Quit-Resistant Reliable Broadcast and Efficient Terminating Gather

Mose Mizrahi Erbes
ETH Zurich, Switzerland

Roger Wattenhofer
ETH Zurich, Switzerland

Abstract

Termination is a central property in distributed computing. A party terminates a protocol once it stops accepting and sending messages. We discover that byzantine reliable broadcast is sometimes used in a manner which leads to non-terminating protocols. We consider an asynchronous network of n parties up to t of which are byzantine, and show that if each party is to broadcast its value and terminate upon obtaining $n-t$ values, then composing n parallel reliable broadcast instances leads to non-termination. The issue is that a party must quit t broadcast instances early in order to terminate, a behaviour not supported by ordinary reliable broadcast. So, we modify Bracha's protocol into a *quit-resistant* reliable broadcast (QBRB) protocol which lets the parties quit early. This protocol retains its termination guarantees as long as no party quits before some party terminates.

Then, we turn our attention to Gather, an all-to-all broadcast primitive which guarantees that the parties obtain $n-t$ common values. Existing error-free deterministic Gather protocols either run forever, or fail to terminate since the parties quit reliable broadcast instances. We design an error-free, deterministic, terminating (and binding) Gather protocol for ℓ-bit inputs with the communication complexity $\mathcal{O}(\ell n^2 + n^3 \log n)$. This matches the state-of-the-art for non-terminating Gather.

Finally, inspired by our QBRB protocol, we design a reliable broadcast protocol which retains its termination guarantees no matter when any party quits. To achieve this, we give each party the option to output $\perp$ if more than q parties quit before some party terminates. The protocol requires $4t + q < n$, which is optimal, and it lets parties quit after they have suffered transient crash failures so that they can help the remaining parties terminate.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Asynchronous networks, byzantine fault tolerance, protocol termination, reliable broadcast, all-to-all broadcast, gather

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2024.15

1 Introduction

Byzantine reliable broadcast (BRB) is a fundamental asynchronous communication primitive. Traditionally, a BRB protocol is run by n parties $P_1, P_2, \dots, P_n$, where up to t of the parties may deviate arbitrarily from the protocol while the rest remain honest. Given a fixed sender party P^* who eventually acquires an input v^*, a BRB protocol ensures the following:

- ***Validity:*** If P^* is honest, and an honest P_i outputs y_i, then P^* has acquired $v^* = y_i$.
- ***Consistency:*** If honest parties P_i and P_j output y_i and y_j, then $y_i = y_j$.
- ***Local Termination:*** If P^* is honest, then some honest party terminates.
- ***Global Termination:*** If some honest party terminates, then all honest parties terminate.

Notice that we split the output correctness properties from the termination properties. Since we focus on termination in this work, the breakdown above will be useful.

When we say that a protocol terminates, what do we mean? In their seminal work on asynchronous byzantine agreement [12], Canetti and Rabin say that a protocol terminates if all the honest parties "complete the protocol (i.e. terminate locally)." The way we understand

28th International Conference on Principles of Distributed Systems (OPODIS 2024).
Editors: Silvia Bonomi, Letterio Galletta, Etienne Rivière, and Valerio Schiavoni; Article No. 15; pp. 15:1–15:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

this is that they do not let parties keep sending messages after they locally terminate, and thus define termination as we will. In [4], Abraham et al. define termination as Canetti and Rabin do, but they also separately define "termination of output" to be the property that every honest party eventually outputs.

Following the definitions in [9], we distinguish between *liveness* and *termination.* A live protocol guarantees that if the honest parties all eventually acquire inputs and run forever, then they all output. Liveness is the property called "termination of output" in [4]. Termination is stronger. A terminating protocol guarantees that if the honest parties all eventually acquire inputs and run until they terminate, then they all terminate. A party can terminate once it has output, and after terminating it can no longer accept or send messages.

Termination as opposed to liveness is useful in that it allows the parties to go offline and forget about a protocol once they terminate. When the parties run an asynchronous protocol that is live but not terminating, they must forever remain alert for further messages, even if the network happens to be fast and they decide on their outputs almost instantly. In contrast, if the parties run a terminating asynchronous protocol, then they can go offline once they terminate. Even in networks where the parties are always active, termination lets a party free the resources it has allocated for a protocol, while liveness does not.

It is common in distributed computing to compose n (or more) instances of BRB to implement *all-to-all* broadcast. In all-to-all broadcast, each party P_i has an input v_i which it broadcasts to the other parties, and each party P_i outputs a set X_i of value-sender pairs (m, P) with at most one value per sender. Typically, one requires at least the following:

- For some constant $z \leq n - t$, the honest output sets contain at least z value-sender pairs.
- If there exists some $(m_j, P_j) \in X_i$ for any honest P_i and P_j, then $m_j = v_j$.
- If there exist some $(m, P) \in X_i$ and $(m', P) \in X_j$ for any honest P_i and P_j, then $m = m'$.

One simple all-to-all broadcast protocol which we name Π_{All} achieves precisely the guarantees above, with z maximally set to $n - t$. It consists of n parallel BRB instances, one for each party to broadcast its input. The idea is that since there exist $n - t$ honest parties, every honest party eventually terminates $n - t$ instances of BRB and thus terminates Π_{All}. In the wild, Π_{All} and its variants are commonly used in agreement protocols. For instance, Canetti and Rabin [12] use Π_{All} (with $z = 2t + 1$) for secret sharing/reconstruction in their seminal work on byzantine agreement [12], and Abraham, Amit and Dolev use a strengthened variant of Π_{All} as a core primitive in their seminal work on approximate agreement [2].

The problem the literature sometimes overlooks is that Π_{All} achieves liveness, but not termination. Observe that if a party P_i terminates Π_{All} upon terminating $n - t$ instances of BRB and obtaining $|X_i| = n - t$, then it stops running the t instances of BRB it has not yet terminated. However, unless the BRB protocol lets honest parties quit before they terminate, honest parties quitting BRB instances early leads to Π_{All} losing its liveness. We show this in Section 3 with an attack when Π_{All} uses Bracha's reliable broadcast protocol Π_{Bracha} [10].

One consequence of this issue is that the byzantine agreement protocol in [12] by Canetti and Rabin, whose termination mechanism is an Π_{All} variant which a party terminates after terminating $t + 1$ broadcasts with identical outputs, does not achieve termination. The protocol does achieve liveness, but it would also do so if one were to remove its termination mechanism entirely. The approximate agreement protocol against $t < \frac{n}{3}$ corruptions in [2] by Abraham et al. is similarly impacted. Contrary to how their protocol is written, a party can never halt. More recent approximate agreement protocols such as [15, 20, 21, 24, 27] based on the witness technique of [2] are impacted as well.

The question we thus ask is if we can redefine reliable broadcast to ensure the termination of protocols like Π_{All} where a party has to quit reliable broadcast instances to terminate. We answer this affirmatively in Section 4 with *quit-resistant* reliable broadcast (QBRB), our new BRB variant where the parties can quit (and inform the other parties of their quitting) before they terminate, with precisely the guarantees to prevent attacks of the sort against Π_{All}. A QBRB protocol retains validity and consistency no matter when any honest party quits, and retains global termination as long as no honest party quits before some honest party terminates. It turns out that one can easily modify Π_{Bracha} into a QBRB protocol resilient against $t < \frac{n}{3}$ corruptions. This is fortunate, as the aforementioned protocols which do not terminate due to BRB not supporting quits become terminating if one replaces the standard BRB invocations in them with QBRB invocations.

Note that a party informing the others when it quits (or wishes to quit) a protocol and this playing a role in termination is an idea that has appeared in the literature before. In [5], the authors design a protocol for byzantine agreement in partial synchrony where upon running a view (i.e. a tagged protocol instance) for too long, a party sends everyone an abort message, and these abort messages play a role in the parties switching to the next view.

We would also be remiss not to mention some of the techniques the literature uses to terminate live agreement protocols. One can upgrade a live byzantine agreement protocol into a terminating one with reliable consensus [13], a constant-round termination procedure based on Bracha's protocol with $\mathcal{O}(n^2)$ messages carrying protocol outputs. More generally, one can upgrade a live agreement protocol where the parties obtain at most ω distinct outputs (e.g. for approximate agreement in a graph [24]) into a terminating one with the constant-round termination procedure in [22] that builds on reliable consensus, if $t < \frac{n}{\max(3,\omega+1)}$.

However, for approximate agreement in $\mathbb{R}^d$, we do not know of any prior termination procedure. So, we offer two approaches to terminate approximate agreement in $\mathbb{R}^d$. One of them is QBRB, our general solution which by itself suffices to upgrade the protocols in [21, 27] into terminating ones. Our second solution that allows the efficient bundling of many approximate agreement instances (as in [18]) is a terminating Gather protocol. Gather is an Π_{All} variant which requires $|\bigcap_{P_i \text{ is honest}} X_i| \geq n - t$. It is especially useful for approximate agreement[1] [2, 15, 19, 20, 21, 24, 27], though its variants (with additional properties such as *binding* and *verifiability*, which we will discuss later) have been used for distributed key generation [4], common coin tossing [14, 18] and asynchronous core set agreement [3, 16].

In Section 5, we design Π_{Gthr}, an error-free deterministic Gather protocol optimally secure against $t < \frac{n}{3}$ corruptions, usable as a termination procedure for any approximate agreement protocol based on Gather iterations since it guarantees that if some honest party terminates, then all of them do. As far as we are aware, no other error-free[2] deterministic[3] Gather protocol in the literature achieves termination. The communication complexity of Π_{Gthr} for ℓ-bit inputs is $\mathcal{O}(\ell n^2 + n^3 \log n)$ bits. This is the complexity of the most efficient live Gather protocol ($\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$) we are aware of [15], when it is instantiated with a state-of-the-art standard BRB protocol with the complexity $\mathcal{O}(\ell n + n^2 \log n)$ [6]. Our Π_{Gthr} protocol additionaly lets one extract a core set of $n - t$ parties whose values are obtained by every honest party from the view of any honest party who outputs; hence, it is binding. Binding Gather was developed in [4], and there are cases involving secret shares which require the binding property [18].

In section 6, we return to QBRB, and design a QBRB variant Π_{Any} which retains local and global termination even if any honest party quits whenever it wishes. The standard output guarantees of reliable broadcast cannot be achieved with such lax participation, and

[1] For approximate agreement, the typical requirement is that $|X_i \cap X_j| \geq n - t$ for all honest P_i and P_j, which is implied by the stronger $|\bigcap_{P_i \text{ is honest}} X_i| \geq n - t$.

[2] We call a protocol *error-free* if it achieves the properties required of it in every possible execution.

[3] Gather is weakening of asynchronous core set agreement (ACS) [8], which requires $X_i = X_j$ for any honest P_i and P_j. Though there exist terminating ACS protocols [8], ACS implies byzantine agreement, and error-free deterministic asynchronous byzantine agreement is impossible against one crash fault [17].

for this reason we introduce the extra outputs $\bot$ and $\top$. If at most q honest parties quit before some honest party terminates, then Π_{Any} is just a QBRB protocol, albeit one with an extra output $\top$ to indicate that the sender quit before it acquired an input. If more than q honest parties quit before some honest party terminates, then we give each party the individual option to output $\bot$. The protocol Π_{Any} optimally requires $4t + q < n$ for this.

