Optimal Byzantine Agreement in the Presence of Message Drops

Hanwen Feng[1], Zhenliang Lu[2], Qiang Tang[1], and Yuchen Ye[1]✉

[1] The University of Sydney, Sydney, Australia
hanwen.feng@sydney.edu.au, qiang.tang@sydney.edu.au, yuchenye19@gmail.com
[2] City University of Hong Kong, Hong Kong, China
zhenliang.lu@cityu.edu.hk

Abstract. To more accurately capture real-world network and adversarial behaviors, recent research has explored Byzantine Agreement (BA) under various mixed-fault models. The breakthroughs by Loss et al. (TCC'23, TCC'24) have established the feasibility of optimally resilient BA in some of these models. Specifically, their protocols tolerate up to t byzantine parties, r receive faulty parties, and s send faulty parties in a network of $n > 2t + r + s$ parties. Initially, Loss et al. (TCC'23) considers a model in which a party may be either receive faulty or send faulty but not at the same time (called the *non-overlapping setting*). The extended model in Loss et al. (TCC'24) further accommodates the *overlapping setting*, where a party can simultaneously exhibit both receive faulty and send faulty behaviors. However, despite this flexibility, *both* protocols incur a prohibitively high communication cost of $O(n^5)$ bits, leaving open the fundamental question of whether the optimal $O(n^2)$-bit complexity achieved by many classical BA protocols is attainable in the optimally resilient mixed-fault model (with overlapping faults or not).

In this work, we answer these open questions affirmatively. We present a mixed-fault BA protocol that achieves the optimal expected $O(n^2\lambda)$ communication complexity while maintaining expected $O(1)$ round complexity and optimal (rushing and strongly adaptive) resilience. Our protocol supports the strongest overlapping setting, while matching the best-known complexity of classical BA protocols. To achieve this, we develop a series of novel techniques, carefully designed to ensure efficient and secure agreement even under the mixed-fault model. Beyond binary BA, we extend our protocol to a multi-valued BA setting, achieving an expected $O(\frac{n^2}{t}L + n^2\lambda^2)$ communication complexity and an $O(1)$ expected round complexity, where t is the number of byzantine faults and L is the bit-length of the input values. In particular, for $t = O(n)$, the communication reduces to $O(nL + n^2\lambda^2)$. Notably, our protocols operate under the same setup and cryptographic assumptions as those in Loss et al.

Keywords: Byzantine Agreement, Omission Fault

1 Introduction

Byzantine Agreement (BA)[1] [29] is arguably one of the most fundamental problems in distributed computing and cryptography. It forms the backbone of state machine replication [5, 14], distributed randomness and key generation [4, 20], verifiable secret sharing [7, 9, 37], and general multi-party computation (MPC) [13, 24]. In essence, the BA problem allows a set of n parties denoted by $P_1, \ldots, P_n$, each holding an input v_i, to reach consensus on a single "valid" output v, even in the presence of a portion of corrupted parties.

The central question in BA research is how to improve performance while tolerating more faulty parties under various setup and network conditions. Classical results [17, 22, 29] showed that without a Public Key Infrastructure (PKI) setup or in asynchronous networks, the maximum number of

[1] Throughout the paper, we refer to BA as the binary version, i.e., each party has one bit as input. For large input values, it is called multi-valued BA.

byzantine faults a BA protocol can tolerate is less than $n/3$. To tolerate more corruptions, specifically up to $t < n/2$, a line of works [2,11,16,27,34] focused on the synchronous settings with PKI setup, where significant efforts have been made to optimize BA performance. For instance, the pioneering BA protocol by Dolev and Strong [16] requires $O(\lambda n^3 L)$ communication and $O(n)$ rounds to agree on a value of L bits. Fitzi-Garay [23] and Katz-Koo [27] later reduced the expected round complexity to $O(1)$. More recently, Abraham et al. [2] and Nayak et al. [35] developed a BA protocol. It, with a threshold signature setup, achieves a communication cost of $O(nL + \lambda n^2)$ and $O(1)$ rounds, which indeed matches the lower bound on communication cost [1].

Characterizing message dropping: BA also with omission faults. While an optimally resilient BA can tolerate at most $t = \lfloor \frac{n-1}{2} \rfloor$ *arbitrarily* faulty (or called malicious) parties, the remaining $> n/2$ parties must *all* stay in a good condition, *e.g.*, with a reliable network, to ensure the security of BA. The requirement may not always be taken for granted. As today's network infrastructure is unfortunately centralized, the adversary does have a better chance to corrupt certain network components and interfere with the network connectivity of a large range of parties. In the latter case, the non-faulty majority assumption no longer holds, and the conventional BA protocols cannot offer security.[2]

Non-malicious parties that fail to receive or send some of their messages are categorized as experiencing omission faults [26,36,38], which encompass both crash faults and faults caused by an adversarial network (i.e., byzantine faults). Since omission and crash failures are more benign but also more common than byzantine faults, mixed-fault models that consider both omission (or crash) faults and byzantine faults have been explored in the literature [25, 39, 40]. These models aim to investigate what is possible in scenarios where only a minority of parties are non-faulty, and where a byzantine-fault-only model offers no viable solutions.

The early work by Garay and Perry [25] introduced a protocol that tolerates a mix of t byzantine and c crash faults, showing that BA is possible when $n > 3t + c$. Abraham et al. [3] later constructed an authenticated consensus protocol that works for $n > 2t + c$.

On omission faults, Siu et al. [39] showed that BA is possible when $n > 3t + k$, where k is the number of arbitrary non-malicious faulty parties. Koo [28] proved that BA and MPC are possible when $n > 3t + 2o$ and $n > 3t + 4o$ respectively, where o is the number of omission faults. Zikas et al. [40] explored a fine-grained characterization by dividing omission faults into *send-omission* faults and *receive-omission* faults. A send faulty party may fail to send some messages, while a receive faulty party may fail to receive some messages, but these parties remain *honest*, i.e., following the protocol. Their work proved that BA and MPC are feasible when $n > 3t + r + s$, where r and s are the number of receive faulty and send faulty parties, without overlapping (implicit in the proof). Recently, Brazitikos and Zikas [8] provided a tight characterization of feasibility of MPC in the presence of general adversaries that mix omission and byzantine faults. Moreover, Eldefrawy et al. [18] proved the corruption lower bound, stating that BA is possible only if $n > 2t + r + s$ even with cryptographic setup. Still, their protocol did not match the bound; it tolerates $n > 2t + 2s + r$, or $n > 2t + r + s$ only when the omissions are spotty, i.e., the adversary has to block either none or all of the messages at once.

BA with *optimal* mixed-fault tolerance remained open until recently, Loss and Stern [31] presented a BA protocol that meets the corruption lower bound of $n > 2t + r + s$, but it assumed that

[2] Of course, one can always view those parties as faulty, but that would immediately reduce the resilience of the protocol, which can only tolerate a much smaller fraction of corruptions; jumping ahead, the number of those send-omission faults s or receive-omission faults r could be as large as $n - 1$.

a party cannot experience send-omission faults and receive-omission faults at the same time (i.e., the non-overlapping setting). Later, Loss et al. [30] finally lifted this restriction and achieved the optimal resilience also in the *overlapping* setting. Nevertheless, both works only established feasibility. Their constructions suffer from large $O(n^5)$ expected communication complexity, and are for binary BA only. Thus, it is a natural question to reduce the communication complexity (indeed an open question left in [31]). In this paper, we explore the following question:

Is BA with omission inherently more communication-costly? Or, can we reduce the communication complexity of BA protocols in such settings, to match the $O(n^2)$ lower bound for classical BA [15]?

In this work, we provide an affirmative answer to this open question. We develop the first protocol that achieves optimal communication complexity in this mixed-fault model. Specifically, we make the following contributions:

- **Optimal Communication Complexity:** We show that it is possible to reduce the communication complexity of byzantine agreement even in the *overlapping* mixed-fault model to $O(n^2\lambda)$ while maintaining expected $O(1)$ round complexity, matching the optimal complexity of classical BA.
- **Multi-valued Byzantine Agreement:** We also extend our protocol to support multi-valued BA in the mixed-fault model. Our protocol achieves expected $O(\frac{n^2}{t}L + n^2\lambda^2)$ communication complexity and expected $O(1)$ round complexity, where t is the number of byzantine faults (communication becomes $O(nL + n^2\lambda^2)$ when $t = O(n)$), and L is the length of the input value. Notably, the classical approach of compiling binary BA to multi-valued BA does not directly work in the mixed-fault settings, so this needs further care.

1.1 Comparison with the Conference Version

Compared to the conference version of this paper [21], the current version presents an improved multi-valued BA protocol, reducing the round complexity from $O(\kappa)$ to $O(1)$, where κ is the statistical security parameter. Moreover, this version also provides complete proofs of the detection soundness property for our undead very weak multicast primitive and BA protocol, which were only sketched in the conference version. In addition, the round and communication complexity analyses of the BA and the multi-valued BA protocols are extended and are now formally presented as theorems, together with their complete proofs. Last but not least, this version contains a more comprehensive preliminaries section, including complete definitions that were omitted from the conference version.

2 Model

Network Model. Throughout this work, we consider a fully connected network of n parties, where each pair of parties communicates directly through an authenticated channel. Moreover, we assume a trusted setup for the unique threshold signature scheme. Namely, before the start of the protocol, each party is given its secret key share, and all parties have access to the same public keys.

The network is assumed to be synchronous, where parties have synchronized clocks, and the protocol progresses in well-defined rounds. At the start of each round, parties send messages, and all messages from honest parties that are not dropped by the adversary are delivered by the end of the round. The adversary can determine the order of messages being delivered within the round.

Adversary Model. Our work aims to design protocols capable of tolerating parties with mixed-fault behaviors. We consider four types of corruptions:

- **Send-Omission Faults.** If a party experiences a send-omission fault, the adversary can choose to drop any messages sent from that party. The party is said to be send faulty. A party that detects itself as being send faulty is a *ghost*.
- **Receive-Omission Faults.** If a party experiences a receive-omission fault, the adversary can choose to drop any messages sent to that party. The party is said to be receive faulty. A party that detects itself as being receive faulty is a *zombie*.
- **Full-Omission Faults.** If a party experiences both send-omission and receive-omission faults, the adversary can choose to drop any messages sent to or from that party. The party is said to be full-omission faulty.
- **Byzantine Faults.** If a party experiences a byzantine fault, it will leak its internal state to the adversary and remain fully controlled by the adversary, allowing it to behave arbitrarily.

We adopt the following terminology to describe parties experiencing different faults: **Honest** parties are all parties except those experiencing **Byzantine** faults; **Non-send-faulty honest** (resp., **Non-receive-faulty honest**, or **Non-full-omission-faulty honest**) parties are honest parties except those experiencing send-omission faults (resp., receive-omission faults, or full-omission faults); **Non-faulty** parties are those not experiencing any faults. We assume that $n > 2t + r + s$, where there are at most r receive faulty, s send faulty, and t byzantine parties; a party experiencing a full-omission fault is counted towards both send faulty and receive faulty parties.

We consider a rushing and strongly adaptive adversary that can corrupt parties, causing them to experience the above fault types at any time throughout the execution. Specifically, if a party is corrupted by the adversary as a byzantine party, the adversary can drop any messages sent by that party if those messages have not been delivered to the receivers.

3 Technical Overview

We start with the simpler *non-overlapping* setting, where a party cannot be simultaneously send faulty and receive faulty, to explain the unique challenges of the mixed-fault model and review the design methods of recent works. Then, we continue to focus on the non-overlapping setting, to illustrate our core ideas for reducing communication complexity. Next, we demonstrate how our techniques can be extended to handle the more challenging *overlapping* setting, resulting in our communication-optimal binary BA in the model. Lastly, we introduce our extension techniques for achieving efficient multi-valued BA in the same model.

3.1 Challenges and the Design Method of Loss and Stern [31]

Challenges in enabling BA to tolerate omission faults. A classical BA protocol, if used directly, immediately fails in this setting because of the presence of receive faulty and send faulty parties. The consistency property of BA requires all honest parties to agree on the same value. However, (1) if a receive faulty party does not receive any messages from other parties, it lacks sufficient information to determine which value to agree on, and (2) message deliveries from send faulty parties are not reliable anymore, leaving room for equivocation. More specifically, the adversary can simply leverage the capability of *send-omission faults* to create partitions among *honest* parties, and emulate an attack similar to those in asynchronous networks.

Detect and amplify: the design method of Loss and Stern [31]. To make BA in this setting feasible, previous works [31, 40] designed an (imperfect) detection mechanism for receive and send faulty parties to try self-detection, and "amplify" limited message delivery, by strengthening the multicast protocols, as shown in Fig. 1. The key observation is that simple (imperfect) detection mechanisms already limit the adversary's attack strategy, not to completely cut off the connections of these receive faulty and send faulty parties, otherwise detection is easy, and the protocol can be designed accordingly. More importantly, the parties are able to jointly amplify the limited connectivity to facilitate effective message propagation to the whole network.

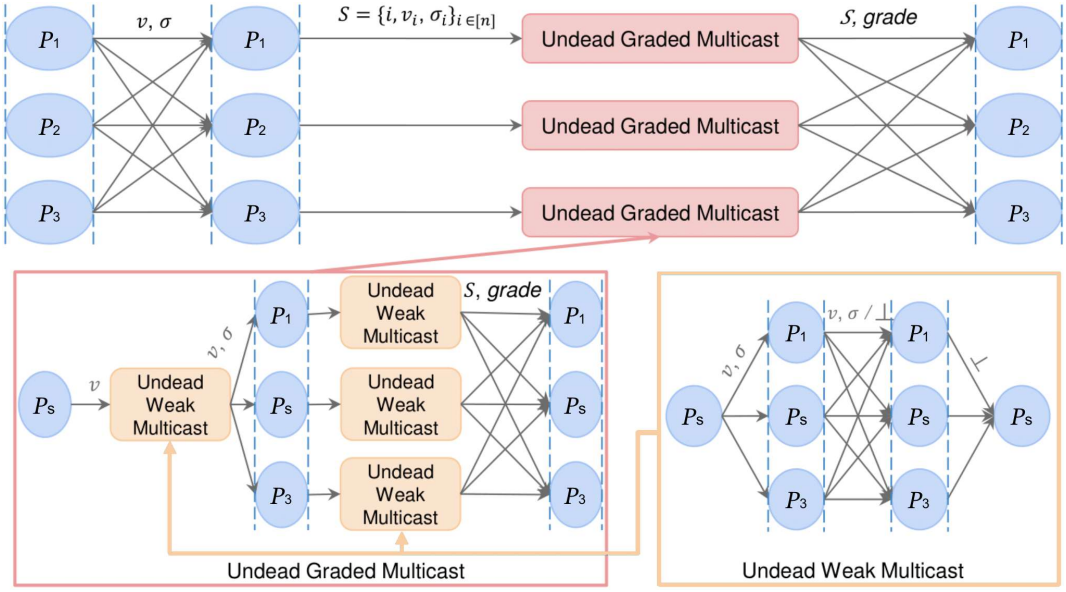

Fig. 1. The structure of weak BA [31].

