Floor-IT: Information-Theoretic BFT in Partial Synchrony with Two Round Good Case Latency and Optimal Resilience

ITTAI ABRAHAM, a16z Crypto Research, USA
YUVAL EFRON, Institute for Advanced Study, USA
JOVAN KOMATOVIC, Category Labs, USA
ALEJANDRO RANCHAL-PEDROSA, Sei Labs, USA

In the information-theoretic model, parties communicate over sender-authenticated point-to-point channels, but use no digital signatures or other transferable cryptographic certificates; the adversary is otherwise computationally unbounded. We present Floor-IT, an information-theoretic Byzantine agreement protocol for partial synchrony with a good-case latency of two rounds that achieves the optimal resilience bound of $n = 5f - 1$ in this setting. When the actual network delay after GST is at most $\delta \leq \Delta$, our protocol achieves a *robust* good-case latency of 2δ. The protocol proceeds in views and guarantees a worst-case view latency of at most $2\Delta + 2\delta$. Moreover, each party requires only $O(1)$ words of persistent storage, and each view incurs $O(n^2)$ messages of $O(1)$ words each.

1 Introduction

In the Byzantine agreement problem, n processes, of which up to f may be corrupted and behave arbitrarily, must decide on a common value [52]. The problem remains central to fault-tolerant computing, underlying both blockchain finality and replicated services. Most deployed protocols assume *partial synchrony* [37]: the network behaves asynchronously until an unknown Global Stabilization Time (GST), after which every message arrives within a known bound Δ; the unknown actual bound on message delays is denoted by $\delta \leq \Delta$. They are also typically *leader-based*: execution is divided into numbered *views*, each with a designated leader that proposes a value while the other processes vote.

Good-case latency and worst-case view length. Beyond correctness, a practical consensus protocol must decide quickly when conditions are favorable and recover quickly from views that do not decide. In partial synchrony, these requirements are captured by two latency measures. Decision latency is measured from GST. *Good-case latency* considers executions in which GST is zero and the first leader is correct. Its robust variant, *robust good-case latency*, considers executions in which GST coincides with a correct leader entering a view, provided no correct process has previously entered that view or any higher view. *View length* measures how long it can take all correct processes to move past a post-GST view, once every correct process has reached that view or a higher one. Together, the two measures bound decision time after GST and the delay that faulty leaders can impose.

With transferable signatures, both measures are well understood. PBFT [31] decides in three rounds, 3δ, at the optimal resilience $n = 3f + 1$, and three rounds is optimal at that resilience [13]. Tendermint [22] reduced the per-view communication from $O(n^3)$ to $O(n^2)$ while storing only $O(1)$ state, and Simplex [24] reduced the view length to $3\Delta + \delta$. Reducing the good-case latency below 3δ requires more processes per fault. FaB Paxos [54] achieves two-round, 2δ, decisions with $n = 5f + 1$ processes; subsequent work established the tight threshold: two-round decisions are achievable if and only if $n \geq 5f - 1$ [13, 50].

Without transferable signatures, fewer matching bounds were known. In the signature-free, *information-theoretic* (IT) setting, point-to-point channels identify the sender, but the adversary reads every message and is computationally unbounded. No process can forward a transferable

Authors' Contact Information: Ittai Abraham, a16z Crypto Research, USA; Yuval Efron, Institute for Advanced Study, USA; Jovan Komatovic, Category Labs, USA; Alejandro Ranchal-Pedrosa, Sei Labs, USA.

certificate proving that a quorum endorsed a claim; each recipient must instead receive and count the relevant messages. This setting avoids trusted setup and cryptographic computation, and its guarantees are unaffected by future cryptanalytic advances, including quantum attacks. For years, the best known IT protocols were slower than their authenticated counterparts. IT-HotStuff [17] and TetraBFT [64] identified closing this gap as an open problem. TetraBFT achieved a good-case latency of 5δ with constant storage, while leaving the optimal good-case latency in this setting open. Forget-IT [4] resolved this question: at the optimal resilience $n = 3f + 1$, it achieves the optimal good-case latency of 3δ, with $O(1)$ words of persistent storage, $O(n^2)$ messages per view, a view length of 7Δ, and optimistic responsiveness. Fast TetraBFT [40] subsequently matched the optimal 3δ by adding a fast path to TetraBFT's first view. Thus, at $n = 3f + 1$, transferable cryptography does not improve the optimal good-case latency.

DGV gave a two-step execution without authentication, but its recovery uses signed view-change proofs [36]. The $n = 5f - 1$ constructions in [12, 13, 50] also use transferable signatures. A recent chained construction is signature-free end to end [6], but does not establish $O(1)$ persistent storage. These results leave open the following question:

> *Does there exist an information-theoretic BFT protocol that decides in two rounds, at the optimal two-round resilience of $n = 5f - 1$, with $O(n^2)$ messages of $O(1)$ words each per view and $O(1)$ persistent storage?*

Main result. We answer this question affirmatively with Floor-IT, a deterministic Byzantine agreement protocol that tolerates a full-information, computationally unbounded adversary corrupting $f \geq 2$ of $n = 5f - 1$ processes, while storing only $O(1)$ words of persistent memory. After GST, when the leader is correct, Floor-IT decides in two rounds: one message delay for voting and one for deciding, so every correct process decides within 2δ of the proposal. Thus, neither the two-round decision procedure nor view change and recovery requires transferable cryptography. The result refutes a conjecture of Abraham *et al.* [13] that two-round signature-free Byzantine broadcast under partial synchrony requires $n \geq 5f + 1$.[1]

Theorem 1 (Informal)**.** *For every $f \geq 2$, there exists a deterministic, information-theoretic Byzantine agreement protocol on $n = 5f - 1$ processes that is optimistically responsive, achieves good-case and robust good-case latencies of 2δ, has a view length of $2\Delta + 2\delta$, uses $O(1)$ words of persistent storage, and sends $O(n^2)$ messages per view.*

Related Work

Byzantine agreement. Introduced by Lamport, Shostak, and Pease [52, 57], Byzantine agreement asks n processes, up to f of which may fail arbitrarily, to converge on a common value; the problem has since grown a literature far too broad to survey here [2, 7, 23, 27, 28, 30, 42, 45, 51, 55, 60, 61, 63]. The FLP impossibility [41] rules out deterministic solutions under full asynchrony; determinism is recovered under synchrony [33] and under partial synchrony [31, 37], the model of this paper.

Latency of Byzantine agreement. The yardstick by which agreement protocols are most commonly measured is *latency*: the number of message delays until every correct process decides. Under synchrony, the Dolev–Strong protocol [33] solves broadcast in $f + 1$ rounds, which is worst-case optimal, while early-stopping protocols [32] terminate within $f' + 2$ rounds when only $f' < f$ faults actually occur. Under asynchrony, Bracha's reliable broadcast [21] completes

[1] Abraham, Ren, and Xiang [16] give an unauthenticated $(2, 3)$-round reliable-broadcast protocol for $n \geq 5f - 1$ under asynchrony and also study Byzantine broadcast under synchrony; neither result addresses Byzantine broadcast under partial synchrony.

in three rounds, and randomization brings the expected latency of full agreement down to a constant [3, 20, 25, 35, 39, 47, 53, 56, 58].

Good-case latency. The study of the happy path opens with PBFT [31] and its good-case latency of 3δ. FaB Paxos [54] was the first to shave off a round, deciding in 2δ with $5f + 1$ acceptors, and presented that resilience as necessary. The claimed lower bound is not a general $5f + 1$ lower bound: it applies to the restricted architecture in which proposer and acceptor roles are disjoint [50]. Without that restriction, Kuznetsov, Tonkikh, and Zhang [50] and Abraham *et al.* [13] established the tight $n = 5f - 1$ boundary for two-round authenticated broadcast and co-located BFT replicas, alongside the optimality of 3δ at $n = 3f + 1$. This issue is separate from the protocol itself: a later audit found a liveness flaw in the original Parameterized FaB view change (and a safety flaw in Zyzzyva), and subsequent work gave repaired fast/slow view changes [10, 11, 44]. In parallel, Tendermint [22] demonstrated that PBFT's latency is compatible with $O(1)$ storage and $O(n^2)$ communication per view, and, following [13], the good-case landscape has been charted along many further axes [9, 14, 16, 19, 38].

The unauthenticated setting. Closest to our work are the unauthenticated protocols that share our concern for persistent storage; Table 2 juxtaposes them against Floor-IT. IT-HotStuff [17] and TetraBFT [64] both combine $O(1)$ persistent memory with $O(n^2)$ messages per view and optimistic responsiveness, reaching good-case latencies of 6δ and 5δ, respectively, and Forget-IT [4] settled the question at optimal resilience, pairing the optimal (robust) good-case latency of 3δ with the same storage and communication guarantees. The very recent Fast TetraBFT [40] then matched the optimal 3δ with a simpler design, a fast-path wrapper around TetraBFT confined to the first view; as the fast path never returns after view 0, the robust variant remains that of the base protocol. An early result comes close to this setting. For Algorithm 1 of Dutta, Guerraoui, and Vukolić [36], constellation C_2 requires $N_a > \max\{2(M-1)+F+2Q, 2M+F+Q\}$ acceptors. Setting $M = Q = F = f$ gives $N_a \geq 5f - 1$ for $f \geq 2$ (and $N_a \geq 5$ for $f = 1$). Its favorable two-round execution needs no authentication, but the complete algorithm uses signed view-change messages and signed view proofs; it is therefore not an end-to-end signature-free protocol. Moreover, N_a is an acceptor count and becomes the replica count only when the roles are co-located. Floor-IT removes signatures from both the fast path and recovery: it delivers 2δ at $n = 5f - 1$ while retaining $O(1)$ persistent storage, $O(n^2)$ logical communication per view, and responsiveness.

Roadmap

We introduce the system model and formally state the problem in §2. Then, in §3, we present Floor-IT, laying out its pseudocode and walking through the ideas that drive it. Finally, in §4, we establish Floor-IT's correctness and complexity in full detail.

2 Preliminaries

Processes & the adversary. Our system comprises a static set $\{p_1, p_2, \ldots, p_n\}$ of $n = 5f - 1$ message-passing processes, for some integer $f \geq 2$. Every process is modeled as a deterministic state machine that possesses a local clock. During an execution, an *adversary* can corrupt as many as f processes, at any point in time; an adversary with this power is said to be *f-bounded*. A corrupted process is Byzantine and can deviate arbitrarily from the protocol, whereas a non-corrupted process behaves exactly as its state machine prescribes; the former processes are called *faulty*, and the latter *correct*. Additionally, the adversary is assumed to be (1) *full-information*, i.e., it has access to the internal state of every process, and (2) *computationally unbounded*. Lastly, since local computation is negligible relative to message delays, we treat it as instantaneous.

Table 1. Good-case 2δ protocols.

	Optimal $5f-1$ threshold	Signature free	$O(1)$ persistent storage proved	Worst-case post-GST view length
Kursawe (2002) [49]	×	×*	×	–
DGV, Alg. 1, C_2 (2005) [36]	✓	×*	×	*InitTimeout* 2^v
FaB (2006) [54]	×	×*	×	–
Zyzzyva (2007, 2018) [11, 48]	×	×*	×	–
SBFT (2019) [44]†	×	×	×	–
ANRZ (2021) [12, 13]	✓	×	×	–
KTZ (2021) [50]	✓	×	×	–
Banyan (2024) [62]	✓	×	×	–
ChonkyBFT (2025) [43]	×	×	×	–
Kudzu (2025) [59]	×	×	×	–
Alpenglow (2025) [46]	×	×	×	–
Two-round Simplex, $5f+1$ (2025) [8]	×	×	×	$2\Delta + 2\delta$
Two-round Simplex, $5f-1$ (2025) [15]	✓	×	×	$2\Delta + 2\delta$
Minimmit (2025) [26]	×	×	×	$2\Delta + 2\delta$
Fast Simplex FVS (2026) [1]‡	✓	×	×	2Δ
Two-round Simplex, $5f-1$, no-commit vector (2026) [1]	✓	×	×	$2\Delta + \delta$
Fast IT BFT post (2026) [6]	✓	✓	×	$2\Delta + 2\delta$
Floor-IT (2026)	✓	✓	✓	$2\Delta + 2\delta$

The $5f-1$ column is checked when the source gives a two-message-delay fast path at that threshold while tolerating f faulty or nonresponsive participants. DGV is counted under role co-location; Fast Simplex FVS for its stated $5f-1$ refinement; Alpenglow's weighted threshold is not counted. "Signature free" means end-to-end, including view change and recovery, without signatures or SNARKs; sender-authenticated channels are allowed. A starred cross means that only the fast path is signature-free. Storage is checked only when the cited source proves $O(1)$ persistent storage under crash recovery. View length is measured from the time every correct process has reached the view or a higher one until every correct process has moved past it. A dash means that the source gives no comparable bound.

† The published SBFT fast path takes 3δ, and setting $c = f$ gives $n = 5f+1$. Broadcasting signature shares all-to-all instead of through the collector yields the displayed 2δ variant.

‡ Fast Simplex FVS assumes synchronized clocks and fixed view starts $s_v = 2v\Delta$; its 2Δ entry is the scheduled slot length.

Communication network. Processes communicate over a point-to-point message-passing network with *authenticated channels*: the recipient of a message can verify who sent it. Reliability, however, is not among our assumptions. Following prior work that shares our concern for constant persistent storage [5, 34], we adopt an *unreliable* network in which all communication unfolds through broadcasts: a broadcast operation unicasts the message to every process individually and retransmits it, again and again, until the operation is explicitly *abandoned*, following the semantics of [34]. Within this model, a message m is *alive* at time τ if its broadcast has been initiated, but not abandoned, by time τ; it is *forever-alive* if its broadcast, once initiated, is never abandoned.

Persistent memory. To formally reason about the memory usage of our protocol, we follow the model introduced in [5]: the state machine of every process contains a designated *persistent memory* component along with rules that specify how this component evolves over time. Namely, beyond Byzantine corruptions, the adversary can also temporarily *crash* correct processes, under

Table 2. A table of previous information-theoretic protocols and their corresponding metrics, juxtapositioned against our protocol Floor-IT.

	Good-case latency	Robust good-case latency	View length[§]	Persistent storage	Message complexity	Responsiveness
PBFT [31][†]	3δ	7δ	9Δ	$O(1)$	$O(n^3)$	✓
IT-HS (blog) [18][‡]	4δ	6δ	7Δ	$O(1)$	$O(n^2)$	×
IT-HS [17]	6δ	8δ	10Δ	$O(1)$	$O(n^2)$	✓
TetraBFT [64]	5δ	7δ	10Δ	$O(1)$	$O(n^2)$	✓
Fast TetraBFT [40]	3δ	7δ	10Δ	$O(1)$	$O(n^2)$	✓
Forget-IT [4]	3δ	3δ	7Δ	$O(1)$	$O(n^2)$	✓
Floor-IT	2δ	2δ	$2\Delta + 2\delta$	$O(1)$	$O(n^2)$	✓

[†] This PBFT variant is suggested in Castro's thesis, not in the peer-reviewed PBFT paper, and is not proved there.

