusted setup for the cryptographic primitives used in the protocol. Each party holds its share of the threshold signing key, while all public parameters are known to every party.

Throughout the work, we consider the following four types of faults:

- **Send-Omission Faults.** A party that is experiencing a send-omission fault is said to be send-faulty. The adversary may selectively prevent any messages sent by the party from being delivered. A party that identifies itself as being send-faulty is referred to as a *ghost*.
- **Receive-Omission Faults.** A party that is experiencing a receive-omission fault is said to be receive-faulty. The adversary may selectively prevent any messages sent to the party from being delivered. A party that identifies itself as being receive-faulty is referred to as a *zombie*.
- **Full-Omission Faults.** A party that is experiencing a full-omission fault is said to be full-omission faulty. The party can be seen as experiencing both the send-omission fault and the receive-omission fault. The adversary may selectively prevent any messages sent by and sent to the party from being delivered.
- **Byzantine Faults.** A Byzantine party is fully controlled by the adversary. Its internal state is leaked and it may behave arbitrarily.

We note that send-faulty, receive-faulty, and full-omission-faulty parties are still honest parties, meaning that they follow the protocol faithfully. For convenience, we refer to honest parties that do not experience send-omission faults, receive-omission faults, and full-omission faults as **non-send-faulty**, **non-receive-faulty**, and **non-full-omission-faulty** honest parties, respectively.

We assume that there are at most t Byzantine parties, r receive-faulty parties, and s send-faulty parties, under the tight feasibility condition $n > 2t + r + s$. Throughout the paper, we consider $t = \Theta(n)$, corresponding to the regime where the number of Byzantine parties is as large as possible. A full-omission faulty party is counted towards both send-faulty and receive-faulty parties. The adversary is *rushing* and *strongly adaptive*, meaning that it may corrupt any party at any time and subject the corrupted party to any of the three fault types described above.

4 Preliminaries

This section introduces the notation and basic primitives. We use $[n]$ to denote the set $\{1, \dots, n\}$, and λ to denote the security parameter.

4.1 Cryptographic Primitives

Erasure code. We use a deterministic (k, n) -erasure coding scheme with two algorithms, EC.Encode and EC.Decode . Specifically, $\text{EC.Encode}(n, k, v) \rightarrow (c_1, \dots, c_n)$ encodes a value v into n coded fragments, and $\text{EC.Decode}(n, k, \{(j, c_j)\}_{j \in S}) \rightarrow v$ reconstructs v from any subset $S \subseteq [n]$ with $|S| = k$. Throughout this paper, we

consider $k = t + 1$. Consequently, any set of $t + 1$ valid coded fragments uniquely determines the encoded value.

Threshold signature scheme. We use a non-interactive (t, n) -threshold signature scheme TSS with five algorithms, TSS.Setup, TSS.PartSign, TSS.PartVrfy, TSS.Combine, and TSS.Vrfy. Specifically, given a security parameter λ and parameters (n, t) , TSS.Setup outputs a master public key mpk , a vector of public keys $\{pk_i\}_{i \in [n]}$, and a vector of secret key shares $\{sk_i\}_{i \in [n]}$. Given a message m and a secret key share sk_i , TSS.PartSign outputs a signature share σ_i . Given a message m , a public key pk_i , and a signature share σ_i , TSS.PartVrfy verifies whether σ_i is a valid signature share generated by party $\mathcal{P}_i$. Given a message m and a set of valid signature shares $\{(j, \sigma_j)\}_{j \in S}$, where $S \subseteq [n]$ and $|S| = t + 1$, TSS.Combine outputs a combined signature Σ . Finally, TSS.Vrfy verifies whether Σ is a valid threshold signature on m .

Position-binding vector commitment. We use a position-binding n -vector commitment (VC) scheme with three algorithms, VC.Commit, VC.Open, and VC.Verify. Specifically, VC.Commit commits to a vector $(m_1, \dots, m_n)$ and outputs a commitment vc . Given a commitment vc , an element m_i , and an index $i \in [n]$, VC.Open(vc, m_i, i) outputs an opening proof π_i proving that m_i is committed at position i . Finally, VC.Verify verifies whether π_i is a valid opening proof for m_i at position i under commitment vc . For simplicity, we omit the auxiliary information utilized in these algorithms.

4.2 Consensus Primitives

Undead common coin (UCOIN). The undead common coin primitive essentially serves as a standard source of randomness. We employ the undead common coin from Loss-Stern [32]. Once at least $t + 1$ honest parties participate in the protocol, all non-receive-faulty parties obtain the same uniformly random binary value, which remains unpredictable to the adversary. In addition, any party that fails to receive the binary value can detect itself as a zombie. Informally, the protocol has the following properties except with negligible probability:

- **Termination.** If at least $t + 1$ honest parties participate in the protocol, then all non-receive-faulty parties output the same uniformly random binary value.
- **Agreement.** Upon completion of the invocation, all non-zombie honest parties output the same value.
- **Unpredictability.** For any given invocation, the output remains indistinguishable to the adversary until at least one honest party invokes the common coin protocol.

In our instantiation, the coin is realized using a unique $(t + 1, n)$ threshold-signature-based common coin. For each coin instance, every party signs the iteration identifier and multicasts its partial signature. Every non-receive-faulty party receives at least $t + 1$ valid partial signatures from distinct parties. Consequently, they can generate a unique threshold signature and derive the coin

by hashing the resulting signature. Parties that receive fewer than $t + 1$ valid partial signatures become zombies and terminate at the end of the iteration. This construction requires only one communication round and incurs $\mathcal{O}(n^2\lambda)$ communication complexity.

Undead multi-valued consensus (UMVC). The undead multi-valued consensus protocol involves n parties, each holding an input value. The goal is to enable the parties to reach agreement on an output value even when some honest parties experience send or receive faults. Formally, the definition is as follows:

Definition 1 (Undead Multi-valued Consensus (UMVC)). *The UMVC protocol [21] is executed among n parties $\mathcal{P}_1, \dots, \mathcal{P}_n$ in the mixed-fault model with at most t Byzantine parties, r receive-faulty parties, and s send-faulty parties, where $n > 2t + r + s$. Each party $\mathcal{P}_i$ holds an input triplet (v_i, z_i, g_i) , where $v_i \in \{\{0, 1\}^{|m|}, \perp\}$ and $z_i, g_i \in \{0, 1\}$. Here, $|m|$ represents the input size, $z_i = 1$ indicates that $\mathcal{P}_i$ considers itself receive-faulty (a zombie), while $g_i = 1$ indicates that $\mathcal{P}_i$ considers itself send-faulty (a ghost). Every party $\mathcal{P}_i$ terminates upon outputting a triplet (v'_i, z'_i, g'_i) , where $v'_i \in \{\{0, 1\}^{|m|}, \perp\}$ and $z'_i, g'_i \in \{0, 1\}$. More formally, an UMVC protocol satisfies the following properties except with negligible probability:*

- **Validity.** *If all the honest non-zombie parties⁴ hold the same input v , and every zombie holds either v or $\perp$, then every honest party $\mathcal{P}_i$ either outputs $v'_i = v$ or becomes a zombie, i.e., $z'_i = 1$.*
- **Agreement.** *If two honest parties $\mathcal{P}_i$ and $\mathcal{P}_j$ are not zombies at the end of the protocol and output (v'_i, z'_i, g'_i) and (v'_j, z'_j, g'_j) , respectively, then $v'_i = v'_j$.*
- **Termination.** *All honest parties eventually output and terminate.*
- **Detection Soundness.** *If an honest party outputs $z'_i = 1$ (resp. $g'_i = 1$), then it is receive-faulty (resp. send-faulty).*

In this paper, we utilize the UMVC protocol by Feng et al. [21]. The protocol achieves an expected communication complexity of $\mathcal{O}(\frac{n^2}{t}|m| + \lambda^2 n^2)$, an expected message complexity of $\mathcal{O}(n^2)$, and an expected round complexity of $\mathcal{O}(1)$, where $|m|$ represents the input size, and λ is the security parameter. When $t = \Theta(n)$, the communication cost reduces to $\mathcal{O}(n|m| + \lambda^2 n^2)$, which is optimal.

⁴ In the original definition of [21], validity is required only when all honest parties hold the same input v . Here, we slightly strengthen the property by additionally allowing zombie parties to hold $\perp$ as input. We emphasize that this modified validity property can be achieved by slightly modifying the original undead multi-valued consensus construction in [21]. Specifically, in the first step, if a party $\mathcal{P}_i$ has input $v_i = \perp$, then it can set its VC_i to a vector of length sz filled with $\perp$, where sz is the bit-length of a vector commitment. Then the parties invoke undead weak binary consensus (WBC) on each bit of VC_i , where $\mathcal{P}_i$ inputs $\perp$. Since WBC allows $\perp$ as receive-faulty parties' input value and still has validity, if all honest non-zombie parties input v and every zombie inputs either v or $\perp$, the bit-wise WBC outputs still correspond to the vector commitment of v , exactly as in the original protocol. No other parts of the construction in [21] need to be modified.

5 Undead Verifiable Information Dispersal

In this section, we introduce our core component, *Undead Verifiable Information Dispersal* (UVID), which is utilized in atomic broadcast to achieve optimal communication complexity. The protocol consists of two subprotocols: DISPERSE and RETRIEVE. During the dispersal phase, the DISPERSE protocol transforms a long value into a short certificate through a *dispersal-check-redispersal* procedure. The resulting certificate also serves as proof of successful dispersal, guaranteeing that a valid value can later be reconstructed. During the reconstruction phase, all honest parties invoke the RETRIEVE protocol to recover the value corresponding to a valid certificate using the fragments stored during the dispersal phase.

To ensure that the reconstructed value is meaningful, we assume the existence of a public deterministic predicate function $\text{Predicate}_{\text{UVID}} : \{0, 1\}^* \rightarrow \{0, 1\}$. It is similar to the predicate function used in MVBA protocols [3, 8, 33]. Specifically, for every input value v held by an honest party and every reconstructed value v' , it holds that $\text{Predicate}_{\text{UVID}}(v) = 1$ and $\text{Predicate}_{\text{UVID}}(v') = 1$.

In UVID, every instance is identified by an identifier pair (ID, s) , where s denotes the identifier of the sender $\mathcal{P}_s$. In the DISPERSE subprotocol, the honest sender $\mathcal{P}_s$ always holds an input value v satisfying $\text{Predicate}_{\text{UVID}}(v) = 1$. Under the mixed-fault setting, we must guarantee that even if the sender later detects itself as a zombie or a ghost, any valid certificate generated during the dispersal phase still guarantees that the corresponding value can be successfully reconstructed. Moreover, even if sufficiently many honest parties receive coded fragments during the dispersal phase, reconstructability is still non-trivial in the presence of Byzantine and ghost faults. In particular, some honest parties may receive valid fragments before realizing that they are ghosts. During the reconstruction phase, such ghost parties may be unable to send fragments to other non-receive-faulty honest parties. Consequently, honest parties may fail to collect enough valid fragments to reconstruct the value, even when a valid certificate exists. Formally, the UVID protocol satisfies the following properties:

Definition 2 (Undead Verifiable Information Dispersal (UVID)). *The UVID protocol is executed among n parties $\mathcal{P}_1, \dots, \mathcal{P}_n$ in the mixed-fault model with at most t Byzantine parties, r receive-faulty parties, and s send-faulty parties, where $n > 2t + r + s$. The protocol consists of two subprotocols, DISPERSE and RETRIEVE, and assumes the existence of a public deterministic predicate function $\text{Predicate}_{\text{UVID}} : \{0, 1\}^* \rightarrow \{0, 1\}$.*

For each instance $\text{DISPERSE}[ID, s]$, the honest sender $\mathcal{P}_s$ inputs a value v satisfying $\text{Predicate}_{\text{UVID}}(v) = 1$ and aims to encode v into n coded fragments and disperse one fragment to each party. Upon termination of $\text{DISPERSE}[ID, s]$, every party obtains fragments output $\text{fragment}[ID, s]$ and a certificate output $\text{dispersed}[ID, s]$. In the RETRIEVE subprotocol with identifier (ID, s) , denoted by $\text{RETRIEVE}[ID, s]$, all honest parties take a fragment from $\text{fragment}[ID, s]$ (output by $\text{DISPERSE}[ID, s]$) as input and aim to reconstruct the dispersed value v_s . Upon termination of $\text{RETRIEVE}[ID, s]$, each party outputs $\text{retrieved}[ID, s] := v_s$. Specifically, the output $\text{dispersed}[ID, s]$ has three possible forms:

- $\text{dispersed}[ID, s] := (\perp, \perp, z_i, g_i)$ if $z_i = 1$;
- $\text{dispersed}[ID, s] := (0, 0, z_i, g_i)$ if $z_i = 0$ but reconstruction cannot be guaranteed, where 0 serves as a default value;
- $\text{dispersed}[ID, s] := (VC_s, \Sigma_s, z_i, g_i)$ if $z_i = 0$, successful reconstruction can be guaranteed, and the reconstructed value v satisfies the predicate function, i.e., $\text{Predicate}_{\text{UVID}}(v) = 1$.