Detection and treatments for receive faulty parties. First, if receive faulty parties detect themselves as being receive faulty (and thus become zombies), then it is hopeless for them to output anything meaningful (as they may never receive any messages from the network). For these parties, we can redefine the validity and consistency properties of BA so that zombies are excluded to avoid the trivial impossibility (thus they simply terminate and claim to be zombies). But for the other receive faulty parties (that have not claimed to be zombies), they still need to be considered for validity and consistency.

One easy way for parties to self-check whether they are zombies is to let them constantly check whether they receive messages from a sufficient number of parties, i.e., $n - t - s$ parties. If not, the party claims to be a *zombie* and terminates. It is easy to see that a non-receive-faulty party can always receive messages from non-send-faulty honest parties.

However, this simple detection mechanism can be trivially circumvented by an adversary by allowing a message from a send faulty party to be delivered to a receive faulty party. For these receive faulty parties, since they will not claim to be zombies, we must still ensure the validity and consistency of their outputs. The challenge is that these receive faulty parties can receive messages but may not receive *enough* to form a reliable "quorum". This issue was mitigated in [31]

by initially focusing on a *weak* Byzantine Agreement (BA) protocol, which provides a relaxed form of consistency, known as *weak consistency*.

Informal Definition 1 (Weak consistency of weak BA [31]) *If an honest party outputs $v \neq \perp$, then every honest party that is not a zombie should output either v or $\perp$.*

This property is easier to achieve, as those receive faulty parties can directly output $\perp$ if they receive messages more than that they can claim to be zombie, but fewer than a reliable quorum size. In addition, while weak BA is strictly weaker than standard BA which guarantees that all honest parties output the same value, fortunately, there is a standard and efficient procedure to convert a weak BA into standard BA [31].

Detection and amplification for send faulty parties. Second, while send faulty parties can still receive messages, their unreliable message deliveries (controlled by the adversary) can disrupt others, potentially emulating network partitions. To mitigate this, a natural idea is to leverage a detection mechanism where a potential *ghost* (a send faulty party) can signal the presence of faults. Specifically, each message that could trigger receiver actions is followed by a second message for confirmation. If a send faulty party identifies itself as a ghost, it simply stays silent, avoiding confirmation. Consequently, a receiver only trusts a message after receiving two legitimate transmissions from the same sender. This ensures that the first message has been properly delivered, even if the sender is send faulty but has not yet declared itself as a ghost.

The above pattern naturally inspires the use of graded multicast. Indeed, Loss and Stern [31] formulated a new primitive called *undead graded multicast* to replace message delivery (see Fig. 1). In an undead graded multicast protocol, if an honest party receives a message m from the sender in the first round and then receives m from some party in the second round, it is said that the party receives the message m with a grade of 2. Otherwise, m is with a grade of 1. The undead graded multicast ensures the following validity and consistency property.

Informal Definition 2 (Undead graded multicast [31]) *It satisfies:*

- ***Validity:*** *For a non-send-faulty honest sender,* ***every honest party*** *will either receive the message* ***or become a zombie****.*
- ***Consistency:*** *For an honest sender, if an honest party receives v with a grade of 2,* ***all*** *honest parties except zombies receive v with a grade of ≥ 1.*

With this property, weak BA illustrated in Fig. 1 can ensure *weak consistency*. Specifically, an honest party outputs v only when it has received v with a grade of 2 from $t+1$ senders via undead graded multicast, and among the senders at least one is honest. In this case, all other honest parties must have received v with a grade of at least 1, so they will not output a conflicting value $v' \neq \perp$.

One remaining module is the detection of send-omission fault itself (which will be the core component of undead graded multicast). The idea is also fairly natural: a sender is expected to infer from the number of responses to each message it sends out. Specifically, [31] implements a module called *undead weak multicast* for this purpose, replacing the previous simple message delivery. In the module, the sender multicasts a message to all parties. The message is forwarded between the parties, and eventually parties that still have not received the message reply abort back to the sender, allowing the sender to learn whether it succeeds in sending its message or not.

An important observation here is that the adversary has to drop or omit sufficient messages sent by a send faulty party P_i to partition him, but that will automatically allow other parties to inform P_i this fact helping him identify himself as a ghost. Specifically, it satisfies the following property.

Informal Definition 3 (Detection of undead weak multicast [31]) *If a send faulty sender does not become a ghost, the message is delivered to at least one* ***non-faulty*** *party.*

This gives us the opportunity to further "amplify" and let the non-faulty party to forward this message to the whole network.

$O(n^5)$-bit communication cost in [31] (and also [30]). However, we can also see (from Fig. 1) that the method above is expensive. Every party will be running n concurrent instances of *undead graded multicast*, each of which takes a vector of size $O(n)$ bits as input, and further invoking n instances of *undead weak multicast*. There is an *all-to-all* communication in undead weak multicast, resulting in $O(n^5)$ communication in total.

3.2 Our Techniques for Achieving Quadratic Communication

Achieving optimal $O(n^2)$ communication in the non-overlapping setting. One may reduce the complexity by a factor of $O(n)$ by leveraging some extension tricks [32,35]. However, to achieve $O(n^2)$ communication, it is necessary to ensure that there is essentially "no overhead" in every step. While the design above strengthens the basic message delivery in multiple layers (i.e., undead graded multicast, and undead weak multicast), it incurs several orders of magnitude communication blowup. As we show below, to achieve our goal, we essentially need to refine *every* layer.

A weaker multicast module. We start with the basic unit of undead weak multicast, which ensures either send-omission fault detection, or a guaranteed delivery to one non-faulty party. The latter is leveraged to further propagate the message.

Observe that the receive faulty parties are honest and, by definition, not send faulty, so it would be possible to have them help propagate messages. This gives us the opportunity to weaken this module to get a more communication-efficient construction, e.g., getting rid of the all-to-all pattern. More specifically, we introduce our *undead very weak multicast* as a replacement for the undead weak multicast, which ensures the following detection property.

Informal Definition 4 (Detection of undead very weak multicast) *If a send faulty sender does not become a ghost, the message is delivered to at least one non-faulty* ***or receive faulty*** *party.*

This weaker guarantee allows us to achieve our undead very weak multicast with only $O(n)$ communication complexity, whereas the undead weak multicast incurs $O(n^2)$ communication complexity.

Our *undead very weak multicast* protocol works simply as follows (see also Fig. 2): the sender multicasts the message to all parties who send 1 back to the sender if they receive the message or 0 if they do not. If the sender receives at least $t + r + 1$ 0 messages, it becomes a ghost. This is sufficient already because if none of the non-faulty or receive faulty parties get the message, they will all send 0 to the sender. If the sender is not receive faulty, it will receive at least $t+r+1$ 0 and thus become a ghost. If the sender is receive faulty, then by definition, it is not send faulty, so all the non-faulty parties should receive the message sent by it. Apart from that, when a party detects itself as being receive faulty and thus becomes a zombie, it should keep sending messages when appropriate until the end of the iteration when it terminates to avoid message loss. See Sec. 5.1 for more details.

Looking ahead, leveraging receive faulty parties turns out to be subtle, because (1) they may not be able to receive the messages that they are supposed to help propagate, and (2) they may even terminate when they detect themselves as being receive faulty.

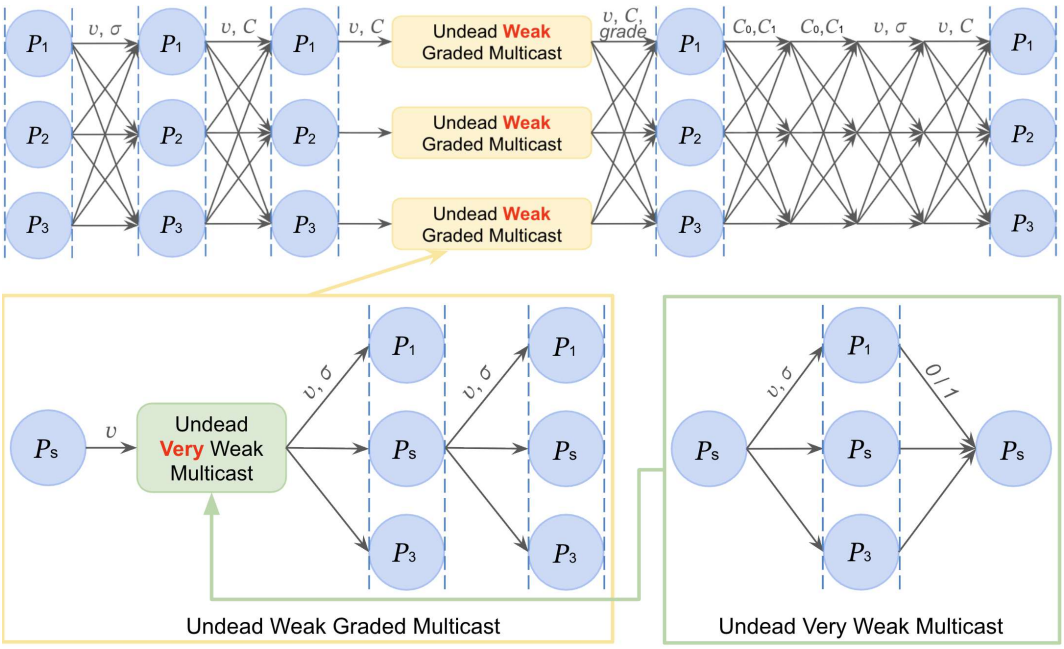

Fig. 2. The structure of our weak BA.

Weaker graded multicast. Next we examine the undead graded multicast. We would need to weaken this module so that we can use *fewer instances* of the *weakened* version of underlying multicast modules (i.e., our *undead very weak multicast*) (see Fig. 2); but still, we could realize a weak BA in the mixed-fault model under optimal corruption. More specifically, we introduce a weaker module called *undead weak graded multicast*, in which both the validity and consistency properties are weakened compared to undead graded multicast.

Informal Definition 5 (Undead weak graded multicast) *It satisfies:*

- ***Weak Validity:*** *For a non-send-faulty honest sender, all* ***non-receive-faulty honest*** *parties can output the message with the grade of* 2.
- ***Weak Consistency:*** *For an honest sender, if an honest party receives* v *with a grade of* 2, ***at least one*** *non-send-faulty honest parties receive* v *with a grade of* ≥ 1.

Compared with the undead graded multicast, the weak validity of our undead weak graded multicast does not guarantee that receive faulty parties who fail to output v will become zombies; the weak consistency only ensures the existence of at least one non-send-faulty honest party, instead of all the honest parties (except zombies) receiving the message with a grade of ≥ 1. In the following, we illustrate how we can efficiently construct a undead weak graded multicast.

First, we notice that, since there is a non-send-faulty honest party having the message at the end of undead very weak multicast (see Fig. 2), it is sufficient for every party to directly multicast (as in the conventional setting) their message instead of using undead very weak multicast, so that the party that holds the message can send the message to all parties.

Also, a closer observation reveals that in (binary) weak BA, where n concurrent undead graded multicast instances are executed, each instance requires every party to multicast once, and the messages being multicast are simply 0s and 1s accompanied by certificates. Consequently, many of these

multicasts are redundant. We optimize this by consolidating each party's n multicast operations into a single multicast.

In more detail, in our undead weak graded multicast, the sender first sends its message using undead very weak multicast, and then multicasts its message if and only if it does not become a ghost by the end of undead very weak multicast. If a party receives the message in the second round, we say the message is with a grade of 2, otherwise the message is with a grade of 1. This ensures that if a party receives the message with a grade of 2, we can deduce that the sender is not a ghost at the end of undead very weak multicast, so that at least one non-send-faulty honest party has received the message (thus with a grade of at least 1). See Sec. 5.2 for a formal description.

The communication complexity of our undead weak graded multicast is $O(n)$ in comparison to the $O(n^3)$ communication cost of the undead graded multicast in [31].

Realizing Weak BA using undead weak graded multicast. Now, with the weaker validity and consistency guarantees, directly applying undead weak graded multicast can lead to issues in both validity and weak consistency in the resulting undead weak binary consensus.

- *Validity Violation.* Our undead weak graded multicast no longer guarantees that receive faulty parties that do not receive the messages can become zombies. Consequently, some receive faulty parties may cause a *validity* violation in undead weak binary consensus, which requires that if all honest parties start the protocol with the same input v, they must either output v or become zombies by the end of the protocol.
- *Weak Consistency Violation.* In the undead weak binary consensus construction of [31] (also illustrated in Fig. 1) and our non-overlapping protocol, a party outputs v when it has received $t+1$ "certificates" for v with a grade of 2 from the undead graded multicast and has not received any certificate for $1-v$. However, unlike the undead graded multicast [31], our undead weak graded multicast only ensures that a certificate of v is available to *one* non-send-faulty honest party (when an honest party sends v and one honest party receives it with a grade of 2). As a result, other parties may collect $t+1$ certificates for $1-v$ with a grade of 2 and output $1-v$ without being aware of the certificate for v, which violates the weak consistency.

Thus, we refine the weak BA design and prevent these violations by applying the following treatments.

- To ensure weak consistency, we let the non-send-faulty honest parties who received a certificate of v with a grade of ≥ 1 help disseminate the certificate. Specifically, we introduce two extra rounds of all-to-all multicasts. In the first round, the parties who received the certificate of v with a grade of ≥ 1 multicast the certificate, such that every non-receive-faulty honest party receives the certificate. In the second round, all parties with the certificate multicast it to the network, ensuring that every receive faulty party will either (1) receive $n-t-s$ "second-round" messages, with at least one of them carrying the certificate, or (2) receive fewer than $n-t-s$ messages and become a zombie. As a result, all honest parties, except the zombies, receive the certificate of v by the end of the two rounds of multicasts.
- To ensure validity, we let all honest parties multicast the value they are going to output, which helps a receive faulty party either output the value or become a zombie. Specifically, we add two more rounds of multicasts at the end of weak BA. Assume without loss of generality that all honest parties plan to output either v or $\perp$. They multicast v (with their partial signatures) to all parties. Upon receiving $t+1$ signed identical values, a party forms a threshold signature

for the value and multicasts it. A party outputs v if it receives a valid threshold signature for v. Recall that a party that does not become a zombie receives from at least one non-receive-faulty honest party. And when all honest parties have the same input v, the non-receive-faulty honest parties can always form threshold signatures (as a certificate) for v. Thus, all honest parties either output v or become zombies, which satisfies the validity property of weak BA.

