

Simple Byzantine Lattice Agreement in $O(\frac{\log f}{\log \log f})$ Rounds

YUVAL EFRON , Institute for Advanced Study, USA

JOVAN KOMATOVIC, Category Labs, USA

Lattice agreement is a relaxed version of the standard consensus problem: correct processes need not decide the same value, but their decisions must be “comparable”. Namely, every process proposes a value from a join semi-lattice, and correct processes decide values that (1) lie on a single chain, (2) include their own proposals, and (3) include nothing beyond what was proposed. Lattice agreement has important practical applications, as it underpins atomic snapshot objects and replicated state machines with commutative operations. Yet, the best known protocols decide in $O(\log f)$ rounds, where f is an upper bound on the number of faulty processes.

In this paper, we break the logarithmic barrier. Namely, we present Join-IT, a synchronous Byzantine lattice agreement protocol that tolerates $f < n/3$ Byzantine processes and decides in $O(\frac{\log f}{\log \log f})$ rounds. Join-IT uses no cryptography whatsoever, and its guarantees thus hold against a computationally unbounded adversary. Perhaps surprisingly, Join-IT is also remarkably simple. It sequentially composes two classical primitives: gradecast, which takes three rounds, and approximate agreement, which, at the precision Join-IT requires, takes $O(\frac{\log f}{\log \log f})$ rounds. A single local step finally turns the output of approximate agreement into comparable decisions, at no additional cost in rounds.

1 Introduction

Lattice agreement [1] is a fundamental problem of distributed computing. It was introduced by Attiya *et al.* as the cornerstone of their implementation of atomic snapshot objects [1], and it has since been used to implement linearizable replicated state machines whose updates commute [6, 13]. Concretely, the lattice agreement problem is posed in a system of n processes, out of which $f < n$ can be faulty, and the rest correct. Each correct process proposes a value from a join semi-lattice, and it eventually decides a value from the same lattice. In this paper, we consider *Byzantine* lattice agreement, where faulty processes can behave arbitrarily. Three properties characterize the problem: (1) *comparability*, requiring the decisions of correct processes to be pairwise comparable, (2) *downward validity*, requiring every correct process to decide a value at least as large as its own proposal, and (3) *upward validity*, preventing correct processes from deciding much beyond what was proposed, as each faulty process can contribute at most a single value of its own.

The logarithmic barrier. The original protocol of Attiya *et al.* [1], designed for crash failures in synchrony, decides in $O(\log n)$ rounds, and Zheng *et al.* [14] later sharpened this to $O(\log f)$. Byzantine failures were considered only recently, and, in synchrony, Zheng and Garg [12] and, concurrently, Di Luna *et al.* [3] decide in $O(\log f)$ rounds. The former does so without signatures and with the optimal resilience of $f < n/3$, whereas the latter tolerates $f < n/3$ with signatures and $f < n/4$ without them. The logarithm is therefore the best known round complexity of lattice agreement, with crash or Byzantine failures, and with or without signatures. Yet, no non-trivial worst-case lower bound on the round complexity of synchronous lattice agreement is known. This leaves a fundamental question open:

Is the logarithmic barrier inherent to (Byzantine) lattice agreement?

Contributions. We answer the above question negatively. Namely, we present Join-IT, the first (Byzantine) lattice agreement protocol to decide in a sublogarithmic number of rounds: Join-IT decides in $O(\frac{\log f}{\log \log f})$ synchronous rounds. Furthermore, Join-IT uses no cryptography whatsoever

Table 1. Lattice agreement protocols. Rounds are worst-case upper bounds for one-shot lattice agreement. Here, $h(X)$ denotes the height of the lattice. Asynchronous round complexities count asynchronous rounds (message delays and round-trips differ only by constant factors in these asymptotic bounds). In the early-stopping bounds, $f_a \leq f$ denotes the actual number of faulty processes.

Protocol	Timing	Failures	Rounds	Resilience	Signatures
Attiya <i>et al.</i> [1]	sync	crash	$O(\log n)$	$f < n$	–
Mavronicolas [9]	sync	crash	$O(\min\{h(X), \sqrt{f_a}\})$	$f < n$	–
Zheng <i>et al.</i> [14]	sync	crash	$O(\log f)$	$f < n$	–
Zheng <i>et al.</i> [14]	sync	crash	$O(\min\{\log^2 h(X), \log^2 f\})$	$f < n$	–
Faleiro <i>et al.</i> [6]	async	crash	$O(n)$	$f < n/2$	–
Zheng <i>et al.</i> [14]	async	crash	$O(\min\{h(X), f\})$	$f < n/2$	–
Zheng <i>et al.</i> [13]	async	crash	$O(\log f)$	$f < n/2$	–
Zheng and Garg [12]	sync	Byzantine	$O(\min\{h(X), \sqrt{f_a}\})$	$f < n/3$	no
Zheng and Garg [12]	sync	Byzantine	$O(\log n)$	$f < n/3$	no
Zheng and Garg [12]	sync	Byzantine	$O(\log f)$	$f < n/3$	no
Zheng and Garg [12]	sync	Byzantine	$O(\min\{h(X), \sqrt{f_a}\})$	$f < n/2$	yes
Zheng and Garg [12]	sync	Byzantine	$O(\log n)$	$f < n/2$	yes
Zheng and Garg [12]	sync	Byzantine	$O(\log f)$	$f < n/2$	yes
Di Luna <i>et al.</i> [3]	sync	Byzantine	$O(\log f)$	$f < n/4$	no
Di Luna <i>et al.</i> [3]	sync	Byzantine	$O(\log f)$	$f < n/3$	yes
Di Luna <i>et al.</i> [4]	async	Byzantine	$O(f)$	$f < n/3$	no
Zheng and Garg [11]	async	Byzantine	$O(\log f)$	$f < n/5$	no
Zheng and Garg [11]	async	Byzantine	$O(\log f)$	$f < n/3$	yes
Join-IT (this paper)	sync	Byzantine	$O(\frac{\log f}{\log \log f})$	$f < n/3$	no