From a practical view, Π_{Any} is of interest in that it lets parties quit after they recover from transient crashes to help the remaining parties terminate. For example, a party can quit when it reconnects after a network disconnection, or when it restarts after a power loss. This ensures that as long as the honest parties that transiently crash (no matter how many) eventually come back online, all parties terminate Π_{Any}. We remark that reliable broadcast with recoverable transient crashes has been studied before. In [7], one can find a live BRB protocol against a polynomially bounded adversary for when $3t + 2f < n$ and at most f honest parties suffer transient crashes for a polynomially bounded number of times.

2 Model

We consider an asynchronous network of n message-sending parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ who are connected pairwise via reliable authenticated channels. The party set $\mathcal{P}$ is publicly known. The parties do not have access to synchronized clocks.

Our basic adversary, which we name the *t-adversary*, corrupts up to t parties. The corrupt parties are *byzantine*; they deviate arbitrarily from the protocol. The rest of the parties are *honest*. The adversary schedules messages as it wishes, and it is only required to eventually deliver all messages from honest senders. The adversary can also adaptively corrupt parties during the execution of a protocol depending on the sent messages and the parties' internal states, and drop messages from parties by corrupting them. So, we call a party honest if it is never corrupted. Unless stated otherwise, our protocols are designed against the t-adversary.

By default, a party must run a protocol until it is instructed to **terminate**. However, a QBRB protocol lets each party **quit** whenever it wishes by invoking the procedure Quit(), in which the party might send some final messages to help the remaining parties terminate. The adversary can influence if and when a party invokes Quit(). The invocation of Quit() is an event which a party handles like any other. A QBRB protocol keeps validity and consistency no matter when any honest party invokes Quit(), but potentially loses global termination if an honest party invokes Quit() before some honest party terminates.

In Section 6, we design the broadcast protocol Π_{Any} against an adversary which can cause a single transient crash for any honest party. This adversary can arbitrarily choose when any crash occurs and how long it lasts. At the end of a crash, the recovering party immediately invokes Quit(), having retained the information that Quit() needs (the kinds of messages the party has sent in Π_{Any}). We define the (t, crash)-adversary to be the t-adversary with the power to cause transient crashes as described above. Against the (t, crash)-adversary, Π_{Any} achieves local and global termination no matter when any honest party quits, though with weaker output guarantees if more than q honest parties quit before some honest termination.

When a party "multicasts" a message m, it sends m to all parties. To keep things simple, we assume that the honest parties do not transiently crash while multicasting, though note that we could remove this assumption by requiring any party that crashes while multicasting any m to remember m so that it can multicast m again when it recovers from the crash.

For composability, the parties do not begin protocols "having" inputs. Instead, they "acquire" inputs while they are running. This matters since we consider protocols where some parties might quit before they acquire inputs, or terminate despite never acquiring inputs.

3 Insufficiency of Standard BRB for Terminating All-to-All Broadcast

Below, we present the classical BRB protocol Π_{Bracha} [10]. Note that a party P_i only accepts a single `ECHO` message and a single `READY` message from any party P_j. We do not require a party to multicast $\langle \texttt{ECHO}, v\rangle$ upon receiving $t+1$ $\langle \texttt{READY}, v\rangle$ messages; this rule is absent in modern renditions of the protocol [1, 11]. The protocol Π_{Bracha} is proven secure in [1, 10, 11].

Protocol Π_{Bracha}

Code for a party P_i

upon acquiring the input v^* **do** *//only run by the sender P^**
  multicast $\langle \texttt{INIT}, v^*\rangle$
upon receiving a message $\langle \texttt{INIT}, v\rangle$ for some v from P^* for the first time **do**
  multicast $\langle \texttt{ECHO}, v\rangle$
upon receiving the message $\langle \texttt{ECHO}, v\rangle$ from $\lfloor \frac{n+t}{2} \rfloor + 1$ distinct parties **do**
  if you have not sent a `READY` message before, **then** multicast $\langle \texttt{READY}, v\rangle$
upon receiving the message $\langle \texttt{READY}, v\rangle$ from $t+1$ distinct parties **do**
  if you have not sent a `READY` message before, **then** multicast $\langle \texttt{READY}, v\rangle$
upon receiving the message $\langle \texttt{READY}, v\rangle$ from $2t+1$ distinct parties **do**
  output v and **terminate**

And now, we present a version of Π_{All} which fails to terminate because it uses Π_{Bracha}.

Protocol Π_{All}

Code for a party P_i

Join n common instances of Π_{Bracha}, named $\Pi^1_{\mathsf{Bracha}}, \dots, \Pi^n_{\mathsf{Bracha}}$. The party P_k is the sender of Π^k_{Bracha}.
$X_i \leftarrow \varnothing$
upon acquiring the input v_i **do**
  set v_i as the input for Π^i_{Bracha}
upon terminating Π^k_{Bracha} with the output m_k **do**
  $X_i \leftarrow X_i \cup \{(m_k, P_k)\}$
  if $|X_i| = n - t$ **then**
    output X_i and **terminate** *//To halt, P_i must halt all Π_{Bracha} instances.*

Let us informally argue that Π_{All} terminates against the t-adversary when $t < \frac{n}{3}$. Since there exist at least $n-t$ honest parties and since Π_{Bracha} guarantees termination for all honest parties when the sender is honest, eventually, some first honest P_i terminates $n-t$ instances of Π_{Bracha} and thus terminates Π_{All}. Then, since Π_{Bracha} guarantees termination for all honest parties when some honest party terminates, every honest party can eventually terminate Π_{All} by terminating the $n-t$ instances of Π_{Bracha} terminated by P_i.

As Theorem 1 below indicates, this argument fails. The intuitive reason why is that an honest party must stop participating in every Π_{Bracha} instance to terminate Π_{All}, which means that the party must prematurely quit t instances of Π_{Bracha} before terminating them. Though some honest party eventually terminates $n-t$ Π_{Bracha} instances, these instances lose global termination if other honest parties quit them before terminating them.

▶ **Theorem 1.** *When $n = 7$, the 2-adversary can prevent a particular party from terminating Π_{All} by making two parties omit sending messages to the party.*

Proof. The adversary corrupts P_2 and P_3. The party P_1 will be precluded from terminating.

The adversary lets every honest P_i acquire its input v_i, and assigns arbitrary inputs to the corrupt parties P_2 and P_3. These corrupt parties do not send P_1 any messages, but otherwise they behave honestly. The adversary begins with the following message scheduling:

- The communication between P_1 and the rest of the parties is indefinitely blocked.
- For $k \in \{4, \ldots, 7\}$, the communication for Π^k_{Bracha} is indefinitely blocked for the party $P_{\mathsf{next}(k)}$, where $\mathsf{next}(k) = k + 1$ if $k < 7$ and $\mathsf{next}(7) = 4$.
- Initially, all `ECHO` and `READY` messages are blocked.

First, each party P_i sends the `INIT` messages of its broadcast instance Π^i_{Bracha}. The `INIT` messages in Π^1_{Bracha} are received by only P_1. The `INIT` messages in Π^2_{Bracha} and Π^3_{Bracha} are received by all parties except P_1. For all $k \geq 4$, the `INIT` messages in Π^k_{Bracha} are received by all parties except P_1 and $P_{\mathsf{next}(k)}$.

Then, the adversary unblocks the `ECHO` messages. The `ECHO` messages in Π^1_{Bracha} are sent and received by only P_1. The `ECHO` messages in Π^2_{Bracha} and Π^3_{Bracha} are sent and received by all parties except P_1; so, all parties except P_1 send `READY` messages in Π^2_{Bracha} and Π^3_{Bracha}. For all $k \geq 4$, the `ECHO` messages in Π^k_{Bracha} are sent and received by all parties except P_1 and $P_{\mathsf{next}(k)}$; so, all parties except P_1 and $P_{\mathsf{next}(k)}$ send `READY` messages in Π^k_{Bracha}.

Afterwards, the adversary unblocks the `READY` messages. No party sends `READY` messages in Π^1_{Bracha}. The `READY` messages in Π^2_{Bracha} and Π^3_{Bracha} are sent and received by all parties except P_1. For all $k \geq 4$, the `READY` messages in Π^k_{Bracha} are sent and received by all parties except P_1 and $P_{\mathsf{next}(k)}$. So, for all $k \geq 4$, the party $P_{\mathsf{next}(k)}$ terminates Π_{All} by receiving at least $n - 2$ `READY` messages in all Π_{Bracha} instances except Π^1_{Bracha} and Π^k_{Bracha}. Since next is a permutation of $\{4, \ldots, 7\}$, we get that the parties $P_4, \ldots, P_7$ all terminate Π_{All}.

Finally, the adversary unblocks all communication. No party sends `READY` messages in Π^1_{Bracha}, and so P_1 does not terminate Π^1_{Bracha}. In any broadcast instance Π^k_{Bracha} where $k \geq 4$, the party P_1 receives `READY` messages from the 3 parties in $\{P_4, \ldots, P_7\} \setminus \{P_{\mathsf{next}(k)}\}$. This suffices for P_1 to send itself a `READY` message in Π^k_{Bracha}. Even then, P_i only receives 4 `READY` messages in Π^k_{Bracha}, which is insufficient for termination. Since P_1 cannot terminate any Π_{Bracha} instance except Π^2_{Bracha} and Π^3_{Bracha}, it cannot terminate Π_{All}. ◀

4 Quit-Resistant BRB

One could thwart the attack with a termination procedure based on reliable consensus [13] by exploiting the fact that if an honest party terminates a Π_{Bracha} instance, then every party learns the instance's output by receiving $n - 2t$ `READY` messages on the output. However, it is hard to come up with a termination procedure that works with any BRB protocol instead of just Π_{Bracha}, and our view is not that Π_{All} is lacking a termination procedure but that the termination properties of standard BRB are lacking for protocols like Π_{All} where a party must quit remaining BRB instances to terminate. Hence, we define quit-resistant BRB (QBRB). A QBRB protocol lets any party quit by invoking $\mathsf{Quit}()$. Given a fixed sender party P^* who might (or might not) eventually acquire an input v^*, a QBRB protocol ensures the following:

- ***Validity:*** If P^* is honest, and an honest P_i outputs y_i, then P^* has acquired $v^* = y_i$.
- ***Consistency:*** If honest parties P_i and P_j output y_i and y_j, then $y_i = y_j$.
- ***Local Termination:*** If P^* is honest, and it acquires an input, then either some honest party terminates, or some honest party quits.
- ***Global Termination:*** If some honest party terminates before any honest party quits, then every honest party either terminates or quits.

We keep the definitions of validity and consistency from standard BRB. However, while a standard BRB protocol requires the parties to run until instructed to terminate, a QBRB protocol lets each party quit whenever it wishes by invoking Quit(), and retains a meaningful global termination guarantee if no honest party quits before some honest party terminates.

QBRB is what Π_{All} needs to terminate. If in Π_{All} one replaces Π_{Bracha} with any QBRB protocol, then the QBRB instances terminated by the first honest party who terminates Π_{All} retain global termination, and thus Π_{All} terminates.

Below, we modify Π_{Bracha} into a QBRB protocol Π_{Quit}, optimally secure when $t < \frac{n}{3}$. We do so with the `QUIT` message which a party multicasts when it invokes Quit(), making it easier for the remaining parties to terminate. The `QUIT` messages do not compromise the safety guarantees since $t + 1$ `READY` messages are required to obtain output, and global termination relies on the fact that if the honest parties do not multicast `QUIT` before some honest party terminates, then the first honest termination occurs after $t + 1$ honest parties multicast $\langle \texttt{READY}, v \rangle$ for some v, which means that every honest party will multicast $\langle \texttt{READY}, v \rangle$ or `QUIT`.