As shown in Fig. 2, the above treatments add four extra rounds of multicast at the end of the protocol, compared to the weak BA in [31].

Handling overlapping faults. The above techniques already lead to a quadratic-communication weak BA in the *non-overlapping* model. Now, we turn to strengthen the protocol to make it secure even in the *overlapping* setting [30].

On the one hand, we observe that our building blocks, including undead very weak multicast and undead weak graded multicast, cannot provide useful guarantees when the sender is full-omission faulty. Specifically, in undead very weak multicast, a full-omission faulty sender may fail to send messages to any non-send-faulty honest parties. However, as it also experiences receive faulty behavior, it may fail to receive enough messages of 0 (indicating unsuccessful message delivery) from receivers, and thus it will not become a ghost, which violates the detection property (cf. Definition 4). Consequently, undead weak graded multicast also loses the security for a full-omission faulty sender.

On the other hand, fortunately, we do not need to "patch" the two building blocks themselves. Instead, we find that, even using these building blocks without guarantees for a full-omission faulty sender (as we will formally define in Definition 1 and Definition 2, respectively), a slightly enhanced version of our weak BA can still achieve all the properties even in the overlapping setting. The enhancements are as follows.

Where the "non-overlapping" weak BA fails in the overlapping setting. Recall that in the weak BA construction of [31], an honest party will output v only when it receives $t+1$ certificates for v with a grade of 2, while no certificate of $1-v$ is received with a grade of ≥ 1. Our non-overlapping weak BA (sketched above) follows this design and uses $t+1$ certificates of v as a threshold to decide whether an honest party should output v. The intuition is that among the $t+1$ certificates with a grade of 2, at least one must be from an honest sender, ensuring that all other honest parties, except zombies, can eventually receive this certificate (a guarantee provided by undead weak graded multicast and the two extra rounds of multicasts in our design).

However, in the overlapping setting, these $t+1$ certificates with a grade of 2 may all be sent by byzantine parties and full-omission faulty parties, while our undead weak graded multicast provides no guarantee for full-omission faulty senders. In this case, an honest party that is send faulty but has not become a ghost may output v, while other parties, failing to receive any certificate of v, may end up outputting $1-v$.

A natural attempt: increasing the threshold. A natural idea is to use a higher threshold to ensure the existence of a non-full-omission faulty honest party who sent the certificate of v. Specifically, instead of outputting v when receiving $t+1$ certificates of v with a grade of 2, an honest party may output only when it has received $t+r+1$ certificates (of v with a grade of 2), among which, at most t are from byzantine parties, and at most r from full-omission faulty parties, ensuring that at least one certificate is from a non-full-omission faulty honest sender.

Increasing threshold harms the validity. Nonetheless, simply increasing the threshold causes a validity issue. Specifically, when all honest parties have the same input v, all alive parties should

output v at the end of the protocol. However, this may be impossible with the higher threshold. In particular, the messages from send faulty parties may be entirely blocked, byzantine parties may remain silent, and receive faulty parties may fail to generate certificates (which require $t+1$ signed messages from different parties) to begin with. Therefore, we can only ensure $t+1$ certificates from non-faulty parties to be delivered properly, which is unable to reach the increased threshold.

Our final solution: adding one round for zombie detection. We resolve the dilemma by leveraging the power of receive faulty parties. Specifically, after the first round of multicasting input values, we introduce an additional round where all honest parties multicast the certificates they generated in the first round. This mechanism ensures that when all honest parties input the same v, a receive faulty party will either (1) receive the certificate of v from the second round, or (2) fail to receive enough messages and become a zombie.

Notably, a zombie will notify all other parties of its status. Then, following the undead weak graded multicast phase, we refine the condition for an honest party to output v as follows: (1) it receives certificates of v with a grade of 2 from k_1 different senders, and it receives k_2 "zombie" messages from different senders, such that $k_1+k_2 \geq t+r+1$, and (2) no certificate of $1-v$ is received. These conditions guarantee that among the k_1 senders, at least one is non-full-omission-faulty and honest.

With this enhancement, we achieve a quadratic-communication binary weak BA protocol in the overlapping setting (see Sec. 5.3). Finally, we apply similar conversion from [30, 31] to obtain a standard binary BA protocol with quadratic communication in the same setting (see Sec. 5.4).

Extending undead binary consensus to undead multi-valued consensus. Now we turn to multi-valued BA in the mixed-fault model, where each party has an ℓ-bit input with $\ell > 1$. In the most straightforward approach, the network can agree on the ℓ-bit value bit by bit, requiring ℓ independent instances of undead binary consensus, which leads to a communication complexity of $O(n^2\ell\lambda)$. In contrast, standard multi-valued BA protocols can achieve much lower complexity, as low as $O(n\ell+n^2\lambda)$ [35]. Our goal is to design a multi-valued BA protocol for the mixed-fault model that matches this lower bound.

Why the standard extension technique fails. A natural idea is to apply the conventional extension technique from [35] to obtain efficient undead multi-valued consensus. However, this approach does not work directly. Recall that the technique in [35] can be viewed as combining two parts: consensus and data availability. For consensus, the network first agrees on the hash of the proposed value, which has $O(\lambda)$ bits, requiring $O(\lambda)$ instances of binary BA. After that, another binary BA is invoked to decide whether a party has an input consistent with the agreed hash. If BA returns 1, then at least one honest party must hold the value. Then the *propagate* function is invoked to ensure the "correct" value becomes available to all honest nodes. In particular, the propagate function is realized through erasure codes which typically require at least $t+1$ fragments to reconstruct the entire value. So the final communication complexity is $(\lambda+1)\mathsf{Cost}_{\mathsf{bin}}+O(\frac{n^2}{t+1}\ell+n^2\lambda)$, where $\textsc{Cost}_{\mathsf{bin}}$ denotes the cost of a binary BA instance.

However, in our mixed-fault setting with omissions, the party holding the correct value may be send faulty. In that case, the binary BA may still return 1, yet no other party can obtain the value through the *propagate* function, thereby violating agreement.

Fix for the non-overlapping setting and the gap towards the overlapping solution. The key to addressing the issue is to ensure that whenever a party inputs 1 to the binary BA, it is convinced that

some non-send-faulty honest party holds the correct value, so that all honest parties can eventually receive it through the subsequent *propagate* function.

In the non-overlapping setting, we propose the following approach. Before invoking the binary BA, each party *multicasts* either 1 (indicating it holds the correct value) or 0 (otherwise) to the network. Then, a party who receives fewer than t zeros and also holds the correct value will confidently input 1 to the binary BA. The reasoning is as follows: If the party is non-receive-faulty, it receives messages from all (at least $t+1$) non-faulty parties; hence, at least one non-faulty party must have sent 1, showing it has the value. Otherwise, if the party is receive faulty, then being non-send-faulty itself, it can ensure that the value is propagated during the subsequent *propagate* step.

However, in the overlapping setting, where a party may be both send faulty and receive faulty simultaneously, this reasoning no longer applies.

Solution via multi-round amplification. We then resolve this challenge via a carefully designed multi-round amplification. As in our undead binary consensus for the overlapping setting, we leverage the "voices" of zombie parties, but in a distinct manner. Specifically, after the $O(\lambda)$ instances of undead binary consensus for agreeing on the hash value, some parties may become zombies. Nevertheless, these parties may still hold the correct input value and can assist in propagation. We therefore allow them to send useful messages "blindly", i.e., directly multicast their hash values rather than 1 or 0, leaving it to the receivers to determine whether their input matches the agreed hash.

Following the non-overlapping solution, each party multicasts 1 if it has a value matching the agreed hash. However, even if a party receives fewer than t zeros in this round and holds the correct value, it still cannot be confident about the existence of a non-send-faulty honest party holding the value. This is because such a party may itself be receive faulty (and thus miss some 0 messages from non-faulty parties), and send faulty (so even if it has the value, data availability is not ensured) simultaneously.

To overcome this, we add an additional round: each party multicasts 1 if and only if it receives fewer than t zeros in the previous round. Then, in the subsequent round, if a party receives at least $t+r+1$ ones, we can guarantee that at least one non-receive-faulty honest party received fewer than t zeros in the previous round, thereby confirming data availability. Any such confident parties input 1 to the binary BA, while non-confident parties (who did not receive $t+r+1$ ones but did not receive more than t zeros either) input $\perp$. The other parties input 0. We slightly adapt our undead weak binary consensus and undead binary consensus definitions to allow $\perp$ inputs, while preserving exactly the same properties as those in [31] when all the inputs are binary.

From $O(\kappa)$ rounds to expected $O(1)$ rounds. We further observe that while running $O(\lambda)$ parallel instances of undead binary consensus is a natural and straightforward approach to agreeing on a hash value, such a construction suffers from two drawbacks: (1) the round complexity is high, as we need to wait for all the undead binary consensus instances to terminate before proceeding to the next step, and (2) within the same undead binary consensus instance, different parties may terminate in different rounds. Thus, it is hard to ensure that all the parties move to the next step at the same time. Executing the subsequent algorithm in an unsynchronized manner, however, may cause inconsistencies. One solution is to stop every undead binary consensus instance after exactly κ iterations; however, this will lead to a round complexity of $O(\kappa)$, and there is a (negligible) probability that parties cannot output within κ rounds.

To address these issues, we replace the $O(\lambda)$ parallel undead binary consensus instances with $O(\lambda)$ parallel undead *weak* binary consensus instances. Since our undead weak binary consensus primitive is deterministic, this gives us $O(1)$ round complexity immediately, but we lose the guar-

antee that all the honest parties that are not zombies will have the same hash value. Instead, some parties may have $\perp$ (if they output $\perp$ in any of the undead weak binary consensus instances). This can lead to complications, e.g., even if a non-send-faulty honest party has the hash value and the corresponding value, and the binary BA outputs 1, the non-send-faulty honest party may still fail to *propagate* the value because the parties that do not have the hash value cannot tell whether the received value or fragments correspond to the correct hash value. We observe that this only happens when not all the honest parties have the same inputs, because otherwise, by the *validity* property of undead weak binary consensus, all parties except the zombies will still have the same hash value.

Now, for the parties that do not have hash values, we apply different strategies for the consensus and data availability parts. For the consensus part, we let these parties pretend not to have the corresponding value. And for the data availability part, we add three rounds. The first two rounds are used to generate the certificate for the hash value, and multicast the hash together with its certificate. The last round is added after the binary BA. We slightly modify the input conditions of the binary BA so that if the binary BA outputs 1, at least one non-send-faulty honest party has the hash and the certificate for the hash. The party can then multicast the hash and the certificate, so that all the non-receive-faulty honest parties will obtain the identical hash, which makes the *propagate* function possible again.

We then rearrange the primitives so that the values are propagated before the binary BA is executed. This avoids the second issue mentioned above. Even if different parties finish the binary BA in different rounds, they can decide whether to output the value immediately without waiting for other parties to finish the binary BA, because the values have been propagated in advance. See Sec. 5.5 for more details.

4 Preliminaries

Below, we provide an overview of the tools used in our construction. We use a public key infrastructure setup.

Digital Signature Scheme. We use a conventional digital signature scheme, denoted as $\{\mathsf{DS.KGen}, \mathsf{DS.Sign}, \mathsf{DS.Verify}\}$. It satisfies the unforgeability against chosen-message attacks.

Threshold Signature Scheme. We use a (k, n)-threshold signature scheme (TSS) [6, 12] that consists of the following algorithms:

- $\mathsf{TSS.Setup}(\lambda, n, k)$. Given the security parameter λ, number of parties n, and threshold k, this algorithm generates a group public key pk, public key shares $(pk_j)_{j\in[n]}$, and secret key shares $(sk_j)_{j\in[n]}$. Each party P_i obtains $(pk, pk_1, \ldots, pk_n, sk_i)$.
- $\mathsf{TSS.PartSign}(sk_i, \mathsf{msg}) \to \sigma_i$. Given a party's secret key share sk_i and the message msg, this algorithm generates a partial signature σ_i.
- $\mathsf{TSS.PartVrfy}(pk_i, \mathsf{msg}, \sigma_i) \to 0/1$. This algorithm verifies whether σ_i is a valid partial signature for message msg with respect to pk_i.
- $\mathsf{TSS.Agg}(\mathsf{msg}, \{(j, \sigma_j)\}_{j\in S}) \to \sigma$. This algorithm combines a set of partial signatures $\{(j, \sigma_j)\}_{j\in S}$ for message msg where $S \subset [n]$ and $|S| = k$ into one signature σ.
- $\mathsf{TSS.Vrfy}(pk, \mathsf{msg}, \sigma) \to 0/1$. This algorithm verifies whether a signature σ is valid for msg under the group public key pk.

Correctness. An (k, n)-TSS is correct if the following two conditions are satisfied:

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

$$\Pr[\mathsf{TSS.PartVrfy}(pk_i, \mathsf{msg}, \mathsf{TSS.PartSign}(sk_i, \mathsf{msg})) = 1] = 1.$$

- Combining any k valid partial signatures for the same message yields a valid threshold signature for that message. Formally, for any set $\{(j, \sigma_j)\}_{j \in S}$ such that $\mathsf{TSS.PartVrfy}(pk_j, \mathsf{msg}, \sigma_j) = 1$ for every $j \in S$ and $|S| = k$,

$$\mathsf{TSS.Vrfy}(pk, \mathsf{msg}, \mathsf{TSS.Agg}(\mathsf{msg}, \{(j, \sigma_j)\}_{j \in S})) = 1.$$

The unforgeability of TSS can be captured by the following security game. Consider a computationally bounded adversary $\mathcal{A}$ that is allowed to corrupt up to $k-1$ parties. When an honest party P_i is corrupted, the adversary is given access to the corresponding secret key share sk_i. Additionally, $\mathcal{A}$ can interact with a signing oracle. For any query (i, msg), the oracle returns a partial signature σ_i for the message msg generated using the secret key share sk_i of party P_i. The goal of $\mathcal{A}$ is to produce a valid threshold signature σ^* for a message msg^*, where the following condition must hold: $|\mathcal{C}| + |\mathcal{Q}_{\mathsf{msg}^*}| < k$, where $\mathcal{C}$ denotes the set of corrupted parties and $\mathcal{Q}_{\mathsf{msg}^*}$ is the set of partial signatures for msg^* obtained via oracle queries. The threshold signature scheme TSS is said to satisfy unforgeability if, for any computationally bounded adversary $\mathcal{A}$, the probability that $\mathcal{A}$ outputs such a valid signature-message pair $(\sigma^*, \mathsf{msg}^*)$ is negligible. In this paper, we assume threshold signature schemes to be correct and unforgeable. Furthermore, for constructing an undead common coin (introduced later in this section), TSS should satisfy the *uniqueness*, which captures that for any message msg, there only exists one valid threshold signature for it under a group public key pk.