[‡] Although the leader must wait 2Δ before proposing (sacrificing responsiveness), the (robust) good-case latency measures time from the issued proposal, hence the 2Δ wait does not appear in the good-case latency metrics.

[§] For simplicity, we do not decompose the view lengths of the prior protocols into their Δ and δ components. Regardless of the decomposition, all significantly exceed the view length of Floor-IT, which never surpasses 4Δ (when $\delta = \Delta$).

the guarantee that each crashed correct process eventually *recovers*. A crashed process sends no messages and performs no local computation; in particular, every broadcasting operation it initiated prior to crashing is implicitly abandoned. Crucially, however, the crash leaves the process's persistent memory intact. Moreover, handlers execute atomically with respect to crashes: a crash may strike only between handler executions.[2] Once it recovers, the process runs a designated *recovery procedure*—a purely local computation whose sole input is the contents of persistent memory, with access to no other information, neither volatile state nor messages received in the past. This procedure yields a new local state, from which the process continues executing its state machine as normal; timers resume from their persisted values. Finally, so that reasoning about sub-linear (in n) storage is meaningful, the identities of the participating processes are assumed to reside in a read-only part of the state machine that remains resilient to crashes and whose size does not count towards the process's persistent memory.

Partial synchrony. We adopt the standard partially synchronous model [37]. In this model, an unknown Global Stabilization Time (GST), selected by the adversary, splits the execution into an initial asynchronous period followed by a synchronous one. During the synchronous period, message delays are bounded by a known constant Δ: every message broadcast at time τ that remains alive throughout $[\tau, \max(\tau, \text{GST}) + \Delta]$ is received by time $\max(\tau, \text{GST}) + \Delta$. Furthermore, we write

[2]This assumption is without loss of generality, as Floor-IT's handlers are non-blocking and straight-line. It can be simulated from word-atomic persistent writes by journaling, in persistent memory, the active handler, its parameters, and an instruction counter, and by having the recovery procedure resume the interrupted handler from the recorded position. As every handler body operates on $O(1)$-word values, the journal occupies $O(1)$ persistent words; moreover, re-executing the single instruction at the crash boundary is harmless, as assignments and broadcast re-initiations are idempotent, and repeated decisions of the same value are permitted.

$\delta \leq \Delta$ for the actual message delay after GST, and assume that every message broadcast at time τ that remains alive throughout $[\tau, \max(\tau, \text{GST}) + \delta]$ is received by time $\max(\tau, \text{GST}) + \delta$.

Additionally, every correct process is assumed to start executing the protocol no later than GST, and hardware clocks, though they may drift arbitrarily before GST, exhibit no drift afterwards. Finally, the adversary is allowed to crash correct processes only prior to GST, and each crashed correct process is guaranteed to also recover before GST. Consequently, all correct processes are continuously online from GST onward.

Byzantine agreement & its metrics. This work designs a Byzantine agreement protocol operating in the model described above. Formally, we address the problem where each correct process proposes an initial value and seeks to decide a value such that the following properties are satisfied:

- *Agreement:* Correct processes never decide different values.
- *Termination:* Each correct process eventually decides.
- *Non-triviality:* If GST = 0, all processes are correct, and all propose the same value v, then no correct process decides a value other than v.

Although various other validity constraints exist [29], we intentionally adopt the simplest formulation so as to keep the focus on the core algorithmic challenges, which do not depend on the specific validity variant.

Efficiency metrics. Three key metrics underpin our evaluation of protocol efficiency. Given an f-bounded adversary $\mathcal{A}$ (capable of corrupting up to f processes) and an f-secure (resilient to f corrupted processes) leader-based Byzantine agreement protocol Π, let $\mathsf{DL}_{\mathcal{A}}(\Pi)$ denote the smallest time T by which, in the presence of $\mathcal{A}$, every correct process decides (but not necessarily terminates), normalized to GST, i.e., $\mathsf{DL}_{\mathcal{A}}(\Pi) = T - \text{GST}(\mathcal{A})$. (Here, $\text{GST}(\mathcal{A})$ denotes the GST of a given adversary $\mathcal{A}$.) Given a family $\mathcal{F}$ of f-bounded adversaries, the decision latency of Π with respect to $\mathcal{F}$ is defined as $\mathsf{DL}_{\mathcal{F}}(\Pi) = \sup_{\mathcal{A} \in \mathcal{F}} \mathsf{DL}_{\mathcal{A}}(\Pi)$.

- *Good-case latency* T_{gc}*:* Let $\mathcal{F}_{\text{gc}}$ denote the family of f-bounded adversaries $\mathcal{A}$ under which GST = 0 and the first leader ℓ is correct. We define $\mathsf{T}_{\text{gc}} \triangleq \mathsf{DL}_{\mathcal{F}_{\text{gc}}}(\Pi)$.
- *Robust good-case latency* T_{rgc}*:* Let $\mathcal{F}_{\text{rgc}}$ denote the family of f-bounded adversaries $\mathcal{A}$ under which GST coincides with a correct leader entering its view, no correct process having entered that view or any greater view beforehand. We define $\mathsf{T}_{\text{rgc}} \triangleq \mathsf{DL}_{\mathcal{F}_{\text{rgc}}}(\Pi)$.
- *Memory complexity* $\mathsf{PM}(\Pi)$*:* Under an adversary $\mathcal{A}$, let $\mathsf{PM}_{\mathcal{A}}(p_i, \tau)$ denote the size (in words) of the persistent memory register of a correct process p_i at time τ, where a single word can hold a constant number of messages, values, and counters. The memory complexity of Π with respect to $\mathcal{A}$ is then $\mathsf{PM}_{\mathcal{A}}(\Pi) = \max_{\text{correct } p_i, \tau \in \mathbb{N}} \mathsf{PM}_{\mathcal{A}}(p_i, \tau)$, while the overall memory complexity of Π is $\mathsf{PM}(\Pi) = \max_{f\text{-bounded } \mathcal{A}} \mathsf{PM}_{\mathcal{A}}(\Pi)$.

Having defined good-case latency, we are now ready to introduce the notion of *responsiveness*. Given a BA protocol Π, let $E(\Pi)$ denote an execution of the protocol; such an execution is fully determined by the adversary's $\mathcal{A}$ choices of corrupt parties & their behaviour, GST, and network delays within the realm of the partially synchronous model. Given an execution $E(\Pi)$ and a message m from correct party p to correct party q, we write $\delta(m)$ for the wall-clock time that elapses between p sending m and q receiving m. Letting M_E be the set of messages sent between correct parties in E, the *actual delay* of the execution E is defined as $\delta(E) = \max_{m \in M_E} \delta(m)$. A protocol Π is said to be *responsive* if its good-case latency depends only on δ. More formally, if there exists a constant $c > 0$ such that $\mathsf{DL}_E(\Pi) < c \cdot \delta(E)$ for every execution $E \in \mathcal{F}_{\text{gc}}$.

Lastly, the *view length* of Π measures how long a single view can detain the correct processes after the network has stabilized: informally, it is the longest it takes them to move from any post-GST view, i.e., any view first entered by a correct process after GST, to greater views. Its formal definition is deferred to §4.4.

Algorithm 1 Floor-IT: Pseudocode (for process p_i) [part 1 of 2]

1: **Constants:**
2: String SAFE = 'Every value is safe' ▷ special outcome: no decision constrains the future
3: **Message formats (field types of each message):**
4: - $\langle$PROPOSE, View V, Value $x\rangle$ ▷ the leader of view V proposes value x
5: - $\langle$VOTE, View V, Value $\cup\ \{\perp\}\ x$, Value $\cup$ {SAFE} $g\rangle$ ▷ ballot x ($\perp$ abstains) and the sender's forwarded guard g
6: - $\langle$ADOPT, View V, Value $\cup$ {SAFE} $c\rangle$ ▷ outcome c distilled from view V
7: **Notation (messages are always counted from distinct senders):**
8: - A *vote sample* S for view V: a set of $4f-1$ received $\langle$VOTE, $V, \cdot, \cdot\rangle$ messages
9: - A *non-leader sample* for view V: a vote sample that excludes leader(V)'s vote
10: - $count_S(x)$: the number of $\langle$VOTE, $V, x, \cdot\rangle$ messages in a vote sample S; Heavy$(S) = \{x \in \text{Value} : count_S(x) \geq 2f-1\}$
11: **Predicates (over the received votes for view V):**
12: - unique(View V, Value x) $\equiv$ some vote sample S has Heavy$(S) = \{x\}$
13: - equiv(View V) $\equiv$ some vote sample S has $|\text{Heavy}(S)| \geq 2$
14: - empty(View V) $\equiv$ some vote sample S has Heavy$(S) = \emptyset$
15: - survivor(View V, Value x) $\equiv$ some non-leader sample N has $count_N(x) \geq 2f$
16: - no_survivor(View V) $\equiv$ some non-leader sample N has no value x with $count_N(x) \geq 2f$
17: - barren(View V) $\equiv$ empty(V) **or** (equiv(V) **and** no_survivor(V))
18: - dominant(View V, Value $\cup$ {SAFE} g) $\equiv$ some vote sample S has $\geq 3f-1$ votes carrying guard g
19: - no_dominant(View V) $\equiv$ some vote sample S has no guard carried by $\geq 3f-1$ of its votes
20: **Rules:**
21: - Process p_i broadcasts at most one vote per view.
22: - Process p_i keeps in persistent memory the view-indexed variables of three views only: $view_i - 2$, $view_i - 1$, and $view_i$.
23: - Upon entering view V, p_i abandons all ongoing broadcasts, except those of messages from views $V-2$ and $V-1$.
24: - Process p_i evaluates the rules at lines 62–68 for a view V only if $V < view_i$ or $timer_i[V] \geq 2\Delta$.
25: **Local variables, indexed by view V where applicable:**
26: - Value $proposal_i$
27: - View $view_i \leftarrow 0$
28: - Value $\cup$ {SAFE} $guard_i[V] \leftarrow \perp$ ▷ p_i's guard for view V
29: - Timer $timer_i[V]$ ▷ view timer
30: - Message $vote_i[V] \leftarrow \perp$ ▷ the vote (if any) broadcast by p_i in view V
31: - Set $adopted_i[V] \leftarrow \emptyset$ ▷ outcomes for which p_i has broadcast an ADOPT message
32: **Functions:**
33: - **function** cast(View V, Value $\cup\ \{\perp\}\ x$, Value $\cup$ {SAFE} g): ▷ broadcast p_i's single vote for view V
34: **if** $vote_i[V] = \perp$: $vote_i[V] \leftarrow \langle$VOTE, $V, x, g\rangle$; **broadcast** $vote_i[V]$
35: **upon starting:**
36: $proposal_i \leftarrow p_i$'s proposal; **enter** view 1 **with guard** SAFE
37: **upon entering** view V **with guard** G:
38: $guard_i[V] \leftarrow G$; $view_i \leftarrow V$; **start** $timer_i[V]$
39: **for each** view $W \in \{V-2, V-1\}$ **with** $W \geq 1$:
40: **invoke** cast$(W, \perp, G)$ ▷ complete the two preceding views' vote records
41: **if** $p_i =$ leader(V):
42: **if** $guard_i[V] =$ SAFE: **broadcast** $\langle$PROPOSE, $V, proposal_i\rangle$ ▷ unconstrained: propose own value
43: **else: broadcast** $\langle$PROPOSE, $V, guard_i[V]\rangle$ ▷ constrained: propose the guarded value
44: **upon** $\langle$PROPOSE, $view_i, x\rangle$ **from** leader$(view_i)$ **with** $guard_i[view_i] \in$ {SAFE, x}, $vote_i[view_i] = \perp$, $timer_i[view_i] \leq 2\Delta$:
45: **invoke** cast$(view_i, x, guard_i[view_i])$
46: **upon** $timer_i[view_i]$ reaching 2Δ **with** $vote_i[view_i] = \perp$:
47: **invoke** cast$(view_i, \perp, guard_i[view_i])$
48: **upon receipt of** $4f-1$ $\langle$VOTE, $V, x, \cdot\rangle$ for the same view V and the same value $x \neq \perp$:
49: **trigger** decide(x)

3 Floor-IT: Pseudocode

As already mentioned, Floor-IT follows the view-based paradigm: execution unfolds through a sequence of views $V = 1, 2, 3, \ldots$, each steered by a leader appointed in round-robin fashion. With this backdrop in place, we now describe the protocol.