Protocol Π_{Quit}

Code for a party P_i

$R \leftarrow \text{List}(n)$ *//list of accepted READY messages, indexed $R[1], \ldots, R[n]$*
$y_i \leftarrow \bot$ *//output of P_i, undetermined for now*
$a_i \leftarrow 0$ *//count of accepted QUIT messages*
upon acquiring the input v^* **do** *//input acquisition code for P^**
  multicast $\langle \texttt{INIT}, v^* \rangle$
upon receiving a message $\langle \texttt{INIT}, v \rangle$ for some v from P^* for the first time **do**
  multicast $\langle \texttt{ECHO}, v \rangle$
upon receiving the message $\langle \texttt{ECHO}, v \rangle$ from $\lfloor \frac{n+t}{2} \rfloor + 1$ distinct parties **do**
  //P_i only accepts a single ECHO message from any P_j.
  if you have not sent a `READY` message before, **then** multicast $\langle \texttt{READY}, v \rangle$
upon receiving some m such that $m = \texttt{QUIT}$ or $m = \langle \texttt{READY}, v \rangle$ for some v for the first time from a party P_j **do** *//do not accept both a QUIT and a READY from P_j*
  if $m = \langle \texttt{READY}, v \rangle$ for some v **then**
    $R[j] \leftarrow v$
    if R contains $t + 1$ copies of v **then**
      $y_i \leftarrow v$
      if you have not sent a `READY` message before, **then** multicast $\langle \texttt{READY}, v \rangle$
  if $m = \texttt{QUIT}$, **then** $a_i \leftarrow a_i + 1$
  if $y_i \neq \bot$ and R contains $2t + 1 - a_i$ copies of y_i **then**
    output y_i and **terminate**
upon invoking Quit() **do**
  if you have not sent a `READY` message before, **then** multicast `QUIT`
  quit

▶ **Theorem 2.** Π_{Quit} *is a secure QBRB protocol when* $t < \frac{n}{3}$.

Proof.

Consistency and Validity. For any v, let P_i be the first honest party who multicasts $\langle \texttt{READY}, v \rangle$. The party P_i cannot have done this after having received $\langle \texttt{READY}, v \rangle$ from $t + 1$ parties, because then there would need to exist a prior honest party who multicast $\langle \texttt{READY}, v \rangle$. Therefore, P_i must have received $\langle \texttt{ECHO}, v \rangle$ from $\lfloor \frac{n+t}{2} \rfloor + 1$ parties.

For contradiction, suppose that for some distinct v and v', some honest parties multicast $\langle \texttt{READY}, v\rangle$ and some honest parties multicast $\langle \texttt{READY}, v'\rangle$. Let P and P' be the first honest parties who respectively multicast $\langle \texttt{READY}, v\rangle$ and $\langle \texttt{READY}, v'\rangle$. Then, P must have received $\langle \texttt{ECHO}, v\rangle$ from $\lfloor \frac{n+t}{2} \rfloor + 1$ parties, and P' must have received $\langle \texttt{ECHO}, v'\rangle$ from $\lfloor \frac{n+t}{2} \rfloor + 1$ parties. These quorums have an intersection of at least $t+1$ parties, and hence we get the contradiction that an honest party must have sent $\langle \texttt{ECHO}, v\rangle$ to P and $\langle \texttt{ECHO}, v'\rangle$ to P'.

Suppose some honest P_i outputs v. Then, P_i must have received $\langle \texttt{READY}, v\rangle$ from at least $t+1$ parties, at least one of which is honest. Since honest parties cannot multicast $\langle \texttt{READY}, v'\rangle$ for any $v' \neq v$, no honest party outputs any $v' \neq v$. That is, we have consistency.

Furthermore, if P^* is honest, then the only message on which a party may receive ECHO messages from $\lfloor \frac{n+t}{2} \rfloor + 1 > t$ parties is the input v^* of P^*, after P^* has acquired v^* and multicast $\langle \texttt{INIT}, v^*\rangle$. This implies validity since an honest party can output any v only after some honest party receives $\langle \texttt{ECHO}, v\rangle$ from $\lfloor \frac{n+t}{2} \rfloor + 1$ parties and multicasts $\langle \texttt{READY}, v\rangle$.

Local Termination. For contradiction, consider a counterexample execution of Π_{Quit} where P^* is honest, and it acquires an input, but no honest party terminates or quits. Then, the honest parties participate forever in the execution. Every honest party receives $\langle \texttt{INIT}, v^*\rangle$ from P^* and thus multicasts $\langle \texttt{ECHO}, v^*\rangle$. Then, every honest party receives $\langle \texttt{ECHO}, v^*\rangle$ from $n - t \geq \lfloor \frac{n+t}{2} \rfloor + 1$ parties and thus multicasts $\langle \texttt{READY}, v^*\rangle$ if it has not multicast a READY message (which would have to be $\langle \texttt{READY}, v^*\rangle$) already. Finally, since all honest parties multicast $\langle \texttt{READY}, v^*\rangle$, some honest party terminates after receiving the message QUIT from f byzantine parties (where $f \leq t$) and receiving $\langle \texttt{READY}, v^*\rangle$ from $2t + 1 - f \leq n - t$ parties. This contradicts the assumption that no honest party terminates.

Global Termination. Suppose some honest P_i terminates with the output v, with no honest parties terminating or quitting earlier. Then, P_i must have received $\langle \texttt{READY}, v\rangle$ from $2t + 1 - f$ parties, where f is the number of QUIT messages P_i received. These f messages must have corrupt senders because honest parties do not quit before P_i terminates. Since P_i does not accept both a QUIT message and a READY message from the same party, at least $2t+1-f-(t-f) = t+1$ of the parties from whom P_i accepted the message $\langle \texttt{READY}, v\rangle$ must be honest. So, every honest party either multicasts QUIT, or receives $\langle \texttt{READY}, v\rangle$ from $t + 1$ parties, which makes it set its output to v and multicast $\langle \texttt{READY}, v\rangle$ if it has not multicast a READY message (which would have to be $\langle \texttt{READY}, v\rangle$) already. Finally, every honest party either quits, or terminates after receiving from $2t + 1$ parties the messages QUIT or $\langle \texttt{READY}, v\rangle$. ◀

We remark that we purposefully defined global termination so that it guarantees nothing if some honest party quits before any honest party terminates. In [23], the authors construct a signature-based reliable broadcast protocol which a party terminates upon receiving/constructing a certificate (that consists of the output y and signatures on y from $t + 1$ parties), after multicasting the certificate so that everyone can terminate by receiving it. This protocol is (with the addition of an empty Quit() procedure) a QBRB protocol which guarantees that if some honest party terminates, then every honest party who does not quit eventually terminates, no matter when any honest party quits. Our Π_{Quit} does not achieve this stronger global termination property, and we do not know whether any error-free QBRB protocol can.

5 Efficient Terminating Gather

In Gather, each party P_i may eventually acquire an ℓ-bit input v_i (where ℓ is publicly known), and each party P_i outputs a set of value-sender pairs X_i (with at most one value per sender). Against the t-adversary, a Gather protocol must achieve the following correctness properties:

- ***Common Core:*** There exists a common core set $\mathcal{C} \subseteq \mathcal{P}$ of size at least $n - t$ such that every honest output set X_i contains some (m_j, P_j) for all $P_j \in \mathcal{C}$.
- ***Validity:*** If there exists some $(m_j, P_j) \in X_i$ for any honest P_i and P_j, then $m_j = v_j$.
- ***Consistency:*** If there exist some $(m, P) \in X_i$ and $(m', P) \in X_j$ for any honest P_i and P_j, then $m = m'$.

If the protocol is *binding*, then the following stronger version of common core must hold:

- ***Binding Common Core:*** At the moment when some honest party outputs for the first time, one can extract a common core set $\mathcal{C}$ from the views of the honest parties.

Finally, the protocol should have one of the following termination-related properties:

- ***Liveness:*** Suppose the honest parties all eventually acquire inputs. If the honest parties all run the protocol forever, then they all obtain outputs from it.
- ***Termination:*** Suppose the honest parties all eventually acquire inputs. If the honest parties all run the protocol until they terminate it, then they all terminate the protocol.
- ***Strong Termination:*** If all honest parties eventually acquire inputs, then some honest party terminates; and if some honest party terminates, all honest parties terminate.

Strong termination is typically the termination property a termination procedure (e.g. reliable consensus [13]) achieves. To see why it matters, let Π_L and Π_T be Gather protocols, the former with liveness only. Let the parties run Π_L on their inputs, transform their Π_L outputs into Π_T inputs, obtain final outputs from Π_T, and terminate when they output. When a party terminates the composed protocol, it necessarily stops running Π_L. Hence, the remaining parties might not output from Π_L, and thus not acquire Π_T inputs. Now, if Π_T strongly terminates, then the composed protocol does so as well since no party quits Π_L before some party terminates Π_T. However, if Π_T is just terminating, then the composed protocol might not terminate since some honest parties might never obtain Π_T inputs. Approximate agreement protocols often consist of Gather iterations composed like this, and the strong termination of the last iteration suffices for the composed protocol to (strongly) terminate.

With $\Pi_{\mathsf{Gthr}}^{\mathsf{Live}}$ we denote the most efficient live Gather protocol we know of [15]. Instantiated with a state-of-the-art standard BRB protocol which requires $\mathcal{O}(\ell n + n^2 \log n)$ bits of communication [6], it achieves the bit complexity $\mathcal{O}(\ell n^2 + n^3 \log n)$. The protocol $\Pi_{\mathsf{Gthr}}^{\mathsf{Live}}$ is presented in [15], though for completeness we restate it and its security proof in the appendix. Note that $\Pi_{\mathsf{Gthr}}^{\mathsf{Live}}$ is not binding, but our strongly terminating Gather protocol Π_{Gthr} will be.

The protocol $\Pi_{\mathsf{Gthr}}^{\mathsf{Live}}$ involves n standard BRB broadcasts of ℓ-bit inputs, n standard BRB broadcasts of n-bit inputs and n multicasts of n-bit inputs. To make $\Pi_{\mathsf{Gthr}}^{\mathsf{Live}}$ (strongly) terminate, it suffices to replace its standard BRB broadcasts and multicasts with QBRB broadcasts. The issue is that against $t < \frac{n}{3}$ corruptions we do not know of a more efficient error-free QBRB protocol than Π_{Quit}, and replacing all standard BRB broadcasts and multicasts in $\Pi_{\mathsf{Gthr}}^{\mathsf{Live}}$ with Π_{Quit} instances results in the bit complexity $\mathcal{O}(\ell n^3 + n^4)$. Our Π_{Gthr} protocol instead achieves strong termination while preserving the complexity $\mathcal{O}(\ell n^2 + n^3 \log n)$.

5.1 k-slot Consensus

To construct Π_{Gthr}, we need a 5-slot consensus[4]protocol with strong termination. In a k-slot consensus protocol, each party P_i may eventually acquire an input $v_i \in \{0, 1\}$ and obtain an output $y_i \in \{0, \frac{1}{k-1}, \dots, \frac{k-2}{k-1}, 1\}$. The following output correctness properties must hold:

[4] In the literature 5-slot consensus is better known as (binary) graded consensus, with the possible outputs $(0, 2), (0, 1), (\perp, 0), (1, 1), (1, 2)$. We use the mapping $[(0, 2), (0, 1), (\perp, 0), (1, 1), (1, 2)] \leftrightarrow [0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}, 1]$. We define k-slot consensus so that we can compare k-slot consensus outputs with numbers.

- ***Validity:*** If for some b every honest input is b, then every honest output is b.
- ***k-Consistency:*** There exists some z such that every honest output is in $\{z, z + \frac{1}{k-1}\}$.