(and is thus secure against a computationally unbounded adversary), and it tolerates $f < n/3$ Byzantine processes, which is the best one can hope for without signatures [3]. Table 1 summarizes how Join-IT compares to prior lattice agreement protocols.

Perhaps surprisingly, the design of Join-IT is simple and modular: Join-IT relies on two well-known primitives, gradecast [2, 7] and approximate agreement [2, 5]. First, processes disseminate their proposals via gradecast, which costs three rounds. Then, for every process, processes run an instance of approximate agreement on whether to include its value, and all these instances run in parallel in $O(\frac{\log f}{\log \log f})$ rounds. Finally, every process applies a purely local rule, which converts the nearly identical outputs of approximate agreement into comparable decisions without exchanging a single message.

Technical Overview

We now explain how Join-IT works, and why. Let us first reformulate the problem in a way that makes approximate agreement a natural tool, and then present the single idea that turns approximate agreement into lattice agreement.

Reformulating the problem. As a decision in lattice agreement is a join of values, it is natural to let every correct process decide the join of the values of a *set of processes*. If correct processes agree on the value of every process they keep, and the sets of processes they keep are nested (related by inclusion), then their decisions are comparable. The two validity properties are then equally simple to satisfy. Downward validity only requires every correct process to keep itself, which Join-IT ensures by letting every correct process keep *all* correct processes. Upward validity, in turn, follows

	correct processes: confirmed by everyone					contested	
	p_1	p_2	p_3	p_4	p_5	p_6	p_7
p_1	●	●	●	●	●	●	○
p_2	●	●	●	●	●	○	●
p_3	●	●	●	●	●	●	○
p_4	●	●	●	●	●	●	○
p_5	●	●	●	●	●	○	○

Fig. 1. The confirmations after gradecast, in an example with $n = 7$ and $f = 2$, where p_6 and p_7 are faulty. Every row shows which processes a correct process confirms (●), i.e., delivers with grade 2, and which ones it does not (○). As p_6 and p_7 are confirmed by some correct process, every correct process knows their values.

from the fact that every process contributes a single value, and faulty processes thus contribute at most one value each.

Consistent values via gradecast. The first obstacle is that a faulty process may disseminate different values to different correct processes. Join-IT neutralizes this obstacle with gradecast [2, 7], in which every correct process delivers the value of a sender together with a *grade* in $\{0, 1, 2\}$. Intuitively, gradecast ensures that the value of a correct sender reaches every correct process with grade 2 (*validity*), that no two correct processes deliver different values with positive grades (*consistency*), and that the grades delivered by any two correct processes differ by at most 1 (*graded agreement*). Henceforth, we say that a correct process *confirms* a sender if it delivers the sender’s value with grade 2. Therefore, once every process has gradecast its proposal, every correct process confirms every correct process. Crucially, if some correct process confirms a (possibly faulty) process, then every correct process delivers the value of that process with a positive grade, and thus knows this very value. In other words, correct processes may differ in *which* processes they confirm, as illustrated in Figure 1, but the value of any process confirmed by some correct process is known to all. Correct processes are confirmed by everyone, and only faulty processes are contested. What remains is to resolve this difference into nested sets of processes (without ever excluding a correct process).

Membership scores via approximate agreement. To this end, correct processes first agree, approximately, on how much every process “deserves to be kept”. Namely, for *every* process, the processes run an instance of approximate agreement [5], in which a process proposes 1 if it confirms that process and 0 otherwise. Intuitively, approximate agreement ensures that every correct process decides a number within the range of the numbers proposed by correct processes (*validity*), and that the numbers decided by any two correct processes are at most ε apart (ε -*agreement*). We call the number that a correct process decides for a process the *score* of that process (at that correct process). Note first that every correct process has score exactly 1 at every correct process: every correct process confirms it and thus proposes 1, and validity leaves no other choice. Similarly, a process that no correct process confirms has score exactly 0 everywhere. Uncertainty is thus confined to the contested processes, i.e., the faulty processes that some correct processes confirm (and propose 1 for) while others do not (and propose 0 for). The scores of a contested process may lie anywhere between 0 and 1, but, by ε -agreement, its scores at any two correct processes differ by

at most ε . It remains to turn nearly identical scores into nested sets, which is where the main idea of Join-IT comes in.