Messages, quorums, & guards. Three kinds of messages circulate within a view V. The leader opens the view with a single proposal $\langle$PROPOSE, $V, x\rangle$ (lines 42 and 43). Every process answers with one vote $\langle$VOTE, $V, x, g\rangle$, whose *ballot* $x \in$ Value $\cup\ \{\perp\}$ either endorses the proposal or abstains ($\perp$), and which additionally forwards g, the sender's current *guard* (line 5). Finally, once the view has spoken, processes summarize its verdict in adopt messages $\langle$ADOPT, $V, c\rangle$, whose *outcome*

Algorithm 1 Floor-IT: Pseudocode (for process p_i) [part 2 of 2]

50: ▷ entering future views
51: **upon receipt of** $4f-1$ $\langle$ADOPT, $V, c\rangle$ for the same view V and outcome c **and** $p_i \neq \text{leader}(V+1)$ **and** $view_i \leq V$:
52: **enter** view $V+1$ **with guard** c
53: **upon receipt of** $3f-1$ $\langle$ADOPT, $V, c\rangle$ for the same view V and outcome c **and** $p_i = \text{leader}(V+1)$ **and** $view_i \leq V$:
54: **enter** view $V+1$ **with guard** c ▷ the leader enters "early"
55: **upon** $\langle$PROPOSE, $V+1, \cdot\rangle$ **from** $\text{leader}(V+1) \neq p_i$ **and** $2f-1$ $\langle$ADOPT, $V, c\rangle$ for the same view V and outcome c **and** $view_i \leq V$:
56: **enter** view $V+1$ **with guard** c ▷ follow the leader's proposal
57: **upon receipt of** a set T of $3f-1$ $\langle$VOTE, $V, \cdot, \cdot\rangle$ messages for the same view V **and** $view_i < V$:
58: **if** some value x has $\geq 2f-1$ ballots in T: **enter** view V **with guard** x; **invoke** $\text{cast}(V, x, x)$ ▷ support the front-runner
59: **else if** some guard g is carried by $\geq 2f-1$ votes in T: **enter** view V **with guard** g; **invoke** $\text{cast}(V, \perp, g)$ ▷ inherit g
60: **else: enter** view V **with guard** $guard_i[view_i]$; **invoke** $\text{cast}(V, \perp, guard_i[V])$ ▷ keep the current guard
61: ▷ issuing ADOPT messages
62: **upon** $\text{unique}(V, x)$ **and** $V \in \{view_i-2, view_i-1, view_i\}$ **and** $x \notin adopted_i[V]$:
63: $adopted_i[V] \leftarrow adopted_i[V] \cup \{x\}$; **broadcast** $\langle$ADOPT, $V, x\rangle$ ▷ x is the unique possible decision of view V
64: **upon** $\text{equiv}(V)$ **and** $\text{survivor}(V, x)$ **and** $V \in \{view_i-2, view_i-1, view_i\}$ **and** $x \notin adopted_i[V]$:
65: $adopted_i[V] \leftarrow adopted_i[V] \cup \{x\}$; **broadcast** $\langle$ADOPT, $V, x\rangle$ ▷ the leader equivocated; x survives the non-leader re-sample
66: **upon** $\text{barren}(V)$ **and** $\text{dominant}(V, g)$ **and** $V \in \{view_i-2, view_i-1, view_i\}$ **and** $g \notin adopted_i[V]$:
67: $adopted_i[V] \leftarrow adopted_i[V] \cup \{g\}$; **broadcast** $\langle$ADOPT, $V, g\rangle$ ▷ no decision in view V: pass the dominant guard forward
68: **upon** $\text{barren}(V)$ **and** $\text{no_dominant}(V)$ **and** $V \in \{view_i-2, view_i-1, view_i\}$ **and** $\text{SAFE} \notin adopted_i[V]$:
69: $adopted_i[V] \leftarrow adopted_i[V] \cup \{\text{SAFE}\}$; **broadcast** $\langle$ADOPT, V, SAFE$\rangle$ ▷ no decision and no dominant guard: everything is safe
70: ▷ divergence: two well-backed outcomes prove that no decision exists anywhere in the past
71: **upon receipt of** $2f-1$ $\langle$ADOPT, $V, c_1\rangle$ **and** $2f-1$ $\langle$ADOPT, $V, c_2\rangle$ **with** $c_1 \neq c_2$ **and** $view_i = V+1$:
72: $guard_i[view_i] \leftarrow \text{SAFE}$ ▷ relax the guard: everything is safe
73: ▷ the recovery procedure
74: **upon recovery:**
75: **for each** view $V \in \{view_i-2, view_i-1, view_i\}$ **with** $V \geq 1$:
76: **if** $vote_i[V] = \perp$: $vote_i[V] \leftarrow \langle$VOTE, $V, \perp, guard_i[view_i]\rangle$ ▷ complete the window's vote records
77: **broadcast** $vote_i[V]$
78: **for each** $c \in adopted_i[V]$: **broadcast** $\langle$ADOPT, $V, c\rangle$

$c \in \text{Value} \cup \{\text{SAFE}\}$ is either a value the future must respect or the token SAFE, declaring that view V constrains nothing (line 6). All counting is over distinct senders: a *vote sample* for view V is any set of $4f-1$ received votes for V (line 8), and a *non-leader sample* is a vote sample that excludes $\text{leader}(V)$'s vote (line 9). Three thresholds recur throughout, $4f-1$, $3f-1$, and $2f-1$, with a fourth, $2f$, reserved for the aftermath of leader equivocation; the first three are spaced exactly f apart, and f is precisely the number of voices the adversary commands, so whatever one correct process establishes with a quorum, every process can establish one rung cheaper within a single message delay (Figure 1). The guard, finally, is a one-word summary of history: a value v reads "v is the only value that may have been decided so far; protect it," whereas SAFE reads "no past decision binds; every value is safe." Each process p_i holds one guard per view, $guard_i[V]$, written at the moment it enters view V (line 38), and guards travel: every vote forwards one, so the record of a view contains, alongside its ballots, the history that its voters carried into it.

Entering a view. Four doors lead into a view. A non-leader ordinarily enters view $V+1$ upon $4f-1$ matching $\langle$ADOPT, $V, c\rangle$ messages (line 51); the leader of view $V+1$ is admitted for $3f-1$ (line 53); and a process already holding the new leader's proposal enters for just $2f-1$ (line 55). The fourth door serves stragglers: $3f-1$ votes for a view V ahead of one's own admit a process directly into V (line 57). On the first three doors, the guard stamped at entry is the adopts' common outcome c; the fourth derives it from the votes themselves: if $2f-1$ of them agree on a ballot x, the newcomer enters with guard x and casts its own ballot for x on the spot (line 58); otherwise, if $2f-1$ of them forward the same guard g, the newcomer inherits g and abstains (line 59); otherwise, it keeps the guard it already had and abstains (line 60). Entry always overwrites the guard and starts the view's timer (line 38).

The happy path. Upon entering view V, the leader broadcasts its proposal: its own input value when its guard is SAFE, and the guarded value otherwise, for then the past has already chosen for it (lines 42 and 43). A process votes for the proposed value x when three conditions align: its guard permits it ($guard_i[V] \in \{\text{SAFE}, x\}$), it has not yet voted in view V, and its timer shows at most 2Δ (line 44). A process decides x upon receiving $4f - 1$ votes for the same value $x \neq \perp$ (line 48). Around this path runs a strict voting discipline. A process broadcasts at most one vote per view, without exception (line 21); this one-shot ballot is what gives quorums their meaning in a world without signatures. If the timer reaches 2Δ and no vote has been cast, the process abstains with a $\perp$ ballot, forwarding its guard (line 47); and no view is left blank: upon entering view V, a process backfills the two preceding views, casting a $\perp$ ballot carrying its entry guard wherever it has not voted yet (line 39). The vote record that the next paragraph feeds on is thus always, eventually, complete.

Distilling a view. The adopt rules read vote samples and answer one question: what does view V oblige the future to remember? Call a value *heavy* in a sample if at least $2f - 1$ of its ballots name it (line 10); the threshold is calibrated to the footprint of a decision, which enlists $4f - 1$ votes for one value, at least $3f - 1$ of them correct, so that at least $2f - 1$ survive into *every* sample, however the adversary assembles it. A decided value, in other words, is heavy everywhere. When a sample exhibits a unique heavy value x, the process broadcasts $\langle \textsc{adopt}, V, x \rangle$ (line 62). When a sample exhibits two heavy values, two correct processes voted differently, which only a two-faced leader can cause: $\text{leader}(V)$ is Byzantine, every correct process is a non-leader, and the reading is repeated over a non-leader sample, where the corrupted leader's voice is absent, at the sharper bar of $2f$; a value reaching $2f$ votes there is adopted (line 64). When, instead, some sample has no heavy value at all, or the leader equivocated and no value survives the non-leader re-sample, the view is *barren* (line 17): it decided nothing. A barren view, however, is not a clean slate, as a decision from an earlier view may still be in force, and the forwarded guards say so: if some guard g is carried by at least $3f - 1$ votes of a sample, it is *dominant*, and the process broadcasts $\langle \textsc{adopt}, V, g \rangle$, the past passed forward (line 66); if instead some sample has no dominant guard, the process broadcasts $\langle \textsc{adopt}, V, \text{SAFE} \rangle$ (line 68). The bar $3f - 1$ mirrors the residue argument one level up: every sample holds at least $3f - 1$ correct votes and, once a decision is in force, every correct guard names the decided value, so dominance is inescapable and the SAFE branch can never fire; conversely, guards too scattered to dominate certify that no such decision exists, and declaring everything safe dissolves constraints that were never real.

Pacing & divergence. A process evaluates the four adopt rules for its *current* view only once its timer has run for 2Δ; the two earlier tracked views are read immediately (line 24). The gate never delays a decision, which reads votes rather than adopts, but it places a floor of 2Δ under every view transition, since entering view $V + 1$ requires adopts for view V that correct processes will not utter early. One further reading of the adopts is needed (line 71): a process in view $V + 1$ that holds $2f - 1$ adopts for an outcome c_1 and $2f - 1$ adopts for a different outcome c_2, both of view V, relaxes its guard for view $V + 1$ to SAFE (line 72). Each batch contains a correct backer, and after a real decision no second outcome can ever gather $2f - 1$ adopts; two well-backed outcomes therefore prove that no decision is in force. The relaxation frees a process whose stamped guard would otherwise block it from supporting a perfectly safe proposal; without it, correct processes admitted through quorums for different, individually safe outcomes could veto each other's leaders forever.

Memory & recovery. A process retains the view-indexed variables of only three views, $view_i - 2$, $view_i - 1$, and $view_i$ (line 22), and, upon entering a view, abandons all ongoing broadcasts except those of messages from the two preceding views (line 23). The 2Δ floor of the adopt gate is what

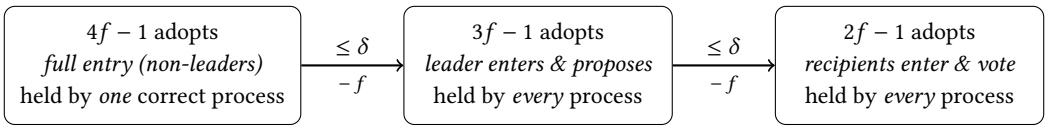


Fig. 1. The three prices of view entry and how evidence cascades between them. The rungs of the threshold ladder are exactly f apart: a full quorum seen by a single correct process becomes a leader quorum everywhere within one message delay (correct adopts are broadcast, and $(4f-1)-f=3f-1$), and the leader quorum behind a proposal becomes a recipient quorum everywhere within one further delay ($3f-1-f=2f-1$).

makes so narrow a window survivable: views cannot advance faster than the evidence that lets stragglers catch up, so nothing a correct process still needs is ever forgotten. The crash-recovery model of §2 is honored by a single rule (line 74): upon recovering, a process republishes what its persistent memory holds for the three tracked views, its votes and its adopts, first completing any still-empty vote record with a $\perp$ ballot that forwards its current guard (line 76). Every message the protocol ever sends is a verbatim copy of a persistently stored variable, so recovery is re-broadcast, and nothing more.

4 Proof of Correctness & Complexity

This section establishes the correctness and complexity of the Floor-IT algorithm.

4.1 Agreement

We first show that Floor-IT satisfies the agreement property, meaning that no two correct processes decide different values. To establish this, we begin by proving that the current view number of every correct process is monotonically increasing throughout the execution.

Claim 1. If any correct process p_i enters a view $V_1 \geq 1$ at time τ_1 and later enters a view $V_2 \geq 1$ at time $\tau_2 > \tau_1$, then $V_2 > V_1$.

Proof. Once process p_i enters view V_1, it sets its local variable $view_i$ to V_1 (line 38); note that $view_i$ is stored in persistent memory. Let V^* denote the next view that process p_i enters. Setting aside the initial entry into view 1 (line 36), which occurs once, at start-up, and is therefore never a subsequent entry, views are entered only at lines 52, 54, and 56, whose conditions (lines 51, 53, and 55) require $view_i \leq V^* - 1$, and at lines 58, 59, and 60, whose condition (line 57) requires $view_i < V^*$. In every case $V^* > V_1$, which completes the proof. □

Next, we show that a correct process casts at most one vote per view, and that it writes and broadcasts vote messages only for its three tracked views.

Claim 2. Let p_i be a correct process and $V \geq 1$ a view. Then:

(1) All $\langle \text{VOTE}, V, \cdot, \cdot \rangle$ messages broadcast by p_i are identical.
(2) Process p_i writes $vote_i[V]$ only at a time when $V \in \{view_i - 2, view_i - 1, view_i\}$.
(3) Process p_i broadcasts a $\langle \text{VOTE}, V, \cdot, \cdot \rangle$ message only at a time when $V \in \{view_i - 2, view_i - 1, view_i\}$.

Proof. We first prove part (2). The variable $vote_i[V]$ is written only by the function $\text{cast}(V, \cdot, \cdot)$ (line 34) and by the recovery rule (line 76); the recovery rule writes $vote_i[V]$ only for $V \in \{view_i - 2, view_i - 1, view_i\}$ (line 75). The function $\text{cast}(V, \cdot, \cdot)$ is invoked at line 45 (with $V = view_i$), at line 47 (with $V = view_i$), at line 40 (with $V \in \{view_i - 2, view_i - 1\}$, by line 39; note that $view_i$ is updated to the entered view beforehand), and at lines 58, 59 and 60 (with V equal to the view

just entered, so $view_i = V$ at invocation time). In all cases, $V \in \{view_i - 2, view_i - 1, view_i\}$ when $vote_i[V]$ is written, which proves part (2).

Next, we prove part (3). A $\langle \text{VOTE}, V, \cdot, \cdot \rangle$ message is broadcast only by the function cast$(V, \cdot, \cdot)$ (line 34) and by the recovery rule (line 77). The former initiates the broadcast of $vote_i[V]$ at the moment of writing it, which by part (2) happens only while $V \in \{view_i - 2, view_i - 1, view_i\}$; the recovery rule initiates the broadcast of $vote_i[V]$ only for $V \in \{view_i - 2, view_i - 1, view_i\}$ (line 75). Hence, every broadcast of a $\langle \text{VOTE}, V, \cdot, \cdot \rangle$ message is initiated at a time when $V \in \{view_i - 2, view_i - 1, view_i\}$. Therefore, $V \in \{view_i - 2, view_i - 1, view_i\}$ throughout, which proves part (3).

Finally, we prove part (1). Both write sites set $vote_i[V]$ only if $vote_i[V] = \perp$ (lines 34 and 76), and every broadcast of a $\langle \text{VOTE}, V, \cdot, \cdot \rangle$ message broadcasts the stored value $vote_i[V]$ (lines 34 and 77). As $vote_i[V]$ resides in persistent memory, it can revert to $\perp$ only when discarded by the three-view rule (line 22), which happens only once $view_i \geq V + 3$. By Claim 1, $view_i \geq V + 3$ holds forever thereafter; by part (2), $vote_i[V]$ is then never written again, and, by part (3), no $\langle \text{VOTE}, V, \cdot, \cdot \rangle$ message is broadcast again either. Hence, all broadcast $\langle \text{VOTE}, V, \cdot, \cdot \rangle$ messages equal the unique value ever stored in $vote_i[V]$, which concludes the proof. □

We now introduce a distinguished view that captures when the first decision by a correct process occurs.

Definition 1 (View V_{dec})**.** Let V_{dec} denote the smallest view in which a correct process decides.

Observe that if no correct process ever decides, then V_{dec} is undefined and agreement holds vacuously; thus, we assume henceforth that V_{dec} is defined. The following claim establishes that the value decided in view V_{dec} is unique.