Of course, a k-slot consensus protocol must also have some termination guarantee. This may be liveness, termination or strong termination, which are defined as they were for Gather.

In [22], the authors present a constant-round quorum-based termination procedure which when $t < \frac{n}{\max(3,\omega+1)}$ turns any live protocol Π where the honest parties obtain at most ω distinct outputs into a strongly terminating[5] protocol where each honest party runs Π with its input and terminates with the Π output of some honest party. The procedure has an overhead of 3 additional rounds and $\mathcal{O}(\omega n^2)$ additional messages carrying Π outputs.

Since from k-slot consensus the honest parties obtain at most 2 distinct outputs and since k-slot consensus remains secure if the honest parties exchange their outputs, the procedure is perfectly suited to upgrade a live k-slot consensus protocol into a strongly terminating one when $t < \frac{n}{3}$. In the appendix, we present the strongly terminating k-slot consensus protocol $\Pi_{\mathsf{k\text{-}slot}}$ where we use the procedure on a live k-slot consensus protocol $\Pi_{\mathsf{k\text{-}slot}}$, alongside a proof since strong termination is not formally considered in [22].

In the rest of this section, we denote by $\Pi_{\mathsf{5\text{-}slot}}$ the protocol $\Pi_{\mathsf{k\text{-}slot}}$ instantiated with the constant-round error-free deterministic live 5-slot consensus protocol in [9] which we call $\Pi_{\mathsf{5\text{-}live}}$. Against $t < \frac{n}{3}$ corruptions, $\Pi_{\mathsf{5\text{-}live}}$ achieves 5-slot consensus with $\mathcal{O}(n^2)$ bits/messages. The 5-slot consensus protocol $\Pi_{\mathsf{5\text{-}slot}}$ has the same corruption tolerance $t < \frac{n}{3}$ and the same asymptotic complexity as $\Pi_{\mathsf{5\text{-}live}}$, and it is error-free and deterministic like $\Pi_{\mathsf{5\text{-}live}}$.

5.2 Online Error Correction

We use online error correction [8] based on Reed-Solomon error correcting codes [25] in order to obtain a communication complexity of the form $\mathcal{O}(\ell n^2 + \dots)$ for Π_{Gthr}. For some $a > \log_2(n)$, we use the following two functions:

- $\mathsf{Encode}(m)$: This function takes a message m of length $a \cdot (n-2t)$, and outputs a codeword $(s_1, \dots, s_n)$ of n symbols in the Galois Field $GF(2^a)$, each of which are a bits long.
- $\mathsf{TryDecode}(s_1, \dots, s_n)$: This function takes as input n symbols $s_1, \dots, s_n \in GF(2^a) \cup \{\bot\}$. With respect to a message m with $\mathsf{Encode}(m) = (s'_1, \dots, s'_n)$; we call a symbol s_j correct if $s_j = s'_j$, missing if $s_j = \bot$, and incorrect otherwise. When used with at most t missing symbols and at most t incorrect symbols with respect to some m, $\mathsf{TryDecode}(s_1, \dots, s_n)$ outputs m if at least $n - t$ of the symbols are correct with respect to m, and $\bot$ otherwise.

To use error correction on messages of any arbitrary length ℓ, we pad the messages to the length $a \cdot (n-2t)$ for some $a > \log_2(n)$. This gives us the symbol bit length $a = \max(\lceil \frac{\ell}{n-2t} \rceil, \lfloor \log_2(n) \rfloor + 1) = \mathcal{O}(\frac{\ell}{n} + \log n)$. For readability, we omit explicit padding/unpadding operations.

Online error correction is useful when there is a message m with $\mathsf{Encode}(m) = (s_1, \dots, s_n)$ such that each honest P_j multicasts s_j. Then, each honest P_i can keep track of the symbols it receives, and upon receiving $n - t + r$ symbols for each $r \in \{0, \dots, t\}$ run $\mathsf{TryDecode}$ in an attempt to obtain m. The output will be m or $\bot$ in each trial as in each trial there will be at most t missing symbols and at most t incorrect symbols with respect to m. Furthermore, since P_i can eventually receive the symbol s_j from each honest P_j, in some $\mathsf{TryDecode}$ trial there will be $n - t$ correct symbols and m will be the output.

[5] In [22], the authors do not consider strong termination, and only prove that their procedure upgrades live protocols into terminating ones. However, upon inspection one can observe that their procedure results in strong termination. This is also the case for the reliable consensus termination procedure [13].

The most efficient standard BRB protocol we are aware of [6] uses the online error correction approach above to achieve the bit complexity $\mathcal{O}(\ell n + n^2 \log n)$. The approach falters for QBRB because an honest party may quit a QBRB instance without ever sending any message related to the sender's input. This prevents us from designing a QBRB protocol against $t < \frac{n}{3}$ corruptions (and not just $t \leq \frac{(1-\varepsilon)n}{3}$ for a positive constant ε) with the bit complexity $\mathcal{O}(\ell n + n^2 \log n)$. Nevertheless, we are able to use online error correction in Π_{Gthr}.

5.3 The Gather Protocol

The Gather protocol $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$ is similar to Π_{All} in that the parties broadcast their inputs with standard BRB instances, and each honest P_i inserts a value-sender pair to its output set X_i when it terminates a BRB instance. In Π_{Gthr} we use $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$, but with external access to the output set X_i which each honest P_i builds in $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$. Hence, we denote the $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$ output of each honest P_i with Z_i. The set Z_i is a snapshot of X_i made when P_i outputs from $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$.

Inspired by the classical ACS protocol in [8] which involves one byzantine agreement instance per party to decide whether the party's input gets into the core set, we run n instances of $\Pi_{\mathsf{5\text{-}slot}}$, named $\Pi^1_{\mathsf{5\text{-}slot}}, \ldots, \Pi^n_{\mathsf{5\text{-}slot}}$. Upon obtaining Z_i from $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$, the party P_i provides an input to each $\Pi_{\mathsf{5\text{-}slot}}$ instance; the input is 1 if there exists a pair $(m_j, P_j) \in Z_i$.

We call an honest P_i happy if it has terminated every $\Pi_{\mathsf{5\text{-}slot}}$ instance, and has inserted some (m_j, P_j) to X_i whenever $g^j_i \geq \frac{1}{4}$. Note that $g^j_i > \frac{1}{4}$ indicates by the validity of $\Pi_{\mathsf{5\text{-}slot}}$ that some honest P_k has $(m_j, P_j) \in Z_k$. So, unless some honest party stops running $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$, eventually P_i can terminate the standard BRB terminated by P_k and insert (m_j, P_j) to X_i. Upon becoming happy, the party P_i lets $(S_{j,1}, \ldots S_{j,n}) = \mathsf{Encode}(m_j)$ if $g^j_i \geq \frac{1}{4}$, and lets $(S_{j,1}, \ldots S_{j,n}) = (\perp, \ldots, \perp)$ otherwise. Then, P_i sends each P_j the message $\langle \texttt{YOURS}, (S_{1,j}, \ldots, S_{n,j}) \rangle$.

Suppose that a party P_i terminates $\Pi^j_{\mathsf{5\text{-}slot}}$ with $g^j_i \geq \frac{2}{4}$, and that there are at least $t+1$ happy parties. Then, by the 5-consistency of $\Pi^j_{\mathsf{5\text{-}slot}}$, every happy party P_k has $g^j_k \geq \frac{1}{4}$, and thus the happy parties send YOURS vectors to P_i with a common non-$\perp$ j^{th} symbol. So, every party P_i can (and must, before it terminates Π_{Gthr}) eventually multicast $\langle \texttt{MINE}, (s_1, \ldots, s_n) \rangle$, where for all j, if $g^j_i \leq \frac{1}{4}$ then $s_j = \perp$, and otherwise s_j is the common non-$\perp$ j^{th} symbol in $t+1$ YOURS vectors which P_i receives. Note that if $s_j \neq \perp$, then s_j is the i^{th} symbol of $\mathsf{Encode}(m_j)$, where m_j is the unique message such that some honest P_k has $(m_j, P_j) \in X_k$.

Finally, a party P_i could terminate $\Pi^j_{\mathsf{5\text{-}slot}}$ with $g^j_i \geq \frac{3}{4}$. Then, to ensure the binding common core property, P_i must insert some (m_j, P_j) to its Π_{Gthr} output set Q_i. By the 5-consistency of $\Pi^j_{\mathsf{5\text{-}slot}}$, every honest P_k has $g^j_k \geq \frac{2}{4}$, which means that P_k multicasts a MINE vector whose j^{th} symbol is the k^{th} symbol of $\mathsf{Encode}(m_j)$, where m_j is the unique message such that some honest P_q has $(m_j, P_j) \in X_q$. So, P_i can obtain m_j via online error correction.

In Π_{Gthr}, we use READY quorums of size $t+1$ and $2t+1$ in the manner of Bracha's protocol to ensure without impacting liveness that no honest party terminates before $t+1$ parties become happy. Since a party must terminate every $\Pi_{\mathsf{5\text{-}slot}}$ instance to become happy, the strong termination of $\Pi_{\mathsf{5\text{-}slot}}$ ensures that every honest party terminates every $\Pi_{\mathsf{5\text{-}slot}}$ instance. Furthermore, the YOURS vectors multicast by at least $t+1$ happy parties ensure that the parties all multicast MINE vectors, and thus that they all construct the messages they need.

When a party P_i terminates Π_{Gthr}, one can use its view to compute the binding common core $\mathcal{C}_i = \{P_j \in \mathcal{P} : g^j_i = 1\}$. The common core property of $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$ and the validity of $\Pi_{\mathsf{5\text{-}slot}}$ ensure that $\mathcal{C}_i \supseteq \mathcal{C}'$ where $\mathcal{C}'$ is any common core of $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$, and therefore that $|\mathcal{C}_i| \geq n - t$. Moreover, $\mathcal{C}_i$ is a common core since the 5-consistency of $\Pi_{\mathsf{5\text{-}slot}}$ ensures that for all $P_k \in \mathcal{C}_i$, every honest P_j obtains $g^k_j \geq \frac{3}{4}$, and therefore inserts some (m_k, P_k) to Q_j.

In Π_{Gthr}, the parties send $\mathcal{O}(\ell n^2 + n^3 \log n)$ bits/$\mathcal{O}(n^3)$ messages in $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$, $\mathcal{O}(n^3 \log n)$ bits/$\mathcal{O}(n^3)$ messages in the n instances of $\Pi_{\mathsf{5\text{-}slot}}$ (the $\log n$ factor is from $\lceil \log_2(n) \rceil$-bit message tags which are necessary to distinguish the $\Pi_{\mathsf{5\text{-}slot}}$ instances), $\mathcal{O}(n^2)$ bits/messages for the `READY` notifications, and finally $\mathcal{O}(ln^2 + n^3 \log n)$ bits/$\mathcal{O}(n^2)$ messages for the `YOURS` and `MINE` vectors. Hence, the complexity of Π_{Gthr} is $\mathcal{O}(\ell n^2 + n^3 \log n)$ bits and $\mathcal{O}(n^3)$ messages.