Key idea. Join-IT turns the scores of a correct process into the set of processes it keeps (and whose values it joins into its decision) with a simple local rule, which we call the *cut*. The cut builds on an elementary observation about sizes. On the one hand, Join-IT ensures that the set kept by a correct process contains every correct process, and thus at least $n - f$ processes. On the other hand, it contains at most n processes. Its size can therefore take only $f + 1$ values, namely $n - f, n - f + 1, \dots, n$, which we call *targets*. The cut assigns a score threshold to every target, and the smaller the target, the higher its threshold. Concretely, the largest target, n , gets threshold $\frac{1}{f+1}$, every other target gets a threshold that is $\frac{1}{f+1}$ higher than that of the next target, and the smallest target, $n - f$, thus gets threshold 1. Accordingly, Join-IT runs every instance of approximate agreement with parameter $\varepsilon = \frac{1}{f+1}$. A correct process then examines the targets *from the smallest to the largest*, and it stops at the first target k for which *exactly* k processes have a score at or above the threshold of k . The processes it keeps are precisely these k processes, i.e., it decides the join of their values. Note that, as long as the process has not stopped, more processes clear the threshold of the current target than the target itself. This holds at the smallest target, $n - f$, as the at least $n - f$ correct processes have score 1, which clears every threshold (unless exactly $n - f$ processes clear it, in which case the process stops right away). Moreover, it is preserved as the target grows by one, since the number of processes clearing the threshold can only grow. As at most n processes clear the threshold of the largest target n , the number of processes must therefore meet the target at some point, so the procedure always stops. Moreover, every process it keeps has a positive score, and was thus confirmed by some correct process, so its value is known to all.

Why are the cuts of correct processes nested? The main reason is that the ε -gap between consecutive thresholds is as wide as any disagreement between correct processes. Hence, a process that clears the threshold of some target at one correct process clears the threshold of the next, lower target at *every* correct process. Now, consider any two correct processes p_i and p_j that stop at targets $k \leq l$, respectively, and any process kept by p_i , i.e., any process whose score at p_i clears the threshold of k . We show that p_j keeps this process as well.

- *Different targets* ($k < l$). By the above, the process clears the threshold of $k + 1$ at p_j . As $l \geq k + 1$ and thresholds decrease as targets grow, it also clears the threshold of l , so p_j keeps it.
- *Same target* ($k = l$). This is the subtle case, as p_j keeps only the processes that clear the threshold of k , while the above only guarantees that the process clears the lower threshold of $k + 1$ at p_j . If $k = l = n - f$, both cuts equal C , as each contains every correct process and only $n - f$ processes in total. Otherwise, the stopping rule comes to the rescue: at p_i , the process in fact clears the *higher* threshold of $k - 1$, and hence the threshold of k at p_j . Indeed, as p_i did not stop at target $k - 1$, more than $k - 1$ (i.e., at least k) processes clear the threshold of $k - 1$ at p_i and these must be the very k processes that clear the lower threshold of k , among which is our process.

In both cases, the cut of p_i is contained in the cut of p_j , and, as correct processes agree on the value of every kept process, comparability follows.

Round complexity. Observe that the cut dictates the precision that approximate agreement must achieve. As there are only $f + 1$ targets, $f + 1$ thresholds spaced ε apart must fit into $(0, 1]$, which yields $\varepsilon = \frac{1}{f+1}$. With this precision, the approximate agreement algorithm of Ben-Or *et al.* [2] takes $O(\frac{\log f}{\log \log f})$ rounds [8]. Altogether, Join-IT decides after three rounds of gradecast and $O(\frac{\log f}{\log \log f})$

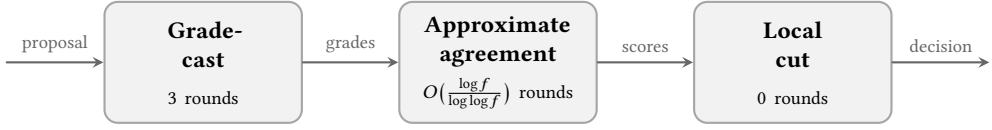


Fig. 2. Join-IT at a glance.

rounds of approximate agreement, while the cut itself costs no communication at all, as summarized in Figure 2.

Roadmap

We define the system model and the Byzantine lattice agreement problem in §2. In §3, we present Join-IT and prove its correctness and round complexity. Finally, we discuss related work in §4.

2 Preliminaries

This section first defines the system model (§2.1), and then formally states the Byzantine lattice agreement problem (§2.2).

2.1 System Model

Processes & the adversary. We consider a static system $\{p_1, p_2, \dots, p_n\}$ of $n \geq 3f + 1$ message-passing processes, for some integer $f > 0$. Every process is modeled as a deterministic state machine. At any point during an execution, an *adversary* may corrupt processes, up to f of them in total. Once corrupted, a process is Byzantine: it may deviate from the protocol in an arbitrary manner. Non-corrupted processes, in contrast, faithfully follow their state machines. We call processes that are corrupted at some point of the execution *faulty* and processes that are never corrupted during the execution *correct*, and we denote by C the set of correct processes. Moreover, the adversary is assumed to be (1) *full-information*, i.e., it observes the internal state of every process, and (2) *computationally unbounded*.