We use $k = t + 1$ throughout the paper, where t is the number of byzantine parties.

Erasure Code. We use a (k, n)-erasure code [33] that consists of the following algorithms:

1. $\mathsf{EC.Encode}(\mathbf{m}, n, k) \rightarrow \mathbf{c}$: Given a value $\mathbf{m}$ as input, this algorithm outputs a vector $\mathbf{c} := \{c_1, \cdots, c_n\}$.
2. $\mathsf{EC.Decode}(\{(i, c_i)\}_{i \in S}, n, k) \rightarrow \mathbf{m}$: Given a set $\{(i, c_i)\}_{i \in S}$ where $S \subset [n]$ and $|S| = k$ as input, this algorithm outputs $\mathbf{m}$.

Correctness. EC is correct if for any $\mathbf{m}$ and any $\mathbb{I} \subset [n]$ with $|\mathbb{I}| = k$,

$$\Pr[\mathsf{EC.Decode}(\{(i, \mathsf{c}_i)\}_{i \in \mathbb{I}}) = \mathbf{m} \mid \mathbf{c} := (\mathsf{c}_1, \cdots, \mathsf{c}_n) \leftarrow \mathsf{EC.Encode}(\mathbf{m}, n, k)] = 1.$$

We use $k = t + 1$ throughout the paper, where t is the maximal number of byzantine parties.

Vector Commitment Scheme. We use a VC scheme [10] that consists of the following algorithms:

1. $\mathsf{VC.Setup}(\lambda, n, \mathcal{M}) \rightarrow pp$: Given the security parameter λ, the size n of the input vector, and the message space $\mathcal{M}$ for each vector element, this algorithm outputs the public parameter pp, which serves as an implicit input to all other algorithms.
2. $\mathsf{VCom}(\mathbf{m}) \rightarrow \mathsf{vc}$: Taking a vector $\mathbf{m} = (m_1, \ldots, m_n)$ as input, this algorithm produces a commitment vc.
3. $\mathsf{Open}(\mathsf{vc}, m_i, i) \rightarrow \pi_i$: Given m_i, position i, and the commitment vc, this algorithm generates an opening string π_i as proof that m_i is the i-th committed element.
4. $\mathsf{VerifyOpen}(\mathsf{vc}, m_i, i, \pi_i) \rightarrow 0/1$: Given m_i, position i, commitment vc, and opening proof π_i, this algorithm outputs 1 if $\mathsf{Open}(\mathsf{vc}, m_i, i) = \pi_i$, otherwise, it outputs 0.

Correctness. VC is correct if for all $\mathbf{m} = \{m_1, \ldots, m_n\} \in \mathcal{M}^n$ and $i \in [n]$,

$$\Pr[\mathsf{VerifyOpen}(\mathsf{vc}, m_i, i, \mathsf{Open}(\mathsf{vc}, m_i, i)) = 1 \mid \mathsf{vc} \leftarrow \mathsf{VCom}(\mathbf{m})] = 1.$$

Position binding. VC is position binding if for any PPT adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c}(\mathsf{vc}, i, m, \pi, m', \pi') \leftarrow \mathcal{A}(\lambda) : \mathsf{VerifyOpen}(\mathsf{vc}, m, i, \pi) = 1 \\ \wedge \ m \neq m' \ \wedge \ \mathsf{VerifyOpen}(\mathsf{vc}, m', i, \pi') = 1\end{array}\right] \leq \mathsf{negl}.$$

Undead Common Coin [31]. Upon $t + 1$ non-faulty parties participating in the protocol, all non-receive-faulty parties can receive the same random binary value, which is uniformly distributed and unpredictable to the adversary. In addition, any party that fails to receive the binary value can detect itself as being receive faulty. Informally, the undead common coin protocol has the following properties:

- For any given iteration, the output remains uniform to the adversary until at least one honest party invokes the common coin protocol.
- At the end of the protocol, all parties output the same value, except for those who are zombies.

Similar to Loss and Stern [31], we use a unique threshold signature-based common coin, such as $(t + 1, n)$-threshold BLS [6, 12], to instantiate such a coin. For each coin, all the parties sign the iteration number and multicast the partial signatures. In our setting, non-receive-faulty parties should receive at least $t+1$ "valid" partial signatures from non-faulty parties. Thus, they can generate a unique threshold signature, and use its hash value as the coin. We note that the remaining parties that receive less than $t + 1$ unique "valid" signatures should become zombies and terminate at the end of the iteration. This simple construction uses only 1 round and incurs $O(n^2\lambda)$ bits of communication.

5 Constructions and Analysis

In this section, we present our undead very weak multicast, undead weak graded multicast, undead weak binary consensus, undead binary consensus, and undead multi-valued consensus protocols, along with their security proofs and complexity analyses. The protocols are resilient to t byzantine parties, r receive faulty parties, and s send faulty parties if $n > 2t+r+s$, and there may be overlaps between send faulty and receive faulty parties (i.e., full-omission faulty parties).

Our construction is inspired by the works of [18, 30, 31] but with key modifications. In essence, we relax the omission detection and use the receive-faulty-only parties to help propagate messages instead of solely relying on non-faulty parties. To deal with full-omission faults, which are hard to detect, we raise the threshold to include them in the receive-omission faults.

In our description, when we say that a sender *multicasts* a value, we mean the sender sends the value to all parties (including the sender itself) via the authenticated channels. Additionally, we assume that the flags Z and G, which indicate whether the party is a zombie or a ghost respectively, are always included in the input and output of a protocol.

Undead Very Weak Multicast, sender P_s with input v, every party P_i with inputs z_i, g_i

1. Party P_s multicasts $(v, \mathsf{DS.Sign}(sk_s, v))$.
2. Let (v_i, σ_i) be the pair P_i receives from P_s. If $\mathsf{DS.Verify}(pk_s, v_i, \sigma_i) = 0$ or P_i receives nothing, $(v_i, \sigma_i) = (\perp, \perp)$. If $v_i \neq \perp$, P_i sends 1 to P_s; otherwise, it sends 0 to P_s.
3. If $P_i \neq P_s$, P_i outputs (v_i, z_i, g_i). P_s sets $Z_s = z_s$, $G_s = g_s$. If P_s receives fewer than $n - t - s$ messages from distinct parties, $Z_s = 1$. If P_s receives at least $t + r + 1$ 0 messages from distinct parties, $G_s = 1$. P_s outputs (v, Z_s, G_s).

Fig. 3. Our undead very weak multicast protocol

5.1 Undead Very Weak Multicast

The most basic primitive in this work is the undead very weak multicast protocol. It is weaker than the undead weak multicast protocol [30, 31], mainly because its detection property only guarantees that if the sender is *not full-omission-faulty* and is not a ghost at the end of the protocol, then its value is delivered to at least one *non-send-faulty honest* party.

Definition 1 (Undead Very Weak Multicast). *Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where a designated sender P_s holds a value $v \neq \perp$, and every party P_i has two boolean values z_i and g_i. Every party P_i terminates upon outputting a triplet (x_i, z_i, g_i) such that x_i is either a value or $\perp$. We say that Π is an undead very weak multicast protocol if it has the following properties:*

- ***Validity.*** *If P_s is a non-send-faulty honest party, then every non-receive-faulty honest party P_i outputs (x_i, z_i, g_i) such that $x_i = v$. Additionally, if P_s is honest, every honest party outputs (x_i, z_i, g_i) such that $x_i = v$ or $\perp$.*
- ***Detection.*** *If P_s is honest and not full-omission faulty, and is not a ghost at the end of the protocol, i.e., $g_s = 0$, then at least one non-send-faulty honest party outputs (x_i, z_i, g_i) such that $x_i = v$.*
- ***Termination.*** *All honest parties eventually output and terminate.*
- ***Detection Soundness.*** *If P_i is honest and it outputs (x_i, z_i, g_i) such that $z_i = 1$ (resp. $g_i = 1$), then it is receive faulty (resp. send faulty).*

The full description of the protocol is provided in Fig. 3. In the protocol, P_s first multicasts its value to all parties. Each party then echoes 1 if it receives the value, or 0 if not. A non-receive-faulty sender should receive echoes from all non-send-faulty honest parties. Thus, if P_s receives fewer than $n-t-s$ messages, it is receive faulty and thus becomes a zombie. Moreover, values from a non-send-faulty sender should be received by all non-receive-faulty honest parties, thus only receive faulty and byzantine parties may echo 0 to the sender unless the sender is send faulty. Therefore, if P_s receives more than $t + r$ 0 messages, it is send faulty and thus becomes a ghost.

We show that the undead very weak multicast protocol satisfies the defined properties. It is easy to see that the protocol is deterministic and terminates after 2 rounds, ensuring termination.

Lemma 1. *The protocol in Fig. 3 guarantees validity.*

Proof. If the sender P_s is a non-send-faulty honest party, and it multicasts the pair (v, σ) to all parties, every non-receive-faulty honest party receives the pair and thus outputs v.

Additionally, if P_s is honest, the honest parties that receive the value output v, and the rest output $\perp$. Since P_s always outputs v, every honest party outputs either $x = v$ or $\perp$. □

Lemma 2. *The protocol in Fig. 3 guarantees detection.*

Proof. Let P_s be a non-full-omission-faulty honest party that is not a ghost at the end of the protocol. We can deduce that P_s receives fewer than $t + r + 1$ 0 messages in step 3. If P_s is not send faulty, by the validity property, all non-receive-faulty honest parties should output $x = v$. If P_s is send faulty and thus not receive faulty, we first assume that no non-send-faulty honest parties output v. These parties send 0 to P_s in step 2. As a result, P_s receives more than $t+r$ 0 and becomes a ghost. By contradiction, at least one non-send-faulty honest party outputs v. □

Lemma 3. *The protocol in Fig. 3 guarantees detection soundness.*

Proof. A non-receive-faulty honest sender always receives messages from all non-send-faulty honest parties. Thus, it always receives at least $n-t-s$ messages in step 3. Conversely, only receive faulty senders may receive fewer than $n-t-s$ messages and become zombies.

A non-send-faulty honest sender always delivers messages to all non-receive-faulty honest parties. Thus, it only receives 0 from receive faulty and byzantine parties, i.e., no more than $t + r$ parties. Conversely, only send faulty senders may receive more than $t+r$ 0 messages and become ghosts. □

5.2 Undead Weak Graded Multicast

The undead weak graded multicast protocol is a much weaker version of the undead graded multicast protocol [30, 31]. It allows parties to output a value with a grade $y \in \{0, 1, 2\}$, which reflects the confidence level that the value has been delivered to a trustworthy party. The protocol is weak in the sense that it does not allow parties to self-detect whether they are receive faulty, and if an honest party outputs a value with a grade of 2, it is only guaranteed that one *non-send-faulty honest* party has outputted the value with a grade of at least 1.

Definition 2 (Undead Weak Graded Multicast). *Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where a designated sender P_s holds a value $v \neq \perp$, and every party P_i has two boolean values z_i and g_i. Every party P_i terminates upon outputting a tuple (x_i, y_i, z_i, g_i) such that x_i is either a value or $\perp$, $y_i \in \{0, 1, 2\}$, and z_i, g_i are boolean values. We say that Π is an undead weak graded multicast protocol if it has the following properties:*

- ***Weak Validity.*** *If P_s is a non-send-faulty honest party, then every non-receive-faulty honest party P_i outputs (x_i, y_i, z_i, g_i) such that $x_i = v$ and $y_i = 2$. Additionally, if P_s is honest, every honest party outputs (x_i, y_i, z_i, g_i) such that $x_i = v$ or $\perp$.*
- ***Weak Consistency.*** *If P_s is honest and not full-omission faulty, and an honest party P_i outputs (x_i, y_i, z_i, g_i) such that $y_i = 2$, then at least one non-send-faulty honest party P_j outputs (x_j, y_j, z_j, g_j) such that $x_j = x_i$ and $y_j \geq 1$.*
- ***Termination.*** *All honest parties eventually output and terminate.*
- ***Detection Soundness.*** *If P_i is honest and it outputs (x_i, y_i, z_i, g_i) such that $z_i = 1$ (resp. $g_i = 1$), then it is receive faulty (resp. send faulty).*

The full description of the protocol is provided in Fig. 4. In the protocol, the sender P_s first multicasts the value v using undead very weak multicast. If P_s becomes a ghost, it outputs v and halts after the next round. Otherwise, by the detection property of undead very weak multicast, at least one non-send-faulty honest party receives v, and P_s multicasts v. A party P_i outputs v with

Undead Weak Graded Multicast, sender P_s with input v, every party P_i with inputs z_i, g_i

1. Party P_s sends v to all parties using undead very weak multicast.
2. Let (x_i, z_i', g_i') be P_i's output from undead very weak multicast. If $g_s' = 1$, P_s outputs $(v, 1, z_s', g_s')$; otherwise, P_s multicasts $(v, \mathsf{DS.Sign}(sk_s, v))$.
3. Let (v_i, σ_i) be the pair P_i receives from P_s. If $\mathsf{DS.Verify}(pk_s, v_i, \sigma_i) = 0$ or P_i receives nothing, $(v_i, \sigma_i) = (\perp, \perp)$. A party only outputs once. If $v_i \neq \perp$, P_i outputs $(v_i, 2, z_i', g_i')$. If $v_i = \perp$ but $x_i \neq \perp$, P_i outputs $(x_i, 1, z_i', g_i')$. Otherwise, P_i outputs $(\perp, 0, z_i', g_i')$.

Fig. 4. Our undead weak graded multicast protocol

a grade of 2 if it receives v from P_s's multicast. Alternatively, P_i outputs v with a grade of 1 if it receives v from P_s's undead very weak multicast. Otherwise, P_i outputs $\perp$ with a grade of 0.

We show that the undead weak graded multicast protocol satisfies the defined properties. It is easy to see that the protocol is deterministic and terminates after a constant number of rounds, ensuring termination. Also, parties become zombies (resp. ghosts) if and only if they become zombies (resp. ghosts) in undead very weak multicast. Since undead very weak multicast satisfies detection soundness, this protocol satisfies detection soundness, too.