Protocol Π_{Gthr}

Code for a party P_i

Join a common instance of $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$. Let X_i be the output set incrementally built in $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$ via n instances of standard BRB.
Join n common instances of $\Pi_{\mathsf{5\text{-}slot}}$, named $\Pi^1_{\mathsf{5\text{-}slot}}, \dots, \Pi^n_{\mathsf{5\text{-}slot}}$. Represent with g_i^j the output from $\Pi^j_{\mathsf{5\text{-}slot}}$.
$Y \leftarrow \text{Matrix}(n \times n)$ *//matrix of accepted `YOURS` symbols, initially filled with $\perp$*
$M \leftarrow \text{Matrix}(n \times n)$ *//matrix of accepted `MINE` symbols, initially filled with $\perp$*
$Q_i \leftarrow \varnothing$ *//Q_i will be the Π_{Gthr} output set of P_i*
upon acquiring the input v_i **do**
    set v_i as the input for $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$
upon obtaining the output Z_i from $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$ **do**
    for each j, set b_j as the input for $\Pi^j_{\mathsf{5\text{-}slot}}$, where $b_j = 1$ if there exists some $(m_j, P_j) \in Z_i$ and $b_j = 0$ otherwise
upon receiving some $\langle$`YOURS`$, (s_1, \dots, s_n)\rangle$ for the first time from a party P_j **do**
    $(Y_{j,1}, \dots, Y_{j,n}) \leftarrow (s_1, \dots, s_n)$
upon receiving `YOURS` vectors from $2t+1$ parties or `READY` from $t+1$ parties **do**
    multicast `READY`
Activate the rules below after terminating all $\Pi_{\mathsf{5\text{-}slot}}$ instances. Before that happens, buffer up to one received `MINE` vector from each P_j.
when there exists some $(m_j, P_j) \in X_i$ whenever $g_i^j \geq \frac{1}{4}$ **do** *//when P_i is happy*
    for $j \in \{1, \dots, n\}$ **do**
        if $g_i^j \geq \frac{1}{4}$, **then** $(S_{j,1}, \dots, S_{j,n}) \leftarrow \mathsf{Encode}(m_j)$, where $(m_j, P_j) \in X_i$
        if $g_i^j = 0$, **then** $(S_{j,1}, \dots, S_{j,n}) \leftarrow (\perp, \dots, \perp)$
    to each party P_j, send the message $\langle$`YOURS`$, (S_{1,j}, \dots, S_{n,j})\rangle$
when a non-$\perp$ symbol s_j is repeated at least $t+1$ times in the j^{th} column of Y whenever $g_i^j \geq \frac{2}{4}$ **do**
    multicast $\langle$`MINE`$, (s'_1, \dots, s'_n)\rangle$, where $s'_j = s_j$ if $g_i^j \geq \frac{2}{4}$ and $s'_j = \perp$ otherwise
upon receiving some $\langle$`MINE`$, (s_1, \dots, s_n)\rangle$ for the first time from a party P_j **do**
    $(M_{1,j}, \dots, M_{n,j}) \leftarrow (s_1, \dots, s_n)$
    for all $k \in \{1, \dots, n\}$ such that $g_i^k \geq \frac{3}{4}$ **do**
        if $(M_{k,1}, \dots, M_{k,n})$ has at most t copies of $\perp$ and $\nexists (m_k, P_k) \in Q_i$ **then**
            $m_k \leftarrow \mathsf{TryDecode}(M_{k,1}, \dots, M_{k,n})$
            if $m_k \neq \perp$, **then** $Q_i \leftarrow Q_i \cup \{(m_k, P_k)\}$
when you have received `READY` from $2t+1$ parties, you have multicast `READY` and some `MINE` vector, and there exists some $(m_j, P_j) \in Q_i$ whenever $g_i^j \geq \frac{3}{4}$ **do**
    stop running $\Pi^{\mathsf{Live}}_{\mathsf{Gthr}}$, **output** Q_i and **terminate**

▶ **Theorem 3.** *Π_{Gthr} is a secure binding Gather protocol with strong termination when $t < \frac{n}{3}$.*

Due to a lack of space, we formally prove Theorem 3 in the appendix.

Some Gather variants also achieve the property *verifiability* [3, 4, 26]. Essentially (as defined in [26]), each honest P_i has a function Verify_i which outputs a bit given P_i's current view and any party set $\mathcal{P}'$. $\mathsf{Verify}_i(\mathcal{P}')$ outputs 0 if $\mathcal{P}'$ does not contain the binding common core, and eventually outputs 1 if $\mathcal{P}'$ is the set of the senders in Q_j for some honest P_j. Also, if $\mathsf{Verify}_i(\mathcal{P}') = 1$ at any time, then $\mathsf{Verify}_i(\mathcal{P}') = 1$ at any later time. To achieve verifiability, we can replace 5-slot consensus in Π_{Gthr} with 6-slot consensus, and require each P_i to include some (m_j, P_j) in its output set Q_i when $g_i^j \geq \frac{3}{5}$. Then, from P_i we can extract the binding common core $\{P_j \in \mathcal{P} : g_i^j = 1\}$, and for each P_j let $\mathsf{Verify}_j(\mathcal{P}')$ output 1 after P_j terminates iff $\mathcal{P}' \supseteq \{P_k \in \mathcal{P} : g_j^k \geq \frac{4}{5}\}$. However, this is insufficient for the "agreement on verification" in [3, 4], which requires for all $\mathcal{P}'$ that if $\mathsf{Verify}_i(\mathcal{P}')$ outputs 1 for some P_i, then eventually $\mathsf{Verify}_j(\mathcal{P}')$ outputs 1 for all P_j. We do not see a way to achieve termination and "agreement on verification" together without switching from Gather to asynchronous core set agreement.

6 QBRB Against Arbitrary Quits

The QBRB protocol Π_{Quit} in Section 4 loses its termination guarantees if any honest party quits before some honest party terminates. Now, we design an error-free deterministic QBRB variant Π_{Any} which keeps its termination guarantees no matter when any honest party quits.

Of course, such lax participation from the honest parties makes the standard output guarantees impossible. Hence, we introduce the extra outputs $\bot$ and $\top$. For some publicly known q, if at most q honest parties quit before some honest party terminates, then Π_{Any} is just a QBRB protocol, albeit one with an extra output $\top$ to indicate that the sender quit before it acquired an input. If more than q honest parties quit before some honest party terminates, then we relax the output guarantees and give each party the individual option to output $\bot$. The protocol Π_{Any} optimally requires $4t + q < n$ for this, which means that if $t \ll n$, then almost every honest party can quit without the output $\bot$ becoming permitted.

Formally, let $\mathcal{M}$ be the input domain of Π_{Any}, with $\mathcal{M} \cap \{\bot, \top\} = \varnothing$. The protocol Π_{Any} has the output domain $\mathcal{M} \cup \{\bot, \top\}$, and given a fixed sender party P^* who may eventually acquire an input $v^* \in \mathcal{M}$, it guarantees the following against the (t, crash)-adversary:

- ***Validity:*** Suppose P^* is honest. If an honest P_i outputs $y_i \in \mathcal{M}$, then P^* has acquired $v^* = y_i$, and if an honest P_i outputs $\top$, then P^* has quit before acquiring an input.
- ***Consistency:*** If honest parties P_i and P_j output $y_i \neq \bot$ and $y_j \neq \bot$, then $y_i = y_j$.
- ***Robustness:*** If at most q honest parties quit before some honest party terminates, then no honest party outputs $\bot$.
- ***Local Termination:*** If P^* is honest, and it either acquires an input or quits before doing so, then either some honest party terminates, or all honest parties quit.
- ***Global Termination:*** If some honest party terminates, then every honest party either terminates or quits.

As per the definition of the (t, crash)-adversary in Section 2, the protocol Π_{Any} achieves security even if the reason a party quits is that it has suffered a transient crash. This makes Π_{Any} of practical interest when the parties are prone to such crashes. For example, a party could get disconnected from the network or run out of battery while running Π_{Any}. Then, upon recovering, the party would invoke $\mathsf{Quit}()$ to help the remaining parties terminate.

In Π_{Any}, transient crash tolerance follows from quit tolerance. If a party invokes $\mathsf{Quit}()$ upon recovering from a crash, then the protocol proceeds almost as if the party never crashed, but just invoked $\mathsf{Quit}()$ while running normally. From before crashing a party must retain only the information that $\mathsf{Quit}()$ needs. In Π_{Any}, a party must remember the kinds of messages (among `INIT`, `ECHO`, `READY`) that it multicast before crashing.

▶ **Theorem 4.** Π_{Any} *is secure when* $4t + q < n$.

Due to a lack of space, we formally prove Theorem 4 in the appendix. Note that $4t+q < n$ is optimal (by Theorem 5) even if the honest parties cannot transiently crash.

The main innovation in Π_{Any} is the flexible ECHO quorum threshold, which allows the honest parties to form ECHO quorums on the INIT value v^* of an honest sender P^* (the input of P^*, or $\top$ if P^* quits before acquiring an input) as long as at most $t+q$ honest parties quit. If P^* is honest, it multicasts $\langle \texttt{INIT}, v^* \rangle$, and at most $t+q$ honest parties quit, then for some $f \leq t+q$ there exist at least $n-t-f$ honest parties who multicast $\langle \texttt{ECHO}, v^* \rangle$ and at least f honest parties who multicast $\langle \texttt{ECHO}, \bot \rangle$. Since $n-t-f > \max(t, \frac{n+t-f}{2})$, this suffices for the honest parties to be able to form quorums on v^*.

As usual, conflicting ECHO quorums do not occur because each honest party multicasts at most one ECHO message. Intuitively, if the adversary can create conflicting quorums on v and v' with $\langle \texttt{ECHO}, \bot \rangle$ messages, then it can also do so with just $\langle \texttt{ECHO}, v \rangle$ and $\langle \texttt{ECHO}, v' \rangle$ messages. So, consider the case where the corrupt parties never send $\langle \texttt{ECHO}, \bot \rangle$. If $f \leq n$ honest parties multicast $\langle \texttt{ECHO}, \bot \rangle$, then a party must receive $\langle \texttt{ECHO}, v \rangle$ from more than $\frac{n+t-f}{2}$ parties to form an ECHO quorum on v. Conflicting quorums on $v \neq v'$ do not occur since at most $n-f$ parties send either $\langle \texttt{ECHO}, v \rangle$ or $\langle \texttt{ECHO}, v' \rangle$. Lastly, an invalid quorum on v (formed without P^* sending $\langle \texttt{INIT}, v \rangle$) is prevented by a quorum on v requiring at least $t+1$ $\langle \texttt{ECHO}, v \rangle$ messages.