Communication network. Processes communicate over an *authenticated* point-to-point network. Moreover, we assume that the communication network is *reliable*: if a correct process sends a message to a correct process, the message is eventually received.

Synchrony. We assume the standard synchronous environment in which the computation unfolds in synchronous rounds. In each round $1, 2, \dots \in \mathbb{N}$, each process (1) performs (deterministic) local computation, (2) sends (possibly different) messages to (a subset of) the other processes, and (3) receives the messages sent to it at the end of the round.

2.2 Problem Definition

Lattices. Let $(X, \leq, \sqcup)$ be a *join semi-lattice*: a (possibly infinite) set X equipped with a partial order $\leq$ and a *join* operation $\sqcup$ that maps every non-empty finite subset $V \subseteq X$ to its least upper bound $\sqcup V$. Two values $u, v \in X$ are *comparable* if and only if $u \leq v$ or $v \leq u$. As is customary, we simply say *lattice* instead of join semi-lattice.

Byzantine lattice agreement. The primitive exposes the following interface:

- input $\text{propose}(x_i \in X)$: process p_i proposes value x_i .
- output $\text{decide}(y_i \in X)$: process p_i decides value y_i .

Every correct process p_i holds its proposal x_i at the common start of the execution, and it invokes $\text{propose}(x_i)$ at the beginning of round 1. The following properties are guaranteed:

- *Comparability*: For every two correct processes p_i and p_j , either $y_i \leq y_j$ or $y_j \leq y_i$.
- *Downward validity*: For every correct process p_i , $x_i \leq y_i$.
- *Upward validity*: There exists a set $B \subseteq X$ with $|B| \leq n - |C|$ such that

$$\bigsqcup \{y_i \mid p_i \in C\} \leq \bigsqcup (\{x_i \mid p_i \in C\} \cup B).$$

- *Termination*: Every correct process eventually decides.

Intuitively, comparability is the lattice counterpart of agreement: correct processes need not decide the same value, but their decisions must be comparable. Downward validity forbids a correct process from “losing” its own proposal along the way. Upward validity, in turn, bounds how far the faulty processes can inflate the decisions. Namely, across the decisions of all correct processes together, the faulty processes may “smuggle in” at most one value each.

3 Join-IT

This section presents Join-IT.

3.1 Building Blocks

Join-IT is assembled from two classical building blocks: *gradecast* and *approximate agreement*.

Gradecast. Gradecast, introduced by Feldman and Micali [7], weakens Byzantine broadcast by attaching a *grade* to the delivered value, i.e., a measure of how confident a recipient may be that every other correct recipient has obtained the same value. We use the three-round gradecast as presented by Ben-Or *et al.* [2]. Each instance of gradecast has a designated *sender* p_s , and it is parameterized by a domain $\mathcal{M}$ of admissible messages. It exposes the following interface:

- input $\text{gradecast}(m_s \in \mathcal{M})$: the sender p_s gradecasts message m_s .
- output $\text{deliver}(v_i \in \mathcal{M} \cup \{\perp\}, g_i \in \{0, 1, 2\})$: process p_i delivers value v_i with grade g_i , where $v_i = \perp$ if and only if $g_i = 0$.

Gradecast guarantees the following properties:

- *Validity*: If p_s is correct, then $v_i = m_s$ and $g_i = 2$ for every correct process p_i .
- *Consistency*: For every two correct processes p_i and p_j with $g_i, g_j \geq 1$, $v_i = v_j$.
- *Graded agreement*: For every two correct processes p_i and p_j , $|g_i - g_j| \leq 1$.
- *Termination*: Every correct process delivers at the end of the third round of the instance.

Approximate agreement. Approximate agreement, introduced by Dolev *et al.* [5], relaxes agreement on a real number. Namely, the decided numbers need not be equal, but merely within some precision ε of each other, and they must stay within the range of the correct proposals. Each instance of approximate agreement is parameterized by a precision $\varepsilon \in (0, 1]$, and it exposes the following interface to every process p_i :

- input $\text{propose}(q_i \in [0, 1])$: process p_i proposes number q_i .
- output $\text{decide}(z_i \in [0, 1])$: process p_i decides number z_i .

Approximate agreement guarantees the following properties:

- *Validity*: For every correct process p_i , $\min\{q_j \mid p_j \in C\} \leq z_i \leq \max\{q_j \mid p_j \in C\}$.
- ε -*Agreement*: For every two correct processes p_i and p_j , $|z_i - z_j| \leq \varepsilon$.
- *Termination*: Every correct process decides within R rounds after proposing, for some publicly known $R \in O\left(\frac{\log(1/\varepsilon)}{\log \log(1/\varepsilon)}\right)$.

We use the approximate agreement algorithm of Ben-Or *et al.* [2], which, by the analysis of Fuchs *et al.* [8, Theorem 2] for arbitrary precision, guarantees these properties as $n \geq 3f + 1$ and all proposals lie in $[0, 1]$.