For simplicity, throughout the paper, whenever we write $\text{dispersed}[ID, s] := (VC_s, \Sigma_s, z_i, g_i)$ or $(VC_s, \Sigma_s, z_i, g_i)$, it is implicitly assumed that Σ_s is a valid $(t+1, n)$ -threshold signature on $ID \| s \| VC_s$. In addition, every party $\mathcal{P}_i$ maintains two status variables $z_i \in \{0, 1\}$ and $g_i \in \{0, 1\}$. Here, $z_i = 1$ indicates that $\mathcal{P}_i$ considers itself receive-faulty (a zombie), while $g_i = 1$ indicates that $\mathcal{P}_i$ considers itself send-faulty (a ghost). We say that an honest party is non-zombie at the end of DISPERSE if $z_i = 0$, and non-ghost if $g_i = 0$. More formally, UVID satisfies the following properties except with negligible probability:

- **Validity.** If the designated sender $\mathcal{P}_s$ is a non-send-faulty honest party and holds an input v_s satisfying $\text{Predicate}_{\text{UVID}}(v_s) = 1$, then every honest party $\mathcal{P}_i$ either outputs $\text{dispersed}[ID, s] := (VC_s, \Sigma_s, z_i, g_i)$ or becomes a zombie (i.e., $z_i = 1$) and outputs $\text{dispersed}[ID, s] := (\perp, \perp, z_i, g_i)$. Furthermore, if all non-receive-faulty honest parties invoke $\text{RETRIEVE}[ID, s]$ with a valid $\text{dispersed}[ID, s] := (VC_s, \Sigma_s, z_i, g_i)$, then every honest party either outputs $\text{retrieved}[ID, s] = v_s$ or becomes a zombie.
- **Termination.** Every honest party that invokes $\text{DISPERSE}[ID, s]$ eventually terminates and outputs $\text{fragment}[ID, s]$ and $\text{dispersed}[ID, s]$. Moreover, if every non-receive-faulty honest party invokes $\text{RETRIEVE}[ID, s]$ on a valid input $\text{dispersed}[ID, s] := (VC_s, \Sigma_s, z_i, g_i)$, then all honest parties eventually terminate and output a value v .
- **Agreement.** If all non-receive-faulty honest parties invoke $\text{RETRIEVE}[ID, s]$ on the same valid input (ID, s, VC_s, Σ_s) , then every honest party either outputs the same non- $\perp$ value or becomes a zombie.
- **Recast-ability.** If some non-receive-faulty honest party outputs a certificate $\text{dispersed}[ID, s] := (VC_s, \Sigma_s, z_i, g_i)$ from $\text{DISPERSE}[ID, s]$, then, if all non-receive-faulty honest parties invoke $\text{RETRIEVE}[ID, s]$ with $\text{dispersed}[ID, s] := (VC_s, \Sigma_s, z_i, g_i)$ as input, every honest party either eventually outputs a value v corresponding to the vector commitment VC_s or becomes a zombie.
- **External validity.** If a non-receive-faulty honest party outputs $v \neq \perp$ from the $\text{RETRIEVE}[ID, s]$ protocol, then $\text{Predicate}_{\text{UVID}}(v) = 1$.
- **Detection Soundness.** If an honest party outputs $z'_i = 1$ (resp. $g'_i = 1$), then it is receive-faulty (resp. send-faulty).

5.1 The DISPERSE Subprotocol

As illustrated in Figure 1, the UVID protocol with identifier (ID, s) consists of two phases: dissemination and amplification. In the dissemination phase, executed at time $\tau = 0$, the sender disperses its input to all parties through the

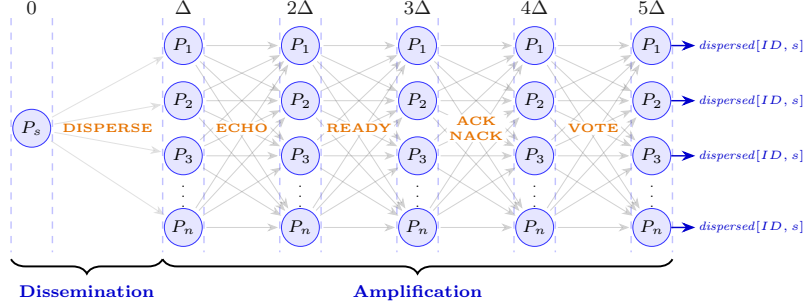


Fig. 1: The execution flow of UVID DISPERSE

DISPERSE message. In the amplification phase, executed from time $\tau = \Delta$ to $\tau = 5\Delta$, the parties verify that the sender has successfully dispersed its input and further re-disperse the coded fragments throughout the network to ensure that the value can later be reconstructed. The amplification phase proceeds through four message rounds, namely ECHO, READY, ACK, and VOTE, ultimately producing a valid certificate.

Algorithms 1 and 2 implement the DISPERSE of UVID with identifier (ID, s) for sender $\mathcal{P}_s$. Each party initializes four flags. The flag $z_i = 1$ indicates that $\mathcal{P}_i$ has detected itself as receive-faulty, i.e., a zombie. The flag $g_i = 1$ indicates that $\mathcal{P}_i$ has detected itself as send-faulty, i.e., a ghost. The flag $\text{lock}_s \neq 0$ indicates that $\mathcal{P}_i$ has locally reconstructed the sender's input value and verified that the vector commitment of the reconstructed value is consistent with the previously received fragments. The flag $\text{Locked}_s = 1$ indicates that $\mathcal{P}_i$ did not receive sufficiently many positive acknowledgments for its READY message. In this case, $\mathcal{P}_i$ considers itself unable to reliably re-disperse the reconstructed value to the entire network. The condition $|A_{i,s}^0| + |A_{i,s}^{\text{zom}}| < t + r + 1$ indicates that $\mathcal{P}_i$ received at least one positive acknowledgment for its READY message from a non-send-faulty honest party. This is because there are $n - t - s \geq t + r + 1$ non-send-faulty honest parties, if all of them send non-positive acknowledgments to $\mathcal{P}_i$, then $\mathcal{P}_i$ will receive at least $t + r + 1$ non-positive acknowledgments and thus become a ghost. By contradiction, at least one non-send-faulty honest party sent the positive acknowledgment. We then note that since the party is not send-faulty, it is not full-omission faulty as well by definition. This guarantees that the value corresponding to VC_s was successfully dispersed and can later be reconstructed if all honest parties invoke RETRIEVE. Therefore, $\mathcal{P}_i$ multicasts a VOTE with value 1 together with its threshold signature share to all parties.

Construction of DISPERSE. We assume that each instance identified by $[ID, s]$ lasts for 5Δ time. For simplicity, we denote time $(ID - 1) \cdot 5\Delta + k\Delta$ in the global execution as time $\tau = k\Delta$ of instance ID . Hence, each instance $\text{DISPERSE}[ID, s]$ starts at time $\tau = 0$. The formal description of DISPERSE is presented in Algorithms 1 and 2. At a high level, the protocol proceeds in two phases with five logical message rounds as follows:

Algorithm 1 UVID Dispersal subprotocol with identifier (ID, s) , code for $\mathcal{P}_i$:

Initialize: $z_i \leftarrow 0$, $g_i \leftarrow 0$, $\text{Lock}_s \leftarrow 0$, $\text{Locked}_s \leftarrow 0$, $\text{fragment}[ID, s] \leftarrow \{\}$ and $\text{dispersed}[ID, s] \leftarrow \emptyset$.

- 1: **if** $\tau = 0$ **then** $\triangleright$ Dissemination for sender $\mathcal{P}_s$
- 2: **if** $\mathcal{P}_i$ is the sender $\mathcal{P}_s$ **then**
- 3: $\overline{m}_s := (m_{s,1}, \dots, m_{s,n}) \leftarrow \text{EC.Encode}(n, t+1, v_s)$ $\triangleright$ Encode v_s
- 4: $VC_s \leftarrow \text{VC.Commit}(\overline{m}_s)$ $\triangleright$ Vector commitment
- 5: **for** $\forall j \in [n]$ **do**
- 6: $\pi_{s,j} \leftarrow \text{VC.Open}(\overline{m}_s, j)$
- 7: $\sigma_s \leftarrow \text{Sign}(sk_s, ID \| s \| VC_s)$ $\triangleright$ Digital signature
- 8: send $(\text{DISPERSE}, ID, s, VC_s, \sigma_s, m_{s,j}, \pi_{s,j})$ to $\mathcal{P}_j$ $\triangleright$ DISPERSE
- 9: **if** $\tau = \Delta$ **then** $\triangleright$ Amplification begins
- 10: **if** received a valid DISPERSE from $\mathcal{P}_s$ **then**
- 11: multicast $(\text{ECHO}, ID, s, 1, VC_s, \sigma_s, m_{s,i}, \pi_{s,i})$ $\triangleright$ ECHO
- 12: **else**
- 13: multicast $(\text{ECHO}, ID, s, 0)$ $\triangleright$ ECHO
- 14: **if** $\tau = 2\Delta$ **then**
- 15: **if** received fewer than $n - t - s$ ECHO from distinct parties **then**
- 16: $z_i \leftarrow 1$ $\triangleright$ detect zombie
- 17: **if** received $t+1$ valid $(\text{ECHO}, ID, s, 1, VC_s, *)$ from distinct parties **then**
- 18: let S be a set of $f+1$ fragments
- 19: $v'_s \leftarrow \text{EC.Decode}(n, t+1, S)$ $\triangleright$ Reconstruct v'_s
- 20: **if** $\text{Predicate}_{\text{wUMVBA}}(v'_s) = 1$ **then**
- 21: $\overline{m}'_s := (m'_{s,1}, \dots, m'_{s,n}) \leftarrow \text{EC.Encode}(n, t+1, v'_s)$
- 22: $VC'_s \leftarrow \text{VC.Commit}(\overline{m}'_s)$
- 23: **if** $VC'_s = VC_s$ **then**
- 24: **for** $\forall j \in [n]$ **do**
- 25: $\pi'_{s,j} \leftarrow \text{VC.Open}(\overline{m}'_s, j)$
- 26: send $(\text{READY}, ID, s, VC_s, m'_{s,j}, \pi'_{s,j})$ to $\mathcal{P}_j$ $\triangleright$ READY
- 27: $\text{Lock}_s \leftarrow VC_s$ $\triangleright$ Already reconstructed locally
- 28: **if** $\tau = 3\Delta$ **then**
- 29: **if** $z_i = 1$ **then**
- 30: multicast $(\text{ACK}, ID, s, \text{ZOMBIE})$ $\triangleright$ zombie report
- 31: **else**
- 32: **for** $\forall j \in [n]$ **do**
- 33: **if** received a valid $(\text{READY}, ID, s, VC_s, m'_{s,i}, \pi'_{s,i})$ message from $\mathcal{P}_j$ **then**
- 34: **if** $\text{Lock}_s = VC_s$ **then**
- 35: record $\text{fragment}[ID, s] \leftarrow \text{fragment}[ID, s] \cup (VC_s, m'_{s,i}, \pi'_{s,i})$
- 36: send $(\text{ACK}, ID, s, 1)$ to $\mathcal{P}_j$ $\triangleright$ ACK
- 37: **else** send $(\text{ACK}, ID, s, 0)$ to $\mathcal{P}_j$ $\triangleright$ ACK
- 38: **else** send $(\text{ACK}, ID, s, 0)$ to $\mathcal{P}_j$ $\triangleright$ ACK