Protocol Π_{Any}

Code for a party P_i

$R, E \leftarrow \mathsf{List}(n), \mathsf{List}(n)$ *//respective lists of accepted READY and ECHO messages*
$y_i \leftarrow \bot$ *//output of P_i, initially set to $\bot$*
upon acquiring the input v^* **do**
  multicast $\langle \texttt{INIT}, v^* \rangle$
upon receiving a message $\langle \texttt{INIT}, v \rangle$ for some $v \neq \bot$ from P^* for the first time **do**
  multicast $\langle \texttt{ECHO}, v \rangle$
upon receiving a message $\langle \texttt{ECHO}, v \rangle$ for the first time from a party P_j **do**
  $E[j] \leftarrow v$
  $f \leftarrow$ the number of $\bot$ symbols in E
  if E contains $\max(t, \lfloor \frac{n+t-f}{2} \rfloor) + 1$ copies of some $v' \neq \bot$ **then**
    if you have not sent a READY message before, **then** multicast $\langle \texttt{READY}, v' \rangle$
upon receiving QUIT from $t+q+1$ parties or $\langle \texttt{READY}, \bot \rangle$ from $t+q+1$ parties **do**
  if you have not sent a READY message before, **then** multicast $\langle \texttt{READY}, \bot \rangle$
upon receiving a message $\langle \texttt{READY}, v \rangle$ for the first time from a party P_j **do**
  $R[j] \leftarrow v$
  if R contains $t+1$ copies of some $v \neq \bot$ **then**
    $y_i \leftarrow v$
    if you have not sent a READY message before, **then** multicast $\langle \texttt{READY}, v \rangle$
  if R contains $n-t$ values in $\mathcal{M} \cup \{\bot, \top\}$ **then**
    output y_i and **terminate**
upon invoking $\mathsf{Quit}()$ **do**
  if $P_i = P^*$ and you have not sent an INIT message before, **then** multicast $\langle \texttt{INIT}, \top \rangle$
  if you have not sent an ECHO message before, **then** multicast $\langle \texttt{ECHO}, \bot \rangle$
  if you have not sent a READY message before, **then** multicast $\langle \texttt{READY}, \bot \rangle$
  multicast QUIT and **quit**

If an honest party quits, then it multicasts both `QUIT` and some `READY` message. If $t+q+1$ honest parties quit, then the $t+q+1$ `QUIT` messages they multicast suffice for every honest party to multicast a `READY` message. Hence, every honest party can receive `READY` messages from $n-t$ parties and terminate. On the other hand, if at most $t+q$ honest parties quit, then we get local termination by the fact that after the honest sender P^* multicasts $\langle \texttt{INIT}, v^* \rangle$, every honest party receives sufficiently many `ECHO` messages to multicast $\langle \texttt{READY}, v^* \rangle$.

If some honest party terminates, then there are at least $n - 2t \geq 2t + q + 1$ honest parties who multicast `READY` messages. Since for some $v \neq \bot$ every honest `READY` message is either $\langle \texttt{READY}, v \rangle$ or $\langle \texttt{READY}, \bot \rangle$, either there are at least $t+1$ honest parties who multicast $\langle \texttt{READY}, v \rangle$, or there are at least $t+q+1$ honest parties who multicast $\langle \texttt{READY}, \bot \rangle$. Either way, every honest party multicasts a `READY` message, and so every honest party can terminate.

Finally, if at most q honest parties quit before some honest P_i terminates, then P_i terminates with at most q honest copies of $\langle \texttt{READY}, \bot \rangle$. Hence, P_i terminates with at least $n - 2t - q$ honest copies of $\langle \texttt{READY}, v \rangle$ for some $v \neq \bot$. The senders of these copies do not send `READY` messages on any $v' \neq v$; so, any honest P_j who terminates (including P_i) does so after receiving $n - 3t - q \geq t + 1$ honest copies of $\langle \texttt{READY}, v \rangle$ and setting its output to v.

▶ **Theorem 5.** *If $|\mathcal{M}| \geq 2$, $n \geq 3$, $t \geq 1$ and $4t + q \geq n$, then no broadcast protocol achieves against the t-adversary the properties validity, consistency, robustness, local termination and global termination (all as defined for Π_{Any}).*

Proof. Suppose $\mathcal{M} \supseteq \{m, m'\}$ for some $m \neq m'$, $n \geq 3$, $t \geq 1$ and $4t + q \geq n$. We partition $\mathcal{P}$ into five sets S_1, S_2, S_3, Q_C, Q_H, where $1 \leq |S_1|, |S_2|, |S_3| \leq t$, $|Q_C| \leq t$ and $|Q_H| \leq q$. Let us consider a broadcast instance where $P^* \in S_2$.

In the five scenarios below, the parties in Q_H are honest, and they quit immediately. The parties in Q_C also quit immediately, but they may be corrupt parties only pretending to quit. Note that the sender's input does not influence the behavior of a non-sender who immediately quits. Finally, the parties in S_1, S_2 and S_3 never quit or pretend to quit. In the first two scenarios, the parties in S_1 and S_2 are honest, and P^* has the input m.

- ***Scenario 1:*** In this scenario, S_3 is the set of corrupt parties, and the parties in S_3 immediately crash. The parties in S_1 must terminate despite never hearing from S_3.
- ***Scenario 2:*** In this scenario, Q_C is the set of corrupt parties, and the messages from S_3 are delayed. The parties in S_1 cannot afford to wait for messages from S_3, because for them this scenario is indistinguishable from Scenario 1. Hence, they must all terminate. Furthermore, they must output m because at most q honest parties quit before some honest party terminates and because the honest sender P^* never quits.

In the next two scenarios, the parties in S_2 and S_3 are honest, and P^* has the input m'.

- ***Scenario 3:*** In this scenario, S_1 is the set of corrupt parties, and the parties in S_1 immediately crash. The parties in S_3 must terminate despite never hearing from S_1.
- ***Scenario 4:*** In this scenario, Q_C is the set of corrupt parties, and the messages from S_1 are delayed. Being in the same predicament as the parties in S_1 in Scenario 2, the parties in S_3 must all terminate with the output m' without waiting for messages from S_1.

Finally, consider Scenario 5, where the adversary successfully executes a split-brain attack.

- ***Scenario 5:*** In this scenario, S_2 is the set of corrupt parties. The parties in S_2 act towards S_1 as if P^* has the input m and the messages from S_3 are delayed, and act towards S_2 as if P^* has the input m' and the messages from S_1 are delayed. Moreover, the communication between S_1 and S_3 is indefinitely delayed. The parties in S_1 output m as they cannot distinguish this scenario from Scenario 2. Likewise, the parties in S_3 output m' as they cannot distinguish this scenario from Scenario 4. This violates consistency. ◀

References

1 Ittai Abraham. Living with asynchrony: Bracha's reliable broadcast. Decentralized Thoughts, 2020. URL: https://decentralizedthoughts.github.io/2020-09-19-living-with-asynchrony-brachas-reliable-broadcast/.

2 Ittai Abraham, Yonatan Amit, and Danny Dolev. Optimal resilience asynchronous approximate agreement. In *Proceedings of the 8th International Conference on Principles of Distributed Systems*, OPODIS '04, pages 229–239, Berlin, Heidelberg, 2004. Springer-Verlag. doi:10.1007/11516798_17.

3 Ittai Abraham, Gilad Asharov, Arpita Patra, and Gilad Stern. Perfectly secure asynchronous agreement on a core set in constant expected time. Cryptology ePrint Archive, Paper 2023/1130, 2023. URL: https://eprint.iacr.org/2023/1130.

4 Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC '21, pages 363–373, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3465084.3467914.

5 Ittai Abraham and Gilad Stern. Information Theoretic HotStuff. In Quentin Bramas, Rotem Oshman, and Paolo Romano, editors, *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*, volume 184 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.OPODIS.2020.11.

6 Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC '22, pages 399–417, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519270.3538475.

7 Michael Backes and Christian Cachin. Reliable broadcast in a computational hybrid model with byzantine faults, crashes, and recoveries. In *2003 International Conference on Dependable Systems and Networks, 2003. Proceedings.*, pages 37–46, 2003. doi:10.1109/DSN.2003.1209914.

8 Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 52–61, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/167088.167109.

9 Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In *Theory of Cryptography*, pages 131–150, Cham, Switzerland, 2019. Springer International Publishing. doi:10.1007/978-3-030-36030-6_6.

10 Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987. doi:10.1016/0890-5401(87)90054-X.

11 Christian Cachin. Secure distributed computing. École Polytechnique Fédérale de Lausanne, 2009. URL: https://dcl.epfl.ch/site/_media/education/sdc_byzconsensus.pdf.

12 Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 42–51, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/167088.167105.

13 Annick Chopard, Martin Hirt, and Chen-Da Liu-Zhang. On communication-efficient asynchronous mpc with adaptive security. In *Theory of Cryptography: 19th International Conference*, TCC'21, pages 35–65, Berlin, Heidelberg, 2021. Springer-Verlag. doi:10.1007/978-3-030-90453-1_2.

14 Ran Cohen, Pouyan Forghani, Juan Garay, Rutvik Patel, and Vassilis Zikas. Concurrent asynchronous byzantine agreement in expected-constant rounds, revisited. In *Theory of Cryptography*, pages 422–451, Cham, Switzerland, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-48624-1_16.

15 Andrei Constantinescu, Diana Ghinea, Roger Wattenhofer, and Floris Westermann. Convex Consensus with Asynchronous Fallback. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:23, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.DISC.2024.15`.

16 Sourav Das, Sisi Duan, Shengqi Liu, Atsuki Momose, Ling Ren, and Victor Shoup. Asynchronous consensus without trusted setup or public-key cryptography. Cryptology ePrint Archive, Paper 2024/677, 2024. Appeared in ACM CCS '24. `doi:10.1145/3658644.3670327`.

17 Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. In *Proceedings of the 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, PODS '83, pages 1–7, New York, NY, USA, 1983. Association for Computing Machinery. `doi:10.1145/588058.588060`.

18 Luciano Freitas, Petr Kuznetsov, and Andrei Tonkikh. Distributed Randomness from Approximate Agreement. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing (DISC 2022)*, volume 246 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:21, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.DISC.2022.24`.

19 Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Optimal synchronous approximate agreement with asynchronous fallback. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC '22, pages 70–80, New York, NY, USA, 2022. Association for Computing Machinery. `doi:10.1145/3519270.3538442`.

20 Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Multidimensional approximate agreement with asynchronous fallback. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '23, pages 141–151, New York, NY, USA, 2023. Association for Computing Machinery. `doi:10.1145/3558481.3591105`.

21 Hammurabi Mendes and Maurice Herlihy. Multidimensional approximate agreement in byzantine asynchronous systems. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 391–400, New York, NY, USA, 2013. Association for Computing Machinery. `doi:10.1145/2488608.2488657`.

22 Mose Mizrahi Erbes and Roger Wattenhofer. Asynchronous approximate agreement with quadratic communication, 2024. `arXiv:2408.05495`, `doi:10.48550/arXiv.2408.05495`.

23 Atsuki Momose and Ling Ren. Multi-threshold byzantine fault tolerance. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 1686–1699, New York, NY, USA, 2021. Association for Computing Machinery. `doi:10.1145/3460120.3484554`.

24 Thomas Nowak and Joel Rybicki. Byzantine Approximate Agreement on Graphs. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.DISC.2019.29`.

25 I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960. `doi:10.1137/0108018`.

26 Gilad Stern and Ittai Abraham. Gather with binding and verifiability. Decentralized Thoughts, 2024. URL: `https://decentralizedthoughts.github.io/2024-01-09-gather-with-binding-and-verifiability/`.

27 Nitin H. Vaidya and Vijay K. Garg. Byzantine vector consensus in complete graphs. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 65–73, New York, NY, USA, 2013. Association for Computing Machinery. `doi:10.1145/2484239.2484256`.

A Strongly Terminating k-slot Consensus

In the protocol $\Pi_{\text{k-slot}}$ below, we use the termination procedure from [22] to upgrade any live k-slot consensus protocol $\Pi_{\text{k-live}}$ into a strongly terminating one. Strong termination is the combination of local termination (some honest party terminates) and global termination (if some honest party terminates, every honest party terminates), and $\Pi_{\text{k-slot}}$ ensures global termination with `READY` quorums. If some honest party terminates, then it has received $2t+1$ `READY` messages, at least $t+1$ of which are honest, and so every honest party receives $t+1$ honest `READY` messages, multicasts `READY`, and receives $2t+1$ `READY` messages. Furthermore, the first honest party who multicasts `READY` does so upon receiving some $\Pi_{\text{k-live}}$ output z from $2t+1$ parties, which means that every honest party can set z as its output upon receiving z from $t+1$ honest parties.