Algorithm 1 Join-IT: Pseudocode (for process p_i)

```

1: Constants:
2:   Integer  $h = f + 1$  ▷ the thresholds of cut are multiples of  $\frac{1}{h}$ 
3:   Real  $\varepsilon = \frac{1}{h}$  ▷ the gap between consecutive thresholds of cut
4: Notation:
5:   - A score map  $z \in \text{ScoreMap}$  assigns a score  $z[s] \in [0, 1]$  to every process  $p_s$ 
6: Uses:
7:   Gradecast with domain  $X$ , instances  $\Pi_{gc}[s]$  with sender  $p_s$ , for every process  $p_s$  ▷ cf. §3.1
8:   Approximate agreement with precision  $\varepsilon$ , instances  $\Pi_{aa}[s]$ , for every process  $p_s$  ▷ cf. §3.1
9: Local variables:
10:   $X \cup \{\perp\}$   $values_i[s] \leftarrow \perp$ , for every process  $p_s$  ▷ the value of  $p_s$  delivered by gradecast
11:  Integer  $grades_i[s] \leftarrow 0$ , for every process  $p_s$  ▷ its grade
12:  ScoreMap  $scores_i$  ▷ the scores decided by the approximate agreement instances
13: Functions:
14:   - cut(ScoreMap  $z$ ): ▷ turns scores into a set of processes
15:     for  $k = n - f, n - f + 1, \dots, n$ : ▷ try the targets in increasing order
16:       if  $|\{p_s \mid z[s] \geq \frac{n-k+1}{h}\}| = k$ : return  $\{p_s \mid z[s] \geq \frac{n-k+1}{h}\}$  ▷ exactly  $k$  processes
17: upon propose( $x_i \in X$ ) do
18:   invoke  $\Pi_{gc}[i].gradecast(x_i)$  ▷ all  $n$  instances run in parallel, in rounds 1–3
19: upon  $\Pi_{gc}[s].deliver(x, g)$  for some process  $p_s$  do
20:    $values_i[s] \leftarrow x$ ;  $grades_i[s] \leftarrow g$  ▷ record the value and the grade
21: upon every instance  $\Pi_{gc}[s]$  having delivered do ▷ end of round 3
22:   for each process  $p_s$  do ▷ all instances run in parallel, in rounds 4 to  $R + 3$ 
23:     if  $grades_i[s] = 2$ : invoke  $\Pi_{aa}[s].propose(1)$  ▷  $p_i$  confirms  $p_s$ 
24:     else: invoke  $\Pi_{aa}[s].propose(0)$  ▷  $p_i$  does not confirm  $p_s$ 
25:   end for
26: upon  $\Pi_{aa}[s].decide(q \in [0, 1])$  for some process  $p_s$  do
27:    $scores_i[s] \leftarrow q$  ▷ record the score of  $p_s$ 
28: upon every instance  $\Pi_{aa}[s]$  having decided do ▷ by the end of round  $R + 3$ 
29:   trigger decide( $\sqcup \{values_i[s] \mid p_s \in \text{cut}(scores_i)\}$ ) ▷ decide the join of the kept values

```

3.2 Pseudocode

Algorithm 1 gives the pseudocode of Join-IT, written for process p_i . Join-IT unfolds in three steps. First, every process gradecasts its proposal (line 18), and p_i records, for every process p_s , the value and the grade it delivers from p_s (line 20). Every correct process participates in all n instances of gradecast, which are scheduled in rounds 1 to 3, including those whose sender stays silent. Second, for every process p_s , the processes run an instance of approximate agreement on whether to include the value of p_s : p_i proposes 1 if it delivered the value of p_s with grade 2, and 0 otherwise (lines 23 and 24). Third, p_i locally cuts the resulting scores (line 27) into a set of processes (lines 15 and 16), and it decides the join of their values (line 29).

3.3 Proof of Correctness & Round Complexity

This subsection proves the main result of the paper.

THEOREM 1 (CORRECTNESS). *Join-IT (Algorithm 1) solves the Byzantine lattice agreement problem, while ensuring that every correct process decides within $O(\frac{\log f}{\log \log f})$ rounds.*

Notation. For every correct process p_i and every process p_s , we denote by $v_i[s]$ and $g_i[s]$ the value and the grade that p_i delivers from $\Pi_{gc}[s]$ (line 20). (By termination of gradecast, all instances $\Pi_{gc}[\cdot]$, which are invoked in round 1, deliver at the end of round 3, so these are well defined.) We

say that p_i confirms p_s if $g_i[s] = 2$. Similarly, we denote by S_i the score map that p_i holds at the end of round $R + 3$ (the value of $scores_i$ at that point), which assigns a score to every process. Lastly, we denote by $V_i = \text{cut}(S_i)$ the set of processes that p_i “keeps” (line 29), and we let

$$K = \{p_s \mid \text{some correct process confirms } p_s\}$$

denote the set of processes confirmed by some correct process. Intuitively, the correct processes must be kept by every correct process, whereas K gathers every process that any correct process could possibly keep.

Intermediate claims. The first claim shows that gradecast pins K down, and that all correct processes agree on the value of every process in K .