Claim 3. If a correct process decides a value v_1 in view $V_{dec} \geq 1$ and another correct process decides a value v_2 in the same view, then $v_1 = v_2$.

Proof. Assume toward contradiction that $v_1 \neq v_2$. For a value v_1 to be decided in view V_{dec}, votes with ballot v_1 must be received from $4f - 1$ distinct senders (line 48); similarly for v_2. Since $n = 5f - 1$, the two sender sets overlap on at least $2(4f - 1) - n = 3f - 1$ processes, of which at least $3f - 1 - f = 2f - 1 \geq 1$ are correct. Each such correct process would have broadcast two different votes for view V_{dec} (one with ballot v_1, one with ballot v_2), contradicting Claim 2. □

We next introduce the value associated with this first decision.

Definition 2 (Value v_{dec})**.** Let v_{dec} denote the value decided by a correct process in view V_{dec}.

By Claim 3, v_{dec} is uniquely determined. The next claim is the counting heart of the proof: the decision on v_{dec} leaves a residue that appears in every vote sample for view V_{dec}, however the sample is assembled.

Claim 4. Let M denote the set of correct processes whose ballot for view V_{dec} is v_{dec}. Then, $|M| \geq 3f - 1$. Moreover:

(1) every vote sample for view V_{dec} contains at least $2f - 1$ votes cast by members of M;
(2) if leader(V_{dec}) is faulty, then every non-leader sample for view V_{dec} contains *(i)* at least $2f$ votes cast by members of M, and *(ii)* for every (non-$\perp$) value $y \neq v_{dec}$, at most $2f - 1$ votes carrying ballot y.

Proof. Let $f^\star \leq f$ denote the actual number of faulty processes. The decision on v_{dec} requires votes with ballot v_{dec} from $4f - 1$ distinct senders, of which at least $4f - 1 - f^\star$ are correct; by Claim 2, these correct senders vote v_{dec} and nothing else, so $|M| \geq 4f - 1 - f^\star \geq 3f - 1$.

For part (1), a vote sample draws $4f-1$ distinct senders from a universe of $n = 5f-1$ processes. The senders outside M number $n - |M| \leq (5f-1) - (3f-1) = 2f$, so the sample contains at least $(4f-1) - 2f = 2f-1$ votes from M.

For part (2), suppose $\mathsf{leader}(V_{dec})$ is faulty; then every member of M is a non-leader. A non-leader sample draws $4f-1$ distinct senders from the $n-1 = 5f-2$ non-leaders; the non-leaders outside M number at most $(5f-2) - (3f-1) = 2f-1$, so the sample contains at least $(4f-1) - (2f-1) = 2f$ votes from M. Finally, fix $y \neq v_{dec}$. The correct processes whose ballot for view V_{dec} is not v_{dec} number at most $(n-f^\star) - (4f-1-f^\star) = f$, and the faulty non-leaders number at most $f^\star - 1 \leq f-1$; hence at most $f + (f-1) = 2f-1$ senders of a non-leader sample can exhibit ballot y. □

Before reading off the consequences of Claim 4, we record that a value ballot cast by a correct process always traces back to a leader proposal.

Claim 5. If a correct process casts a vote for view V with ballot $x \neq \perp$, then some correct process received $\langle \textsc{propose}, V, x\rangle$ from $\mathsf{leader}(V)$.

Proof. We proceed by induction on the order in which correct processes cast value ballots for view V. A correct process casts a ballot $x \neq \perp$ for view V only at line 45, upon receiving $\langle \textsc{propose}, V, x\rangle$ from $\mathsf{leader}(V)$ (the claim follows directly), or at line 58, upon receiving a set T of $3f-1$ votes for view V in which at least $2f-1$ ballots equal x. In the latter case, the $2f-1$ ballots for x come from distinct senders, of which at most f are faulty; hence at least $f-1 \geq 1$ (as $f \geq 2$) are correct processes that cast ballot x for view V earlier, and the inductive hypothesis applied to any of them yields the claim. □

The next claim translates the residue into the language of the predicates: in view V_{dec}, the ballot-reading rules can endorse only v_{dec}, and the barren rules cannot fire at all.

Claim 6. For every correct process p_i and at every time:

(1) if $\mathsf{unique}(V_{dec}, y)$ holds at p_i, then $y = v_{dec}$;
(2) $\mathsf{empty}(V_{dec})$ does not hold at p_i;
(3) if $\mathsf{equiv}(V_{dec})$ holds at p_i, then $\mathsf{leader}(V_{dec})$ is faulty;
(4) if $\mathsf{equiv}(V_{dec})$ holds at p_i and $\mathsf{survivor}(V_{dec}, y)$ holds at p_i, then $y = v_{dec}$;
(5) if $\mathsf{equiv}(V_{dec})$ holds at p_i, then $\mathsf{no_survivor}(V_{dec})$ does not hold at p_i;
(6) $\mathsf{barren}(V_{dec})$ does not hold at p_i.

Proof. By Claim 4(1), every vote sample S for view V_{dec} satisfies $count_S(v_{dec}) \geq 2f-1$, i.e., $v_{dec} \in \mathsf{Heavy}(S)$. Part (2) of the claim follows immediately, and part (1) follows since $\mathsf{Heavy}(S) = \{y\}$ forces $y = v_{dec}$.

For part (3), suppose some sample S has $|\mathsf{Heavy}(S)| \geq 2$, and let $y \neq v_{dec}$ be a second heavy value. Then ballots v_{dec} and y are each exhibited by at least $2f-1$ distinct senders of S, hence each by at least $2f-1-f = f-1 \geq 1$ correct processes. By Claim 5, $\mathsf{leader}(V_{dec})$ sent both $\langle \textsc{propose}, V_{dec}, v_{dec}\rangle$ and $\langle \textsc{propose}, V_{dec}, y\rangle$ to correct processes; a correct leader broadcasts exactly one proposal per view (line 41), so $\mathsf{leader}(V_{dec})$ is faulty.

Parts (4) and (5) now follow from Claim 4(2): since $\mathsf{leader}(V_{dec})$ is faulty, every non-leader sample N satisfies $count_N(v_{dec}) \geq 2f$ and $count_N(y) < 2f$ for all $y \neq v_{dec}$. Part (6) follows from parts (2) and (5) and the definition of barren (line 17). □

The following claim is the heart of the agreement proof. It establishes that once a decision on v_{dec} occurs in view V_{dec}, every message broadcast by a correct process in any subsequent view remains consistent with v_{dec}.

Claim 7. Let m be any message for some view V broadcast by a correct process. Then:

(1) If $m = \langle \text{ADOPT}, V, c \rangle$ with $V \geq V_{dec}$, then $c = v_{dec}$.
(2) If $m = \langle \text{VOTE}, V, x, g \rangle$ with $V > V_{dec}$, then $g = v_{dec}$ and $x \in \{v_{dec}, \perp\}$.

Proof. We proceed by induction on the order in which correct processes broadcast messages. Fix a message m broadcast by a correct process p_i whose type and view match one of the conditions of the claim (i.e., m is an ADOPT message for a view $V \geq V_{dec}$, or a VOTE message for a view $V > V_{dec}$), and suppose that every such message previously broadcast by a correct process is of the form required by the claim. Since the recovery rule (lines 77 and 78) re-broadcasts only messages previously created (and line 76 creates a vote covered by Subclaim 1 and Subclaim 2 below), it suffices to analyze the rules that create messages. We establish the claim through three subclaims.

Subclaim 1. *If p_i enters a view $V + 1 > V_{dec}$ with guard G, then $G = v_{dec}$; moreover, $guard_i[V + 1] = v_{dec}$ holds from that moment onward (for as long as the variable is retained).*

Proof of Subclaim 1. We first consider the value of G at entry, according to the entry path:

- *Adopt-based entry* (lines 52, 54, 56): the guard $G = c$ is backed by $\langle \text{ADOPT}, V, c \rangle$ messages from at least $2f - 1$ distinct senders, where $V \geq V_{dec}$. At least $2f - 1 - f = f - 1 \geq 1$ of them are correct, so by the inductive hypothesis (part 1), $c = v_{dec}$.
- *Vote-based catch-up* (line 57): the trigger set T contains $3f - 1$ votes for view $V + 1 > V_{dec}$ from distinct senders, of which at least $2f - 1$ are from correct processes. By the inductive hypothesis (part 2), every correct vote in T carries guard v_{dec} and a ballot in $\{v_{dec}, \perp\}$. If the branch at line 58 is taken with value x, then at least $2f - 1$ ballots in T equal x, of which at least $f - 1 \geq 1$ are correct, so $x = v_{dec}$ and $G = v_{dec}$. If the branch at line 59 is taken with guard g, the same counting gives $g = v_{dec}$. The branch at line 60 is unreachable as the guard v_{dec} is carried by all $\geq 2f - 1$ correct votes in T.

After entry, $guard_i[V + 1]$ can change only through the divergence rule (line 72), which requires $\langle \text{ADOPT}, V, c_1 \rangle$ and $\langle \text{ADOPT}, V, c_2 \rangle$ messages, with $c_1 \neq c_2$, from $2f - 1$ distinct senders each, where $V \geq V_{dec}$. Each batch contains a correct sender, so by the inductive hypothesis $c_1 = v_{dec} = c_2$, which is a contradiction. ◇

Subclaim 2. *If $m = \langle \text{VOTE}, V, x, g \rangle$ with $V > V_{dec}$, then $g = v_{dec}$ and $x \in \{v_{dec}, \perp\}$.*

Proof of Subclaim 2. We distinguish the site at which m is created. Suppose first that g is the guard held for a view $V' \in \{V + 1, V + 2\}$: this happens when m is created by the backfill invocation of cast (line 40), where $g = G$ is the guard with which p_i enters view V', and when m is created by the recovery rule while $view_i = V' > V$ (line 76), where g is the value of $guard_i[V']$ at that moment. In both cases, $x = \perp$ and, since $V' > V > V_{dec}$, Subclaim 1 yields $g = v_{dec}$. Otherwise, m is created by an invocation of cast other than the backfill, or by the recovery rule with $view_i = V$, hence while $view_i = V$ (see the proof of Claim 2), i.e., after p_i entered view $V > V_{dec}$; by Subclaim 1, $guard_i[V] = v_{dec}$ throughout p_i's stay in view V. All these sites attach the value of $guard_i[V]$ as the guard: the recovery rule directly (line 76), the invocations at lines 45 and 47 by passing $guard_i[view_i]$ with $view_i = V$, and the invocations at lines 58–60 by passing the guard written upon the entry into view V that immediately precedes them (line 38). Hence, $g = v_{dec}$. As for the ballot: the rules at lines 47, 59, 60, and 76 cast $x = \perp$; the rule at line 45 requires $guard_i[V] \in \{\text{SAFE}, x\}$ (line 44), and since $guard_i[V] = v_{dec} \neq \text{SAFE}$, it casts $x = v_{dec}$; and the rule at line 58 casts a value x exhibited by at least $2f - 1$ ballots of its trigger set T, of which at least $f - 1 \geq 1$ are correct, so $x = v_{dec}$ by the inductive hypothesis. ◇

Subclaim 3. *If $m = \langle \text{ADOPT}, V, c \rangle$ with $V \geq V_{dec}$, then $c = v_{dec}$.*

Proof of Subclaim 3. We analyze the four rules that create adopt messages.

- *Line 63* (unique(V, c) holds at p_i). If $V = V_{dec}$, then $c = v_{dec}$ by Claim 6(1). If $V > V_{dec}$, then c is exhibited as a ballot by at least $2f - 1$ distinct senders of some sample, hence by at least $f - 1 \geq 1$ correct processes; by Subclaim 2 (applied through the inductive order), $c = v_{dec}$.
- *Line 65* (equiv(V) and survivor(V, c) hold at p_i). If $V = V_{dec}$, then $c = v_{dec}$ by Claim 6(4). If $V > V_{dec}$, then c is exhibited as a ballot by at least $2f$ senders of some non-leader sample, hence by at least $2f - f = f \geq 1$ correct processes, so $c = v_{dec}$ as above.
- *Line 67* (barren(V) and dominant(V, c) hold at p_i). The case $V = V_{dec}$ is impossible by Claim 6(6). If $V > V_{dec}$, then c is carried as a guard by at least $3f - 1$ distinct senders' votes of some sample, hence by at least $3f - 1 - f = 2f - 1 \geq 1$ correct votes; by the inductive hypothesis (part 2), $c = v_{dec}$.
- *Line 69* (barren(V) and no_dominant(V) hold at p_i; $c = \text{SAFE}$). The case $V = V_{dec}$ is impossible by Claim 6(6). The case $V > V_{dec}$ is impossible as well: every vote sample for view V contains votes from at least $(4f - 1) - f = 3f - 1$ correct processes, all of which carry guard v_{dec} by the inductive hypothesis, so v_{dec} is dominant in *every* sample and no_dominant(V) cannot hold.

In all cases $c = v_{dec}$, which concludes the proof. ◇

Since the subclaims cover all message types, the proof is complete. □

We are finally ready to prove that Floor-IT satisfies agreement.

Lemma 1 (Agreement)**.** Floor-IT (Algorithm 1) satisfies the agreement property.

Proof. We show that no correct process decides a value different from v_{dec} in any view $V \geq V_{dec}$; since V_{dec} is the smallest view with a decision, this proves agreement. In view V_{dec} itself, this follows from Claim 3. For any view $V > V_{dec}$, a decision on a value y requires votes with ballot $y \neq \perp$ from $4f - 1$ distinct senders (line 48), of which at least $3f - 1 \geq 1$ are correct. By Claim 7(2), every ballot cast by a correct process in view V belongs to $\{v_{dec}, \perp\}$. Hence, $y = v_{dec}$ and agreement holds. □

4.2 Termination

In this subsection, we prove that Floor-IT satisfies the termination property, meaning that all correct processes eventually decide. We first prove that ADOPT messages for any view V can only exist if some correct process has previously entered view V.

Claim 8. If a correct process broadcasts an $\langle \text{ADOPT}, V, \cdot \rangle$ message for a view $V \geq 1$, then some correct process has previously entered view V.

Proof. If $V = 1$, the claim is immediate: every correct process enters view 1 at the start of the execution (line 36), before broadcasting any message. Hence, assume $V \geq 2$.

We rely on two observations. First, for any view $U \geq 1$, consider the first time a correct process broadcasts an $\langle \text{ADOPT}, U, \cdot \rangle$ message: this broadcast is initiated by one of the rules at lines 62–68, since the recovery rule (line 78) only re-initiates broadcasts performed earlier. Consequently, a vote sample for U precedes it: votes for U from $4f - 1$ distinct senders, hence from at least $3f - 1$ correct processes, all broadcast beforehand. Second, by Claim 2(3), a correct process broadcasts a vote for view U only while its current view belongs to $\{U, U + 1, U + 2\}$; as $view_i$ is set to a view only upon entering it (line 38), every correct process that broadcasts a vote for view U has previously entered a view $\geq U$. Combining the two observations, some correct process enters a view $\geq V$ before the assumed $\langle \text{ADOPT}, V, \cdot \rangle$ message is broadcast.

