Synchronous BFT with Provability and Fast Path for the Age of Blockchains

ITTAI ABRAHAM, a16z Crypto Research
KARTIK NAYAK, Duke University
LING REN, University of Illinois at Urbana-Champaign
ERTEM NUSRET TAS, a16z Crypto Research

Synchronous Byzantine fault tolerant (BFT) protocols offer stronger security guarantees, increasing resilience from one third to one half. However, existing constructions suffer from three drawbacks that are critical in blockchain settings.

First, blockchain systems require rotating leaders, but the existing constructions are not optimized for low latency under leader rotation. Second, blockchain systems require *provable commitments* that can be forwarded to clients or offchain entities. Third, modern blockchains optimize for low latency in the good case, and often further optimize for a fast path under even milder adversarial conditions.

In this work, we formalize and study fast paths in the context of provable synchronous BFT. Our first result is that for $n \leq 2t + 2p + c - 1$, it is impossible to obtain a provable synchronous BFT protocol that tolerates t Byzantine and c crash faults while also achieving a two round fast path resilient to p Byzantine faults. Guided by this bound, we then present four provable synchronous protocols tailored to two deployment paradigms: optimistic responsivness and fixed view schedules.

For optimistic responsiveness, we present Δ-*Sync Simplex* and 2Δ-*Sync Simplex*, which for $n = 2t + 2p + c + 1$ produce provable commit certificates in time 2δ under a correct leader when the number of faulty parties f satisfies $f \leq p$. 2Δ-Sync Simplex also achieves 3δ commit latency when the total number of faulty parties is $f \leq \lfloor (p + t)/2 \rfloor$ with Byzantine faults $f_b \leq t$ and crash faults $f_c \leq c$. Outside these optimistic regimes, the protocols remain safe and live for t Byzantine and c crash faults, and output provable commitments within $2\Delta + 2\delta$ and $1\Delta + 3\delta$, respectively. Under faulty leaders, the protocols guarantee a worst case view length of $6\Delta + \delta$ and $5\Delta + \delta$ respectively.

For fixed view schedules, we present two additional protocols, *TenderSync* and *SyncMint*, that trade off commit latency and view duration. They achieve either 3Δ provable commit with 4Δ views, or 4Δ provable commit with 3Δ views. Both protocols obtain a provable commit in 2δ under a correct leader when $f \leq p$.

1 Introduction

The design of Byzantine fault tolerant (BFT) consensus protocols has traditionally been dominated by the partial synchrony model [16], which assumes that communication is usually fast but may occasionally be interrupted by unbounded delays. This abstraction underlies most deployed low latency blockchain protocols today such as Tendermint [11] (Cosmos), Alpenglow [13, 23] (Solana), Mysticeti [9] (Mysten) and Raptr [37] (Aptos). In contrast, it has been the prevailing view that *synchronous* BFT protocols necessarily entail high latency, proportional to a network delay upper bound Δ [30]. This in turn has hindered deployment of synchronous BFT protocols in practice, with the notable exception of Bitcoin and other protocols based on Nakamoto consensus[1] [29].

Recent changes in the infrastructure underlying the deployed consensus protocols motivate a renewed focus on synchrony with low latency. An increasing number of validator networks now operate over carefully provisioned infrastructure, with dedicated fiber links, controlled routing domains, and active network monitoring. In such environments, assuming a known upper bound on message delay is no longer a purely theoretical convenience, but a

[1]Unlike in many synchronous BFT protocols, Nakamoto consensus continues to provide Byzantine fault tolerance even if the network delay exceeds Δ, albeit with a smaller resilience that degrades in proportion to the gap between the actual network delay and Δ.

realistic engineering assumption, as already reflected in modern interchain and networking layers such as LayerZero [26]. When synchrony is a valid assumption, its implications for blockchains are substantial: the resilience threshold increases from one third Byzantine parties to one half. This shift directly improves the capital efficiency of security, allowing the same adversarial tolerance with significantly fewer parties.

Moving from partially synchronous to synchronous blockchain design, however, is not a straightforward substitution of models. Modern blockchains impose performance, safety, and usability requirements that are not fully addressed by the classical synchronous Byzantine agreement literature. In this paper, we revisit synchronous Byzantine consensus through the lens of two central requirements for modern blockchain systems: **provable commitments** and **low latency**. In the case of view-based protocols, the latter can further be analyzed for a good-case view with a correct leader, and a worst-case latency when there are faulty leaders. We study both requirements under two dominant blockchain design paradigms: **fixed view schedules** and **optimistic responsiveness**.