CLAIM 1. *The following holds:*

- (1) *Every correct process confirms every correct process p_s , and delivers its proposal x_s from it. In particular, $C \subseteq K$.*
- (2) *For every process $p_s \in K$, all correct processes deliver the same value $u_s \neq \perp$ from $\Pi_{gc}[s]$, where $u_s = x_s$ if p_s is correct.*

PROOF. Consider any correct process p_s . As p_s gradecasts x_s (line 18), validity of $\Pi_{gc}[s]$ ensures that $v_i[s] = x_s$ and $g_i[s] = 2$ for every correct process p_i , which proves part (1).

Consider now any process $p_s \in K$, and let p_j be a correct process that confirms p_s , i.e., $g_j[s] = 2$. By graded agreement of $\Pi_{gc}[s]$, every correct process p_i has $g_i[s] \geq 1$, and thus $v_i[s] \neq \perp$. By consistency of $\Pi_{gc}[s]$, all these values are identical, and, if p_s is correct, they equal x_s by part (1), which proves part (2). $\square$

Next, we show that correct processes score every correct process with 1 and every process outside K with 0, and that, for every process, their scores differ by at most ε .

CLAIM 2. *For every two correct processes p_i and p_j and every process p_s , the following holds:*

- (1) *If $p_s \in C$, then $S_i[s] = 1$.*
- (2) *If $p_s \notin K$, then $S_i[s] = 0$.*
- (3) *$|S_i[s] - S_j[s]| \leq \varepsilon$.*

PROOF. Fix a process p_s . If $p_s \in C$, then every correct process confirms p_s by part (1) of Claim 1 and thus proposes 1 to $\Pi_{aa}[s]$ (line 23), and validity of $\Pi_{aa}[s]$ yields $S_i[s] = 1$, which proves part (1). If $p_s \notin K$, then no correct process confirms p_s , every correct process proposes 0 (line 24), and validity of $\Pi_{aa}[s]$ yields $S_i[s] = 0$, which proves part (2). Finally, part (3) is exactly ε -agreement of $\Pi_{aa}[s]$. $\square$

We say that a correct process p_i selects target k if $\text{cut}(S_i)$ returns in the iteration k of its loop (line 16). For every target $k \in \{n - f, \dots, n\}$, we call $\frac{n-k+1}{h}$ the *threshold* of target k , and we let $c_i(k) = |\{p_s \mid S_i[s] \geq \frac{n-k+1}{h}\}|$ denote the number of processes that clear it at p_i . Thus, p_i selects target k if and only if k is the smallest target with $c_i(k) = k$, in which case $V_i = \{p_s \mid S_i[s] \geq \frac{n-k+1}{h}\}$. Note that c_i is non-decreasing, as thresholds decrease as targets grow.

The crucial observations. We first show that every correct process selects a target, and that its cut contains C , lies within K , contains exactly as many processes as its target, and clears the threshold of the previous target.

CLAIM 3. *Every correct process p_i selects some target k , and the following holds:*

- (1) $C \subseteq V_i \subseteq K$.
- (2) $|V_i| = k$.
- (3) If $k = n - f$, then $V_i = C$.

(4) If $k > n - f$, then every process $p_s \in V_i$ satisfies $\mathcal{S}_i[s] \geq \frac{n-k+2}{h}$, i.e., it clears the threshold of target $k - 1$.

PROOF. We first show that p_i selects some target. Namely, we show that, as long as p_i has not selected a target, more processes clear the threshold of the current target than the target itself. At the smallest target, $n - f$, every correct process has score 1 at p_i (part (1) of Claim 2), which clears the threshold $\frac{f+1}{h} = 1$, and thus $c_i(n - f) \geq |C| \geq n - f$. Hence, if p_i does not select target $n - f$, then $c_i(n - f) > n - f$. Moreover, if $c_i(k - 1) > k - 1$ for some target $k > n - f$, then $c_i(k) \geq c_i(k - 1) \geq k$, as c_i is non-decreasing. Again, if p_i does not select target k , then $c_i(k) > k$. As $c_i(n) \leq n$ (there are only n processes), p_i thus selects target n at the latest (line 15).

Let k be the target selected by p_i , so that $V_i = \{p_s \mid \mathcal{S}_i[s] \geq \frac{n-k+1}{h}\}$. As $k \geq n - f$, this threshold is at most $\frac{f+1}{h} = 1$, so $C \subseteq V_i$ by part (1) of Claim 2. As this threshold is at least $\frac{1}{h} > 0$, we have $V_i \subseteq K$ by part (2) of Claim 2, which proves part (1). Moreover, $|V_i| = c_i(k) = k$, which proves part (2). If $k = n - f$, then $C \subseteq V_i$ and $|V_i| = n - f \leq |C|$ give $V_i = C$, which proves part (3).

Finally, let $k > n - f$. As p_i did not select target $k - 1$, we showed above that $c_i(k - 1) > k - 1$, i.e., at least k processes clear the threshold of target $k - 1$. Every such process also clears the lower threshold of target k , which exactly k processes clear, namely the processes of V_i . Hence, every process of V_i clears the threshold of target $k - 1$, which proves part (4). $\square$

Next, we show that the cuts of correct processes are nested: a correct process that selects a smaller (or equal) target obtains a smaller (or equal) cut.