Lemma 4. *The protocol in Fig. 4 guarantees weak validity.*

Proof. If P_s is honest and non-send-faulty, by the validity property of undead very weak multicast, every non-receive-faulty honest party P_i has $x_i = v$. Since P_s is non-send-faulty, by the detection soundness property of undead very weak multicast, P_s will not become a ghost. Then P_s multicasts (v, σ). Every non-receive-faulty honest party should receive v and thus output $(v, 2, z_i, g_i)$. Additionally, if P_s is honest, by the validity property of undead very weak multicast, every honest party P_i has $x_i = v$ or $\perp$. Then P_s multicasts v depending on whether it becomes a ghost. Either way, since P_s is honest, it only multicasts the same v, so that every honest party outputs $x = v$ or $\perp$. □

Lemma 5. *The protocol in Fig. 4 guarantees weak consistency.*

Proof. If an honest party P_i outputs v with a grade of 2, then P_i receives v from P_s's multicast, meaning that P_s is not a ghost after undead very weak multicast. Therefore, by the detection property of undead very weak multicast, at least one non-send-faulty honest party P_j outputs v in undead very weak multicast. Thus, P_j will output v either with a grade of 2 (if it later receives v from P_s's multicast), or with a grade of 1. □

5.3 Undead Weak Binary Consensus

We now use our undead weak graded multicast primitive to construct the undead weak binary consensus protocol. The undead weak binary consensus protocol is weak in the sense that the honest parties that are not zombies at the end of protocol output either the same value v *or* $\perp$. Our protocol extends the undead weak consensus protocol in [31] by allowing $\perp$ as inputs, but has the same properties when all the inputs are binary.

Definition 3 (Undead Weak Binary Consensus). *Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where every party P_i holds a triplet (v_i, z_i, g_i) such that $v_i \in \{0, 1, \perp\}$, and z_i, g_i are boolean values. Every party P_i terminates upon outputting a triplet (x_i, z_i, g_i) such that $x_i \in \{0, 1, \perp\}$, and z_i, g_i are boolean values. We say that Π is an undead weak binary consensus protocol if it has the following properties:*

Undead Weak Binary Consensus, every party P_i with inputs $v_i \in \{0,1,\perp\}, z_i, g_i$

If at any point throughout the execution P_i becomes a zombie, it multicasts $\langle Zombie \rangle$.

1. Initially, each party P_i sets $Z_i = z_i, G_i = g_i, V_i = \perp, C_i = \perp, C_{0,i} = \perp, C_{1,i} = \perp$. If $v_i \neq \perp$, P_i multicasts $(v_i, \mathsf{TSS.PartSign}(sk_i, v_i))$.
2. If there exists $v' \in \{0,1\}$, and P_i receives at least $t+1$ (v'_j, σ'_j) pairs from distinct P_j such that $v'_j = v'$ and $\mathsf{TSS.PartVrfy}(pk_j, v'_j, \sigma'_j) = 1$, P_i sets $V_i = v'$ and $C_i = \mathsf{TSS.Agg}(v', \{(j, \sigma'_j)\}_{j \in S})$, where $S \subseteq [n]$ and $|S| = t+1$. If there are multiple (V_i, C_i) pairs, P_i chooses one arbitrarily. P_i multicasts (V_i, C_i).
3. If P_i receives fewer than $n-t-s$ (v'_j, c'_j) pairs from distinct parties, $Z_i = 1$. If $V_i = \perp$, upon receiving a (v'_j, c'_j) pair from P_j, and $\mathsf{TSS.Vrfy}(pk, v'_j, c'_j) = 1$, $(V_i, C_i) = (v'_j, c'_j)$. If $V_i \neq \perp$, P_i sends (V_i, C_i) to all parties using undead weak graded multicast.
4. Let $(v_{i,j}, c_{i,j}, y_{i,j}, z_{i,j}, g_{i,j})$ be P_i's output from undead weak graded multicast with the sender being P_j. If any $z_{i,j} = 1$, $Z_i = 1$. If any $g_{i,j} = 1$, $G_i = 1$. For each $v' \in \{0,1\}$, if there exists a tuple $(v_{i,j}, c_{i,j}, y_{i,j}, z_{i,j}, g_{i,j})$ such that $v_{i,j} = v'$, $y_{i,j} \geq 1$ and $\mathsf{TSS.Vrfy}(pk, v_{i,j}, c_{i,j}) = 1$, then $C_{v',i} = c_{i,j}$. If multiple such tuples exist, P_i chooses one arbitrarily. P_i multicasts $(C_{0,i}, C_{1,i})$.
5. For each $v' \in \{0,1\}$, if $C_{v',i} = \perp$, then upon receiving $(c_{0,j}, c_{1,j})$ from P_j and $\mathsf{TSS.Vrfy}(pk, v', c_{v',j}) = 1$, $C_{v',i} = c_{v',j}$. P_i multicasts $(C_{0,i}, C_{1,i})$.
6. If P_i receives fewer than $n-t-s$ pairs from distinct parties, $Z_i = 1$. If $Z_i = 1$, P_i outputs $(\perp, Z_i, G_i)$. For each $v' \in \{0,1\}$, if $C_{v',i} = \perp$, then upon receiving a $(v', c_{v',j})$ pair from P_j and $\mathsf{TSS.Vrfy}(pk, v', c_{v',j}) = 1$, $C_{v',i} = c_{v',j}$.
   P_i sets $V_i = \perp$. Let $\mathbb{T} = \{(v_{i,j}, c_{i,j}, y_{i,j}, z_{i,j}, g_{i,j})\}$ be the set of all tuples P_i outputs in undead weak graded multicast. P_i updates $V_i = v'$, if there exists $v' \in \{0,1\}$ such that the following conditions are met:
   (a) Let $E1$ be the set of parties from which P_i receives $\langle Zombie \rangle$. Let $E2$ be the set of parties P_j such that $P_j \notin E1$, and in $\mathbb{T}$, $v_{i,j} = v'$, $\mathsf{TSS.Vrfy}(pk, v_{i,j}, c_{i,j}) = 1$, and $y_{i,j} = 2$. Then $|E1| + |E2| \geq t + r + 1$.
   (b) $C_{1-v',i} = \perp$.
   P_i multicasts $(V_i, \mathsf{TSS.PartSign}(sk_i, V_i))$.
7. P_i sets $V'_i = \perp, C'_i = \perp$. If there exists $v'' \in \{0,1\}$, and P_i receives at least $t+1$ (v_j, σ_j) pairs from distinct P_j such that $v_j = v''$ and $\mathsf{TSS.PartVrfy}(pk_j, v_j, \sigma_j) = 1$, P_i sets $V'_i = v'', C'_i = \mathsf{TSS.Agg}(v'', \{(j, \sigma_j)\}_{j \in S})$, where $S \subseteq [n]$ and $|S| = t+1$. P_i multicasts (V'_i, C'_i).
8. Let $(v'_{i,j}, c'_{i,j})$ be the pair that P_i receives from P_j. If P_i receives fewer than $n-t-s$ pairs, $Z_i = 1$. Suppose there is a $v'' \in \{0,1\}$. P_i outputs (v'', Z_i, G_i) if P_i receives a $(v'_{i,j}, c'_{i,j})$ pair such that $v'_{i,j} = v''$ and $\mathsf{TSS.Vrfy}(pk, v'', c'_{i,j}) = 1$; otherwise, P_i outputs $(\perp, Z_i, G_i)$.

Fig. 5. Our undead weak binary consensus protocol

- ***Validity.*** *If all the non-faulty parties hold the same input $v \neq \perp$, and the other honest parties hold either v or $\perp$, then every honest party P_i either outputs (x_i, z_i, g_i) such that $x_i = v$, or becomes a zombie by the end of the protocol, i.e., $z_i = 1$.*
- ***Weak Consistency.*** *If two honest parties that are not zombies output (x_i, z_i, g_i) and (x_j, z_j, g_j), and $x_i, x_j \neq \perp$, then $x_i = x_j$.*
- ***Termination.*** *All honest parties eventually output and terminate.*
- ***Detection Soundness.*** *If P_i is honest and it outputs (x_i, z_i, g_i) such that $z_i = 1$ (resp. $g_i = 1$), then it is receive faulty (resp. send faulty).*

The full description of the protocol is provided in Fig. 5. In the protocol, each party P_i first multicasts the value along with its partial signature σ to all parties. Upon receiving $t+1$ partial signatures for the same v, P_i generates a certificate C for v, and multicasts (v, C). The receivers update (v, C) if they did not have one. Then the parties multicast (v, C) using undead weak graded multicast.

The parties store the received certificates for 0 and 1 in C_0 and C_1, which are set to $\perp$ initially. When P_i receives a (v, C) pair with a grade of at least 1 from undead weak graded multicast, it sets $C_v = C$. Then P_i multicasts (C_0, C_1). For each $v' \in \{0,1\}$, P_i updates $C_{v'}$ if $C_{v'} = \perp$. Note that if

(v, C_v) is outputted by some honest parties with a grade of 2, by the weak consistency property of undead weak graded multicast, at least one non-send-faulty honest party receives the (v, C_v) pair with a grade of at least 1, thus every non-receive-faulty honest party has a certificate for v after this multicast. Then P_i multicasts (C_0, C_1) again (in step 5). Parties that receive fewer than $n - t - s$ pairs become zombies. Otherwise, it receives (C_0, C_1) from at least one non-receive-faulty honest party. As a result, every party except the zombies receives a certificate for 0 (resp. 1) unless there are no certificates generated for 0 (resp. 1).

Throughout the execution, when a party becomes a zombie, it multicasts $\langle Zombie \rangle$. Consider a value $v' \in \{0, 1\}$. In step 6, we let $E1$ be the set of parties that P_i receives $\langle Zombie \rangle$ from, and let $E2$ be the set of parties P_j such that P_j is not in $E1$, and P_i outputs (v', C'_v) with a grade of 2 in undead weak graded multicast with the sender being P_j. If $|E1| + |E2| \geq t + r + 1$, and P_i does not receive any certificates for $1 - v'$, i.e., $C_{1-v'} = \perp$, P_i sets $V = v'$. Intuitively, since $|E1| + |E2| \geq t + r + 1$, at least one (v', C'_v) with a grade of 2 is outputted when the sender is a non-receive-faulty honest party. Then by the weak consistency property of undead weak graded multicast, at least one non-send-faulty honest party outputs (v', C'_v) with a grade of at least 1. The party multicasts C'_v. Thus, every party except the zombies has a certificate for v', and no honest parties will set $V = 1 - v'$, ensuring weak consistency. Then each party P_i multicasts V and its partial signature.

In step 7, upon receiving $t+1$ partial signatures for V, P_i generates a certificate C' for V. Since we have shown that all honest parties have $V = v'$ or $\perp$, all C' are for v'. Then P_i multicasts (v', C'). In step 8, if a party P_i receives fewer than $n - t - s$ pairs, it becomes a zombie. If it receives a valid (v', C') pair, it outputs v', otherwise $\perp$.

We show that the undead weak binary consensus protocol satisfies the defined properties. It is easy to see that the protocol is deterministic and terminates after a constant number of rounds. Moreover, parties become zombies (resp. ghosts) if they become zombies (resp. ghosts) in undead weak graded multicast. Besides that, parties only become zombies if they receive fewer than $n-t-s$ messages. Since undead weak graded multicast satisfies detection soundness, this protocol satisfies detection soundness, too.

Lemma 6. *The protocol in Fig. 5 guarantees validity.*

Proof. If all the non-faulty parties hold the same input $v \neq \perp$ and multicast (v, σ_i), all the non-receive-faulty honest parties receive at least $t+1$ (v, σ) pairs from non-faulty parties, so they generate certificates C for v. Since no honest parties hold the input $1 - v$ and there are up to t byzantine parties, certificates for $1 - v$ will not be generated. Then the parties multicast (v, C) in step 2. A party either receives $n - t - s$ (v, C) pairs in which at least one is from a non-receive-faulty honest party and thus get (v, C), or becomes a zombie. Then the parties send (v, C) to all parties using undead weak graded multicast in step 3. By the weak validity property of undead weak graded multicast, if the sender is honest, non-send-faulty and is not a zombie, every non-receive-faulty honest party outputs $(v_{i,j}, c_{i,j}, y_{i,j}, z_{i,j}, g_{i,j})$ such that $v_{i,j} = v$, $c_{i,j}$ is for v, and $y_{i,j} = 2$. As a result, non-send-faulty honest parties will be either in $E1$ or $E2$ in step 6. For non-receive-faulty honest parties, $|E1| + |E2| \geq t + r + 1$, thus $V = v$. The receive faulty parties, on the other hand, will have $V = v$ or $\perp$ or become zombies, because there is no certificate for $1 - v$. Then the parties multicast (V, σ) in step 6. All the non-receive-faulty honest parties receive $t + 1$ valid $(V = v, \sigma)$ pairs from non-faulty parties, so that they have $V' = v$ and generate C' for v. Then all the parties multicast V' and C' in step 7. Since a party that receives fewer than $n - t - s \geq t + r + 1$ messages becomes

a zombie, an honest party that does not become a zombie receives at least one message from a non-receive-faulty honest party. Thus, an honest party either receives C' for v and thus outputs v, or becomes a zombie. □

Lemma 7. *The protocol in Fig. 5 guarantees weak consistency.*

Proof. Assume an honest party P_i outputs $v \neq \perp$, which means that P_i is not a zombie and P_i receives (v, c'). The certificate implies that at least one honest party P_j multicasts (v, σ) at the end of step 6. Notice that P_j multicasts (v, σ) only if it has $|E1|+|E2| \geq t+r+1$. Since there are up to t byzantine parties, and by the detection soundness property, up to r receive faulty parties may send $\langle Zombie \rangle$, we can deduce that among the $|E1|+|E2|$ parties, there exists at least one honest party P_k such that P_j outputs $(v, c_{j,k}, 2, z_{j,k}, g_{j,k})$ in undead weak graded multicast when P_k is the sender. By the weak consistency property of undead weak graded multicast, at least one non-send-faulty honest party P_r receives the (v', c', y') tuple such that $v' = v$, $c' = c_{j,k}$, and $y' \geq 1$. So for P_r, $C_{v,r} \neq \perp$. Then P_r multicasts $(v, C_{v,j})$ to all parties. Since P_r is not send faulty, every non-receive-faulty honest party P_t will receive the pair and have $C_{v,t} \neq \perp$. They multicast $(C_{v,t}, C_{1-v,t})$ to all parties. Notice that a party that receives fewer than $n-t-s$ pairs becomes a zombie, thus, if an honest party P_l does not become a zombie, it receives at least $n-t-s$ messages in which at least one is from a non-receive-faulty honest party. Therefore, P_l will have $C_{v,l} \neq \perp$, and P_l will never have $V_l = 1-v$. This means that every honest party that is not a zombie has $V = v$ or $\perp$. Since there are up to t byzantine parties, no C' for $1-v$ can be generated in step 7. Thus, every honest party that is not a zombie outputs v or $\perp$. □

5.4 Undead Binary Consensus

We first present the definition of undead binary consensus, which is a mixed-fault version of the classical BA. Our protocol extends the undead consensus protocol in [31] by allowing $\perp$ as inputs, but has the same properties when all the inputs are binary.