Let $\mathcal{E}^*$ denote the first entry by a correct process into a view $\geq V$, and let $Y \geq V$ be the entered view; since $Y \geq 2$, the entry $\mathcal{E}^*$ occurs at one of the view-entry rules. It suffices to show that $Y = V$.

The entry $\mathcal{E}^*$ does not occur at the catch-up rule (line 57): the rule's trigger contains votes for view Y from $3f-1$ distinct senders, hence from at least $2f-1 \geq 1$ correct processes, each of which, by the second observation, entered a view $\geq Y \geq V$ before broadcasting its vote, and thus before $\mathcal{E}^*$, contradicting the fact that $\mathcal{E}^*$ is first. Hence, $\mathcal{E}^*$ occurs at line 52, 54, or 56, and its entry evidence contains $\langle \text{ADOPT}, Y-1, \cdot \rangle$ messages from at least $2f-1$ distinct senders, hence from at least $f-1 \geq 1$ correct processes. By the two observations, votes for view $Y-1$ from at least $3f-1$ correct processes precede $\mathcal{E}^*$, and each of their senders has previously entered a view in $\{Y-1, Y, Y+1\}$. As entries into views Y and $Y+1$ (both $\geq V$) cannot precede $\mathcal{E}^*$, these senders entered view $Y-1$; moreover, $Y-1 < V$, as an entry into view $Y-1$ with $Y-1 \geq V$ cannot precede $\mathcal{E}^*$ either. Therefore, $Y-1 < V \leq Y$, i.e., $Y = V$, which completes the proof. □

An analogous statement holds for VOTE messages.

Claim 9. If a correct process broadcasts a $\langle \text{VOTE}, V, \cdot, \cdot \rangle$ message for a view $V \geq 1$, then some correct process has previously entered view V.

PROOF. If $V = 1$, the claim is immediate: every correct process enters view 1 at the start of the execution (line 36), before broadcasting any message. Therefore, let us assume $V \geq 2$.

By Claim 2(3), the broadcasting process initiates the broadcast at a time when $V \in \{view_i - 2, view_i - 1, view_i\}$; as $view_i$ is set to a view only upon entering it (line 38), some correct process enters a view $\geq V$ before the broadcast. Let $\mathcal{E}^*$ denote the first entry by a correct process into a view $\geq V$, and let $Y \geq V$ be the entered view; since $Y \geq 2$, the entry $\mathcal{E}^*$ occurs at one of the view-entry rules. It suffices to show that $Y = V$.

The entry $\mathcal{E}^*$ does not occur at the catch-up rule (line 57): the rule's trigger contains votes for view Y from $3f-1$ distinct senders, hence from at least $2f-1 \geq 1$ correct processes, each of which, by Claim 2(3), broadcast its vote while its current view was at least $Y \geq V$, and had therefore entered a view $\geq V$ before $\mathcal{E}^*$, contradicting the fact that $\mathcal{E}^*$ is first. Hence, $\mathcal{E}^*$ occurs at line 52, 54, or 56, and its trigger contains $\langle \text{ADOPT}, Y-1, \cdot \rangle$ messages from at least $2f-1$ distinct senders, hence from at least $f-1 \geq 1$ correct processes. By Claim 8, some correct process enters view $Y-1$ before any such message is broadcast, and thus before $\mathcal{E}^*$. By the minimality of $\mathcal{E}^*$, $Y-1 < V$; combined with $Y \geq V$, this yields $Y = V$, which completes the proof. □

Next, we show that if any correct process enters view $V > 1$, then some correct process previously entered view $V-1$.

Claim 10. If any correct process enters any view $V > 1$, then some correct process previously entered view $V-1$.

PROOF. Let $\mathcal{E}$ denote the first entry by a correct process into view V; since $V > 1$, the entry $\mathcal{E}$ does not occur at line 36, so it occurs at one of the view-entry rules. The entry $\mathcal{E}$ does not occur at the catch-up rule (line 57): the rule's trigger contains votes for view V from $3f-1$ distinct senders, hence from at least $2f-1 \geq 1$ correct processes; by Claim 9, some correct process would then have entered view V before those votes were broadcast, and thus before $\mathcal{E}$, contradicting the fact that $\mathcal{E}$ is first. Hence, $\mathcal{E}$ occurs at line 52, 54, or 56, and its trigger contains $\langle \text{ADOPT}, V-1, \cdot \rangle$ messages from at least $2f-1$ distinct senders, hence from at least $f-1 \geq 1$ correct processes. By Claim 8, some correct process enters view $V-1$ before any such message is broadcast, and thus before $\mathcal{E}$, which precedes every entry into view V by a correct process. □

We next prove that correct processes keep advancing through views. To this end, we first show that if no correct process ever advances past some view V, then every non-PROPOSE message broadcast by a correct process for view V or view $V-1$ is forever-alive.

Claim 11. Let $V \geq 1$ be a view such that no correct process ever enters any view greater than V. If a correct process broadcasts a VOTE or an ADOPT message m for view V or view $V-1$, then m is forever-alive.

PROOF. Let p_i denote the correct process broadcasting m, and let $W \in \{V-1, V\}$ denote m's view. First, we show that m resides in p_i's persistent memory, verbatim, from no later than the first initiation of its broadcast and forever onward. Indeed, every broadcast of m transmits a stored record, written no later than the initiation: $vote_i[W]$ for a vote (lines 34 and 77), and an outcome $c \in adopted_i[W]$ for $m = \langle \text{ADOPT}, W, c \rangle$ (lines 63–69 and 78). This record is written while $W \in \{view_i - 2, view_i - 1, view_i\}$: for votes, this is Claim 2(2), while every rule extending $adopted_i[W]$ requires it explicitly (lines 62–68). Hence, $W \leq view_i$ at the write; as $view_i$ never decreases (Claim 1) and never exceeds $V \leq W + 1$, $W \in \{view_i - 1, view_i\}$ holds from the write onward, so the three-view rule (line 22) never discards the record. Moreover, the record never changes: $vote_i[W]$ is written only if $vote_i[W] = \perp$ (lines 34 and 76), whereas $adopted_i[W]$ only grows.

Second, we show that no broadcast of m is ever explicitly abandoned. The rule at line 23 abandons broadcasts only when p_i enters a view; consider any entry by p_i into a view V' that follows an initiation of m's broadcast. By the first step, $W \leq view_i$ at the initiation, so $V' > view_i \geq W$ (Claim 1), while $V' \leq V \leq W + 1$ by assumption. Hence, $V' = W + 1$, and the exception at line 23 spares the broadcast of m.

Third, we exhibit an initiated broadcast of m that is never implicitly abandoned. A crash of p_i implicitly abandons all its ongoing broadcasts; however, crashes occur only prior to GST, so there is a time after which p_i never crashes. If p_i never crashes after first initiating the broadcast of m, that broadcast is never implicitly abandoned. Otherwise, the final crash of p_i occurs after the first initiation, and thus after m is stored; upon the final recovery, m is still in persistent memory with $W \in \{view_i - 2, view_i - 1, view_i\}$ (by the first step), so the recovery procedure initiates a broadcast of m (line 77 or 78), and no subsequent crash abandons it. In both cases, the exhibited broadcast is, by the second step, never explicitly abandoned either; thus, m is forever-alive. □

Next, we prove that a correct process that never leaves a view eventually broadcasts a vote message for it.

Claim 12. If a correct process p_i enters a view $V \geq 1$ and never enters any view greater than V, then p_i eventually broadcasts a VOTE message for view V.

PROOF. First, note that $view_i = V$ forever after the entry: $view_i$ changes only upon entering a view (line 38), entered views strictly increase (Claim 1), and views greater than V are excluded by assumption. Suppose, toward a contradiction, that p_i never broadcasts a VOTE message for view V. Then, $vote_i[V] = \perp$ forever: $vote_i[V]$ is written only at line 34, which broadcasts it on the spot, and at line 76, whose run of the recovery procedure broadcasts it at line 77. If p_i crashes after entering view V, then, upon its final recovery, $view_i = V$ and $vote_i[V] = \perp$, so the recovery procedure writes $vote_i[V]$ (line 76) and broadcasts it (line 77), a contradiction. Otherwise, p_i never crashes after entering view V; its timer $timer_i[V]$, started upon entry (line 38), advances at the rate of real time from GST onward and therefore eventually reaches 2Δ, at which point the rule at line 46 invokes $\text{cast}(V, \perp, guard_i[V])$ (line 47), which writes $vote_i[V]$ and broadcasts it (line 34), again a contradiction. □

The next claim shows that the first correct process to enter any view $V > 1$ does so upon receiving ADOPT messages (and not upon receiving VOTE messages).

Claim 13. If some correct process enters a view $V > 1$, then the first such entry occurs at line 52, 54, or 56, and never at the catch-up rule (line 57).

Proof. Suppose, by contradiction, that the first correct process to enter view $V > 1$ does so at line 58, 59, or 60. That means it received $\langle \text{VOTE}, V, \cdot, \cdot \rangle$ messages from $3f-1$ distinct senders (line 57), out of which at least $2f-1 \geq 1$ are broadcast by correct processes. By Claim 9, some correct process entered view V before those messages were broadcast, and thus before the considered entry, which contradicts the fact that this is the first correct process to enter view V. Hence, as $V > 1$ excludes the initial entry (line 36), the claim holds. □

We now prove that if some correct process enters a view V beyond which no correct process ever advances, then every correct process eventually enters view V or view $V-1$.

Claim 14. Let $V \geq 1$ be a view such that (1) some correct process enters view V, and (2) no correct process ever enters any view greater than V. Then, every correct process eventually enters view V or view $V-1$.

Proof. If $V = 1$, then every correct process enters view $V = 1$ upon starting the execution of the protocol (line 36). Hence, let $V > 1$.

Due to Claim 13, the first correct process that enters view V does so at line 52, 54, or 56, and has thus received $\langle \text{ADOPT}, V-1, \cdot \rangle$ messages from at least $2f-1$ distinct senders, out of which at least $f-1 \geq 1$ are correct. The first such $\langle \text{ADOPT}, V-1, \cdot \rangle$ message from a correct process could not have been broadcast via the recovery rule (line 78), which only re-initiates broadcasts performed earlier; hence, it was broadcast via one of the rules at lines 62–68, each of which requires a vote sample for view $V-1$: $\langle \text{VOTE}, V-1, \cdot, \cdot \rangle$ messages from $4f-1$ distinct senders, out of which at least $3f-1$ are broadcast by correct processes. All these $3f-1$ votes are forever-alive (Claim 11), so every correct process p_i eventually receives them. If $view_i \geq V-1$ at that point, then p_i has already entered view $V-1$ or view V, and the claim follows. Otherwise, the catch-up rule (line 57) activates, and p_i enters view $V-1$ (line 58, 59, or 60). □

Next, we prove that if every correct process eventually enters view V or view $V-1$, and no correct process ever enters any view greater than V, then every correct process eventually enters view V.

Claim 15. Let $V \geq 1$ be a view such that (1) every correct process eventually enters view V or view $V-1$ (if $V-1 \geq 1$), and (2) no correct process ever enters any view greater than V. Then, every correct process eventually enters view V.

Proof. If $V = 1$, then every correct process enters view $V = 1$ upon starting the execution of the protocol (line 36), and the claim trivially holds. Therefore, suppose $V > 1$.

First, we show that every correct process eventually broadcasts its vote for view $V-1$. Consider any correct process p_i, and distinguish two scenarios:

- Suppose that p_i eventually enters view V. Then, p_i broadcasts its vote for view $V-1$ at line 40 at the latest, unless it has broadcast it earlier.
- Suppose that p_i never enters view V. By assumption (1), p_i then eventually enters view $V-1$; moreover, p_i never enters any view greater than $V-1$, as such an entry would target view V or a view greater than V, both excluded. Hence, by Claim 12, p_i eventually broadcasts its vote for view $V-1$.

Second, let D denote the set of the $4f-1$ correct processes with the smallest indices, and let R contain their broadcast votes for view $V-1$. As shown above, $|R| = 4f-1$, and each is forever-alive by Claim 11; hence, every correct process eventually receives all of R, at which point R constitutes a

vote sample S for view $V-1$ (line 8), identical at every correct process. We now exhibit an outcome c, common to all correct processes, for which one of the rules at lines 62–68 applies:

- If $\mathsf{Heavy}(S) = \{x\}$, then $\mathsf{unique}(V-1, x)$ holds with S as its witness, and the rule at line 62 applies with $c = x$.
- If $\mathsf{Heavy}(S) = \emptyset$, then $\mathsf{empty}(V-1)$, and hence $\mathsf{barren}(V-1)$, holds. If some guard g is carried by at least $3f-1$ votes of S, then $\mathsf{dominant}(V-1, g)$ holds and the rule at line 66 applies with $c = g$; otherwise, S witnesses $\mathsf{no_dominant}(V-1)$ and the rule at line 68 applies with $c = \mathsf{SAFE}$.
- If $|\mathsf{Heavy}(S)| \geq 2$, then $\mathsf{equiv}(V-1)$ holds. Moreover, two distinct values appear as ballots on the votes of S, all cast by correct processes; by Claim 5, $\mathsf{leader}(V-1)$ therefore sent two different proposals, and, as a correct leader broadcasts at most one proposal per view (lines 42 and 43; views are entered at most once by Claim 1), $\mathsf{leader}(V-1)$ is faulty. Consequently, S is also a non-leader sample for view $V-1$ (line 9). If some value x has $count_S(x) \geq 2f$, then $\mathsf{survivor}(V-1, x)$ holds and the rule at line 64 applies with $c = x$; otherwise, S witnesses $\mathsf{no_survivor}(V-1)$, so $\mathsf{barren}(V-1)$ holds, and we conclude with line 66 or line 68 as in the previous case.

The remaining conditions of the exhibited rule are eventually satisfied at every correct process p_j: view $V-1$ lies within its window, as $view_j \in \{V-1, V\}$ forever; and the adopt gate (line 24) eventually opens, as either p_j enters view V, making $V-1 < view_j$, or p_j remains in view $V-1$ forever and $timer_j[V-1]$ eventually reaches 2Δ. Hence, once past its final crash, every correct process broadcasts $\langle \textsc{adopt}, V-1, c\rangle$: the rule fires unless $c \in adopted_j[V-1]$ already, in which case the broadcast accompanied the insertion (lines 63–69). Every such message is forever-alive by Claim 11.