Protocol $\Pi_{\text{k-slot}}$

Code for a party P_i

$y_i \leftarrow \bot$ *//output of P_i, undetermined for now*
upon acquiring the input v_i **do**
  set v_i as the input for $\Pi_{\text{k-live}}$
upon obtaining the output z_i from $\Pi_{\text{k-live}}$ **do**
  if you have not multicast z_i before, **then** multicast z_i
upon receiving any $z \in \{0, \frac{1}{k-1}, \ldots, 1\}$ from $t+1$ distinct parties **do**
  if $y_i = \bot$, **then** $y_i \leftarrow z$
  if you have not multicast z before, **then** multicast z
upon receiving any $z \in \{0, \frac{1}{k-1}, \ldots, 1\}$ from $2t+1$ distinct parties **do**
  if you have not multicast `READY` before, **then** multicast `READY`
upon receiving `READY` from $t+1$ distinct parties **do**
  if you have not multicast `READY` before, **then** multicast `READY`
upon having received `READY` from $2t+1$ distinct parties and having $y_i \neq \bot$ **do**
  output y_i and **terminate**

▶ **Theorem 6.** *Suppose $\Pi_{\text{k-live}}$ achieves validity, k-consistency and liveness when $t < \frac{n}{3}$. Then, $\Pi_{\text{k-slot}}$ achieves validity, k-consistency and strong termination when $t < \frac{n}{3}$.*

Proof.

k-Consistency and Validity. An honest party must receive z from $t+1$ parties (at least one of which is honest) to output z, and for any z, the first honest party who multicasts z must have output z from $\Pi_{\text{k-live}}$. Hence, every honest $\Pi_{\text{k-slot}}$ output is the $\Pi_{\text{k-live}}$ output of some honest party. The k-consistency and validity of $\Pi_{\text{k-slot}}$ thus follow from the k-consistency and validity of $\Pi_{\text{k-live}}$.

Strong Termination. To prove strong termination, we prove local termination and global termination. Local termination guarantees that if the honest parties all eventually acquire inputs, then some honest party terminates. Global termination guarantees that if some honest party terminates, then all honest parties terminate.

For the sake of contradiction, consider an execution of $\Pi_{\text{k-slot}}$ where local termination fails. Every honest party provides an input to $\Pi_{\text{k-live}}$, and hence every honest party obtains a $\Pi_{\text{k-live}}$ output. By the k-consistency of $\Pi_{\text{k-live}}$ there exists some z such that every honest $\Pi_{\text{k-live}}$ output is either z or $z + \frac{1}{k-1}$. By the pigeonhole principle there exists some $z' \in \{z, z + \frac{1}{k-1}\}$

such that at least $\lceil \frac{n-t}{2} \rceil \geq t+1$ honest parties output z' from $\Pi_{\text{k-live}}$. At least $t+1$ honest parties multicast z', and this suffices for every honest P_i to receive z' from $t+1$ parties, set $y_i \leftarrow z'$ if $y_i = \bot$ and multicast z'. Since all honest parties multicast z', all honest parties can receive z' from $2t+1$ parties and thus multicast READY. Finally, since all honest parties multicast READY, all honest parties can receive READY from $2t+1$ parties and thus terminate.

Now, let us show global termination. If some honest party terminates, then it must have received READY from $2t+1$ parties, at least $t+1$ of which are honest. Hence, every honest party multicasts READY, at the latest upon receiving READY from $t+1$ parties, and so any honest P_i who sets $y_i \neq \bot$ becomes able to terminate by receiving READY from $2t+1$ parties. Furthermore, the first honest party who multicasts READY does so when it has received some z from $2t+1$ parties, at least $t+1$ of which are honest. Since $t+1$ honest parties multicast z, every honest P_j can receive z from $t+1$ parties and set $y_j \leftarrow z$. ◀

B The Live Gather Protocol

Below, we present $\Pi^{\text{Live}}_{\text{Gthr}}$, the most efficient live Gather protocol we are aware of [15]. It achieves the bit complexity $\mathcal{O}(\ell n^2 + n^3 \log n)$ when instantiated with the state-of-the-art standard BRB protocol Π_{LBRB} with the bit complexity $\mathcal{O}(\ell n + n^2 \log n)$ [6].

Protocol $\Pi^{\text{Live}}_{\text{Gthr}}$

Code for a party P_i

Join $2n$ common instances of Π_{LBRB}, named $\Pi^{0,1}_{\text{LBRB}}, \dots, \Pi^{0,n}_{\text{LBRB}}, \Pi^{1,1}_{\text{LBRB}}, \dots, \Pi^{1,n}_{\text{LBRB}}$. The party P_k is the sender of $\Pi^{0,k}_{\text{LBRB}}$ and $\Pi^{1,k}_{\text{LBRB}}$.
$W^i_0, W^i_1, W^i_2 \leftarrow \varnothing, \varnothing, \varnothing$ *//"witness sets" of P_i, as per the witness technique of [2]*
$X_i \leftarrow \varnothing$ *//X_i is the output set of P_i, updated even after P_i outputs*
upon acquiring the input v_i **do**
    set v_i as the input for $\Pi^{0,i}_{\text{LBRB}}$
upon terminating $\Pi^{0,j}_{\text{LBRB}}$ with the output m_j **do**
    $X_i \leftarrow X_i \cup \{(m_j, P_j)\}$
    $W^i_0 \leftarrow W^i_0 \cup \{P_j\}$
when $|W^i_0| = n - t$ **do**
    set W^i_0 as the input for $\Pi^{1,i}_{\text{LBRB}}$
upon terminating $\Pi^{1,j}_{\text{LBRB}}$ with an output $W_0 \subseteq \mathcal{P}$ of size $n-t$ **do**
    once $W_0 \subseteq W^i_0$, add P_j to W^i_1
when $|W^i_1| = n - t$ **do**
    multicast W^i_1
upon receiving a set $W_1 \subseteq \mathcal{P}$ of size $n-t$ for the first time from a party P_j **do**
    once $W_1 \subseteq W^i_1$, add P_j to W^i_2
when $|W^i_0| \geq n-t$, $|W^i_1| \geq n-t$ and $|W^i_2| \geq n-t$ **do**
    output X_i, but keep running the protocol and updating sets (including X_i)

▶ **Theorem 7.** *Π_{Gthr} is a secure Gather protocol with liveness when $t < \frac{n}{3}$.*

Proof.

Consistency and Validity. The consistency and validity of $\Pi^{\text{Live}}_{\text{Gthr}}$ follow from the consistency and validity of $\Pi^{0,1}_{\text{LBRB}}, \dots, \Pi^{0,n}_{\text{LBRB}}$.

Liveness. Since all honest parties broadcast their inputs, every honest P_i terminates $n-t$ input broadcasts, and thus broadcasts W_0^i. All honest parties terminate all honest W_0 set broadcasts, and if some honest P_i broadcasts W_0^i, then, by the global termination of Π_{LBRB}, every honest P_j terminates all input broadcasts terminated by P_i, obtains $W_0^j \supseteq W_0^i$, and adds P_i to W_1^j. This means that every honest P_i adds every honest P_j to W_1^i, and therefore that eventually $|W_1^i| \geq n-t$ holds and P_i multicasts W_1^i. Afterwards, every honest P_j adds P_i to W_2^j after receiving W_1^i from P_i because eventually $W_1^j \supseteq W_1^i$ holds, due to the fact that whenever P_i adds any P_k to W_1^i, eventually P_j adds P_k to W_1^j as well since eventually $W_0^j \supseteq W_0^i$ holds and since P_j terminates P_k's W_0 set broadcast with the same W_0 set as P_i. So, since every honest party adds every honest party to its W_2 set, eventually $|W_2^i| \geq n-t$ holds for every honest P_i. Finally, we conclude that every honest P_i obtains $|W_0^i| \geq n-t$, $|W_1^i| \geq n-t$ and $|W_2^i| \geq n-t$, which leads to P_i outputting X_i.

Common Core. Let $\mathcal{H}$ be the set of the first $n-t$ honest parties P_i who obtain $|W_1^i| = n-t$. For each $P_i \in \mathcal{H}$, the set W_1^i which P_i multicasts contains $n-t$ parties, at least $n-2t$ of which are in $\mathcal{H}$. Hence, $\sum_{P_i \in \mathcal{H}} |W_1^i \cap \mathcal{H}| \geq (n-2t)(n-t)$.

There exists some $P_k \in \mathcal{H}$ included in the multicast W_1 sets of at least $n-2t$ parties in $\mathcal{H}$. For contradiction, suppose otherwise. Then, for each $P_j \in \mathcal{H}$, less than $n-2t$ parties in $\mathcal{H}$ have W_1 sets which include P_j. This implies $\sum_{P_j \in \mathcal{H}} |\{P_i \in \mathcal{H} : P_j \in W_1^i\}| < (n-2t)(n-t)$, which contradicts $\sum_{P_j \in \mathcal{H}} |\{P_i \in \mathcal{H} : P_j \in W_1^i\}| = \sum_{P_i \in \mathcal{H}} |W_1^i \cap \mathcal{H}| \geq (n-2t)(n-t)$.

Suppose some honest P_i outputs from $\Pi_{\mathsf{Gthr}}^{\mathsf{Live}}$. Consider the set W_0^k at the instant when P_k broadcasts it. Since at least $n-2t \geq t+1$ honest parties include P_k in the W_1 sets which they multicast, P_i must have added some honest P_j to W_2^i from whom P_i received some $W_1^j \supseteq \{P_k\}$. This implies $W_1^i \supseteq \{P_k\}$, and $W_1^i \supseteq \{P_k\}$ implies $W_0^k \subseteq W_0^i = \{P_j \in \mathcal{P} : \exists (m, P_j) \in X_i\}$. So, W_0^k is a common core. ◀

C Skipped Proofs

▶ **Theorem 3.** Π_{Gthr} *is a secure binding Gather protocol with strong termination when* $t < \frac{n}{3}$.

Proof. As we did for $\Pi_{\mathsf{k\text{-}slot}}$, we split strong termination into local and global termination.

Consistency and Validity. Suppose some honest P_i includes some (m_j, P_j) in its output set Q_i. This requires $g_i^j \geq \frac{3}{4}$; hence, by the validity of $\Pi_{\mathsf{5\text{-}slot}}$, there exists some (unique, by the consistency of $\Pi_{\mathsf{Gthr}}^{\mathsf{Live}}$) m'_j such that some honest parties have (m'_j, P_j) in their $\Pi_{\mathsf{Gthr}}^{\mathsf{Live}}$ output sets. We show below that $m_j = m'_j$. Note that this gives us consistency because if some P_k has some $(m''_j, P_j) \in Q_k$, then we have $m'_j = m_j = m''_j$. We also get validity: If P_j is honest, then $m'_j = m_j$ must be the input of P_j by the validity of $\Pi_{\mathsf{Gthr}}^{\mathsf{Live}}$.

Let $(s_1, \ldots, s_n) \leftarrow \mathsf{Encode}(m'_j)$. The party P_i obtains m_j via $\mathsf{TryDecode}(M_{j,1}, \ldots, M_{j,n})$, where $M_{j,k}$ is either $\perp$, or the non-$\perp$ j^{th} symbol in the $\texttt{MINE}$ vector P_k sends to P_i. Assume now that if the latter is the case and P_k is honest, then $M_{j,k} = s_k$. This implies that in any $\mathsf{TryDecode}$ attempt the vector $(M_{j,1}, \ldots, M_{j,n})$ contains with respect to m'_j at most t incorrect symbols (from corrupt parties) and at most t missing symbols. Thus, the correctness of $\mathsf{TryDecode}$ ensures that $\mathsf{TryDecode}(M_{j,1}, \ldots, M_{j,n}) \in \{m'_j, \perp\}$ always holds.