Definition 4 (Undead Binary Consensus). *Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where every party P_i holds a triplet (v_i, z_i, g_i) such that $v_i \in \{0, 1, \perp\}$, and z_i, g_i are boolean values. Every party P_i terminates upon outputting a triplet (x_i, z_i, g_i) such that $x_i \in \{0, 1, \perp\}$, and z_i, g_i are boolean values. We say that Π is an undead binary consensus protocol if it has the following properties:*

- ***Validity.*** *If all the non-faulty parties hold the same input $v \neq \perp$, and the other honest parties hold either v or $\perp$, then every honest party P_i either outputs (x_i, z_i, g_i) such that $x_i = v$, or becomes a zombie by the end of the protocol, i.e., $z_i = 1$.*
- ***Consistency.*** *If two honest parties that are not zombies at the end of the protocol output (x_i, z_i, g_i) and (x_j, z_j, g_j), then $x_i = x_j$.*
- ***Termination.*** *All honest parties eventually output and terminate.*
- ***Detection Soundness.*** *If P_i is honest and it outputs (x_i, z_i, g_i) such that $z_i = 1$ (resp. $g_i = 1$), then it is receive faulty (resp. send faulty).*

The full protocol is in Fig. 6, in which the messages are domain-separated by round numbers. At a high level, we convert our undead weak binary consensus protocol into a undead binary consensus protocol by using a coin, enabling us to achieve an expected round complexity of $O(1)$. This conversion largely follows the same approach as in [31], except: (1) we use threshold signatures to

Undead Binary Consensus, every party P_i with inputs $v_i \in \{0, 1, \perp\}, z_i, g_i$

1. Initially, each party P_i sets $Z_i = z_i, G_i = g_i, V_i = v_i, D_0 = D_1 = \emptyset, C_0 = C_1 = \perp$. Loop until the termination condition is met:
   i) P_i calls undead weak binary consensus with input (V_i, Z_i, G_i). Let (u_i, z_i', g_i') be P_i's output. If $z_i' = 1$, $Z_i = 1$. If $g_i' = 1$, $G_i = 1$.
   ii) P_i invokes the common coin protocol. Let (b, z_i'', g_i'') be P_i's output. If $z_i'' = 1$, $Z_i = 1$. If $g_i'' = 1$, $G_i = 1$. If $u_i = b$, P_i multicasts $(u_i, \mathsf{TSS.PartSign}(sk_i, u_i))$ to all parties. If $u_i \neq \perp$, $V_i = u_i$; otherwise, $V_i = b$.
   iii) If P_i receives (u, σ) from a party P_j such that $\mathsf{TSS.PartVrfy}(pk_j, u, \sigma) = 1$ and no pair $(j, \sigma') \in D_u$ exists yet, then P_i adds (j, σ) to D_u.
   iv) If there exists $v \in \{0, 1\}$ such that $C_v = \perp$ and $|D_v| \geq t + 1$, P_i generates $C_v = \mathsf{TSS.Agg}(v, S)$, where $S \subseteq D_v$ and $|S| = t + 1$. P_i multicasts (v, C_v) to all parties and outputs (v, Z_i, G_i). Then P_i continues for one more iteration before terminating, while the output remains unchanged.
2. If at any point in the protocol P_i receives a (v', C') pair such that $\mathsf{TSS.Vrfy}(pk, v', C') = 1$ and $C_{v'} = \perp$, P_i sets $C_{v'} = C'$, multicasts (v', C') to all parties, and outputs (v', Z_i, G_i). P_i then continues for one more iteration before terminating, while the output remains unchanged.
3. If at any point in the protocol $Z_i = 1$, P_i stops receiving messages but keeps sending messages until the end of the iteration. Then it multicasts $\langle Zombie \rangle$ to all parties, outputs $(\perp, Z_i, G_i)$ and terminates. If P_i receives $\langle Zombie \rangle$ from a party P_j, P_i behaves as if P_j forwards 0 in all subsequent undead very weak multicast and undead weak graded multicast protocols, and $\perp$ in any other protocols.
4. If at any point in the protocol $G_i = 1$, P_i stops sending messages but keeps receiving and processing messages.

Fig. 6. Our undead binary consensus protocol

optimize the communication complexity, and (2) when a receive faulty party detects itself as being receive faulty, it keeps sending messages until the end of the undead weak binary consensus protocol rather than terminating immediately. This is because our primitives rely on receive faulty parties to propagate messages. Thus, immediate termination may result in message loss.

In the protocol, the parties keep performing undead weak binary consensus until they terminate. In particular, each party P_i sets $V_i = v_i$ as their input for the first iteration. At the start of each iteration, P_i calls the undead weak binary consensus protocol with V_i, and gets the output u_i. Then P_i calls the common coin protocol and gets the output b. If $u_i \neq \perp$, $V_i = u_i$; otherwise $V_i = b$.

Note that by the weak consistency property of undead weak binary consensus, all parties except the zombies get the same value u or $\perp$. If $b = u$, all the parties except the zombies have $V = u$ and start the next iteration with the same input u. By the validity property of undead weak binary consensus, all parties except zombies will get u at the end of the next iteration.

Unfortunately, the parties may not be aware that they have reached consensus, so we implement an additional mechanism for them to complete the protocol. If $u_i = b$, P_i signs u_i and multicasts (u_i, σ) to all parties. Upon receiving (u_i, σ) from P_i, parties store (i, σ) in a set D_{u_i}. Once a party receives $t + 1$ such pairs on the same value v, i.e., $|D_v| \geq t + 1$, it generates a threshold signature C_v using partial signatures in D_v, multicasts C_v, and outputs v. It then continues for one more iteration to help the other parties terminate, but the output remains unchanged. Any parties that receive C_v output v and also wait for one more iteration before terminating.

Note that if all parties except zombies get u at the end of the undead weak binary consensus protocol and $u = b$, the parties send (u, σ) to all parties. A non-faulty party P_h will receive at least $t + 1$ such pairs from the non-faulty parties, so P_h outputs u and generates a threshold signature C_u for u. Then P_h multicasts C_u to all parties. All honest parties that receive C_u will output u and terminate at the end of the next iteration. Since P_h is non-faulty, every party that does not

receive C_u is receive faulty, and they will become zombies once the other parties terminate and stop sending messages to them, so that they can no longer receive at least $n - t - s$ messages.

We show that the undead binary consensus protocol satisfies the defined properties. Our proofs follow a similar approach to that in Loss-Stern [31].

Lemma 8. *The protocol in Fig. 6 guarantees validity.*

Proof. Assume all the non-faulty parties hold the same input v and the other honest parties hold either v or $\perp$. They call the undead weak binary consensus protocol with input v. By the validity property of undead weak binary consensus, all the honest parties either get $u = v$ or become zombies. Since $u \neq \perp$, the parties that are not zombies will start the next iteration with input v. This inductively proves that the honest parties will always have $v = u$ as their input for all following iterations and have $u = v$ as the output from undead weak binary consensus, unless they become zombies. Then the parties call the common coin protocol and get b. If $b \neq v$, none of these parties will sign b. Since there are up to t byzantine parties, D_b collects up to t pairs, which means no honest parties will ever generate C_b and output $b \neq v$. If $b = v$, these parties sign v and multicast (v, σ) to all parties. The non-receive-faulty honest parties receive at least $t + 1$ signatures from non-faulty parties and thus have $|D_v| \geq t + 1$. They generate C_v and multicast (v, C_v) to all parties, then they output v. All the non-receive-faulty parties receive the messages, so they output v and terminate at the end of the next iteration. Once they terminate, the remaining parties can only receive messages from byzantine and receive faulty parties that have not terminated. Since they can receive at most $t + r < n - t - s$ messages, they will become zombies. Thus, all the honest parties either output v or become zombies at the end of the protocol. □

Lemma 9. *The protocol in Fig. 6 guarantees consistency.*

Proof. Assume that an honest party P_i that is not a zombie at the end of the protocol outputs v. By the construction of the protocol, P_i must have obtained a valid threshold signature C_v for the value v.

If P_i is not send faulty, it multicasts (v, C_v) to all parties. Every party that is not receive faulty receives the pair. If it has not outputted yet, the party outputs v and terminates after the next iteration ends. The remaining parties will only be able to receive messages from up to $t + r$ parties, and thus become zombies.

The existence of C_v indicates that at least one honest party signed the value v, and according to the protocol, the honest party must have obtained the value v from the undead weak binary consensus protocol, and obtained $b = v$ as the output from the common coin protocol. By the weak consistency property of undead weak binary consensus, all honest parties have obtained either the same value v or $\perp$ from undead weak binary consensus. Thus, if a party outputs before P_i sends (v, C_v) to it, the party outputs the same v.

If P_i is send faulty, the parties that obtained v from undead weak binary consensus will use v as input for the next iteration. For the remaining honest parties, since the common coin protocol returns $b = v$, they will also use v as their input. Thus, all honest parties except the zombies will have the same input v for the next iteration. Then the consistency proof follows the validity proof above, even if P_i terminates in advance. This is because there is no difference between send faulty parties stopping sending messages because they have terminated, and send faulty parties failing to send messages because the messages have been omitted by the adversary. □

Lemma 10. *The protocol in Fig. 6 guarantees termination.*

Proof. If a non-faulty party P_i outputs v, it must have multicasted (v, C_v) to all parties. All honest parties that receive (v, C_v) will terminate after a finite number of rounds. Since P_i is non-faulty, the parties that do not receive the message are receive faulty. They will become zombies once the other parties terminate and stop sending messages to them, so that they can no longer receive at least $n - t - s$ messages. This means that proving at least one non-faulty party completes the protocol within a finite number of rounds is sufficient to prove the termination property of the protocol.

Assume that there is an iteration in which no non-faulty parties terminate yet, and by the weak consistency property of undead weak binary consensus, assume without loss of generality that all the honest parties that are not zombies at the end of undead weak binary consensus have $u = v$ or $\perp$. With a probability of $\frac{1}{2}$, all the parties have $b = v$ as the output from the common coin protocol. If so, all the honest parties that are not zombies will have v as their input for the next iteration. By the validity property of the protocol, all non-faulty parties will output v in the next iteration, which means that all the honest parties will eventually terminate.

The number of iterations required for b to equal v follows a geometric random variable with a success probability of $\frac{1}{2}$ in each iteration. Therefore, the expected number of iterations needed is bounded by 2. Additionally, the probability that more than k iterations are required is bounded by 2^{-k}. Then the protocol will need to run an additional iteration to aid termination, so the total expected number of iterations is 3. □

Lemma 11. *The protocol in Fig. 6 guarantees detection soundness.*

Proof. An honest party becomes a zombie when it receives fewer than $n - t - s$ messages. Unless the party is receive faulty, it receives messages from all non-send-faulty honest parties. Also, even if a party becomes a zombie, other parties will treat it as if it sends $\perp$. Thus, a non-receive-faulty honest party will never become a zombie unless a non-faulty party terminates and stops sending messages to it. However, as stated in the termination property of the protocol, if a non-faulty party terminates, all non-receive-faulty honest parties output the same value and terminate at the end of the next iteration. By contradiction, non-receive-faulty honest parties will never become zombies.

Apart from that, parties become ghosts after they become ghosts in the undead weak binary consensus protocol. Parties become zombies after they become zombies in the undead weak binary consensus protocol. Since the undead weak binary consensus protocol satisfies detection soundness, the undead binary consensus protocol satisfies detection soundness, too. □

Below, we analyze the round and communication complexity of our undead binary consensus protocol.

Theorem 1. *The expected round complexity of the undead binary consensus protocol described in Fig. 6 is* $O(1)$.

Proof. As proven in Lemma 10, the expected number of iterations needed in undead binary consensus is constant. In each iteration, the parties call undead weak binary consensus in substep (i), and call the common coin protocol in substep (ii). Since these two primitives and the other multicasts all require constant rounds only, the total expected round complexity of the undead binary consensus protocol is $O(1)$. □

Theorem 2. *The expected communication complexity of the undead binary consensus protocol described in Fig. 6 is $O(n^2\lambda)$.*

Proof. At the start of undead weak binary consensus, the parties multicast their value and verifying signature to each other. Since the value is binary, it takes $O(n^2\lambda)$ bits. Then each of n parties multicasts a value and a threshold signature using undead weak graded multicast. In undead weak graded multicast, a sender multicasts the value and threshold signature to all parties which echo binary values back to the sender, and the sender multicasts again correspondingly, which incurs a total of $O(n\lambda)$ bits. Since there are n calls, the total complexity is $O(n^2\lambda)$. Then each party multicasts two certificates which are essentially threshold signatures for 0 and 1. This is repeated twice and such an all-to-all round requires $O(n^2\lambda)$ bits. Based on certificates received, each party decides their V. They multicast V and its verifying signatures to all parties, which requires $O(n^2\lambda)$ bits. Some parties use the signatures to generate a threshold signature and multicast it along with the value, which also requires $O(n^2\lambda)$ bits. Thus, the total communication complexity of the undead weak binary consensus protocol is $O(n^2\lambda)$ bits.

As proved in Lemma 10, the expected number of iterations needed in undead binary consensus is constant. In each iteration, the parties call undead weak binary consensus and the common coin protocol first which requires $O(n^2\lambda)$ bits. Then the parties multicast values and signatures, and upon having $t+1$ valid signatures in D_v, they multicast values and threshold signatures. These are two all-to-all multicasts which requires $O(n^2\lambda)$ bits. Thus, the total communication complexity of each iteration is $O(n^2\lambda)$. Note that when a party outputs a value, it multicasts a value and a threshold signature. Alternatively, when a party becomes a zombie, it multicasts $\langle Zombie \rangle$. A party at most outputs or becomes a zombie once, so the communication complexity here is $O(\lambda)$ for each message and $O(n^2\lambda)$ for all-to-all communication. Overall, the total expected communication complexity of the undead binary consensus protocol is $O(n^2\lambda)$ bits. □

5.5 Undead Multi-valued Consensus

Our undead multi-valued consensus protocol is the multi-valued version of our undead binary consensus protocol.