Finally, every correct process p_i eventually receives $\langle \textsc{adopt}, V-1, c\rangle$ from the $4f-1$ distinct senders in D. If $view_i = V$ at that point, then p_i has already entered view V. Otherwise, $view_i \leq V-1$ and, once p_i never crashes again, the rule at line 51 (if $p_i \neq \mathsf{leader}(V)$) or at line 53 (if $p_i = \mathsf{leader}(V)$, with the lower threshold of $3f-1$) fires, and p_i enters view V. The claim holds. □

We are finally ready to prove that all views are eventually entered by a correct process.

Claim 16. For every view $V \geq 1$, at least one correct process eventually enters view V.

Proof. Suppose, by contradiction, that some view is never entered by any correct process, and let $V_{\max}$ denote the greatest view such that every view $\leq V_{\max}$ is entered by a correct process; $V_{\max}$ is well-defined, as every correct process enters view 1 upon starting (line 36) and, by Claim 10, the set of views entered by correct processes is closed downward. By the maximality of $V_{\max}$ and Claim 10, no correct process ever enters any view greater than $V_{\max}$. Hence, by Claim 14, every correct process eventually enters view $V_{\max}$ or view $V_{\max}-1$, and, by Claim 15, every correct process eventually enters view $V_{\max}$, where it remains forever.

We now apply the argument of Claim 15 to view $V_{\max}$ itself. Every correct process eventually broadcasts a vote for view $V_{\max}$ (Claim 12), and every such vote is forever-alive (Claim 11). As in the proof of Claim 15, let D denote the set of the $4f-1$ correct processes with the smallest indices: their votes eventually form a vote sample S for view $V_{\max}$, identical at every correct process, and the case analysis over $\mathsf{Heavy}(S)$ yields an outcome c, common to all correct processes, for which one of the rules at lines 62–68 applies. The rule's remaining conditions are eventually satisfied at every correct process p_j: view $V_{\max} = view_j$ lies within its window, and the adopt gate (line 24) opens once $timer_j[V_{\max}]$ reaches 2Δ, which eventually happens as p_j remains in view $V_{\max}$ forever. Hence, every correct process eventually broadcasts $\langle \textsc{adopt}, V_{\max}, c\rangle$, either upon the rule firing

or, if c entered $adopted_j[V_{\max}]$ earlier, upon that insertion; every such message is forever-alive (Claim 11).

Finally, let p_j be a correct process with $p_j \neq \mathsf{leader}(V_{\max}+1)$; one exists, as at least $4f-1 \geq 2$ processes are correct. Process p_j eventually receives $\langle \textsc{adopt}, V_{\max}, c\rangle$ from the $4f-1$ distinct senders in D while $view_j = V_{\max}$; the rule at line 51 therefore fires, and p_j enters view $V_{\max}+1$, contradicting the maximality of $V_{\max}$. □

Throughout the remainder of the proof, for every view $V \geq 1$, let $\tau(V)$ denote the earliest time at which a correct process enters view V; by Claim 16, $\tau(V)$ is well-defined, and, by Claim 10, $\tau(V) < \tau(V+1)$. We also define *post-GST* views as follows.

Definition 3 (Post-GST view)**.** We say that a view $V \geq 1$ is a *post-GST* view if and only if $\tau(V) \geq \text{GST}$.

We next prove that no correct process broadcasts an ADOPT message "too early".

Claim 17. For any post-GST view $V \geq 1$, no correct process broadcasts an $\langle \textsc{adopt}, V, \cdot\rangle$ message before time $\tau(V) + 2\Delta$.

Proof. Consider the first $\langle \textsc{adopt}, V, \cdot\rangle$ message broadcast by a correct process p_i, at some time τ_a (if no such message exists, the claim trivially holds). Its broadcast is not initiated by the recovery rule (line 78), which only re-initiates broadcasts performed earlier. Hence, the broadcast is initiated by one of the rules at lines 62–68, all subject to the adopt gate (line 24): $V < view_i$ or $timer_i[V] \geq 2\Delta$ at time τ_a.

Suppose $V < view_i$ at time τ_a. Then, p_i has previously entered a view greater than V, so, by repeated applications of Claim 10, some correct process enters view $V+1$ before τ_a, i.e., $\tau(V+1) < \tau_a$. By Claim 13, the entry at time $\tau(V+1)$ occurs at line 52, 54, or 56; each of these rules requires $\langle \textsc{adopt}, V, \cdot\rangle$ messages from at least $2f-1$ distinct senders (lines 51, 53, and 55), hence from at least $f-1 \geq 1$ correct processes, broadcast before $\tau(V+1) < \tau_a$, contradicting the fact that the considered message is first.

Hence, $timer_i[V] \geq 2\Delta$ at time τ_a, so p_i has entered view V: the timer $timer_i[V]$ is started upon that entry (line 38), at some time $e \geq \tau(V) \geq \text{GST}$. From GST onward, p_i neither crashes nor suffers clock drift, so $timer_i[V] \geq 2\Delta$ implies $\tau_a \geq e + 2\Delta \geq \tau(V) + 2\Delta$. □

A direct consequence of the previous claim is that consecutive post-GST views are separated by at least 2Δ time.

Claim 18. For any post-GST view $V \geq 1$, $\tau(V+1) \geq \tau(V) + 2\Delta$.

Proof. The entry occurring at time $\tau(V+1)$ is the first entry by a correct process into view $V+1 > 1$; by Claim 13, it occurs at line 52, 54, or 56. Each of these rules requires $\langle \textsc{adopt}, V, \cdot\rangle$ messages from at least $2f-1$ distinct senders (lines 51, 53, and 55), hence from at least $f-1 \geq 1$ correct processes; these messages are received by the entrant, and thus broadcast, by $\tau(V+1)$. Due to Claim 17, no correct process broadcasts an $\langle \textsc{adopt}, V, \cdot\rangle$ message before time $\tau(V) + 2\Delta$; therefore, $\tau(V+1) \geq \tau(V) + 2\Delta$. □

The following claim shows that ADOPT messages for the previous view remain alive throughout the current view, provided the current view is a post-GST view.

Claim 19. Consider any post-GST view $V > 1$, and suppose that a correct process broadcasts an $\langle \textsc{adopt}, V-1, \cdot\rangle$ message m by some time $\tau \in [\tau(V), \tau(V+1)]$. Then, m is alive throughout $[\tau, \tau(V+1)]$.

Proof. Let p_i denote the correct process broadcasting $m = \langle \text{ADOPT}, V-1, c\rangle$. Before m is broadcast, the outcome c is stored in $adopted_i[V-1]$ (lines 63–69), which thus occurs by time τ; moreover, every initiation of m's broadcast happens while $V-1 \in \{view_i - 2, view_i - 1, view_i\}$, i.e., while $view_i \in \{V-1, V, V+1\}$ (lines 62–68 and 75). By repeated applications of Claim 10, no correct process enters a view greater than V before time $\tau(V+1)$; as entered views only increase (Claim 1), $view_i \in \{V-1, V\}$ holds from the first initiation of m's broadcast until time $\tau(V+1)$. Consequently, m is not explicitly abandoned before time $\tau(V+1)$: the only view p_i can enter in this period is view V, whose entry "spares" the ongoing broadcasts of messages from view $V-1$ (line 23). For the same reason, the three-view rule (line 22) does not discard the variables of view $V-1$ before time $\tau(V+1)$, so c remains stored in $adopted_i[V-1]$ throughout.

Finally, let $\tau_b \leq \tau$ denote the last time at which a broadcast of m is initiated by time τ. Crashes occur only before GST $\leq \tau(V) \leq \tau$; if a crash struck after τ_b, the subsequent recovery, which occurs before GST and hence by τ, would re-initiate the broadcast of m (line 78, as c is stored with view $V-1$ in the window), contradicting the choice of τ_b. Hence, no crash strikes after τ_b: the broadcast initiated at time τ_b is never implicitly abandoned, nor, as shown above, explicitly abandoned before time $\tau(V+1)$. Since $\tau_b \leq \tau$, message m is alive throughout $[\tau, \tau(V+1)]$. □

Next, we show that a correct leader enters its post-GST view "quickly": within δ time of the view's first correct entrant.

Claim 20. Let $V \geq 1$ be a post-GST view whose leader $p_\ell = \text{leader}(V)$ is correct. Then, p_ℓ enters view V by time $\tau(V) + \delta$.

Proof. If $V = 1$: every correct process starts executing by GST and enters view 1 upon starting (line 36), so $\tau(1) \leq$ GST; as V is post-GST, $\tau(1) =$ GST, and every correct process, p_ℓ included, starts and enters view 1 exactly at time $\tau(1)$. Hence, let $V > 1$.

Let p_i denote the first correct process to enter view V, doing so at time $\tau(V)$. If $p_i = p_\ell$, the claim trivially holds; thus, let $p_i \neq p_\ell$. By Claim 13, p_i enters view V at line 52, 54, or 56. It cannot enter at line 54, as $p_i \neq \text{leader}(V)$. It cannot enter at line 56 either: this would require a $\langle \text{PROPOSE}, V, \cdot\rangle$ message from p_ℓ, which the correct p_ℓ broadcasts only upon entering view V (lines 42 and 43), impossible before time $\tau(V)$. Therefore, p_i enters view V at line 52, having received $\langle \text{ADOPT}, V-1, c\rangle$ messages, for some outcome c, from $4f-1$ distinct senders (line 51); at least $3f-1$ of them are broadcast by correct processes by time $\tau(V)$.

By Claim 19 (with $\tau = \tau(V)$), all these $3f-1$ messages are alive throughout $[\tau(V), \tau(V+1)]$; as $\tau(V+1) \geq \tau(V) + 2\Delta$ (Claim 18) and $\delta \leq \Delta$, they are alive throughout $[\tau(V), \tau(V)+\delta]$. Since GST $\leq \tau(V)$, every correct process, p_ℓ included, therefore receives all of them by time $\tau(V) + \delta$. Consider the moment p_ℓ holds all $3f-1$ messages, at some time in $[\tau(V), \tau(V)+\delta]$. By repeated applications of Claim 10, no correct process enters a view greater than V before time $\tau(V+1) \geq \tau(V) + 2\Delta > \tau(V) + \delta$; hence, $view_\ell \leq V$ at that moment. If $view_\ell = V$, then p_ℓ has already entered view V, and the claim follows. Otherwise, $view_\ell \leq V-1$, so the rule at line 53 activates, with the $3f-1$ matching $\langle \text{ADOPT}, V-1, c\rangle$ messages and $p_\ell = \text{leader}(V)$, and p_ℓ enters view V (line 54) by time $\tau(V) + \delta$. □

We are finally ready to prove Floor-IT's termination property.

Lemma 2 (Termination)**.** Floor-IT (Algorithm 1) satisfies the termination property and achieves a robust good-case latency of 2δ.

Proof. Let $V^\star$ denote the first post-GST view whose leader $p_\ell = \text{leader}(V^\star)$ is correct. Such a view exists: as every process performs finitely many steps by any given time, only finitely many views are entered before GST, so, by Claim 16, post-GST views exist; moreover, the leader role

rotates in a round-robin fashion and at least $4f - 1 \geq 1$ processes are correct. Let τ_ℓ denote the time at which p_ℓ enters view $V^\star$; by Claim 20, $\tau_\ell \leq \tau(V^\star) + \delta$. Throughout the proof, we use the following bounds: by Claim 18 (applied to the post-GST views $V^\star$, $V^\star + 1$, and $V^\star + 2$), $\tau(V^\star + 1) \geq \tau(V^\star) + 2\Delta$, $\tau(V^\star + 2) \geq \tau(V^\star) + 4\Delta$, and $\tau(V^\star + 3) \geq \tau(V^\star) + 6\Delta$; moreover, by repeated applications of Claim 10, no correct process is in a view greater than $V^\star$ before time $\tau(V^\star + 1)$.

We first show that, if $V^\star > 1$, process p_ℓ enters view $V^\star$ at line 54; consequently, its guard $c_\ell = guard_\ell[V^\star]$, written upon that entry (line 38), is an outcome for which p_ℓ received matching $\langle \textsc{adopt}, V^\star - 1, c_\ell \rangle$ messages from $3f - 1$ distinct senders, at least $2f - 1$ of them correct, all broadcast before τ_ℓ. The rules at lines 51 and 55 exclude leader$(V^\star)$, and line 36 concerns view 1 only. An entry via the catch-up rule (line 57) at time τ_ℓ would require votes for view $V^\star$ from $3f - 1$ distinct senders, hence from at least $2f - 1 \geq 1$ correct processes, broadcast before τ_ℓ; however, no correct process broadcasts a vote for view $V^\star$ before τ_ℓ. Indeed, consider the first such vote: the rule at line 44 presupposes a $\langle \textsc{propose}, V^\star, \cdot \rangle$ message from p_ℓ, broadcast only upon p_ℓ's entry (lines 42 and 43), hence not before τ_ℓ; the timeout (line 47) fires at the earliest 2Δ after an entry into view $V^\star$, i.e., not before $\tau(V^\star) + 2\Delta > \tau_\ell$; the backfill (line 40) presupposes an entry into view $V^\star + 1$ or $V^\star + 2$, impossible before $\tau(V^\star + 1) > \tau_\ell$; the catch-up branches (lines 58–60) presuppose earlier correct votes for view $V^\star$; and the recovery rule (line 76) runs only before GST, when no correct process has entered view $V^\star$. Hence, p_ℓ enters at line 54, and its trigger (line 53) supplies the claimed $\langle \textsc{adopt}, V^\star - 1, c_\ell \rangle$ messages. Upon entry, p_ℓ broadcasts a single $\langle \textsc{propose}, V^\star, v^\star \rangle$ message, where $v^\star$ is p_ℓ's own proposal if $c_\ell = \mathsf{SAFE}$, and $v^\star = c_\ell$ otherwise (lines 42 and 43); in particular, $v^\star \in \mathsf{Value}$. If $V^\star = 1$, every correct process enters view 1 at time $\tau(1) = \tau_\ell$ with guard SAFE (Claim 20, line 36), and $v^\star$ is p_ℓ's own proposal.