CLAIM 4. If a correct process p_i selects target k and a correct process p_j selects target $l \geq k$, then $V_i \subseteq V_j$.

PROOF. If $k = n - f$, then $V_i = C \subseteq V_j$ by part (3) of Claim 3 and part (1) of Claim 3. Otherwise, consider any process $p_s \in V_i$. By part (4) of Claim 3, we have $\mathcal{S}_i[s] \geq \frac{n-k+2}{h}$. By part (3) of Claim 2, we thus have $\mathcal{S}_j[s] \geq \mathcal{S}_i[s] - \varepsilon \geq \frac{n-k+2}{h} - \frac{1}{h} = \frac{n-k+1}{h} \geq \frac{n-l+1}{h}$, as $l \geq k$. Hence, $p_s \in V_j$. $\square$

Finally, p_i decides the join of the values $v_i[s]$ of all processes $p_s \in V_i$ (line 29). As $V_i \subseteq K$ (part (1) of Claim 3), every such value equals u_s (part (2) of Claim 1), and hence

$$y_i = \bigsqcup \{u_s \mid p_s \in V_i\}.$$

We are now ready to prove the four properties of Byzantine lattice agreement.

Comparability. First, we prove comparability.

LEMMA 1 (COMPARABILITY). *Join-IT (Algorithm 1) satisfies comparability.*

PROOF. Let p_i and p_j be any two correct processes. By Claim 4, we may assume without loss of generality that $V_i \subseteq V_j$. Both sets are non-empty as they contain C (part (1) of Claim 3) and $|C| \geq n - f > 0$. As the join of a subset is below the join of its superset, we obtain

$$y_i = \bigsqcup \{u_s \mid p_s \in V_i\} \leq \bigsqcup \{u_s \mid p_s \in V_j\} = y_j. \quad \square$$

Downward validity. Next, we prove downward validity.

LEMMA 2 (DOWNWARD VALIDITY). *Join-IT (Algorithm 1) satisfies downward validity.*

PROOF. Let p_i be any correct process. By part (1) of Claim 3, p_i “keeps” itself and $u_i = x_i$ by part (2) of Claim 1. As p_i decides the join of the values u_s of the processes $p_s \in V_i$, and as x_i is one of these values, we obtain $x_i \leq y_i$. $\square$

Upward validity. The following lemma proves upward validity.

LEMMA 3 (UPWARD VALIDITY). *Join-IT (Algorithm 1) satisfies upward validity.*

PROOF. Let $B = \{u_s \mid p_s \in K \setminus C\}$ be the set of values of the faulty processes in K . As every process contributes a single value, $|B| \leq |K \setminus C| \leq n - |C|$. For every correct process p_i , we have $V_i \subseteq K = C \cup (K \setminus C)$ by part (1) of Claim 3, and $u_s = x_s$ for every $p_s \in C$ by part (2) of Claim 1. Hence, every value u_s with $p_s \in V_i$ belongs to $\{x_j \mid p_j \in C\} \cup B$, which gives $y_i \leq \sqcup (\{x_j \mid p_j \in C\} \cup B)$. $\square$

Termination & round complexity. Finally, we prove termination, and we bound the round complexity of Join-IT.

LEMMA 4 (TERMINATION & ROUND COMPLEXITY). *Join-IT (Algorithm 1) satisfies termination. Moreover, every correct process decides by the end of round $R + 3 \in O(\frac{\log f}{\log \log f})$.*

PROOF. All instances $\Pi_{aa}[\cdot]$ decide by the end of round $R + 3$. Thus, every correct process decides by the end of round $R + 3$ (line 29). Finally, as $\frac{1}{\varepsilon} = h = f + 1$, we have $R \in O(\frac{\log f}{\log \log f})$ (cf. §3.1). $\square$

4 Related Work

Lattice agreement with crash failures. Lattice agreement was introduced by Attiya *et al.* [1] as a stepping stone towards atomic snapshot objects, together with a synchronous protocol that tolerates any number of crash failures and decides in $O(\log n)$ rounds. Mavronicolas [9] later gave an early-stopping protocol whose round complexity also depends on the height $h(X)$ of the lattice, namely $O(\min\{h(X), \sqrt{f}\})$, and Zheng *et al.* [14] brought the synchronous round complexity down to $O(\log f)$. Unlike consensus, lattice agreement remains solvable in asynchronous systems, as long as a majority of processes is correct. There, Faleiro *et al.* [6] introduced *generalized* lattice agreement, in which processes keep proposing and deciding values, and used it to build linearizable replicated state machines whose updates commute. For one-shot (i.e., non-generalized) lattice agreement, their protocol takes $O(n)$ asynchronous rounds, which was subsequently improved to $O(f)$ [14] and then to $O(\log f)$ [13].