Definition 5 (Undead Multi-valued Consensus). *Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where every party P_i holds a triplet (v_i, z_i, g_i) such that $v_i \in \{0,1\}^L$ is an L-bit value, and z_i, g_i are boolean values. Every party P_i terminates upon outputting a triplet (x_i, z_i, g_i) such that x_i is either a value or $\perp$, and z_i, g_i are boolean values. We say that Π is an undead multi-valued consensus protocol if it has the following properties:*

- ***Validity.*** *If all honest parties hold the same input v, every honest party P_i either outputs (x_i, z_i, g_i) such that $x_i = v$, or becomes a zombie by the end of the protocol, i.e., $z_i = 1$.*
- ***Consistency.*** *If two honest parties that are not zombies at the end of the protocol output (x_i, z_i, g_i) and (x_j, z_j, g_j), then $x_i = x_j$.*
- ***Termination.*** *All honest parties eventually output and terminate.*
- ***Detection Soundness.*** *If P_i is honest and it outputs (x_i, z_i, g_i) such that $z_i = 1$ (resp. $g_i = 1$), then it is receive faulty (resp. send faulty).*

We first introduce our undead propagate primitive. Similar to a dispersal protocol, it splits a large piece of data into small fragments using erasure codes and distributes them across multiple

Undead Propagate, every party P_i with inputs v_i, vc_i, z_i, g_i

1. Each party P_i sets $Z_i = z_i, G_i = g_i, V_i = \perp$. If $v_i \neq \perp$, P_i computes $c_i = \mathsf{EC.Encode}(v_i, n, t+1)$ and $VC_i = \mathsf{VCom}(c_i)$. Let $c_i[j]$ be the j-th bit of c_i. Then for each $j \in [n]$, P_i computes $\pi_i[j] = \mathsf{Open}(VC_i, c_i[j], j)$, and sends $(c_i[j], \pi_i[j])$ to P_j. If $v_i = \perp$, P_i sends nothing.
2. If $Z_i = 1$, P_i multicasts $\perp$. Otherwise, upon receiving $(c_j[i], \pi_j[i])$ from P_j such that $\mathsf{VerifyOpen}(vc_i, c_j[i], i, \pi_j[i]) = 1$, P_i multicasts $(i, c_j[i], \pi_j[i])$. If no such pair exists, P_i multicasts $\perp$.
3. If P_i receives fewer than $n - t - s$ messages, $Z_i = 1$. If $Z_i = 1$, P_i outputs $(\perp, Z_i, G_i)$ and terminates. Otherwise, upon receiving at least $t + 1$ $(j, c[j], \pi[j])$ tuples from distinct P_j, and for each tuple, $\mathsf{VerifyOpen}(vc_i, c[j], j, \pi[j]) = 1$, P_i reconstructs the value by computing $V_i = \mathsf{EC.Decode}(\{(j, c[j])\}_{j \in S}, n, t+1)$, where $S \subseteq [n]$ and $|S| = t + 1$. P_i outputs (V_i, Z_i, G_i).

Fig. 7. Our undead propagate primitive

parties. A party can reconstruct the data if it has a sufficient number of fragments. More specifically, in undead propagate, each party starts by encoding its value v (if it has one) into a vector c of size n with erasure code. It computes the vector commitment of c and generates proofs for each fragment of c. It then sends the fragment $c[i]$, along with its proof $\pi[i]$, to the corresponding party P_i. When P_i receives a valid $(c[i], \pi[i])$ pair, it forwards the pair to all parties. When a party receives $t + 1$ such valid pairs, it reconstructs the value. The full description is provided in Fig. 7. Note that a pair is considered valid by P_i if $\pi[i]$ correctly opens $c[i]$ against the commitment vc_i inputted by P_i at the start of the protocol. It is guaranteed that if all non-faulty parties input the same vector commitment vc_i, and at least one non-send-faulty party inputs the value that corresponds to vc_i, then all the non-receive-faulty parties can reconstruct the value.

Now we construct the undead multi-valued consensus protocol. The full description is provided in Fig. 8.. The protocol starts with each party P_i encoding its value v_i and generating its commitment VC_i. It then runs $O(\lambda)$ undead weak binary consensus in parallel, one for each bit of VC_i. By concatenating all the outputs from the undead weak binary consensus protocols, P_i obtains a commitment vc_i. If any of the outputs is $\perp$, $vc_i = \perp$. By the weak consistency property of undead weak binary consensus, vc_i should be identical across all the honest parties unless $vc_i = \perp$. We say that $vc_i \neq \perp$ is the chosen commitment, and the corresponding value is the chosen value. Then the parties multicast their vc_i together with the partial signatures of it to all parties if $vc_i \neq \perp$.

In step 3, if a party receives $t+1$ partial signatures for a commitment vc, it generates a certificate C for vc and sets $(vc_i, C_i) = (vc, C)$, and it multicasts (vc_i, C_i) to all parties. The receivers update their (vc_i, C_i) if they did not have such a pair. Note that if a party ends up not having a C_i, then $vc_i = \perp$, because without C_i, it cannot prove vc_i is the chosen commitment. The party P_i multicasts $(\langle Zombie1 \rangle, VC_i)$ if it is a zombie; otherwise, it multicasts 1 if $vc_i = VC_i$ which means P_i has the chosen value, or 0 if not.

In step 5, P_i multicasts $(\langle Zombie2 \rangle, VC_i)$ if it is a zombie. Otherwise, it collects the parties that send 0 and the parties that send $(\langle Zombie1 \rangle, VC_j)$ such that $VC_j \neq vc_i$. If $vc_i = VC_i$ and the total number of such parties is no more than t, P_i multicasts 1, otherwise 0. Note that if none of the non-faulty parties have the chosen value, they all multicast 0 in step 4. Thus, if P_i is not receive faulty, it will collect at least $t + 1$ 0, and thus multicast 0 in step 5.

In step 6, if P_i is a zombie, $B_i = \perp$. B_i will later be used as the input to an undead binary consensus protocol to decide whether there is a party that is able to propagate the chosen value to all parties. If P_i has no vc_i, $B_i = 0$. Otherwise, it adds the parties sending 1, 0, and $(\langle Zombie2 \rangle, VC_j)$ with $VC_j = vc_i$ into $E1$, $E2$, and $E3$ respectively. If $vc_i = VC_i$ and at least $t + r + 1$ parties are in $E1$ or $E3$, $B_i = 1$. This indicates that at least one non-receive-faulty honest party multicasts 1 in

Undead Multi-valued Consensus, every party P_i with inputs $v_i \in \{0,1\}^L, z_i, g_i$

1. Initially, each party P_i sets $VC_i = \mathsf{VCom}(\mathsf{EC.Encode}(v_i, n, t+1)), Z_i = z_i, G_i = g_i, C_i = \perp, B_i = \perp$. Let $VC_i[k]$ be the k-th bit of VC_i. Let sz denote the bit-length of VC_i. For each bit position $k \in [sz]$, P_i calls the undead weak binary consensus protocol denoted $\mathsf{WBC[k]}$, using $VC_i[k]$ as input.
2. Let $(vc_i[k], z_{i,k}, g_{i,k})$ be P_i's output from $\mathsf{WBC[k]}$, where $k \in [sz]$. If any $z_{i,k} = 1$, $Z_i = 1$. If any $g_{i,k} = 1$, $G_i = 1$. If any $vc_i[k] = \perp$, P_i sets $vc_i = \perp$ and multicasts $\perp$. Otherwise, P_i lets vc_i be the concatenation of $\{vc_i[k]\}_{k\in[sz]}$ and multicasts $(vc_i, \mathsf{TSS.PartSign}(sk_i, vc_i))$
3. If there exists vc', and P_i receives at least $t+1$ (vc_j, σ_j) pairs from distinct P_j such that $vc_j = vc'$ and $\mathsf{TSS.PartVrfy}(pk_j, vc', \sigma_j) = 1$, P_i sets $vc_i = vc', C_i = \mathsf{TSS.Agg}(vc', \{(j, \sigma_j)\}_{j\in S})$, where $S \subseteq [n]$ and $|S| = t+1$. If multiple (vc_i, C_i) pairs exist, P_i chooses one arbitrarily. P_i multicasts (vc_i, C_i).
4. If P_i receives fewer than $n-t-s$ pairs in the form of (vc_j, c_j) from distinct parties, $Z_i = 1$. If $C_i = \perp$, upon receiving a (vc_j, c_j) pair from P_j and $\mathsf{TSS.Vrfy}(pk, vc_j, c_j) = 1$, $(vc_i, C_i) = (vc_j, c_j)$. If C_i is still $\perp$, $vc_i = \perp$. If $Z_i = 1$, P_i multicasts $(\langle Zombie1\rangle, VC_i)$. Otherwise, if $vc_i = \perp$, P_i multicasts 0; if $vc_i = VC_i$, P_i multicasts 1; else, P_i multicasts 0.
5. If $Z_i = 1$, P_i multicasts $(\langle Zombie2\rangle, VC_i)$. Otherwise, if $vc_i = \perp$, P_i multicasts 0. If not, let $E1$ be the set of parties from which P_i receives 0. Let $E2$ be the set of parties P_j from which P_i receives $(\langle Zombie1\rangle, VC_{i,j})$ and $VC_{i,j} \neq vc_i$. If $vc_i = VC_i$ and $|E1| + |E2| \leq t$, P_i multicasts 1; otherwise it multicasts 0.
6. If $Z_i = 1$, $B_i = \perp$. Otherwise, if $vc_i = \perp$, $B_i = 0$. Otherwise, let $E1$ be the set of parties from which P_i receives 1. Let $E2$ be the set of parties from which P_i receives 0. Let $E3$ be the set of parties P_j from which P_i receives $(\langle Zombie2\rangle, VC_{i,j})$ and $VC_{i,j} = vc_i$. If $vc_i = VC_i$ and $|E1| + |E3| \geq t + r + 1$, $B_i = 1$. If $vc_i = VC_i$ and $|E2| \leq t$, $B_i = \perp$, else $B_i = 0$. Then P_i multicasts (vc_i, C_i).
7. If $Z_i = 0$ and $C_i = \perp$, upon receiving a (vc_j, c_j) pair from P_j and $\mathsf{TSS.Vrfy}(pk, vc_j, c_j) = 1$, $(vc_i, C_i) = (vc_j, c_j)$. P_i calls undead propagate with input (v_i, vc_i, Z_i, G_i).
8. Let (V_i, Z_i', G_i') be P_i's output from undead propagate. If $Z_i' = 1$, $Z_i = 1$. If $G_i' = 1$, $G_i = 1$. P_i calls undead binary consensus with input B_i.
9. Let (b_i, Z_i', G_i') be P_i's output from undead binary consensus. If $Z_i' = 1$, $Z_i = 1$. If $G_i' = 1$, $G_i = 1$. If $Z_i = 1$ or $b_i = 0$, P_i outputs $(\perp, Z_i, G_i)$. Otherwise, if $V_i = \perp$, P_i sets $Z_i = 1$. Then P_i outputs (V_i, Z_i, G_i).

Fig. 8. Our undead multi-valued consensus protocol

step 5. Else, if $vc_i = VC_i$ and there are at most t parties in $E2$, $B_i = \perp$; otherwise, $B_i = 0$. Then the parties multicast and update (vc_i, C_i).

The parties propagate v_i using undead propagate, and run undead binary consensus with input B_i. Let b_i be P_i's output from undead binary consensus. If $b_i = 0$, all parties output $\perp$. Otherwise, at least one non-send-faulty honest party has the chosen value, and the value should have been propagated to all parties with the undead propagate primitive. The party outputs the chosen value. If a party does not have the value, it is receive faulty, so it becomes a zombie.

We show that the undead multi-valued consensus protocol satisfies the defined properties. Without formal proof, since the undead weak binary consensus protocol (in step 1) and the undead propagate protocol (in step 7) are deterministic, and the undead binary consensus protocol (in step 8) terminates after expected $O(1)$ rounds (as proven in Lemma 10), this protocol terminates after a finite number of rounds, ensuring termination.

Lemma 12. *The protocol in Fig. 8 guarantees validity.*

Proof. Assume all the honest parties hold the same input v_i, and thus have the same VC_i. The undead weak binary consensus protocols are called with input being each bit of VC_i. By the validity property of undead weak binary consensus, the honest parties that are not zombies will have the same vc_i. These parties multicast the partial signatures of vc_i in step 2.

In step 3, all the non-receive-faulty parties receive $t+1$ partial signatures from non-faulty parties and thus generate C_i for vc_i. They then multicast (vc_i, C_i).

In step 4, an honest party either receives fewer than $n - t - s > t + r + 1$ pairs and becomes a zombie, or it receives at least one (vc, C) pair from a non-receive-faulty party. The zombies multicast $(\langle Zombie1\rangle, VC_i)$, and the remaining honest parties multicast 1.

In step 5, the zombies multicast $(\langle Zombie2\rangle, VC_i)$. For the other parties, $E1$ and $E2$ will only contain byzantines parties, since no honest parties multicast 0 in step 4, and all honest parties have the same $VC_i = vc_i$. As a result, these parties multicast 1.

In step 6, the zombies have $B_i = \perp$. For the non-receive-faulty honest parties, a total of $t+r+1$ non-faulty and receive-faulty-only parties will be in either $E1$ or $E3$. Thus, $|E1| + |E3| \geq t + r + 1$, and these parties have $B_i = 1$. For the receive faulty parties that are not zombies, they may have $B_i = 1$ for the same reason. If not, note that their $E2$ contains at most t parties that are byzantine, because no honest parties multicast 0 in step 5. Thus, these parties will have $B_i = \perp$. The parties then multicast (vc_i, C_i).

Since in step 4 all the honest parties except the zombies already obtain C, no parties update C_i in step 7. The zombies call the undead propagate primitive with input either $(v_i, \perp)$ or (v_i, vc_i), and the other honest parties call the undead propagate primitive with the same input (v_i, vc_i).

In the undead propagate primitive, all the honest parties compute the same erasure code c and vector commitment VC', and thus the same proofs π. They send the corresponding fragment of erasure code $(c[j], \pi[j])$ to P_j. At least $t+1$ non-faulty parties P_j receive valid $(c[j], \pi[j])$ pairs from non-send-faulty honest parties and forward them to all parties. All the honest parties, if not receive faulty, receive $t + 1$ such pairs, reconstruct v, and output v. In other words, the remaining parties that cannot reconstruct v are receive faulty and thus become zombies.

In step 8, the parties call the undead binary consensus protocol with input B_i. By the validity property of undead binary consensus, all honest parties except the zombies output 1. Finally, the honest parties that are not zombies output v, and the zombies output $\perp$.