Dissemination. At time $\tau = 0$ (Algorithm 1, lines 1-8), only the sender $\mathcal{P}_s$ performs the dissemination procedure. It first encodes v_s into $\overline{m}_s$ using the $(t+1, n)$ erasure code, then computes the vector commitment VC_s and signs $ID \| s \| VC_s$. For each party $\mathcal{P}_j$, the sender transmits the corresponding coded

Algorithm 2 UVID Dispersal subprotocol with identifier (ID, s) , code for $\mathcal{P}_i$ (continued):

```

39: if  $\tau = 4\Delta$  then
40:   if  $z_i = 1$  then
41:     multicast (VOTE,  $ID, s$ , ZOMBIE) ▷ zombie report
42:   else
43:      $A_{i,s}^1 \leftarrow \{j \in [n] : \mathcal{P}_i \text{ receives (ACK, } ID, s, 1) \text{ from } \mathcal{P}_j\}$ 
44:      $A_{i,s}^0 \leftarrow \{j \in [n] : \mathcal{P}_i \text{ receives (ACK, } ID, s, 0) \text{ from } \mathcal{P}_j\}$ 
45:      $A_{i,s}^{zom} \leftarrow \{j \in [n] : \mathcal{P}_i \text{ receives (ACK, } ID, s, \text{ZOMBIE) from } \mathcal{P}_j\}$ 
46:     if  $|A_{i,s}^1| + |A_{i,s}^0| + |A_{i,s}^{zom}| < n - t - s$  then ▷ detect zombie
47:        $z_i \leftarrow 1$ 
48:       multicast (VOTE,  $ID, s$ , ZOMBIE)
49:     else
50:       if  $|A_{i,s}^1| \leq t$  then ▷ READY not sufficiently confirmed
51:          $\text{Locked}_s \leftarrow 1$ 
52:         if  $\text{Lock}_s \neq 0$  and  $|A_{i,s}^0| + |A_{i,s}^{zom}| \geq t + r + 1$  then
53:            $g_i \leftarrow 1$  ▷ detect ghost
54:           if  $\text{Lock}_s \neq 0$  and  $\text{Locked}_s = 0$  and  $g_i = 0$  then
55:              $\rho_{i,s} \leftarrow \text{TSS.PartSign}(sk_i, ID \| s \| VC_s)$ 
56:             multicast (VOTE,  $ID, s, 1, VC_s, \rho_{i,s}$ ) ▷ VOTE
57:           else
58:             multicast (VOTE,  $ID, s, 0$ ) ▷ no positive share
59:   if  $\tau = 5\Delta$  then
60:     if  $z_i = 1$  then ▷ zombie
61:       output  $\text{dispersed}[ID, s] \leftarrow (\perp, \perp, z_i, g_i)$  ▷ output  $\perp$ 
62:     else
63:        $S_{i,s}^{yes} \leftarrow \{j \in [n] : \mathcal{P}_i \text{ receives a valid (VOTE, } ID, s, 1, VC_s, \rho_{j,s}) \text{ from } \mathcal{P}_j\}$ 
64:        $S_{i,s}^0 \leftarrow \{j \in [n] : \mathcal{P}_i \text{ receives (VOTE, } ID, s, 0) \text{ from } \mathcal{P}_j\}$ 
65:        $S_{i,s}^{zom} \leftarrow \{j \in [n] : \mathcal{P}_i \text{ receives (VOTE, } ID, s, \text{ZOMBIE) from } \mathcal{P}_j\}$ 
66:       if  $|S_{i,s}^{yes}| + |S_{i,s}^0| + |S_{i,s}^{zom}| < n - t - s$  then ▷ detect zombie
67:          $z_i \leftarrow 1$  ▷ zombie
68:         output  $\text{dispersed}[ID, s] \leftarrow (\perp, \perp, z_i, g_i)$  ▷ output  $\perp$ 
69:       else ▷ not zombie
70:         if  $|S_{i,s}^{yes}| \geq t + 1$  then
71:            $\Sigma_s \leftarrow \text{TSS.Combine}(ID \| s \| VC_s, S_{i,s}^{yes})$  ▷ Aggregation
72:           output  $\text{dispersed}[ID, s] \leftarrow (VC_s, \Sigma_s, z_i, g_i)$ 
73:         else
74:           output  $\text{dispersed}[ID, s] \leftarrow (0, 0, z_i, g_i)$  ▷ output default 0

```

fragment by sending the message $(\text{DISPERSE}, ID, s, VC_s, \sigma_s, m_{s,j}, \pi_{s,j})$ to $\mathcal{P}_j$. Consequently, each receiver obtains only its own coded fragment, while all fragments remain bound to the same commitment and protocol instance.

Amplification. As discussed previously, even if sufficiently many honest parties claim that they received coded fragments during the dispersal phase, reconstructability is still challenging under the mixed-fault setting. To address this issue, we introduce an amplification mechanism based on a *check-redispersal* methodology. The detailed procedure is as follows:

At time $\tau = \Delta$ (Algorithm 1, lines 9-13), party $\mathcal{P}_i$ checks the message received from the sender. If both the signature on $ID\|s\|VC_s$ and the opening proof for position i are valid, then $\mathcal{P}_i$ multicasts a positive ECHO message. Otherwise, it multicasts a negative ECHO message. Here, a positive ECHO indicates that the sender successfully delivered a verifiable coded fragment, while a negative ECHO still serves as feedback in the receive-fault detection procedure.

At time $\tau = 2\Delta$ (Algorithm 1, lines 14-27), $\mathcal{P}_i$ first performs zombie detection. Specifically, it expects to receive at least $n - t - s$ ECHO messages. This threshold follows from the mixed-fault model: among the n parties, at most t Byzantine parties may send arbitrary or no messages, and at most s send-faulty parties may have their outgoing messages omitted. Therefore, a non-receive-faulty honest party should still receive at least $n - t - s$ messages. If $\mathcal{P}_i$ receives fewer than this threshold, it sets z_i to 1. Meanwhile, if $\mathcal{P}_i$ receives $t + 1$ valid positive ECHO messages from distinct parties, it selects a set S of $t + 1$ such fragments and decodes a candidate value v'_s . Party $\mathcal{P}_i$ then checks whether $\text{Predicate}_{\text{wUMVBA}}(v'_s) = 1$, re-encodes v'_s through $\text{EC.Encode}(n, t + 1, v'_s)$ to obtain $\bar{m}'_s = (m'_{s,1}, \dots, m'_{s,n})$, and recomputes VC'_s using $\text{VC.Commit}(\bar{m}'_s)$. If $VC'_s = VC_s$, then for every party $\mathcal{P}_j$, party $\mathcal{P}_i$ computes the opening proof $\pi'_{s,j}$ and sends READY message to $\mathcal{P}_j$. After sending READY messages, $\mathcal{P}_i$ sets lock_s to VC_s . The purpose of the READY phase is to re-disperse the coded fragments throughout the network. Intuitively, a positive READY message demonstrates that $\mathcal{P}_i$ can successfully reconstruct the sender's value.

At time $\tau = 3\Delta$ (Algorithm 1, lines 28-38), a party with $z_i = 1$ multicasts (ACK, ID, s, ZOMBIE). This message indicates that the party has detected itself as receive-faulty. If $z_i = 0$, then for every party $\mathcal{P}_j$, $\mathcal{P}_i$ checks whether it received a valid READY message from $\mathcal{P}_j$ and whether $\text{lock}_s = VC_s$. If both conditions hold, $\mathcal{P}_i$ stores the carried fragment in $\text{fragment}[ID, s]$ and replies with ACK carrying value 1. Otherwise, it replies with ACK carrying value 0. Here, value 1 indicates that a verifiable coded fragment carried by a READY message in the previous round has been successfully stored, and that the corresponding value has been successfully dispersed to the network.

At time $\tau = 4\Delta$ (Algorithm 2, lines 39-58), if $z_i = 1$, then $\mathcal{P}_i$ multicasts $\text{VOTE}(\text{ZOMBIE})$. Otherwise, $\mathcal{P}_i$ collects three acknowledgment sets: $A_{i,s}^1$ for positive acknowledgments, $A_{i,s}^0$ for negative acknowledgments, and $A_{i,s}^{\text{zom}}$ for zombie reports. Party $\mathcal{P}_i$ first performs the same receive-fault detection procedure. If $|A_{i,s}^1| + |A_{i,s}^0| + |A_{i,s}^{\text{zom}}| < n - t - s$, then too few feedback messages have been received. In this case, $\mathcal{P}_i$ sets z_i to 1 and announces itself as ZOMBIE through a VOTE message. Otherwise, $\mathcal{P}_i$ checks whether its READY messages have been sufficiently acknowledged. First, $\mathcal{P}_i$ checks whether it should regard itself as a ghost. If $\text{lock}_s \neq 0$ and $|A_{i,s}^0| + |A_{i,s}^{\text{zom}}| \geq t + r + 1$, then $\mathcal{P}_i$ sets g_i to 1. The threshold $t + r + 1$ is derived from the mixed-fault model: at most t Byzantine parties and at most r receive-faulty parties can explain negative acknowledgments or zombie reports. Therefore, receiving at least $t + r + 1$ such messages implies that even some non-zombie honest parties failed to receive the re-dispersed fragments.

Algorithm 3 UVID Retrieval subprotocol with identifier (ID, s) , code for $\mathcal{P}_i$:

Input: $\text{dispersed}[ID, s] := (VC_s, \Sigma_s, z_i, g_i)$

- 1: **if** $\tau = 0$ **then** $\triangleright$ for $\mathcal{P}_i$
- 2: **if** $(VC_s, m'_{s,i}, \pi'_{s,i}) \in \text{fragment}[ID, s]$ **then**
- 3: multicast $(\text{FRAGMENT}, ID, s, VC_s, m'_{s,i}, \pi'_{s,i})$
- 4: **if** $\tau = (k+1)\Delta$ **then** $\triangleright$ for $\mathcal{P}_i$
- 5: **if** received $t+1$ valid $(\text{FRAGMENT}, ID, s, VC_s, *)$ from distinct parties **then**
- 6: let S be a set of $f+1$ fragments
- 7: $v_s \leftarrow \text{EC.Decode}(n, t+1, S)$ $\triangleright$ reconstruct v_s
- 8: **output** $\text{retrieved}[ID, s] := v_s$
- 9: **else**
- 10: **output** $\text{retrieved}[ID, s] := \perp$ $\triangleright$ receive fault

Party $\mathcal{P}_i$ sends a positive VOTE message (i.e., with value 1) only if all three conditions hold: $\text{lock}_s \neq 0$, $\text{locked}_s = 0$, and $g_i = 0$. In this case, $\mathcal{P}_i$ generates a threshold signature share on $ID \| s \| VC_s$. Otherwise, it sends a negative VOTE message (i.e., with value 0). Therefore, a positive VOTE carries a strong guarantee: the voter has successfully reconstructed a valid value, verified that the reconstructed value recommit to VC_s , confirmed that sufficiently many parties can store fragments of the value, and has not detected itself as a ghost.