Byzantine lattice agreement. Nowak and Rybicki [10] were the first to consider Byzantine failures, as a special case of their approximate agreement on graphs. Their protocol works on finite cycle-free semi-lattices, tolerates $f < n/(h(X) + 1)$ Byzantine processes, and guarantees a different validity property: every decision lies above *some* correct proposal and below the join of the correct proposals. Di Luna *et al.* [4] gave the first protocol for arbitrary lattices. Their protocol works in asynchronous systems, decides within $O(f)$ message delays, tolerates the optimal $f < n/3$ Byzantine processes, and extends to generalized lattice agreement and Byzantine replicated state machines. Zheng and Garg [11] later decided in $O(\log f)$ asynchronous rounds, at the price of tolerating only $f < n/5$ Byzantine processes without signatures (and $f < n/3$ with them). In synchrony, Zheng and Garg [12] gave the first protocols, which decide in $O(\min\{h(X), \sqrt{f}\})$, $O(\log n)$, and $O(\log f)$ rounds, and tolerate $f < n/3$ Byzantine processes without signatures (and $f < n/2$ with them). Like Join-IT, all of them rely on gradecast. Concurrently, Di Luna *et al.* [3] decided in $O(\log f)$ rounds with signatures and $f < n/3$, or without signatures and $f < n/4$, and showed that, without signatures, no protocol tolerates $f \geq n/3$.

AI Disclosure

The authors constructed a reduction from Byzantine Lattice Agreement to Byzantine Approximate Agreement. Inserting their ideas into GPT 6.1-Sol, the model brought to their attention the existence

of [2, 8], which obtain Byzantine Approximate Agreement in $O(\frac{\log f}{\log \log f})$ rounds. Additional rounds of interaction served to simplify the protocol and assist with typesetting and editing.

Acknowledgments

Yuval Efron’s research is supported by the Institute for Advanced Study and a Kovner Foundation fellowship.

References

- [1] Hagit Attiya, Maurice Herlihy, and Ophir Rachman. 1995. Atomic Snapshots Using Lattice Agreement. *Distributed Computing* 8, 3 (1995), 121–132.
- [2] Michael Ben-Or, Danny Dolev, and Ezra N. Hoch. 2010. Simple Gradecast Based Algorithms. arXiv:1007.1049 [cs.DC]
- [3] Giuseppe Antonio Di Luna, Emmanuelle Anceaume, Silvia Bonomi, and Leonardo Querzoni. 2020. Synchronous Byzantine Lattice Agreement in $O(\log(f))$ Rounds. In *Proceedings of the 40th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, Singapore, 146–156.
- [4] Giuseppe Antonio Di Luna, Emmanuelle Anceaume, and Leonardo Querzoni. 2019. Byzantine Generalized Lattice Agreement. arXiv:1910.05768 [cs.DC]
- [5] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. 1986. Reaching Approximate Agreement in the Presence of Faults. *J. ACM* 33, 3 (1986), 499–516.
- [6] Jose M. Faleiro, Sriram Rajamani, Kaushik Rajan, G. Ramalingam, and Kapil Vaswani. 2012. Generalized Lattice Agreement. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, New York, NY, USA, 125–134.
- [7] Paul Feldman and Silvio Micali. 1988. Optimal Algorithms for Byzantine Agreement. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, New York, NY, USA, 148–161.
- [8] Marc Fuchs, Diana Ghinea, Zahra Parsaeian, and Joel Rybicki. 2025. Round and Resilience-Optimal Approximate Agreement on Trees and Block Graphs. arXiv:2502.05591v5 [cs.DC]
- [9] Marios Mavronicolas. 2000. A Bound on the Rounds to Reach Lattice Agreement. Manuscript, <http://www.cs.ucy.ac.cy/~mavronic/pdf/lattice.pdf>.
- [10] Thomas Nowak and Joel Rybicki. 2019. Byzantine Approximate Agreement on Graphs. In *Proceedings of the 33rd International Symposium on Distributed Computing (DISC) (LIPIcs, Vol. 146)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 29:1–29:17.
- [11] Xiong Zheng and Vijay K. Garg. 2020. Byzantine Lattice Agreement in Asynchronous Systems. In *Proceedings of the 24th International Conference on Principles of Distributed Systems (OPODIS) (LIPIcs, Vol. 184)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 4:1–4:16. doi:10.4230/LIPIcs.OPODIS.2020.4
- [12] Xiong Zheng and Vijay K. Garg. 2020. Byzantine Lattice Agreement in Synchronous Message Passing Systems. In *Proceedings of the 34th International Symposium on Distributed Computing (DISC) (LIPIcs, Vol. 179)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 32:1–32:16. doi:10.4230/LIPIcs.DISC.2020.32
- [13] Xiong Zheng, Vijay K. Garg, and John Kaippallimalil. 2019. Linearizable Replicated State Machines with Lattice Agreement. In *Proceedings of the 23rd International Conference on Principles of Distributed Systems (OPODIS) (LIPIcs, Vol. 153)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 29:1–29:16. doi:10.4230/LIPIcs.OPODIS.2019.29
- [14] Xiong Zheng, Changyong Hu, and Vijay K. Garg. 2018. Lattice Agreement in Message Passing Systems. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC) (LIPIcs, Vol. 121)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 41:1–41:17. doi:10.4230/LIPIcs.DISC.2018.41