Next, we show that, by time $\tau_\ell + \delta$, every correct process has entered view $V^\star$ and received both the $\langle \textsc{propose}, V^\star, v^\star \rangle$ message and, if $V^\star > 1$, $\langle \textsc{adopt}, V^\star - 1, c_\ell \rangle$ messages from at least $2f - 1$ distinct senders. For the proposal: p_ℓ initiates its broadcast at time $\tau_\ell \geq$ GST and, being correct, never crashes afterwards; the broadcast is abandoned only upon p_ℓ entering view $V^\star + 3$ (line 23), which does not happen by $\tau_\ell + \delta < \tau(V^\star) + 6\Delta$. Hence, the proposal is alive throughout $[\tau_\ell, \tau_\ell + \delta]$ and received by every correct process by time $\tau_\ell + \delta$. For the $\langle \textsc{adopt}, V^\star - 1, c_\ell \rangle$ messages (case $V^\star > 1$): they are broadcast by time $\tau_\ell \in [\tau(V^\star), \tau(V^\star + 1)]$, so, by Claim 19, they are alive throughout $[\tau_\ell, \tau(V^\star + 1)] \supseteq [\tau_\ell, \tau_\ell + \delta]$ and received by every correct process by time $\tau_\ell + \delta$. For the entries: consider any correct process p_j that has not entered view $V^\star$ by time $\tau_\ell + \delta$; since $\tau_\ell + \delta \leq \tau(V^\star) + 2\delta \leq \tau(V^\star + 1)$, we have $view_j \leq V^\star - 1$. By the above, p_j then holds the $\langle \textsc{propose}, V^\star, \cdot \rangle$ message from leader$(V^\star) \neq p_j$ and $\langle \textsc{adopt}, V^\star - 1, c_\ell \rangle$ messages from $2f - 1$ distinct senders, so the rule at line 55 fires and p_j enters view $V^\star$. Hence, every correct process enters view $V^\star$ at some time in $[\tau(V^\star), \tau_\ell + \delta]$.

Step 1: Every correct process broadcasts a $\langle \textsc{vote}, V^\star, v^\star, \cdot \rangle$ message by time $\tau_\ell + \delta$, and each of these messages is alive from its broadcast until time $\tau(V^\star + 2)$.
We first show that every correct process casts its vote for view $V^\star$ by time $\tau_\ell + \delta$ (line 34). To begin, note that no correct process casts a vote for view $V^\star$ at line 40, 47, or 76 by that time: the backfill presupposes an entry into view $V^\star + 1$ or $V^\star + 2$, impossible before $\tau(V^\star + 1) \geq \tau(V^\star) + 2\Delta$; the timeout of a correct process p_i fires at time $e_i + 2\Delta \geq \tau(V^\star) + 2\Delta$, where e_i denotes p_i's entry time into view $V^\star$; and the recovery rule runs only before GST, when no correct process has entered view $V^\star$; in the first two cases, no earlier than $\tau(V^\star) + 2\Delta \geq \tau(V^\star) + 2\delta \geq \tau_\ell + \delta$. Now, consider any correct process p_i. If p_i enters view $V^\star$ via the catch-up rule, it casts its vote upon entry (lines 58–60), which occurs by $\tau_\ell + \delta$ (as shown above). Otherwise, p_i enters view $V^\star$ with some guard $c_i = guard_i[V^\star]$, written upon entry (line 38; $c_i = \mathsf{SAFE}$ if $V^\star = 1$); by time $\tau_\ell + \delta$, process p_i has entered view $V^\star$ and holds the proposal (as shown above), while

$timer_i[V^\star] \leq \tau_\ell + \delta - \tau(V^\star) \leq 2\delta \leq 2\Delta$. If $guard_i[V^\star] \in \{\text{SAFE}, v^\star\}$ at any point by then, the rule at line 44 fires, and p_i casts its vote by time $\tau_\ell + \delta$. Otherwise, $c_i \notin \{\text{SAFE}, v^\star\}$; then $V^\star > 1$ and $c_i \neq c_\ell$: if $c_\ell = \text{SAFE}$, then $c_i \neq \text{SAFE} = c_\ell$, and if $c_\ell \neq \text{SAFE}$, then $v^\star = c_\ell \neq c_i$. Process p_i entered view $V^\star$ at line 52 or 56 (line 54 concerns only p_ℓ, whose guard satisfies $c_\ell \in \{\text{SAFE}, v^\star\}$ by the definition of $v^\star$), so its trigger supplies $\langle \text{ADOPT}, V^\star - 1, c_i \rangle$ messages from at least $2f - 1$ distinct senders; as shown above, p_i also holds $\langle \text{ADOPT}, V^\star - 1, c_\ell \rangle$ messages from $2f - 1$ distinct senders by time $\tau_\ell + \delta$. As $c_i \neq c_\ell$ and $view_i = V^\star = (V^\star - 1) + 1$, the divergence rule (line 71) fires by time $\tau_\ell + \delta$, setting $guard_i[V^\star]$ to SAFE (line 72); the rule at line 44 then fires, and p_i casts its vote by time $\tau_\ell + \delta$.

We now show that every vote cast by a correct process for view $V^\star$ carries ballot $v^\star$; we proceed by induction on the order in which these votes are cast. By the above, every correct process casts its single vote for view $V^\star$ (line 21) by time $\tau_\ell + \delta$, and thus at line 45 or upon a catch-up entry (lines 58–60), the only casting sites available by that time. In the former case, the ballot is the value carried by a $\langle \text{PROPOSE}, V^\star, \cdot \rangle$ message from p_ℓ; the correct p_ℓ broadcasts exactly one proposal, $\langle \text{PROPOSE}, V^\star, v^\star \rangle$, and broadcasts retransmit messages verbatim, so the ballot is $v^\star$. In the latter case, the trigger T contains votes for view $V^\star$ from $3f - 1$ distinct senders, hence from at least $2f - 1$ correct processes; these votes are cast earlier, so, by the induction hypothesis, their ballots all equal $v^\star$. Hence, $v^\star$ commands at least $2f - 1$ ballots in T, while any other value commands at most $f < 2f - 1$ (only faulty senders can back it); the branch at line 58 therefore fires with $x = v^\star$, and the cast ballot is $v^\star$.

Finally, each of these votes is broadcast at some time $\geq \tau(V^\star) \geq \text{GST}$, after which its correct caster never crashes; the broadcast is abandoned only upon the caster entering view $V^\star + 3$ (line 23), impossible before $\tau(V^\star + 3) > \tau(V^\star + 2)$. Hence, each of these messages is alive from its broadcast until time $\tau(V^\star + 2)$.

Step 2: Every correct process decides by time $\tau_\ell + 2\delta$.
By Step 1, all correct processes (at least $4f - 1$ of them) broadcast $\langle \text{VOTE}, V^\star, v^\star, \cdot \rangle$ messages by time $\tau_\ell + \delta \geq \text{GST}$, and these messages remain alive until time $\tau(V^\star + 2) \geq \tau(V^\star) + 4\Delta \geq \tau_\ell + 2\delta$. Hence, every correct process receives votes for view $V^\star$ carrying ballot $v^\star$ from $4f - 1$ distinct senders by time $\tau_\ell + 2\delta$. Since $v^\star \neq \perp$, the rule at line 48 fires, and every correct process decides by time $\tau_\ell + 2\delta$. As the leader's proposal is broadcast at time τ_ℓ, every correct process decides within 2δ of it.

Finally, we deduce the claimed robust good-case latency. Consider any adversary in $\mathcal{F}_{\text{rgc}}$ (§2): GST coincides with a correct leader entering its view V, no correct process having entered view V or any greater view beforehand. Hence, $\tau(V) = \text{GST}$, making V a post-GST view; moreover, as $\tau(W) < \tau(V)$ for every view $W < V$, view V is the *first* post-GST view, and its leader is correct, so $V = V^\star$ and $\tau_\ell = \text{GST}$. Every correct process thus decides by time GST + 2δ, which establishes the robust good-case latency of 2δ. □

4.3 Non-Triviality

This subsection proves the non-triviality property of Floor-IT.

Lemma 3 (Non-triviality)**.** Floor-IT (Algorithm 1) satisfies the non-triviality property.

Proof. Suppose GST = 0, all processes are correct, and all propose the same value $v^\star$. Every process enters view 1 with guard SAFE at the start of the execution (line 36), so leader(1) broadcasts $\langle \text{PROPOSE}, 1, v^\star \rangle$ upon entry (line 42), $v^\star$ being its own proposal. Every process receives this proposal within $\delta \leq \Delta < 2\Delta$; since its guard is SAFE and its timer has not expired, the rule at line 44 fires and it casts ballot $v^\star$. Hence, all $n \geq 4f - 1$ processes cast, and broadcast, ballot $v^\star$ for view 1 by

time δ. As no process ever crashes and these broadcasts are abandoned only upon their senders entering view 4 (line 23), impossible before time $\tau(4) \geq 6\Delta > 2\delta$ (Claim 18, applied to views 1, 2, and 3), every process receives $4f - 1$ of these votes by time 2δ and decides (line 49). It remains to argue that no process ever decides a value different from $v^\star$, in view 1 or later. This follows directly from agreement (Lemma 1): correct processes never decide different values, and, as shown above, they decide $v^\star$. □

4.4 Complexity

This subsection proves the complexity of Floor-IT: its per-view message complexity, its memory complexity, and its view length.

Message complexity. We begin by bounding the number of distinct outcomes that correct processes can ever adopt for a single view.

Claim 21. For every view V, let

$$O_V = \left\{c \in \mathsf{Value} \cup \{\mathsf{SAFE}\} \,\middle|\, \text{at least one correct process broadcasts } \langle \textsc{adopt}, V, c\rangle\right\}.$$

Then, $|O_V| \in O(1)$.

Proof. As the recovery rule re-broadcasts only previously created adopts, every $\langle \textsc{adopt}, V, c\rangle$ message of a correct process is created by one of the rules at lines 62–68. Hence, every $c \in O_V$ is adoptable by one of these rules, and we bound each source separately, using the facts that a correct process casts a single vote per view (Claim 2), carrying a single ballot and a single guard.

- *Line 62:* c appears as a ballot on the votes of at least $2f - 1$ distinct senders, hence of at least $f - 1$ correct processes. Distinct such values require disjoint sets of correct ballot-casters, drawn from the at most $n = 5f - 1$ correct processes, so at most $\frac{5f-1}{f-1} \in O(1)$ values qualify.
- *Line 64:* c appears as a ballot on the votes of at least $2f$ distinct non-leader senders, hence of at least $2f - f = f$ correct processes; at most $\frac{5f-1}{f} \in O(1)$ values qualify.
- *Line 66:* c is carried as a guard by the votes of at least $3f - 1$ distinct senders, hence of at least $2f - 1$ correct processes; at most $\frac{5f-1}{2f-1} \in O(1)$ guards qualify.
- *Line 68:* only $c = \mathsf{SAFE}$.

Summing, $|O_V| \in O(1)$. □

We are now ready to prove the message complexity of Floor-IT.

Lemma 4 (Message complexity)**.** Floor-IT (Algorithm 1) exchanges $O(n^2)$ messages per view.

Proof. Fix a correct process p_i and a view V. Process p_i broadcasts at most one $\langle \textsc{propose}, V, \cdot\rangle$ message (only if $p_i = \mathsf{leader}(V)$, upon entering V; views are entered at most once by Claim 1). It broadcasts at most one vote for V (Claim 2) and at most one $\langle \textsc{adopt}, V, c\rangle$ message per outcome c, by the $adopted_i[V]$ latch present in every adopt rule (lines 62–68), and only for outcomes in O_V, which number $O(1)$ by Claim 21. Hence, p_i broadcasts at most $O(1)$ messages in view V, which concludes the proof. □

Memory complexity. Next, we prove Floor-IT's memory complexity.

Lemma 5 (Memory complexity)**.** In Floor-IT (Algorithm 1), each correct process stores $O(1)$ words in persistent memory.

Proof. By the rule at line 22, a correct process retains in persistent memory the view-indexed variables of at most three views, alongside $proposal_i$ and $view_i$ (one word each). For each retained view V: $guard_i[V]$ occupies one word; $vote_i[V]$ is a single message consisting of a view, a ballot,

and a guard ($O(1)$ words); and $adopted_i[V]$ holds one outcome per broadcast adopt, i.e., $O(1)$ outcomes by Claim 21. In total, $O(1)$ words. □

View length. Finally, we formally define and bound the view length of Floor-IT.

Definition 4 (View length)**.** The *view length* of Floor-IT is the smallest bound Λ with the following guarantee: for every post-GST view V, if every correct process enters view V or a higher view by some time τ, then every correct process enters a view greater than V by time $\tau + \Lambda$.

Lemma 6 (View length)**.** The view length of Floor-IT (Algorithm 1) is at most $2\Delta + 2\delta$.

Proof. Fix a post-GST view V, and suppose that every correct process enters view V or a higher view by some time τ; note that $\tau \geq \tau(V) \geq$ GST, as no view $\geq V$ is entered before time $\tau(V)$ (Claim 10). We distinguish two cases according to $\tau(V+2)$.

- Suppose first that $\tau(V+2) \leq \tau + 2\Delta + \delta$. Arguing exactly as in the proof of Claim 14, at least $3f-1$ correct processes broadcast votes for view $V+1$ before time $\tau(V+2)$; each such vote is broadcast after its sender entered a view $\geq V+1$ (Claim 2(3)), hence at some time $\geq \tau(V+1) \geq$ GST, after which the sender never crashes. These broadcasts are abandoned only upon their senders entering view $V+4$ (line 23), impossible before $\tau(V+4) \geq \tau(V+2) + 4\Delta$ (Claim 18, applied to the post-GST views $V+2$ and $V+3$). Hence, every correct process receives votes for view $V+1$ from $3f-1$ distinct senders by time $\tau(V+2) + \delta \leq \tau + 2\Delta + 2\delta$, and every correct process whose view is then still at most V enters view $V+1$ via the catch-up rule (line 57).
- Suppose now that $\tau(V+2) > \tau + 2\Delta + \delta$; in particular, no correct process enters a view greater than $V+1$ by time $\tau + 2\Delta + \delta$. Every correct process broadcasts its vote for view V by time $\tau + 2\Delta$: by time τ, it enters view V or view $V+1$ (greater views being excluded); if it enters view $V+1$ by time $\tau + 2\Delta$, the backfill at that entry casts the vote (line 40); otherwise, it enters view V at some time $e \leq \tau$ and is still there at time $e + 2\Delta \leq \tau + 2\Delta$, when the timeout casts the vote (line 47), unless it has voted even earlier. These broadcasts are abandoned only upon their senders entering view $V+3$ (line 23), impossible before $\tau(V+3) \geq \tau(V+2) + 2\Delta > \tau + 4\Delta + \delta$ (Claim 18, applied to the post-GST view $V+2$); hence, all these votes reach every correct process by time $\tau + 2\Delta + \delta$. Arguing exactly as in the proof of Claim 15 (the sample of the $4f-1$ correct processes with the smallest indices, identical at every correct process, and the case analysis over Heavy), there is an outcome c, common to all correct processes, for which one of the rules at lines 62–68 applies for view V. By time $\tau + 2\Delta + \delta$, the adopt gate (line 24) is open at every correct process, as a process in view V entered it by time τ, so its timer reaches 2Δ by time $\tau + 2\Delta$, while a process in view $V+1$ satisfies $V < view_i$; moreover, every correct process is in view V or $V+1$, so view V lies within its window. Hence, every correct process broadcasts $\langle \text{ADOPT}, V, c \rangle$ by time $\tau + 2\Delta + \delta$, at the exhibited rule or upon the earlier insertion of c into its *adopted* set; these broadcasts, too, are abandoned only upon their senders entering view $V+3$, so the $4f-1$ matching adopts reach every correct process by time $\tau + 2\Delta + 2\delta$, and every correct process still in view V enters view $V+1$ (lines 51 and 53).