At time $\tau = 5\Delta$ (Algorithm 2, lines 59-74), $\mathcal{P}_i$ determines its final output. If $z_i = 1$, then it outputs $(\perp, \perp, z_i, g_i)$. Otherwise, $\mathcal{P}_i$ collects three vote sets: $S_{i,s}^{\text{yes}}$ for positive votes, $S_{i,s}^0$ for negative votes, and $S_{i,s}^{\text{zom}}$ for zombie reports. If the total number of received votes is smaller than $n - t - s$, then $\mathcal{P}_i$ detects a receive fault, sets z_i to 1, and outputs the $\perp$ value. If $\mathcal{P}_i$ receives at least $t+1$ valid positive VOTE messages for the same VC_s , it combines the corresponding threshold signature shares into a threshold signature Σ_s and outputs $(VC_s, \Sigma_s, z_i, g_i)$. The valid threshold signature guarantees the existence of at least one honest positive voter. Finally, if sufficiently many VOTE messages are received but fewer than $t+1$ are positive, then $\mathcal{P}_i$ outputs the default value $(0, 0, z_i, g_i)$, indicating that no valid certificate was produced.

5.2 The RETRIEVE Subprotocol

In the reconstruction phase, all non-receive-faulty honest parties invoke the RETRIEVE protocol with the same input $\text{dispersed}[ID, s] := (VC_s, \Sigma_s, z_i, g_i)$ generated during the dispersal phase. The goal of this phase is to reconstruct the common value v_s corresponding to the vector commitment VC_s , satisfying $VC_s = \text{VC.Commit}(\text{EC.Encode}(n, t+1, v_s))$. We remark that parties participating in the RETRIEVE protocol are not required to start simultaneously. We assume that if the first non-receive-faulty honest party invokes RETRIEVE at time $\tau = w\Delta$, then the last non-receive-faulty honest party invokes RETRIEVE no later than time $\tau = (w+k)\Delta$. In other words, the invocation times of non-receive-faulty honest parties differ by at most $k\Delta$. Algorithm 3 implements the reconstruction phase. The detailed construction of RETRIEVE is as follows:

1. At time $\tau = 0$ (Algorithm 3, lines 1-3), all non-receive-faulty honest parties $\mathcal{P}_i$ invoke $\text{RETRIEVE}[ID, s]$ with input $\text{dispersed}[ID, s] := (VC_s, \Sigma_s, z_i, g_i)$. Each party $\mathcal{P}_i$ first checks whether its local storage $\text{fragment}[ID, s]$ contains a tuple $(VC_s, m'_{s,i}, \pi'_{s,i})$ corresponding to the same commitment VC_s . If such a tuple exists, then $\mathcal{P}_i$ multicasts $(\text{FRAGMENT}, ID, s, VC_s, m'_{s,i}, \pi'_{s,i})$.
2. At time $\tau = (k+1)\Delta$ (Algorithm 3, lines 4-10), party $\mathcal{P}_i$ considers only valid FRAGMENT messages received from distinct parties. If $\mathcal{P}_i$ has collected at least $t+1$ such valid fragments, it reconstructs v_s through the EC.Decode algorithm and outputs $\text{retrieved}[ID, s] := v_s$. Otherwise, $\mathcal{P}_i$ outputs $\text{retrieved}[ID, s] := \perp$, indicating that it failed to collect sufficiently many fragments before the end of the current retrieval attempt and therefore considers itself receive-faulty.

5.3 Analysis of UVID

Lemma 1 (Validity). *Algorithms 1, 2, and 3 guarantee the validity property in Definition 2.*

Proof. Let $\mathcal{P}_s$ be a non-send-faulty honest sender with input v_s . The input is valid, i.e., $\text{Predicate}_{\text{UVID}}(v_s) = 1$. At $\tau = 0$, $\mathcal{P}_s$ encodes v_s , computes $VC_s = \text{VC.Commit}(\overline{m}_s)$, signs $ID \| s \| VC_s$, and sends each party $\mathcal{P}_j$ a valid fragment $(m_{s,j}, \pi_{s,j})$. At $\tau = \Delta$, every non-receive-faulty honest party receives this valid DISPERSE message and multicasts a positive ECHO . At $\tau = 2\Delta$, consider any non-receive-faulty honest party $\mathcal{P}_i$. By the counting fact, $\mathcal{P}_i$ receives at least $n - t - s$ ECHO messages, so it does not set $z_i = 1$. Among all parties, at least $n - t - r - s \geq t + 1$ are non-full-omission-faulty honest parties. These parties receive the sender's valid DISPERSE message and send valid positive ECHO messages for the same VC_s . Party $\mathcal{P}_i$ decodes v_s , verifies $\text{Predicate}_{\text{UVID}}(v_s) = 1$, re-encodes v_s , confirms $VC'_s = VC_s$, sends every party $\mathcal{P}_j$ a valid READY message carrying the re-encoded fragment $m'_{s,j}$, and sets $\text{Lock}_s \leftarrow VC_s$.

Now consider any non-full-omission-faulty honest party $\mathcal{P}_h$. At $\tau = 3\Delta$, $\mathcal{P}_h$ receives a valid READY message from every such party. Since $\mathcal{P}_h$ has already set $\text{Lock}_s = VC_s$, it verifies the opening proof, stores $(VC_s, m'_{s,h}, \pi'_{s,h})$ in $\text{fragment}[ID, s]$, and replies with $\text{ACK}(1)$. Therefore, every such party that sent a valid READY receives at least $n - t - r - s > t$ positive acknowledgments from parties in this set at $\tau = 4\Delta$, so Locked_s remains 0. Moreover, for such a party, all non-receive-faulty honest parties receive its valid READY message and have $\text{Lock}_s = VC_s$, hence they reply with $\text{ACK}(1)$. Negative acknowledgments or zombie reports can only be explained by Byzantine parties or receive-faulty honest parties, at most $t + r$ parties in total. Thus the ghost-detection condition $|A_{i,s}^0| + |A_{i,s}^{\text{zom}}| \geq t + r + 1$ is not triggered, and g_i remains 0. Consequently, every such party sends a valid positive VOTE share on $ID \| s \| VC_s$. At $\tau = 5\Delta$, every non-receive-faulty honest party receives at least $t + 1$ positive VOTE shares from parties in this set, combines them into a threshold signature Σ_s , and outputs $\text{dispersed}[ID, s] := (VC_s, \Sigma_s, z_i, g_i)$. Any honest party that fails a receive-fault check at some step sets $z_i = 1$ and outputs $(\perp, \perp, z_i, g_i)$.

During DISPERSE, at least $n - t - r - s \geq t + 1$ non-full-omission-faulty honest parties store valid fragments under VC_s : each of them has locked on VC_s and has received a valid re-dispersed fragment through a READY message. Suppose all non-receive-faulty honest parties invoke RETRIEVE[ID, s] with the valid input $\text{dispersed}[ID, s] = (VC_s, \Sigma_s, z_i, g_i)$. Then these honest fragment holders invoke RETRIEVE as well. At local time $\tau = 0$, each of them multicasts its stored valid FRAGMENT message. By the bounded-start assumption and synchronous delivery, every non-receive-faulty honest retriever receives at least $t + 1$ valid fragments from distinct honest parties by its local deadline $\tau = (k + 1)\Delta$. By erasure-code correctness, it decodes v_s and outputs $\text{retrieved}[ID, s] = v_s$. Any honest party that does not receive enough valid fragments by the retrieval deadline is receive-faulty and becomes a zombie. Therefore, every honest party either outputs $\text{retrieved}[ID, s] = v_s$ or becomes a zombie.

Lemma 2 (Termination). *Algorithms 1, 2, and 3 guarantee the termination property in Definition 2.*

Proof. DISPERSE has a fixed synchronous schedule ending at time 5Δ . At that time, Algorithm 2 enters exactly one output branch. If $z_i = 1$, or if the final receive test detects too few votes, the party outputs $(\perp, \perp, z_i, g_i)$. If it receives at least $t + 1$ valid positive votes for some VC_s , it combines the shares and outputs $(VC_s, \Sigma_s, z_i, g_i)$. Otherwise it outputs the default value $(0, 0, z_i, g_i)$. The local variable $\text{fragment}[ID, s]$ is initialized at the beginning of the instance and is updated only by finite message-processing steps, so it is available when DISPERSE terminates.

RETRIEVE also runs on a fixed schedule: at local time $\tau = 0$, a party multicasts a FRAGMENT message if it stores a valid fragment for the input commitment, at deadline $\tau = (k + 1)\Delta$, it either decodes a value from $t + 1$ valid FRAGMENT messages and outputs it, or becomes a zombie. Hence every honest party that invokes either subprotocol terminates at its prescribed deadline.

Lemma 3 (Agreement). *Algorithms 1, 2, and 3 satisfy the agreement property in Definition 2.*

Proof. Fix a valid certificate (ID, s, VC_s, Σ_s) . Since Σ_s is a valid $(t + 1, n)$ -threshold signature on $ID \| s \| VC_s$ and at most t parties are Byzantine, at least one positive VOTE share in Σ_s was generated by an honest party $\mathcal{P}_h$. The positive-voting branch of Algorithm 2 requires $\text{Lock}_s \neq 0$, $\text{Locked}_s = 0$, and $g_h = 0$. Since the vote is for VC_s , the local lock of $\mathcal{P}_h$ is VC_s . Thus $\mathcal{P}_h$ decoded some value v^* , verified $\text{Predicate}_{\text{UVD}}(v^*) = 1$, re-encoded it, and confirmed $VC_s = \text{VC.Commit}(\text{EC.Encode}(n, t + 1, v^*))$. By the binding property of the vector commitment, VC_s uniquely determines this value v^* .

We next show that the certificate is supported by stored fragments. Since $\mathcal{P}_h$ casts a positive vote, it does not set $\text{Locked}_s = 1$, and therefore it has received more than t positive acknowledgments for its READY message. As at most t parties are Byzantine, at least one of these ACK(1) messages is sent by an honest party. An honest party sends ACK(1) only after it receives a valid READY

message carrying a fragment opened under VC_s , has $\text{Lock}_s = VC_s$, and stores this fragment in $\text{fragment}[ID, s]$. Hence a positive vote certifies that the re-dispersed codeword under VC_s has been checked and stored by an honest non-zombie party. Moreover, because $\mathcal{P}_h$ does not set $g_h = 1$, the ghost-detection test gives $|A_{h,s}^0| + |A_{h,s}^{\text{zom}}| \leq t + r$. Thus the failed acknowledgments and zombie reports received by $\mathcal{P}_h$ can be accounted for by Byzantine parties and receive-faulty parties. In particular, if too few non-full-omission-faulty honest parties had stored fragments for VC_s , then more than $t + r$ parties would have returned either $\text{ACK}(0)$ or $\text{ACK}(\text{ZOMBIE})$, forcing $\mathcal{P}_h$ to set $g_h \leftarrow 1$ and preventing a positive vote. Therefore, the positive vote implies that at least $t + 1$ non-full-omission-faulty honest parties store valid fragments for the codeword committed by VC_s . Being non-send-faulty, these parties can also act as fragment senders in RETRIEVE.