It remains to prove the assumption that if $M_{j,k} \neq \perp$ and P_k is honest, then $M_{j,k} = s_k$. If $M_{j,k} \neq \perp$ and P_k is honest, then $M_{j,k}$ is a non-$\perp$ symbol repeated at least $t+1$ times in the j^{th} column of the Y matrix of P_k. Hence, some honest P_q must have sent P_k a message $\langle \texttt{YOURS}, (S_{1,k}, \ldots, S_{n,k}) \rangle$ with $S_{j,k} = M_{j,k}$. The party P_q must have had $(m'_j, P_j) \in X_q$ and computed $(S_{j,1}, \ldots, S_{j,n}) \leftarrow \mathsf{Encode}(m'_j)$. Therefore, we conclude that $M_{j,k} = S_{j,k} = s_k$.

Global Termination. If some honest party terminates, then it has received `READY` from $2t+1$ parties. We show below that for all honest parties to terminate, it suffices for some honest party to receive `READY` from $2t+1$ parties.

If an honest party has received `READY` from $2t+1$ parties, then at least $t+1$ honest parties have multicast `READY`. Every honest party becomes able to multicast `READY` by receiving `READY` from $t+1$ parties, and so every honest party receives `READY` from $2t+1$ parties.

The first honest party who multicasts `READY` must have done so because it has received `YOURS` vectors from $2t+1$ parties. Hence, there must exist at least $t+1$ happy parties P_i such that P_i has terminated all $\Pi_{\textsf{5-slot}}$ instances and there exists some $(m_j, P_j) \in X_i$ whenever $g_i^j \geq \frac{1}{4}$. Furthermore, because the happy parties have terminated all $\Pi_{\textsf{5-slot}}$ instances, the strong termination of $\Pi_{\textsf{5-slot}}$ implies that every honest party terminates every $\Pi_{\textsf{5-slot}}$ instance.

Suppose some honest P_i has $g_i^j \geq \frac{2}{4}$ for some j. Then, every happy P_k has $g_k^j \geq \frac{1}{4}$ and thus has some $(m_j, P_j) \in X_k$. This m_j is the same for all happy parties by the consistency of standard BRB. Hence, every happy P_k computes $(s_1, \dots, s_n) \leftarrow \textsf{Encode}(m_j)$ and sends P_i a `YOURS` vector with the j^{th} symbol s_i. The party P_i can thus find s_i to be the non-$\bot$ symbol repeated at least $t+1$ times in the j^{th} column of its matrix Y. This happens eventually for any j where $g_i^j \geq \frac{2}{4}$; hence, every honest party eventually multicasts some `MINE` vector.

Finally, suppose some honest P_i has $g_i^j \geq \frac{3}{4}$ for some j. Then, every honest P_k has $g_k^j \geq \frac{2}{4}$, which means that the j^{th} symbol of the `MINE` vector which P_k multicasts is the non-$\bot$ symbol s_k, which (as we showed while proving consistency) is such that $(s_1, \dots, s_n) = \textsf{Encode}(m_j)$ for some m_j that is consistent for all honest parties. Since P_i can eventually store s_k as $M_{j,k}$, eventually P_i's vector $(M_{j,1}, \dots, M_{j,n})$ contains $n-t$ correct symbols with respect to m_j, $\textsf{TryDecode}(M_{k,1}, \dots, M_{k,n})$ outputs m_j, and P_i inserts (m_j, P_j) to Q_i.

In the end, every honest P_i terminates $\Pi_{\textsf{Gthr}}$ after terminating every $\Pi_{\textsf{5-slot}}$ instance, multicasting `READY` and receiving `READY` from $2t+1$ parties, multicasting some `MINE` vector and inserting some (m_j, P_j) to Q_i whenever $g_i^j \geq \frac{3}{4}$. This gives us global termination.

Local Termination. For contradiction, suppose no honest party terminates. Then, every honest P_i eventually outputs some Z_i from $\Pi_{\textsf{Gthr}}^{\textsf{Live}}$ and thus provides inputs to all $\Pi_{\textsf{5-slot}}$ instances. Hence, the honest parties terminate all $\Pi_{\textsf{5-slot}}$ instances. For any honest P_i, if for some j it is the case that $g_i^j \geq \frac{1}{4}$, then by the validity of $\Pi_{\textsf{5-slot}}$ some honest party must have terminated the standard BRB instance in $\Pi_{\textsf{Gthr}}^{\textsf{Live}}$ with which P_j broadcasts its input, and hence P_i terminates this BRB instance and inserts some (m_j, P_j) to X_i as well. So, every honest P_i eventually becomes happy and sends out `YOURS` vectors. Consequently, every honest party receives `YOURS` vectors from $2t+1$ parties and multicasts `READY`, and then every honest party receives `READY` from $2t+1$ parties. As we showed to prove global termination, for all honest parties to terminate it suffices for some honest party to receive `READY` from $2t+1$ parties. So, we reach the contradictory conclusion that all honest parties terminate. ◀

▶ **Theorem 4.** $\Pi_{\textsf{Any}}$ *is secure when* $4t + q < n$.

Proof.

Consistency. We show that there exist no distinct $v \neq \bot$ and $v' \neq \bot$ such that some honest parties multicast $\langle \texttt{READY}, v \rangle$ while others multicast $\langle \texttt{READY}, v' \rangle$. As in $\Pi_{\textsf{Bracha}}$ and $\Pi_{\textsf{Quit}}$, the first honest party who multicasts $\langle \texttt{READY}, v \rangle$ for any particular $v \neq \bot$ must have done so upon receiving enough `ECHO` messages to form an `ECHO` quorum on v. For the sake of contradiction, let P_i and P_j be the first honest parties who respectively multicast $\langle \texttt{READY}, v_i \rangle$ and $\langle \texttt{READY}, v_j \rangle$ for some distinct $v_i \neq \bot$ and $v_j \neq \bot$, let $\mathcal{H}$ be a set of $n-t$ honest parties, and let f be the number of parties in $\mathcal{H}$ who multicast $\langle \texttt{ECHO}, \bot \rangle$. For each $k \in \{i, j\}$, let

- b_k be the number of $\langle \texttt{ECHO}, v_k \rangle$ messages from parties in $\mathcal{H}$ counted in P_k's quorum,
- c_k be the number of $\langle \texttt{ECHO}, v_k \rangle$ messages from parties outside $\mathcal{H}$ counted in P_k's quorum,
- d_k be the number of $\langle \texttt{ECHO}, \perp \rangle$ messages from parties in $\mathcal{H}$ counted in P_k's quorum,
- e_k be the number of $\langle \texttt{ECHO}, \perp \rangle$ messages from parties outside $\mathcal{H}$ counted in P_k's quorum.

For each $k \in \{i, j\}$, we have $d_k \leq f$ and $c_k + e_k \leq t$. For the quorum of P_k to be of sufficient size, we must have $b_k + c_k > \max(t, \frac{n+t-d_k-e_k}{2})$, which implies $b_k > \frac{n+t-d_k-e_k}{2} - c_k \geq \frac{n+t-d_k-2(c_k+e_k)}{2} \geq \frac{n-t-f}{2}$. However, $\mathcal{H}$ consists of f honest parties who multicast $\langle \texttt{ECHO}, \perp \rangle$ and thus contribute to neither b_i nor b_j, and $n - t - f$ honest parties who contribute to at most one of b_i and b_j since an honest party cannot send $\langle \texttt{ECHO}, v_i \rangle$ to P_i and $\langle \texttt{ECHO}, v_j \rangle$ to P_j. This gives us $b_i + b_j \leq n - t - f$, which contradicts $b_i > \frac{n-t-f}{2} \wedge b_j > \frac{n-t-f}{2}$. Finally, consistency follows because in order to output $v \neq \perp$, an honest party must receive $\langle \texttt{READY}, v \rangle$ from $t + 1$ parties, at least one of which is honest.

Validity. For some honest party to output $v \neq \perp$, there must exist a first honest P_i who multicasts $\langle \texttt{READY}, v \rangle$. To do so, P_i must receive $\langle \texttt{ECHO}, v \rangle$ from at least $t + 1$ parties, at least one of which is honest. This can happen only after the sender P^* sends $\langle \texttt{INIT}, v \rangle$ to some honest party. Now suppose P^* is honest. If P^* acquires an input v^*, then it multicasts $\langle \texttt{INIT}, v^* \rangle$, and this makes v^* the unique possible non-$\perp$ output. If P^* quits before acquiring an input, then it multicasts $\langle \texttt{INIT}, \top \rangle$, and thus $\top$ becomes the unique possible non-$\perp$ output.

Local Termination. For contradiction, suppose that the sender P^* is honest, that it either acquires an input or quits before doing so, that not all honest parties quit, and that despite all the preceding no honest party terminates. We separately consider the case where at least $t + q + 1$ honest parties quit, and the case where at most $t + q$ honest parties quit. Note that an honest party does not quit without multicasting a READY message.

- Suppose at least $t + q + 1$ honest parties quit. Because an honest party multicasts QUIT when it quits, every honest party either quits, or eventually receives QUIT from $t + q + 1$ parties and thus becomes able to multicast a READY message.
- Suppose $f \leq t + q$ honest parties quit. The sender P^* eventually multicasts $\langle \texttt{INIT}, v^* \rangle$, where v^* is either its input or $\top$. Then, at least $n - t - f$ honest parties multicast $\langle \texttt{ECHO}, v^* \rangle$. Since we have $n - t - f > (\frac{n}{2} + \frac{4t+q}{2}) - t - (\frac{f}{2} + \frac{t+q}{2}) = \frac{n+t-f}{2} \geq \frac{n-q}{2} > 2t$, every honest party either quits, or eventually receives $\langle \texttt{ECHO}, v^* \rangle$ from sufficiently many parties to multicast a READY message.

In both cases, every honest party multicasts a READY message, which means that every honest party either quits, or terminates after receiving READY messages from $n - t$ parties.

Global Termination. If an honest party terminates, then it has received READY messages from $n - t$ parties, at least $n - 2t \geq 2t + q + 1$ of which are honest. Since there is some $v \neq \perp$ such that every honest READY message is on either v or $\perp$, either there are at least $t + 1$ honest parties who multicast $\langle \texttt{READY}, v \rangle$, or there are at least $t + q + 1$ honest parties who multicast $\langle \texttt{READY}, \perp \rangle$. In either case, every honest party can multicast a READY message either before quitting, or after receiving sufficiently many honest READY messages. So, every honest party either quits, or terminates after receiving READY messages from $n - t$ parties.

Robustness. Suppose at most q honest parties have quit when some honest P_i terminates for the first time. Then, an honest P_j can multicast $\langle \texttt{READY}, \perp \rangle$ before P_i terminates only if it quits, as P_j cannot receive QUIT or $\langle \texttt{READY}, \perp \rangle$ from $t + q + 1$ parties before P_i terminates. So, P_i terminates with at most t corrupt READY messages, at most q honest $\langle \texttt{READY}, \perp \rangle$ messages, and at least $n - 2t - q$ honest READY messages not on $\perp$. Since for some $v \neq \perp$ every honest READY message is on either v or $\perp$, there are at least $n - 2t - q$ honest parties who multicast $\langle \texttt{READY}, v \rangle$ for some v. Hence, any honest P_j (including P_i) can terminate only after receiving at least $n - 3t - q \geq t + 1$ honest $\langle \texttt{READY}, v \rangle$ messages and setting $y_j \leftarrow v$. ◀