In both cases, every correct process enters a view greater than V by time $\tau + 2\Delta + 2\delta$. □

Acknowledgments

We thank Gilad Stern and Sourav Das for many valuable discussions and their feedback on this work.

References

[1] Ittai Abraham. 2026. Fast Simplex on a Fixed View Schedule. Decentralized Thoughts. https://decentralizedthoughts.github.io/2026-05-17-fast-simplex-fvs/

[2] Ittai Abraham, Yonatan Amit, and Danny Dolev. 2004. Optimal Resilience Asynchronous Approximate Agreement. In *Principles of Distributed Systems, 8th International Conference, OPODIS 2004, Grenoble, France, December 15-17, 2004, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 3544)*, Teruo Higashino (Ed.). Springer, 229–239. doi:10.1007/11516798_17

[3] Ittai Abraham, Naama Ben-David, and Sravya Yandamuri. 2022. Efficient and Adaptively Secure Asynchronous Binary Agreement via Binding Crusader Agreement. In *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, Alessia Milani and Philipp Woelfel (Eds.). ACM, 381–391. doi:10.1145/3519270.3538426

[4] Ittai Abraham, Sourav Das, Yuval Efron, and Jovan Komatovic. 2026. Forget-IT: Optimal Good-Case Latency For Information-Theoretic BFT. *Cryptology ePrint Archive* (2026).

[5] Ittai Abraham, Sourav Das, Yuval Efron, and Jovan Komatovic. 2026. Forget-IT: Optimal Good-Case Latency For Information-Theoretic BFT. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2026, Egham, United Kingdom, July 6-10, 2026*, Dennis Olivetti, Eric Ruppert, and Sean Ovens (Eds.). ACM, 478–488. doi:10.1145/3796701.3815965

[6] Ittai Abraham, Sourav Das, Yuval Efron, Jovan Komatovic, Alejandro Ranchal-Pedrosa, and Gilad Stern. 2026. Two Round Information-Theoretic Chained BFT at n=5f-1. Decentralized Thoughts. https://decentralizedthoughts.github.io/2026-07-12-fast-IT-BFT/

[7] Ittai Abraham, Srinivas Devadas, Kartik Nayak, and Ling Ren. 2017. Brief Announcement: Practical Synchronous Byzantine Consensus. In *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[8] Ittai Abraham, Yuval Efron, and Ling Ren. 2025. 2-round BFT in Simplex Style. Decentralized Thoughts. https://decentralizedthoughts.github.io/2025-07-18-two-round-Simplex/

[9] Ittai Abraham, Yuval Efron, and Ling Ren. 2025. The Latency Cost Of Censorship Resistance. *IACR Cryptol. ePrint Arch.* (2025), 2136.

[10] Ittai Abraham, Guy Gueta, Dahlia Malkhi, Lorenzo Alvisi, Rama Kotla, and Jean-Philippe Martin. 2017. Revisiting Fast Practical Byzantine Fault Tolerance. arXiv:1712.01367. https://arxiv.org/pdf/1712.01367

[11] Ittai Abraham, Guy Gueta, Dahlia Malkhi, and Jean-Philippe Martin. 2018. Revisiting Fast Practical Byzantine Fault Tolerance: Thelma, Velma, and Zelma. arXiv:1801.10022. https://arxiv.org/pdf/1801.10022

[12] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. 2021. Brief Note: Fast Authenticated Byzantine Consensus. arXiv:2102.07932. https://arxiv.org/pdf/2102.07932

[13] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. 2021. Good-case Latency of Byzantine Broadcast: a Complete Categorization. In *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen (Eds.). ACM, 331–341. doi:10.1145/3465084.3467899

[14] Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. 2021. Optimal Good-Case Latency for Rotating Leader Synchronous BFT. In *25th International Conference on Principles of Distributed Systems, OPODIS 2021, Strasbourg, France, December 13-15, 2021 (LIPIcs, Vol. 217)*, Quentin Bramas, Vincent Gramoli, and Alessia Milani (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 27:1–27:19. doi:10.4230/LIPICS.OPODIS.2021.27

[15] Ittai Abraham and Ling Ren. 2025. 2-round BFT in Simplex Style for n=5f-1. Decentralized Thoughts. https://decentralizedthoughts.github.io/2025-08-06-5fminus1-simplex/

[16] Ittai Abraham, Ling Ren, and Zhuolun Xiang. 2021. Good-Case and Bad-Case Latency of Unauthenticated Byzantine Broadcast: A Complete Categorization. In *25th International Conference on Principles of Distributed Systems, OPODIS 2021, Strasbourg, France, December 13-15, 2021 (LIPIcs, Vol. 217)*, Quentin Bramas, Vincent Gramoli, and Alessia Milani (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 5:1–5:20. doi:10.4230/LIPICS.OPODIS.2021.5

[17] Ittai Abraham and Gilad Stern. 2021. Information Theoretic HotStuff. In *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 11–1.

[18] Ittai Abraham and Gilad Stern. 2021. Information Theoretic HotStuff (IT-HS) Part One. https://decentralizedthoughts.github.io/2021-09-20-information-theoretic-hotstuff-it-hs-part-one/.

[19] Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani. 2024. Good-case early-stopping latency of synchronous byzantine reliable broadcast: the deterministic case. *Distributed Computing* 37, 2 (2024), 121–143.

[20] Michael Ben-Or. 1983. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols (Extended Abstract). In *PODC*. ACM, 27–30.

[21] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. *Information and computation* 75, 2 (1987), 130–143.

[22] Ethan Buchman. 2016. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains.* Ph. D. Dissertation. University of Guelph.

[23] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and Efficient Asynchronous Broadcast Protocols. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings (Lecture Notes in Computer Science, Vol. 2139)*, Joe Kilian (Ed.). Springer, 524–541. doi:10.1007/3-540-44647-8_31

[24] Benjamin Y. Chan and Rafael Pass. 2023. Simplex Consensus: A Simple and Fast Consensus Protocol. In *TCC (4) (Lecture Notes in Computer Science, Vol. 14372)*. Springer, 452–479. https://eprint.iacr.org/2023/463.pdf

[25] Jinyuan Chen. 2024. OciorMVBA: Near-Optimal Error-Free Asynchronous MVBA. arXiv:2501.00214v1 [cs.CR]. arXiv:2501.00214v1 [cs.CR] http://arxiv.org/abs/2501.00214v1

[26] Brendan Kobayashi Chou, Andrew Lewis-Pye, and Patrick O'Grady. 2025. Minimmit: Fast Finality with Even Faster Blocks. arXiv:2508.10862. https://arxiv.org/pdf/2508.10862

[27] Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. 2022. Byzantine Consensus is $\Theta(n^2)$: The Dolev-Reischuk Bound is Tight even in Partial Synchrony!. In *36th International Symposium on Distributed Computing (DISC 2022) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 246)*, Christian Scheideler (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 14:1–14:21. doi:10.4230/LIPIcs.DISC.2022.14

[28] Pierre Civit, Seth Gilbert, and Vincent Gramoli. 2021. Polygraph: Accountable Byzantine Agreement. In *Proceedings of the 41st IEEE International Conference on Distributed Computing Systems (ICDCS'21)*.

[29] Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. 2023. On the Validity of Consensus. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*. 332–343.

[30] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. 2018. DBFT: Efficient Leaderless Byzantine Consensus and its Applications to Blockchains. In *Proceedings of the 17th IEEE International Symposium on Network Computing and Applications (NCA'18)*. IEEE.

[31] Miguel Oom Temudo de Castro. 2000. *Practical Byzantine fault tolerance.* Ph. D. Dissertation. Massachusetts Institute of Technology, Cambridge, MA, USA. https://hdl.handle.net/1721.1/86581

[32] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. 1990. Early Stopping in Byzantine Agreement. *J. ACM* 37, 4 (1990), 720–741. doi:10.1145/96559.96565

[33] Danny Dolev and H. Raymond Strong. 1983. Authenticated Algorithms for Byzantine Agreement. *SIAM J. Comput.* 12, 4 (1983), 656–666.

[34] Manu Drijvers, Tim Gretler, Yotam Harchol, Tobias Klenze, Ognjen Maric, Stefan Neamtu, Yvonne-Anne Pignolet, Rostislav Rumenov, Daniel Sharifi, and Victor Shoup. 2024. A New Broadcast Primitive for BFT Protocols. *CoRR* abs/2410.22080 (2024). arXiv:2410.22080 doi:10.48550/ARXIV.2410.22080

[35] Sisi Duan, Xin Wang, and Haibin Zhang. 2023. FIN: Practical Signature-Free Asynchronous Common Subset in Constant Time. In *CCS*. ACM, 815–829.

[36] Partha Dutta, Rachid Guerraoui, and Marko Vukolić. 2005. *Best-Case Complexity of Asynchronous Byzantine Consensus.* Technical Report EPFL/IC/200499. EPFL. https://lpdwww.epfl.ch/upload/documents/publications/567931850DGV-feb-05.pdf

[37] C. Dwork, N. Lynch, and L. Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *Journal of the Association for Computing Machinery, Vol. 35, No. 2, pp.288-323* (1988).

[38] Yuval Efron, Joachim Neu, Ling Ren, and Ertem Nusret Tas. 2025. Optimal Good-Case Latency for Sleepy Consensus. *CoRR* abs/2510.06023 (2025). arXiv:2510.06023 doi:10.48550/ARXIV.2510.06023

[39] Pesech Feldman and Silvio Micali. 1997. An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement. *SIAM J. Comput.* 26, 4 (1997), 873–933. doi:10.1137/S0097539790187084

[40] Antonio J. Fernández-Pinto, Manuel Bravo, Gregory Chockler, and Alexey Gotsman. 2026. Fast TetraBFT: Optimizing Latency Where It Matters. arXiv:2606.03754.

[41] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM (JACM)* 32, 2 (1985), 374–382.

[42] Matthias Fitzi and Juan A Garay. 2003. Efficient Player-Optimal Protocols for Strong and Differential Consensus. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*. 211–220.

[43] Bruno França, Denis Kolegov, Igor Konnov, and Grzegorz Prusak. 2025. ChonkyBFT: Consensus Protocol of ZKsync. arXiv:2503.15380. https://arxiv.org/pdf/2503.15380

[44] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2019. SBFT: a scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 568–580. https://arxiv.org/pdf/1804.01626

[45] Kim P. Kihlstrom, Louise E. Moser, and Peter M. Melliar-Smith. 2003. Byzantine Fault Detectors for Solving Consensus. *British Computer Society* (2003).

[46] Quentin Kniep, Kobi Sliwinski, and Roger Wattenhofer. 2025. Alpenglow: A New Consensus for Solana. https://github.com/rogerANZA/Alpenglow-White-Paper/blob/main/Alpenglow-v1.1.pdf

[47] Jovan Komatovic, Joachim Neu, and Tim Roughgarden. 2024. Toward Optimal-Complexity Hash-Based Asynchronous MVBA with Optimal Resilience. *CoRR* abs/2410.12755 (2024). arXiv:2410.12755 doi:10.48550/ARXIV.2410.12755

[48] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2007. Zyzzyva: Speculative Byzantine Fault Tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. 45–58.

[49] Klaus Kursawe. 2002. Optimistic Byzantine Agreement. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS)*. doi:10.1109/RELDIS.2002.1180196

[50] Petr Kuznetsov, Andrei Tonkikh, and Yan X. Zhang. 2021. Revisiting Optimal Resilience of Fast Byzantine Consensus. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 343–353. https://arxiv.org/pdf/2102.12825

[51] Leslie Lamport. 1983. The weak Byzantine generals problem. *Journal of the ACM (JACM)* 30, 3 (1983), 668–676.

[52] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* 4, 3 (1982), 382–401.

[53] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited. In *PODC*. ACM, 129–138.

[54] Jean-Philippe Martin and Lorenzo Alvisi. 2006. Fast Byzantine Consensus. *IEEE Trans. Dependable Secur. Comput.* 3, 3 (July 2006), 202–215. doi:10.1109/TDSC.2006.35

[55] Darya Melnyk and Roger Wattenhofer. 2018. Byzantine Agreement with Interval Validity. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 251–260.

[56] Achour Mostéfaoui and Michel Raynal. 2017. Signature-Free Asynchronous Byzantine Systems: From Multivalued to Binary Consensus with t< n/3, O(n^2) Messages, and Constant Time. *Acta Informatica* 54, 5 (2017), 501–520. doi:10.1007/S00236-016-0269-Y

[57] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. 1980. Reaching Agreement in the Presence of Faults. *J. ACM* 27, 2 (1980), 228–234. doi:10.1145/322186.322188

[58] Michael O. Rabin. 1983. Randomized Byzantine Generals. In *FOCS*. IEEE Computer Society, 403–409.

[59] Victor Shoup, Jakub Sliwinski, and Yann Vonlanthen. 2025. Kudzu: Fast and Simple High-Throughput BFT. In *DISC (LIPIcs, Vol. 356)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 42:1–42:19. https://arxiv.org/pdf/2505.08771

[60] Hin-Sing Siu, Yeh-Hao Chin, and Wei-Pang Yang. 1998. Reaching strong consensus in the presence of mixed failure types. *Information Sciences* 108, 1-4 (1998), 157–180.

[61] David Stolz and Roger Wattenhofer. 2016. Byzantine Agreement with Median Validity. In *19th International Conference on Principles of Distributed Systems (OPODIS 2015)*, Vol. 46. Schloss Dagstuhl–Leibniz-Zentrum für Informatik GmbH, 22.

[62] Yann Vonlanthen, Jakub Sliwinski, Massimo Albarello, and Roger Wattenhofer. 2024. Banyan: Fast Rotating Leader BFT. Proceedings of the 25th ACM/IFIP International Middleware Conference. https://arxiv.org/pdf/2312.05869

[63] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 347–356.

[64] Qianyu Yu, Giuliano Losa, and Xuechao Wang. 2024. TetraBFT: Reducing Latency of Unauthenticated, Responsive BFT Consensus. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, Ran Gelles, Dennis Olivetti, and Petr Kuznetsov (Eds.). ACM, 257–267. doi:10.1145/3662158.3662783