When all non-receive-faulty honest parties invoke RETRIEVE on the same valid input, these honest fragment holders also invoke RETRIEVE. At their local time $\tau = 0$, they multicast valid FRAGMENT messages for VC_s . So every non-receive-faulty honest retriever receives at least $t + 1$ such fragments by its local deadline $\tau = (k + 1)\Delta$. Algorithm 3 accepts only fragment openings that verify under VC_s , by position-binding, each position h admits at most one valid fragment. Since erasure decoding of any $t + 1$ fragments consistent with the same codeword yields a unique result, all non-receive-faulty honest retrievers output the same value $v^* \neq \perp$. Any honest party that does not receive enough valid fragments by the retrieval deadline is receive-faulty and becomes a zombie. Therefore, every honest party either outputs the same non- $\perp$ value v^* or becomes a zombie.

Lemma 4 (Recast-ability). *Algorithms 1, 2, and 3 satisfy the recast-ability property in Definition 2.*

Proof. Suppose a non-receive-faulty honest party $\mathcal{P}_i$ outputs $\text{dispersed}[ID, s] := (VC_s, \Sigma_s, z_i, g_i)$. By the certificate branch of Algorithm 2, $\mathcal{P}_i$ collected at least $t + 1$ valid positive VOTE shares and combined them into Σ_s , thus (VC_s, Σ_s) is a valid certificate for $ID \| s \| VC_s$. The conclusion now follows directly from Lemma 3: since the certificate is valid, when all non-receive-faulty honest parties invoke $\text{RETRIEVE}[ID, s]$ on it, every honest party either outputs the same non- $\perp$ value corresponding to VC_s or becomes a zombie.

Lemma 5 (External validity). *Algorithms 1, 2, and 3 guarantee the External validity property in Definition 2.*

Proof. Because the certificate is valid, its threshold signature contains at least one honest positive vote, except with negligible probability. An honest party sends a vote only after it reconstructs some value v^* , checks $\text{Predicate}_{\text{UVID}}(v^*) = 1$, re-encodes v^* , and verifies that the resulting commitment is VC_s .

The value v output by RETRIEVE is decoded from $t + 1$ fragments whose openings verify under VC_s . By position-binding, these fragments must be consistent with the same committed codeword used by the honest positive voter. By erasure-code uniqueness, $v = v^*$. Therefore $\text{Predicate}_{\text{UVID}}(v) = 1$.

Lemma 6 (Detection soundness). *Algorithms 1, 2, and 3 guarantee the Detection soundness property in Definition 2.*

Proof. An honest party sets $z_i \leftarrow 1$ only after receiving fewer than $n - t - s$ messages in one of the receive-checking steps for ECHO, ACK, or VOTE. If this party were not receive-faulty, then all messages sent by non-send-faulty honest parties in that step would be delivered to it. There are at least $n - t - s$ such parties, because at most t parties are Byzantine and at most s honest parties are send-faulty. Hence a non-receive-faulty honest party cannot receive fewer than $n - t - s$ such messages, so setting $z_i = 1$ is sound.

An honest party sets $g_i \leftarrow 1$ only in the branch $\text{Lock}_s = 1$ and $|A_{i,s}^0| + |A_{i,s}^{\text{zom}}| \geq t + r + 1$. If the party were not send-faulty, then every non-receive-faulty honest party would receive its valid READY message and reply with ACK(1). Negative acknowledgments or zombie reports could then come only from Byzantine parties and receive-faulty honest parties, at most $t + r$ in total. This contradicts the threshold $t + r + 1$. Thus setting $g_i = 1$ is sound.

Complexity analysis. We now analyze the complexity of DISPERSE and RETRIEVE. Let ℓ denote the input size. Each coded fragment has size $\mathcal{O}(\ell/(t + 1))$, while vector commitments, opening proofs, signatures, threshold signature shares, and certificates each have size $\mathcal{O}(\lambda)$.

1. In DISPERSE, the dissemination phase is executed only by the sender. The sender transmits one coded fragment through a DISPERSE message to each party. Each DISPERSE message has size $\mathcal{O}(\ell/t + \lambda)$. Therefore, in this phase, the communication complexity is $\mathcal{O}(n\ell/t + n\lambda)$ and the message complexity is $\mathcal{O}(n)$.
2. In DISPERSE, the amplification phase consists of the ECHO, READY, ACK, and VOTE message rounds. Each ECHO and READY message has size at most $\mathcal{O}(\ell/t + \lambda)$. Therefore, each of these two rounds incurs communication complexity $\mathcal{O}(n^2\ell/t + n^2\lambda)$ bits. The ACK and VOTE rounds carry only short acknowledgments, zombie reports, or threshold signature shares. Hence, each such message has size at most $\mathcal{O}(\lambda)$, and therefore each round incurs communication complexity $\mathcal{O}(n^2\lambda)$ bits.

Consequently, in the amplification phase of DISPERSE, the communication complexity is $\mathcal{O}(n^2\ell/t + n^2\lambda)$ and the message complexity is $\mathcal{O}(n^2)$.

3. In RETRIEVE, every party multicasts at most one FRAGMENT message, and each FRAGMENT message has size $\mathcal{O}(\ell/t + \lambda)$. Hence, in this phase, the communication complexity is $\mathcal{O}(n^2\ell/t + n^2\lambda)$ and the message complexity is $\mathcal{O}(n^2)$.

In summary, both DISPERSE and RETRIEVE use $\mathcal{O}(n^2)$ messages and $\mathcal{O}(1)$ synchronous rounds. When $t = \Theta(n)$, the communication complexity of each subprotocol is $\mathcal{O}(n\ell + n^2\lambda)$ bits.

6 Weak Undead Multi-valued Validated Byzantine Agreement

In this section, we introduce *Weak Undead Multi-valued Validated Byzantine Agreement* (wUMVBA), a single-shot weak variant of multi-valued validated Byzantine agreement [3, 8, 33]. The protocol employs the UVID protocol as its data-dispersal layer and uses the undead multi-valued consensus primitive of Feng et al. [21] as its agreement component. The goal of wUMVBA is to allow all non-receive-faulty honest parties to output a common value. Unlike standard MVBA, the common output is allowed to be $\perp$. If the output is not $\perp$, then it must satisfy the public predicate function $\text{Predicate}_{\text{wUMVBA}}$ defined in Definition 3. More precisely, the definition of wUMVBA is as follows:

Definition 3 (Weak Undead MVBA (wUMVBA)). *The wUMVBA protocol is executed by n parties $\mathcal{P} = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ in the mixed-fault model. Each party $\mathcal{P}_i$ holds an input v_i and a global public predicate function $\text{Predicate}_{\text{UVID}} : \{0, 1\}^* \rightarrow \{0, 1\}$. The goal of the protocol is to output a common value v , where it is possible that $v = \perp$. If $v \neq \perp$, then it must satisfy $\text{Predicate}_{\text{UVID}}(v) = 1$. Formally, wUMVBA satisfies the following properties except with negligible probability:*

- **Agreement.** *If two non-zombie honest parties output v_i and v_j , respectively, then $v_i = v_j$.*
- **Termination.** *Every honest party eventually outputs a value v .*
- **External validity.** *If a non-zombie honest party outputs $v \neq \perp$, then $\text{Predicate}_{\text{UVID}}(v) = 1$.*
- **Quality.** *If a non-receive-faulty honest party outputs v , then the probability that v was proposed by the adversary is at most t/n . In addition, the probability that $v = \perp$ is at most $(t + s)/n$.*
- **Detection Soundness.** *If an honest party outputs $z'_i = 1$ (resp. $g'_i = 1$), then it is receive-faulty (resp. send-faulty).*

6.1 The Weak Undead MVBA Protocol

As illustrated in Figure 2, the protocol consists of four stages: dispersal, election, consensus, and retrieval. During the dispersal stage, every party invokes n instances of the UVID DISPERSE protocol while acting as the designated sender in its corresponding instance. During the election stage, the undead common coin protocol outputs a common index. The parties then invoke UMVC using the UVID output corresponding to the elected index as input. Finally, if the agreed output is a certified UVID tuple, the parties invoke RETRIEVE to reconstruct the corresponding value.

Algorithm 4 implements the wUMVBA protocol with identifier ID and public predicate $\text{Predicate}_{\text{wUMVBA}}$. Before the timed stages begin, each party $\mathcal{P}_i$ initializes n dispersal instances $\text{DISPERSE}[ID, j]$ for every $j \in [n]$, where all instances

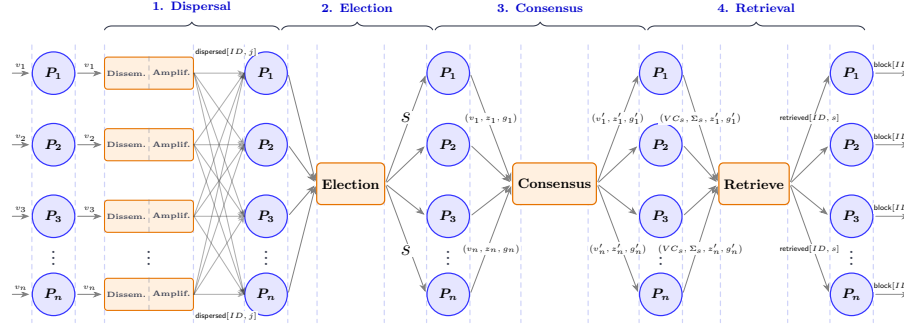


Fig. 2: The execution flow of Weak Undead MVBA

are configured with the same predicate $\text{Predicate}_{\text{wUMVBA}}$. Party $\mathcal{P}_i$ only inputs its local value v_i into the corresponding instance $\text{DISPERSE}[ID, i]$.

Construction of wUMVBA. The formal description is given in Algorithm 4. At a high level, the protocol proceeds through the following four stages:

1. **Dispersal.** At time $\tau = 0$ (Algorithm 4, lines 1-2), party $\mathcal{P}_i$ inputs its local proposal v_i into the UVID subprotocol $\text{DISPERSE}[ID, i]$ and participates in all n dispersal instances.
2. **Election.** At time $\tau = 5\Delta$ (Algorithm 4, lines 3-6), each $\text{DISPERSE}[ID, j]$ outputs $\text{dispersed}[ID, j]$ for every $j \in [n]$. Party $\mathcal{P}_i$ invokes the common coin instance $\text{UCOIN}[ID]$.
3. **Consensus.** At time $\tau = 6\Delta$ (Algorithm 4, lines 7-11), the common coin outputs a sender index, denoted by s . Party $\mathcal{P}_i$ then parses the output $\text{dispersed}[ID, s]$ as (x, y, z_i, g_i) , sets its local value v_i to (x, y) , and inputs the triplet (v_i, z_i, g_i) into $\text{UMVC}[ID]$.
4. **Retrieval.** Upon receiving (v'_i, z'_i, g'_i) from $\text{UMVC}[ID]$, party $\mathcal{P}_i$ inspects the agreed value v'_i (Algorithm 4, lines 12-18). If $v'_i = (VC_s, \Sigma_s)$ is a valid certificate, then $\mathcal{P}_i$ invokes $\text{RETRIEVE}[ID, s]$ with input $(VC_s, \Sigma_s, z'_i, g'_i)$, obtains the output $\text{retrieved}[ID, s]$, and sets $\text{block}[ID]$ to $\text{retrieved}[ID, s]$. If v'_i is not a valid certificate, then $\mathcal{P}_i$ sets $\text{block}[ID]$ to $\perp$. In both cases, $\mathcal{P}_i$ outputs $\text{block}[ID]$ as the result of the $\text{wUMVBA}[ID]$ execution.

6.2 Analysis of Weak Undead MVBA

Lemma 7 (Agreement). *Algorithm 4 guarantees the Agreement property in Definition 3.*

Proof. Let $\mathcal{P}_i$ and $\mathcal{P}_j$ be two non-zombie honest parties. By the agreement property of UCOIN , all non-zombie honest parties obtain the same elected index $s \in [n]$ from $\text{UCOIN}[ID]$, so both $\mathcal{P}_i$ and $\mathcal{P}_j$ read the same s in Stage 3.