Lemma 13. *The protocol in Fig. 8 guarantees consistency.*

Proof. We first show that if a non-send-faulty party calls the undead propagate primitive with input $(m, *)$, and every non-receive-faulty party has the input, the vector commitment vc for erasure code of m, then all the non-receive-faulty honest parties can reconstruct m at the end of the undead propagate primitive. For the non-send-faulty party with input $(m, *)$, in step 1 of the undead propagate primitive, the erasure code c, the vector commitment $VC = vc$, and the proof π are computed. When the party sends $(c[i], \pi[i])$ to a non-faulty party P_i, P_i can verify whether $c[i]$ is valid with its own input vc. Then P_i forwards the pair to all parties. All the non-receive-faulty parties receive $t+1$ such pairs from non-faulty parties, verify the pairs with their vc, and reconstruct m.

Now we assume an honest party that is not a zombie at the end of undead multi-valued consensus has $b = 0$ in step 9. By the consistency property of undead binary consensus, all the honest parties that are not zombies also output $b = 0$ from the undead binary consensus protocol. Thus, all the honest parties that are not zombies output $\perp$.

Assume an honest party that is not a zombie at the end of undead multi-valued consensus has $b = 1$ in step 9, and outputs $v \neq \perp$. By the validity property of undead binary consensus, we can deduce that at least one honest party calls undead binary consensus with input $B = 1$. This means that at least one honest party P_h has $VC = vc$ and $|E1| + |E3| \geq t + r + 1$ in step 6. Note that in step 4, $vc = \perp$ if $C = \perp$, thus $C_h \neq \perp$.

If P_h is not send faulty, it multicasts (vc, C_h) in step 6, and all the non-receive-faulty honest parties will receive it. P_h then calls undead propagate with input $(v, *)$. Because the commitment for v's erasure code $VC = vc$, as we have proven above, all the non-receive-faulty honest parties will be able to reconstruct v. The parties that cannot reconstruct v become zombies. In step 9, since $b = 1$, the honest parties that are not zombies output v, and the zombies output $\perp$.

If P_h is send faulty, since $|E1| + |E3| \geq t + r + 1$, there is at least one party P_j that is not byzantine or receive faulty, and is contained in either $E1$ or $E3$. In other words, P_j is either a non-faulty or a send-faulty-only party. By the detection soundness property of undead binary consensus, P_j is not a zombie, so it does not multicast $(\langle Zombie2 \rangle, *)$ in step 6. This means it is in $E1$, and it satisfies $|E1| + |E2| \leq t$ in step 5 because it multicasts 1 in step 5. Suppose that none of the non-faulty parties have (vc, C) and the corresponding value v in step 4. As a result, all these parties multicast 0 in step 4. Since P_j is not receive faulty, it should receive all these messages and contain these non-faulty parties in $E1$ in step 5, so $|E1| \geq t + 1$. However, P_j has $|E1| + |E2| \leq t$. By contradiction, at least one non-faulty party has $(vc = VC, C)$ and v in step 4. By following the same approach as above, we can deduce that all the honest parties that are not zombies output v, and the zombies output $\perp$. □

Lemma 14. *The protocol in Fig. 8 guarantees detection soundness.*

Proof. As proven in Lemma 13, if undead binary consensus outputs 1 in step 9, all non-receive-faulty honest parties get the chosen value from undead propagate. Thus, only receive faulty parties become zombies in step 9. Moreover, parties become zombies (resp. ghosts) if they become zombies (resp. ghosts) in undead weak binary consensus and undead binary consensus. Since undead weak binary consensus and undead binary consensus satisfy detection soundness, this protocol satisfies detection soundness, too. □

Below, we analyze the round and communication complexity of our undead multi-valued consensus protocol.

Theorem 3. *The expected round complexity of the undead multi-valued consensus protocol described in Fig. 8 is* $O(1)$.

Proof. In our undead multi-valued consensus protocol, the parties first execute $O(\lambda)$ undead weak binary consensus in step 1, each for a bit of the vector commitment of the input's erasure code. Note that these undead weak binary consensus protocols are executed in parallel, and the undead weak binary consensus protocol is deterministic, thus the number of rounds required is $O(1)$.

Our undead propagate primitive encodes the value, distributes the fragments, forwards the fragments, and reconstructs the value. Since distributing and forwarding the fragments each take only one round, and the other steps are computed locally, the round complexity of the undead propagate primitive is $O(1)$.

In step 9, the parties run the undead binary consensus protocol, which by Theorem 1, needs expected $O(1)$ rounds. Since all the other steps require a constant number of rounds, the total expected round complexity of the undead multi-valued consensus protocol is $O(1)$. □

Theorem 4. *The expected communication complexity of the undead multi-valued consensus protocol described in Fig. 8 is* $O(\frac{n^2}{t}L + n^2\lambda^2)$.

Proof. In our undead propagate primitive, each party encodes the L-bit value first, and the size of each fragment is $O(\frac{L}{t})$. The parties send the fragments to the corresponding parties, requiring $O(\frac{n^2 L}{t})$ bits. Then each party verifies the fragments and forwards the valid ones to all parties, requiring $O(\frac{n^2 L}{t})$ bits. Thus, the total communication complexity of the undead propagate primitive is $O(\frac{n^2 L}{t})$.

In our undead multi-valued consensus protocol, the parties run $O(\lambda)$ undead weak binary consensus protocols in parallel in step 1, each for a bit of the vector commitment of the input's erasure code. By Theorem 2, each undead weak binary consensus instance requires expected $O(n^2\lambda)$ bits, so a total of $O(n^2\lambda^2)$ bits is required. The parties also run the undead binary consensus protocol in step 9, which requires expected $O(n^2\lambda)$ bits.

The other steps contain all-to-all multicasts of values, signatures, and vector commitments, which uses up to $O(n^2\lambda)$ bits. As a result, the total expected communication complexity of the undead multi-valued consensus protocol is $O(\frac{n^2}{t}L + n^2\lambda^2)$ bits. □

6 Future Directions

This work presents a binary BA protocol in the mixed-fault model with overlapping faults that matches the lower bounds on both round complexity and communication complexity known for conventional BA. For multi-valued BA, our protocol achieves an expected communication complexity of $O(\frac{n^2}{t}L + n^2\lambda^2)$ and an expected round complexity of $O(1)$. The communication cost reduces to $O(nL + n^2\lambda^2)$ only when $t = O(n)$, where t is the maximum number of byzantine faults. How to close this gap and obtain a multi-valued BA protocol that is communication-optimal for arbitrary t remains an open problem.

Beyond efficiency, a critical challenge involves eliminating the requirement for a private setup. While a common coin can be instantiated using unique threshold signatures (as shown in Sec. 4), this necessitates a distributed key generation (DKG) protocol secure in the mixed-fault model. Constructing an optimally resilient DKG or common coin under mixed faults is currently an open question. Alternatively, as suggested by Loss and Stern [31], one could eliminate the use of threshold cryptography altogether and generate randomness through other means. For instance, classical BA protocols, such as those of Feldman and Micali [19] and Katz and Koo [27], employ verifiable secret sharing to derive randomness.

Lastly, our binary and multi-valued BA protocols can serve as foundational building blocks for more complex primitives, such as atomic broadcast, state machine replication, and secure multiparty computation (MPC). Building upon the fine-grained modeling of omission faults by Zikas et al. [40] and the tight feasibility characterization for MPC in the presence of mixed faults by Brazitikos and Zikas [8], extending the existing constructions to the mixed-fault model represents a promising avenue for achieving robust and efficient MPC.

Acknowledgment. This work is supported in part by ARC Discovery Project (DP250101739), and research awards from the Stellar Development Foundation.

References

1. Abraham, I., Chan, T.H., Dolev, D., Nayak, K., Pass, R., Ren, L., Shi, E.: Communication complexity of byzantine agreement, revisited. Distributed Comput. **36**(1), 3–28 (2023)

2. Abraham, I., Devadas, S., Dolev, D., Nayak, K., Ren, L.: Synchronous byzantine agreement with expected O(1) rounds, expected o(n^2) communication, and optimal resilience. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 11598, pp. 320–334. Springer (2019)
3. Abraham, I., Dolev, D., Kagan, A., Stern, G.: Brief announcement: Authenticated consensus in synchronous systems with mixed faults. In: DISC. LIPIcs, vol. 246, pp. 38:1–38:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022)
4. Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Reaching consensus for asynchronous distributed key generation. In: PODC. pp. 363–373. ACM (2021)
5. Abraham, I., Nayak, K., Ren, L., Xiang, Z.: Brief announcement: Byzantine agreement, broadcast and state machine replication with optimal good-case latency. In: DISC. LIPIcs, vol. 179, pp. 47:1–47:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
6. Bacho, R., Loss, J.: On the adaptive security of the threshold BLS signature scheme. In: CCS. pp. 193–207. ACM (2022)
7. Basu, S., Tomescu, A., Abraham, I., Malkhi, D., Reiter, M.K., Sirer, E.G.: Efficient verifiable secret sharing with share recovery in BFT protocols. In: CCS. pp. 2387–2402. ACM (2019)
8. Brazitikos, K., Zikas, V.: General adversary structures in byzantine agreement and multi-party computation with active and omission corruption. In: TCC (4). Lecture Notes in Computer Science, vol. 15367, pp. 200–233. Springer (2024)
9. Cachin, C., Kursawe, K., Lysyanskaya, A., Strobl, R.: Asynchronous verifiable secret sharing and proactive cryptosystems. In: CCS. pp. 88–97. ACM (2002)
10. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 7778, pp. 55–72. Springer (2013)
11. Civit, P., Dzulfikar, M.A., Gilbert, S., Guerraoui, R., Komatovic, J., Vidigueira, M.: DARE to agree: Byzantine agreement with optimal resilience and adaptive communication. In: PODC. pp. 145–156. ACM (2024)
12. Das, S., Ren, L.: Adaptively secure BLS threshold signatures from DDH and co-cdh. In: CRYPTO (7). Lecture Notes in Computer Science, vol. 14926, pp. 251–284. Springer (2024)
13. Deligios, G., Hirt, M., Liu-Zhang, C.: Round-efficient byzantine agreement and multi-party computation with asynchronous fallback. In: TCC (1). Lecture Notes in Computer Science, vol. 13042, pp. 623–653. Springer (2021)
14. Distler, T.: Byzantine fault-tolerant state-machine replication from a systems perspective. ACM Comput. Surv. **54**(1), 24:1–24:38 (2022)
15. Dolev, D., Reischuk, R.: Bounds on information exchange for byzantine agreement. J. ACM **32**(1), 191–204 (1985)
16. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
17. Dwork, C., Lynch, N.A., Stockmeyer, L.J.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
18. Eldefrawy, K., Loss, J., Terner, B.: How byzantine is a send corruption? In: ACNS. Lecture Notes in Computer Science, vol. 13269, pp. 684–704. Springer (2022)
19. Feldman, P., Micali, S.: Optimal algorithms for byzantine agreement. In: STOC. pp. 148–161. ACM (1988)
20. Feng, H., Lu, Z., Tang, Q.: Dragon: Decentralization at the cost of representation after arbitrary grouping and its applications to sub-cubic DKG and interactive consistency. In: PODC. pp. 469–479. ACM (2024)
21. Feng, H., Lu, Z., Tang, Q., Ye, Y.: Optimal byzantine agreement in the presence of message drops. In: ASIACRYPT (5). Lecture Notes in Computer Science, vol. 16249, pp. 163–195. Springer (2025)
22. Fischer, M.J., Lynch, N.A., Merritt, M.: Easy impossibility proofs for distributed consensus problems. In: PODC. pp. 59–70. ACM (1985)
23. Fitzi, M., Garay, J.A.: Efficient and player-optimal strong consensus. IACR Cryptol. ePrint Arch. p. 85 (2002)
24. Fitzi, M., Gisin, N., Maurer, U.M., von Rotz, O.: Unconditional byzantine agreement and multi-party computation secure against dishonest minorities from scratch. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 2332, pp. 482–501. Springer (2002)
25. Garay, J.A., Perry, K.J.: A continuum of failure models for distributed computing. In: WDAG. Lecture Notes in Computer Science, vol. 647, pp. 153–165. Springer (1992)
26. Hadzilacos, V.: Issues of Fault Tolerance in Concurrent Computations (Databases, Reliability, Transactions, Agreement Protocols, Distributed Computing). Ph.D. thesis, Harvard University (1985)
27. Katz, J., Koo, C.: On expected constant-round protocols for byzantine agreement. In: CRYPTO. Lecture Notes in Computer Science, vol. 4117, pp. 445–462. Springer (2006)
28. Koo, C.: Secure computation with partial message loss. In: TCC. Lecture Notes in Computer Science, vol. 3876, pp. 502–521. Springer (2006)

29. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
30. Loss, J., Shi, K., Stern, G.: Consensus in the presence of overlapping faults and total omission. In: TCC (1). Lecture Notes in Computer Science, vol. 15364, pp. 353–382. Springer (2024)
31. Loss, J., Stern, G.: Zombies and ghosts: Optimal byzantine agreement in the presence of omission faults. In: TCC (4). Lecture Notes in Computer Science, vol. 14372, pp. 395–421. Springer (2023)
32. Lu, Y., Lu, Z., Tang, Q., Wang, G.: Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In: PODC. pp. 129–138. ACM (2020)
33. Massey, J.L.: Review of 'theory and practice of error control codes' (blahut, r.e.; 1983). IEEE Trans. Inf. Theory **31**(4), 553–554 (1985)
34. Momose, A., Ren, L.: Optimal communication complexity of authenticated byzantine agreement. In: DISC. LIPIcs, vol. 209, pp. 32:1–32:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
35. Nayak, K., Ren, L., Shi, E., Vaidya, N.H., Xiang, Z.: Improved extension protocols for byzantine broadcast and agreement. In: DISC. LIPIcs, vol. 179, pp. 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
36. Parvédy, P.R., Raynal, M.: Uniform agreement despite process omission failures. In: IPDPS. p. 212. IEEE Computer Society (2003)
37. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: STOC. pp. 73–85. ACM (1989)
38. Raynal, M.: Consensus in synchronous systems: A concise guided tour. In: PRDC. pp. 221–228. IEEE Computer Society (2002)
39. Siu, H., Chin, Y., Yang, W.: Byzantine agreement in the presence of mixed faults on processors and links. IEEE Trans. Parallel Distributed Syst. **9**(4), 335–345 (1998)
40. Zikas, V., Hauser, S., Maurer, U.M.: Realistic failures in secure multi-party computation. In: TCC. Lecture Notes in Computer Science, vol. 5444, pp. 274–293. Springer (2009)