By the detection-soundness property of UMVC (Definition 1), if an honest party outputs $z'_i = 1$ then it is receive-faulty. Since $\mathcal{P}_i$ and $\mathcal{P}_j$ are non-zombie,

Algorithm 4 wUMVBA protocol with identifier ID and external $\text{Predicate}_{\text{wUMVBA}}$, code for $\mathcal{P}_i$:

```

for each  $j \in [n]$  do
    initialize a dispersal instance  $\text{DISPERSE}[ID, j]$ 
    let the  $\text{Predicate}_{\text{UVID}}$  of  $\text{DISPERSE}[ID, j]$  be the same as  $\text{Predicate}_{\text{wUMVBA}}$ .

1: if  $\tau = 0$  then ▷ Stage 1: Disperse
2:   input  $v_i$  into  $\text{DISPERSE}[ID, i]$ 

3: if  $\tau = 5\Delta$  then ▷ Stage 2: Election
4:   for  $j \in [n]$  do
5:      $\text{dispersed}[ID, j] \leftarrow \text{DISPERSE}[ID, j]$  ▷ see Alg. 1 and 2
6:   invoke  $\text{UCOIN}[ID]$ 

7: if  $\tau = 6\Delta$  then ▷ Stage 3: Consensus
8:    $s \leftarrow \text{UCOIN}[ID]$ 
9:   parse  $\text{dispersed}[ID, s] := (x, y, z_i, g_i)$ 
10:  let  $v_i \leftarrow (x, y)$ 
11:  input  $(v_i, z_i, g_i)$  into  $\text{UMVC}[ID]$ 

12: upon  $(v'_i, z'_i, g'_i) \leftarrow \text{UMVC}[ID]$  do ▷ Stage 4: Retrieve
13:  if  $v'_i := (VC_s, \Sigma_s)$  then
14:     $\text{retrieved}[ID, s] \leftarrow \text{RETRIEVE}[ID, s](VC_s, \Sigma_s, z'_i, g'_i)$ 
15:     $\text{block}[ID] \leftarrow \text{retrieved}[ID, s]$ 
16:  else
17:     $\text{block}[ID] \leftarrow \perp$ 
18:  output  $\text{block}[ID]$ 

```

they output $z'_i = 0$ and $z'_j = 0$ at the end of $\text{UMVC}[ID]$. By the agreement property of UMVC , all non-zombie honest parties obtain the same output value. In particular, $\mathcal{P}_i$ and $\mathcal{P}_j$ obtain the same v' .

If $v' = (\perp, \perp)$ or $v' = (0, 0)$, both $\mathcal{P}_i$ and $\mathcal{P}_j$ output $\perp$. Otherwise $v' = (VC_s, \Sigma_s)$ is a valid UVID certificate. By the detection-soundness property of UMVC , an honest party that is a zombie at the end of $\text{UMVC}[ID]$ is receive-faulty. Hence every honest party that is not receive-faulty obtains the same v' and invokes $\text{RETRIEVE}[ID, s]$ with the same input (VC_s, Σ_s) . By the convention of Definition 2, Σ_s is a valid $(t+1, n)$ -threshold signature on $ID \| s \| VC_s$, so this is a valid input to RETRIEVE . By Lemma 3, every honest party either outputs the same non- $\perp$ value or becomes a zombie. Since $\mathcal{P}_i$ and $\mathcal{P}_j$ are non-zombie, they output the same value.

Lemma 8 (Termination). *Algorithm 4 guarantees the Termination property in Definition 3.*

Proof. Each of the n UVID dispersal instances runs on a fixed synchronous schedule of length 5Δ . By the termination property of UVID (Definition 2), every honest party terminates every $\text{DISPERSE}[ID, j]$ instance and obtains a local output $\text{dispersed}[ID, j]$ by time $\tau = 5\Delta$. At time $\tau = 5\Delta$, after reading all dispersal outputs via the **for** loop, every party invokes $\text{UCOIN}[ID]$. By the termination

property of **UCOIN**, every honest party either obtains the elected index s within one round or becomes a zombie. At time $\tau = 6\Delta$, every party inputs (v_i, z_i, g_i) into **UMVC**[ID]. By the termination property of **UMVC** (Definition 1), all honest parties eventually receive an output triplet (v'_i, z'_i, g'_i) .

If v'_i is not a valid certificate, the wrapper immediately sets $\text{block}[ID] \leftarrow \perp$ and outputs $\perp$. If $v'_i = (VC_s, \Sigma_s)$ is a valid certificate, the wrapper invokes **RETRIEVE**[ID, s], which runs on a fixed synchronous schedule of length $(k + 1)\Delta$. By the termination property of **UVID**, every honest party that invokes **RETRIEVE** eventually terminates and outputs a value or $\perp$. Hence every honest party eventually produces an output.

Lemma 9 (External validity). *Algorithm 4 guarantees the External validity property in Definition 3.*

Proof. Suppose a non-zombie honest party $\mathcal{P}_i$ outputs $v \neq \perp$. This occurs only when **RETRIEVE**[ID, s] returns a non- $\perp$ value from a valid certificate $v' = (VC_s, \Sigma_s)$ agreed upon by **UMVC**[ID].

By the certificate convention of Definition 2, Σ_s is a valid $(t+1, n)$ -threshold signature on $ID \| s \| VC_s$. Since at most t parties are Byzantine, at least one positive **VOTE** share contributing to Σ_s was produced by an honest party $\mathcal{P}_k$. By the positive-voting branch of Algorithm 2, $\mathcal{P}_k$ must have set $\text{lock}_s = 1$, $\text{locked}_s = 0$, and $g_k = 0$. The condition $\text{lock}_s = 1$ means $\mathcal{P}_k$ decoded some value v^* from $t+1$ valid positive **ECHO** messages, verified $\text{Predicate}_{\text{wUMVBA}}(v^*) = 1$, re-encoded v^* , and confirmed that the resulting vector commitment equals VC_s . By the position-binding property of the vector commitment scheme, VC_s uniquely determines v^* .

The value v output by **RETRIEVE** is decoded from at least $t+1$ **FRAGMENT** messages whose opening proofs verify under VC_s (Algorithm 3). By position-binding, each position admits at most one valid fragment, so all accepted fragments are consistent with the same committed codeword that $\mathcal{P}_k$ used. By erasure-code uniqueness, $v = v^*$, and therefore $\text{Predicate}_{\text{wUMVBA}}(v) = \text{Predicate}_{\text{wUMVBA}}(v^*) = 1$.

Lemma 10 (Quality). *Algorithm 4 guarantees the Quality property in Definition 3.*

Proof. Let $s = \text{UCOIN}[ID]$ be the elected index. By the unpredictability and agreement properties of **UCOIN**, s is uniformly random over $[n]$ and independent of all party inputs. At most t parties are Byzantine, so $\Pr[\mathcal{P}_s \text{ is Byzantine}] \leq t/n$. If $\mathcal{P}_s$ is not Byzantine, the output value originates from $\mathcal{P}_s$'s dispersal and is not proposed by the adversary. Hence the probability that v was proposed by a Byzantine party is at most t/n .

For the second claim, we show that $v \neq \perp$ whenever $\mathcal{P}_s$ is a non-send-faulty honest sender with $\text{Predicate}_{\text{wUMVBA}}(v_s) = 1$. By the validity property of **UVID** (Lemma 1), every honest party $\mathcal{P}_i$ either outputs $\text{dispersed}[ID, s] := (VC_s, \Sigma_s, z_i, g_i)$ from **DISPERSE**[ID, s] or becomes a zombie (sets $z_i = 1$, in which case $\text{dispersed}[ID, s] := (\perp, \perp, z_i, g_i)$). Therefore: every non-zombie honest party

inputs $(VC_s, \Sigma_s, z_i, g_i)$ into $\text{UMVC}[ID]$. And every zombie honest party ($z_i = 1$) inputs $(\perp, \perp, 1, g_i)$, i.e., $v_i = \perp$. By the validity property of UMVC (Definition 1), since all non-zombie honest parties hold the same input (VC_s, Σ_s) and every zombie holds $\perp$, every honest party either outputs $v'_i = (VC_s, \Sigma_s)$ or becomes a zombie ($z'_i = 1$). By the detection-soundness property of UMVC , if $z'_i = 1$ then $\mathcal{P}_i$ is receive-faulty. While non-receive-faulty honest parties output $z'_i = 0$, i.e., they are non-zombie at the end of $\text{UMVC}[ID]$. They therefore all output $v'_i = (VC_s, \Sigma_s)$ and invoke $\text{RETRIEVE}[ID, s]$ with the valid certificate. By the second clause of Lemma 1, every honest party either outputs $\text{retrieved}[ID, s] = v_s$ or becomes a zombie. By the detection-soundness property of UVID , if an honest party becomes a zombie, then it is receive-faulty. Hence every non-receive-faulty honest party outputs $\text{retrieved}[ID, s] = v_s \neq \perp$. Consequently, $v = \perp$ can occur only when $\mathcal{P}_s$ is Byzantine or send-faulty. Since there are at most $t + s$ Byzantine or send-faulty parties among n parties, we have $\Pr[v = \perp] \leq (t + s)/n$.

Lemma 11 (Detection soundness). *Algorithm 4 guarantees the Detection soundness property in Definition 3.*

Proof. The wrapper passes the local undead status bits (z_i, g_i) from the UVID dispersal output directly into $\text{UMVC}[ID]$ as part of the input triplet (v_i, z_i, g_i) . The output bits (z'_i, g'_i) of wUMVBA are exactly those returned by $\text{UMVC}[ID]$; the wrapper does not set, modify, or independently inspect these bits.

If an honest party outputs $z'_i = 1$, then $\text{UMVC}[ID]$ produced $z'_i = 1$ for that party. By the detection-soundness property of UMVC (Definition 1), the party is receive-faulty. Similarly, if an honest party outputs $g'_i = 1$, it is send-faulty by the same guarantee. Hence any fault indicator raised at the output of wUMVBA correctly reflects an actual fault of the corresponding type.

Complexity analysis. We now analyze the complexity of wUMVBA . Let ℓ denote the input size and λ the security parameter. Each coded fragment has size $\mathcal{O}(\ell/t)$, while vector commitments, opening proofs, signatures, threshold signature shares, and certificates each have size $\mathcal{O}(\lambda)$.

By Section 5.3, each DISPERSE dissemination phase incurs communication complexity $\mathcal{O}(n\ell/t + n\lambda)$ with message complexity $\mathcal{O}(n)$, while each amplification phase incurs communication complexity $\mathcal{O}(n^2\ell/t + n^2\lambda)$ with message complexity $\mathcal{O}(n^2)$. Moreover, each invocation of RETRIEVE incurs communication complexity $\mathcal{O}(n^2\ell/t + n^2\lambda)$ with message complexity $\mathcal{O}(n^2)$.

The wUMVBA protocol consists of four stages: dispersal, election, consensus, and retrieval, which are implemented by n concurrent DISPERSE instances (including both dissemination and amplification phases), one election procedure, one UMVC consensus instance, and one RETRIEVE invocation, respectively.

1. **Dispersal.** Each DISPERSE instance consists of a dissemination phase and an amplification phase. The dissemination phase incurs communication complexity $\mathcal{O}(n\ell/t + n\lambda)$ with message complexity $\mathcal{O}(n)$, while the amplification phase incurs communication complexity $\mathcal{O}(n^2\ell/t + n^2\lambda)$ with message complexity $\mathcal{O}(n^2)$. Therefore, n concurrent DISPERSE instances incur total communication complexity $\mathcal{O}(n^3\ell/t + n^3\lambda)$ with message complexity $\mathcal{O}(n^3)$.

Algorithm 5 Opt-wUMVBA protocol with identifier ID and external $\text{Predicate}_{\text{wUMVBA}}$, code for $\mathcal{P}_i$:

```

for each  $j \in [n]$  do
    initialize the dissemination phase of  $\text{DISPERSE}[ID, j]$ 
    let the  $\text{Predicate}_{\text{UID}}$  of  $\text{DISPERSE}[ID, j]$  be the same as  $\text{Predicate}_{\text{wUMVBA}}$ .

1: if  $\tau = 0$  then ▷ Stage 1: Dissemination
2:   input  $v_i$  into the dissemination phase of  $\text{DISPERSE}[ID, i]$ 

3: if  $\tau = \Delta$  then ▷ Stage 2: Election
4:   invoke  $\text{UCOIN}[ID]$ 

5: if  $\tau = 2\Delta$  then ▷ Stage 3: Amplification
6:    $s \leftarrow \text{UCOIN}[ID]$ 
7:   initiate the amplification phase of  $\text{DISPERSE}[ID, s]$ 

8: if  $\tau = 6\Delta$  then ▷ Stage 4: UMVC
9:    $\text{dispersed}[ID, s] \leftarrow \text{DISPERSE}[ID, s]$  ▷ see Alg. 1 and 2
10:  parse  $\text{dispersed}[ID, s] := (x, y, z_i, g_i)$ 
11:  let  $v_i \leftarrow (x, y)$ 
12:  input  $(v_i, z_i, g_i)$  into  $\text{UMVC}[ID]$ 

13: upon  $(v'_i, z'_i, g'_i) \leftarrow \text{UMVC}[ID]$  do ▷ Stage 5: Retrieve
14:   if  $v'_i := (VC_s, \Sigma_s)$  then
15:     $\text{retrieved}[ID, s] \leftarrow \text{RETRIEVE}[ID, s](VC_s, \Sigma_s, z'_i, g'_i)$ 
16:     $\text{block}[ID] \leftarrow \text{retrieved}[ID, s]$ 
17:   else
18:     $\text{block}[ID] \leftarrow \perp$ 
19:   output  $\text{block}[ID]$ 

```

2. **Election.** The undead common coin protocol $\text{UCOIN}[ID]$ incurs communication complexity $\mathcal{O}(n^2\lambda)$ and message complexity $\mathcal{O}(n^2)$.
3. **Consensus.** We utilize the $\text{UMVC}[ID]$ protocol from [21]. Since the input size is $\mathcal{O}(\lambda)$, this stage incurs communication complexity $\mathcal{O}(n^2\lambda^2)$ with message complexity $\mathcal{O}(n^2)$. The expected round complexity is $\mathcal{O}(1)$.
4. **Retrieval.** At most one RETRIEVE invocation is executed. Therefore, this stage incurs communication complexity $\mathcal{O}(n^2\ell/t + n^2\lambda)$ with message complexity $\mathcal{O}(n^2)$.

The time cost consists of 5Δ for the dispersal stage, followed by Δ for the election stage, $\mathcal{O}(1)$ expected rounds for the consensus stage, and $(k+1)\Delta$ for the retrieval stage. In summary, when $t = \Theta(n)$, the wUMVBA protocol achieves communication complexity $\mathcal{O}(n^2\ell + n^3\lambda + n^2\lambda^2)$, message complexity $\mathcal{O}(n^3)$, and time complexity $\mathcal{O}(1)$.

7 The Optimized Weak Undead MVBA Protocol

In Section 6, the proposed wUMVBA protocol achieves communication complexity $\mathcal{O}(n^2\ell + n^3\lambda + n^2\lambda^2)$. As observed in [33], the $n^2\ell$ term dominates the

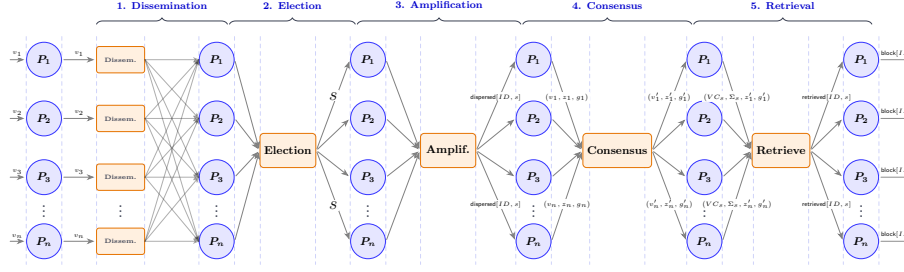


Fig. 3: The execution flow of the Optimized Weak Undead MVBA

overall performance when constructing atomic broadcast. We further observe that this $n^2\ell$ term mainly originates from the dissemination phases of the n concurrent DISPERSE instances. Therefore, in this section, we propose an Optimized Weak Undead MVBA protocol that reduces the communication complexity to $\mathcal{O}(n\ell + n^2\lambda^2)$.

Algorithm 5 implements an optimized variant of the wUMVBA protocol, referred to as Opt-wUMVBA. The key difference is that the UVID dispersal stage is divided into two phases: a *dissemination* phase, executed in parallel for all n senders, and an *amplification* phase, executed only for the dispersal instance corresponding to the elected sender after the undead common coin outputs its result. In wUMVBA, there are n amplification procedures executed concurrently. In contrast, Opt-wUMVBA reduces the number of amplification procedures to only one, while preserving all other stages identical to those in wUMVBA. The detailed execution flow of Opt-wUMVBA is illustrated in Figure 3.

Security Proof. Similar to the security analysis in Section 6.2, it is straightforward to prove that Algorithm 5 satisfies the properties of wUMVBA in Definition 3.

Complexity analysis. Compared with wUMVBA, Opt-wUMVBA executes only one amplification procedure while keeping all other procedures unchanged. Hence, by an analysis similar to that in Section 6.2, when $t = \Theta(n)$, the Opt-wUMVBA protocol achieves communication complexity $\mathcal{O}(n\ell + n^2\lambda^2)$, message complexity $\mathcal{O}(n^2)$, and time complexity $\mathcal{O}(1)$.

8 Undead Atomic Broadcast

In this section, we construct undead atomic broadcast (UABC) by sequentially executing continuous Opt-wUMVBA instances (Section 7) to generate a totally ordered log. Recall that the consensus component is implemented using the UMVC[ID] protocol from [21], which does not guarantee that all honest parties terminate in the same round. However, the DISPERSE protocol requires all honest parties to participate within the same round window. Therefore, directly starting Opt-wUMVBA[ID + 1] only after Opt-wUMVBA[ID] outputs may cause honest parties to participate in Opt-wUMVBA[ID + 1] at different times, which

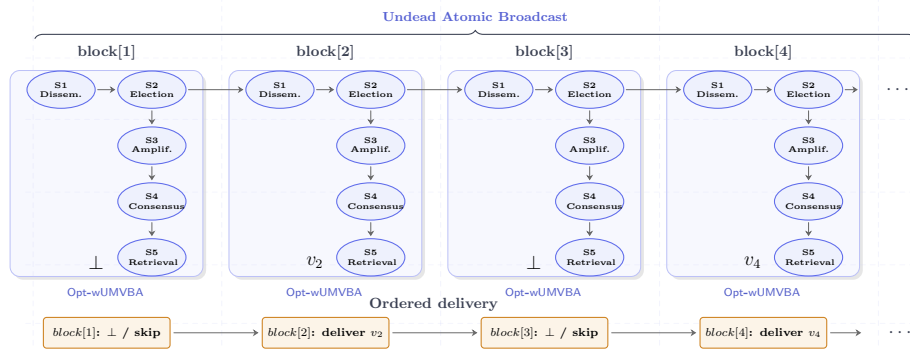


Fig. 4: The execution flow of Undead Atomic Broadcast

may prevent the protocol from functioning correctly. To address this issue, as highlighted in Figure 4, we start executing $\text{Opt-wUMVBA}[ID + 1]$ immediately after the election stage of $\text{Opt-wUMVBA}[ID]$ completes, rather than waiting for the output of $\text{Opt-wUMVBA}[ID]$.

Algorithm 6 therefore starts $\text{Opt-wUMVBA}[k]$ at time $\tau = (k - 1) \cdot 2\Delta$, since the election result of $\text{Opt-wUMVBA}[k - 1]$ becomes available at time $\tau = k \cdot 2\Delta$. Considering Opt-wUMVBA selects only one party's input as output, the inputs of other honest parties may not be delivered in the current instance. Therefore, starting a new Opt-wUMVBA instance immediately after the election stage serves two purposes. First, the elected party can avoid repeatedly proposing the same value in subsequent Opt-wUMVBA instances. Second, other honest parties whose values were not selected can continue proposing their inputs in later instances, thereby increasing the chance that their values are output earlier. We formally define UABC as follows.

Definition 4 (Undead Atomic Broadcast (UABC)). *The UABC protocol is executed by n parties $\mathcal{P} = \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ in the mixed-fault model. Formally, we say that UABC satisfies the following properties except with negligible probability:*

- **Validity.** *If a value v is input by all non-send-faulty honest parties, then v is eventually output by all non-zombie honest parties.*
- **Agreement.** *If an honest party outputs a value v , then every non-zombie honest party outputs the same value v eventually.*
- **Total order.** *If a non-zombie honest party $\mathcal{P}_i$ outputs a sequence of values $\{v_1, v_2, \dots, v_s\}$ and another non-zombie honest party $\mathcal{P}_j$ outputs a sequence of values $\{v'_1, v'_2, \dots, v'_{s'}\}$, then $v_k = v'_k$ for all $k \in [1, \min\{s, s'\}]$.*

8.1 Analysis of Undead Atomic Broadcast

Lemma 12 (Validity). *Algorithm 6 satisfies the Validity property of Definition 4.*

Algorithm 6 Undead Atomic broadcast, code for $\mathcal{P}_i$:

```

1: for  $k = 1, 2, 3, \dots$  do
2:   if  $\tau = (k - 1) \cdot 2\Delta$  then
3:     input  $v_i$  into  $\text{Opt-wUMVBA}[k]$ 

4: upon  $\text{block}[k] \leftarrow \text{Opt-wUMVBA}[k]$  do
5:   wait until  $\text{block}[k - 1]$  has already been output       $\triangleright \text{block}[0]$ : genesis block.
6:   output  $\text{block}[k]$ 

```

Proof. Suppose all non-send-faulty honest parties hold the same value v that has not yet been output. In every instance $\text{Opt-wUMVBA}[k]$ where v remains pending, each non-send-faulty honest party inputs v into $\text{Opt-wUMVBA}[k]$.

The common coin $\text{UCOIN}[ID_k]$ elects a uniformly random sender index s_k . Since there are at least $n - t - s$ non-send-faulty honest parties, the probability that $\mathcal{P}_{s_k}$ is a non-send-faulty honest party is at least $(n - t - s)/n$. By Lemma 10, whenever $\mathcal{P}_{s_k}$ is non-send-faulty honest, every non-receive-faulty honest party outputs v from $\text{Opt-wUMVBA}[k]$, and thus $\text{block}[k] = v$. By Lemma 11, every such party is non-zombie. Then, by Lemma 7, every non-zombie honest party outputs the same value v from $\text{Opt-wUMVBA}[k]$.

Each $\text{UCOIN}[ID_k]$ uses a distinct identifier, so the probability that none of the first q executions outputs $\text{block}[k] = v$ is at most $((t + s)/n)^q$. Therefore, with overwhelming probability, there exists some k' such that $\text{block}[k'] = v$. By Lemma 8, every honest party eventually obtains an output for each preceding instance. According to the output rule in Algorithm 6, every non-zombie honest party eventually outputs v .

Lemma 13 (Agreement). *Algorithm 6 satisfies the Agreement property of Definition 4.*

Proof. Suppose an honest party $\mathcal{P}_i$ outputs a value v . Then $\text{block}[k] = v$ for some k . By Lemma 7 of Opt-wUMVBA , every non-zombie honest party obtains the same output v from $\text{Opt-wUMVBA}[k]$. Furthermore, by Lemma 8, every such party eventually obtains an output from $\text{Opt-wUMVBA}[s]$ for every $s < k$. Therefore, according to the output rule in Algorithm 6, every non-zombie honest party eventually outputs the same value v .

Lemma 14 (Total order). *Algorithm 6 satisfies the Total order property of Definition 4.*

Proof. Let $\mathcal{P}_i$ and $\mathcal{P}_j$ be two non-zombie honest parties. By Lemma 7, they obtain the same output from $\text{Opt-wUMVBA}[k]$ for every identifier k . Both parties process the instances in the same increasing order of identifiers. By the agreement property of Opt-wUMVBA , all non-zombie honest parties obtain the same output for each instance. Therefore, they skip the same $\perp$ slots and collect the same non- $\perp$ values in the same order. Hence, $v_m = v'_m$ for all $m \in [1, \min\{s, s'\}]$.

Complexity analysis. We now analyze the complexity of UABC. Each output of UABC is obtained from an execution of Opt-wUMVBA. By the quality property of Opt-wUMVBA, the probability that the output is $\perp$ is at most $(t+s)/n$. Since $n > 2t + r + s$, the expected number of Opt-wUMVBA executions required to output one non- $\perp$ value proposed by an honest party is at most $n/(t+r)$, which is a constant when $t = \Theta(n)$.

When $t = \Theta(n)$, the Opt-wUMVBA protocol achieves communication complexity $\mathcal{O}(n\ell + n^2\lambda^2)$, and message complexity $\mathcal{O}(n^2)$. Therefore, UABC achieves the same asymptotic expected complexity: communication complexity $\mathcal{O}(n\ell + n^2\lambda^2)$, message complexity $\mathcal{O}(n^2)$, and time complexity $\mathcal{O}(1)$ per delivered value.

8.2 Discussion

Regarding the time complexity, each execution of Opt-wUMVBA outputs the input of only one honest party. Consequently, the inputs of other honest parties may need to wait for later Opt-wUMVBA executions to be output. Since the probability of selecting any particular honest party is only $\Theta(1/n)$, the worst-case expected latency for an individual honest input may grow to $\mathcal{O}(n)$.

To reduce the latency to expected constant time, one can further incorporate ideas from asynchronous common subset (ACS) protocols [25, 26, 35]. In particular, instead of directly using Opt-wUMVBA, we can use wUMVBA to construct a synchronous common subset protocol. The high-level idea is as follows. Each party first inputs its value into one wUMVBA instance. Running n wUMVBA instances in parallel produces an n -dimensional output vector. Each party then multicasts its vector and obtains an updated vector. For every non- $\perp$ entry in the updated vector, the corresponding certificate guarantees reconstructability of the associated value. Since the network is synchronous, all non-send-faulty honest parties can ensure that the certificates corresponding to their input values appear consistently in every honest party's updated vector. Subsequently, parties use the updated vector as input to Algorithm 6. In this way, each execution can output $\Theta(n)$ values simultaneously. Furthermore, following the batching technique of HoneyBadgerBFT [35], parties may aggregate a batch of transactions, partition the batch into n pieces before dispersal, and randomly select one piece as the input of wUMVBA. This allows the communication overhead to be amortized across many transactions, yielding amortized communication complexity $\mathcal{O}(n\ell + n^2\lambda^2)$ together with expected constant latency.

Acknowledgments. We thank our anonymous reviewers for their constructive comments. The work of Qiang and Yuchen was 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. The work of Zhenliang and Dalin was supported in part by a grant from City University of Hong Kong (Project No. 9610761).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Abraham, I., Dolev, D., Kagan, A., Stern, G.: Authenticated consensus in synchronous systems with mixed faults. *Cryptology ePrint Archive* (2022)
2. Abraham, I., Malkhi, D., Nayak, K., Ren, L., Yin, M.: Sync hotstuff: Simple and practical synchronous state machine replication. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 106–118. IEEE (2020)
3. Abraham, I., Malkhi, D., Spiegelman, A.: Asymptotically optimal validated asynchronous byzantine agreement. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. pp. 337–346 (2019)
4. Alhaddad, N., Das, S., Duan, S., Ren, L., Varia, M., Xiang, Z., Zhang, H.: Brief announcement: Asynchronous verifiable information dispersal with near-optimal communication. In: Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing. pp. 418–420 (2022)
5. Aublin, P.L., Mokhtar, S.B., Quéma, V.: Rbft: Redundant byzantine fault tolerance. In: 2013 IEEE 33rd international conference on distributed computing systems. pp. 297–306. IEEE (2013)
6. Bessani, A., Sousa, J., Alchieri, E.E.: State machine replication for the masses with bft-smart. In: 2014 44th Annual IEEE/IFIP international conference on dependable systems and networks. pp. 355–362. IEEE (2014)
7. Bracha, G.: Asynchronous byzantine agreement protocols. *Information and Computation* **75**(2), 130–143 (1987)
8. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: Annual International Cryptology Conference. pp. 524–541. Springer (2001)
9. Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: 24th IEEE Symposium on Reliable Distributed Systems (SRDS’05). pp. 191–201. IEEE (2005)
10. Cachin, C., Vukolić, M.: Blockchain consensus protocols in the wild. In: 31 International Symposium on Distributed Computing (2017)
11. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation. vol. 99, pp. 173–186. The USENIX Association (1999)
12. Clement, A., Wong, E., Alvisi, L., Dahlin, M., Marchetti, M., et al.: Making byzantine fault tolerant systems tolerate byzantine faults. In: Proceedings of the 6th USENIX symposium on Networked systems design and implementation. pp. 153–168. The USENIX Association (2009)
13. Cristian, F., Aghili, H., Strong, R., Dolev, D.: Atomic broadcast: From simple message diffusion to byzantine agreement. *Information and Computation* **118**(1), 158–179 (1995)
14. Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.: Narwhal and tusk: a dag-based mempool and efficient bft consensus. In: Proceedings of the Seventeenth European Conference on Computer Systems. pp. 34–50 (2022)
15. Défago, X., Schiper, A., Urbán, P.: Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys (CSUR)* **36**(4), 372–421 (2004)
16. Duan, S., Reiter, M.K., Zhang, H.: Beat: Asynchronous bft made practical. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 2028–2041 (2018)
17. D’Amato, F., Saltini, R., Tran, T.H., Zanolini, L.: Tob-svd: Total-order broadcast with single-vote decisions in the sleepy model. In: 2025 IEEE 45th International

- Conference on Distributed Computing Systems (ICDCS). pp. 1033–1043. IEEE (2025)
18. Efron, Y., Tas, E.N.: Dynamically available common subset. *Cryptology ePrint Archive* (2025)
 19. Eldefrawy, K., Loss, J., Terner, B.: How byzantine is a send corruption? In: International Conference on Applied Cryptography and Network Security. pp. 684–704. Springer (2022)
 20. Feng, H., Lu, Z., Mai, T., Tang, Q.: Faster hash-based multi-valued validated asynchronous byzantine agreement. In: 2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 303–316. IEEE (2025)
 21. Feng, H., Lu, Z., Tang, Q., Ye, Y.: Optimal byzantine agreement in the presence of message drops. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 163–195. Springer (2025)
 22. Gao, Y., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 1187–1201 (2022)
 23. Garay, J.A., Perry, K.J.: A continuum of failure models for distributed computing. In: International Workshop on Distributed Algorithms. pp. 153–165. Springer (1992)
 24. Gramoli, V., Lu, Z., Tang, Q., Zarbafian, P.: Aoab: optimal and fair ordering of financial transactions. In: 2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 377–388. IEEE (2024)
 25. Guo, B., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Speeding dumbo: Pushing asynchronous bft closer to practice. *Cryptology ePrint Archive* (2022)
 26. Guo, B., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo: Faster asynchronous bft protocols. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 803–818 (2020)
 27. Hadzilacos, V.: Issues of fault tolerance in concurrent computations. *Dissertation Abstracts International Part B: Science and Engineering*[DISS. ABST. INT. PT. B- SCI. & ENG.], **46**(7) (1986)
 28. Kamp, S.H., Loss, J., Nayak, K., Shi, K.: Omniba: Round-efficient ba with quadratic communication under mixed faults. *Cryptology ePrint Archive* (2025)
 29. Keidar, I., Kokoris-Kogias, E., Naor, O., Spiegelman, A.: All you need is dag. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing. pp. 165–175 (2021)
 30. Koo, C.Y.: Secure computation with partial message loss. In: Theory of Cryptography Conference. pp. 502–521. Springer (2006)
 31. Loss, J., Shi, K., Stern, G.: Consensus in the presence of overlapping faults and total omission. In: Theory of Cryptography Conference. pp. 353–382. Springer (2024)
 32. Loss, J., Stern, G.: Zombies and ghosts: Optimal byzantine agreement in the presence of omission faults. In: Theory of Cryptography Conference. pp. 395–421. Springer (2023)
 33. Lu, Y., Lu, Z., Tang, Q., Wang, G.: Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In: Proceedings of the 39th symposium on principles of distributed computing. pp. 129–138 (2020)
 34. Malkhi, D., Momose, A., Ren, L.: Towards practical sleepy bft. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 490–503 (2023)

35. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of bft protocols. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. pp. 31–42 (2016)
36. Momose, A., Ren, L.: Constant latency in sleepy consensus. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2295–2308 (2022)
37. Parvédy, P.R., Raynal, M.: Uniform agreement despite process omission failures. In: Proceedings International Parallel and Distributed Processing Symposium. pp. 7–pp. IEEE (2003)
38. Pass, R., Shi, E.: The sleepy model of consensus. In: International conference on the theory and application of cryptology and information security. pp. 380–409. Springer (2017)
39. Perry, K.J., Toueg, S.: Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering* (3), 477–482 (1986)
40. Ramasamy, H.V., Cachin, C.: Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast. In: International Conference On Principles Of Distributed Systems. pp. 88–102. Springer (2005)
41. Schmiedel, H., Han, R., Tang, Q., Steinfeld, R., Yu, J.: Constant latency and finality for dynamically available dag. In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 1910–1927. IEEE (2025)
42. Siu, H.S., Chin, Y.H., Yang, W.P.: Byzantine agreement in the presence of mixed faults on processors and links. *IEEE Transactions on Parallel and Distributed Systems* 9(4), 335–345 (1998)
43. Spiegelman, A., Girdharan, N., Sonnino, A., Kokoris-Kogias, L.: Bullshark: Dag bft protocols made practical. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2705–2718 (2022)
44. Sui, X., Wang, X., Duan, S.: Signature-free atomic broadcast with optimal $\mathcal{O}(n^2)$ messages and $\mathcal{O}(1)$ expected time. In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 1547–1565. IEEE (2025)
45. Tang, Q., Ye, Y.: On the communication complexity of sleepy consensus. *Cryptology ePrint Archive* (2026)
46. Veronese, G.S., Correia, M., Bessani, A.N., Lung, L.C.: Spin one’s wheels? byzantine fault tolerance with a spinning primary. In: 2009 28th IEEE International Symposium on Reliable Distributed Systems. pp. 135–144. IEEE (2009)
47. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: Bft consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM symposium on principles of distributed computing. pp. 347–356 (2019)
48. Zikas, V., Hauser, S., Maurer, U.: Realistic failures in secure multi-party computation. In: Theory of Cryptography Conference. pp. 274–293. Springer (2009)