Provable commitments

In partially synchronous protocols, a committed value is typically accompanied by an explicit cryptographic certificate that can be verified by any external observer, including light clients, wallets, bridges, or cross domain verifiers, without relying on timing assumptions or further interaction. In contrast, many synchronous protocols emphasize small good case latency, but define commitment only relative to the shared timing model of the parties. For blockchains, provability is fundamental: certificates must be portable and independently verifiable. We formalize provable synchronous state machine replication, Byzantine broadcast, and Byzantine agreement, requiring that every committed value is accompanied by a certificate whose validity does not depend on synchrony. We show that provability fundamentally constrains achievable latency. In particular, any synchronous BFT protocol that tolerates t Byzantine and c crash faults and guarantees a good case commit latency of $T \leq 2\Delta$ when there are up to p crash faults, must satisfy $n \geq 2t + 2p + c$. This lower bound explains why provable commitment cannot be achieved in two synchronous rounds at optimal (Byzantine) resilience of $1/2$.

Low latency

Modern blockchains typically employ a revolving leader paradigm, where progress depends on a sequence of leaders. When the current leader is non-faulty (so-called the good case), commit latency is captured by the regular or optimistic execution paths. When the leader is faulty, however, the system must advance to the next view, and the view length becomes the dominant contributor to latency and the end-to-end confirmation time for transactions. This is captured by the notion of worst case view latency. Existing synchronous protocols incur long worst case view lengths even when the protocol is otherwise stable. We present view-based synchronous protocols with worst case view length of at most $5\Delta + \delta$ and $6\Delta + \delta$, improving over prior approaches that require substantially longer views under faulty leaders.

Deployed blockchains follow two distinct timing paradigms. The first one is fixed view schedules, where the start and end of each view are determined as a function of a globally synchronized clock. This approach simplifies engineering and acknowledges the fact that in practice, view leaders are incentivized to delay block proposal for capturing higher-value transactions [8, 31, 34]. It is used by Cosmos, Solana and Ethereum [12], as well as by several other production systems built around slot based execution. A second class of systems (*e.g.*, Espresso [10, 17] and Flow [20, 21]) prioritizes optimistic responsiveness, aiming to commit as soon as messages arrive when network delays are small, while retaining worst

case guarantees under synchrony. In practice, the actual message delay δ is often significantly smaller than the configured upper bound Δ. Thus, optimistic responsiveness formally requires that a protocol commits in time proportional to δ when conditions are favorable, while retaining worst case guarantees proportional to Δ. These two paradigms impose fundamentally different constraints on protocol design when optimizing both provable good case latency and worst case view latency, and motivate distinct algorithmic techniques.

1.1 Contributions

Foundational limits of fast commit with provability. As our first contribution, we explore fast path provable commit and show that any synchronous BFT protocol that is resilient to t Byzantine and c crash faults, and achieve a good case commit latency of $T \leq 2\Delta$ when there are up to p crash faults, must have at least $n = 2t + 2p + c$ parties in total.

Theorem 1 (Lower bound for $< 3\Delta$ provable commit, Section 3)**.** *For any $t, c, p > 0$ and $n = 2t + 2p + c - 1$, there does not exist a provable Byzantine broadcast protocol that can tolerate t Byzantine faults and c crash faults and also obtain a provable commit certificate in time $< 3\Delta$ in an execution where the leader is correct and there are at most p crash faults.*

Guided by this lower bound, we design two optimistically responsive protocols and two protocols with fixed view schedules for $n = 2t + 2p + c + 1$ with a fast path provable commit given $p < t$ Byzantine faults and a regular provable path for t Byzantine and c crash faults.

Synchronous BFT with optimistic responsiveness. Combining responsiveness with provability raises new challenges, because a responsive commit must still gather sufficient evidence to form a certificate that remains valid without timing assumptions. Guided by our lower bounds, we design Simplex-style [13] synchronous protocols that expose optimistic execution paths alongside a synchronous fallback.

Theorem 2. *Δ-Sync Simplex in Section 4.1 is a view-based rotating-leader synchronous validated BFT protocol with $n = 2t + 2p + c + 1$.*

- *Fast path: all correct parties obtain a commit certificate responsively in 2δ time in any execution with a correct leader and $\leq p$ Byzantine faults.*
- *Synchronous path: all correct parties obtain a commit certificate in $\Delta + 3\delta$ in any execution with a correct leader, $\leq t$ Byzantine faults, and $\leq c$ crash faults.*
- *If all correct parties enter a view by time t, then all correct parties leave the view by time $t + 5\Delta + \delta$ in any execution with $\leq t$ Byzantine faults and $\leq c$ crash faults.*

Compared to the protocol in [35, Section 5], which we denote by the name Δ-OptSync, Δ-Sync Simplex improves the view length, provides a generalization of the resiliences for the fast path, and achieves provable commit (see Table 1).

Theorem 3. *2Δ-Sync Simplex in Section 4.2 is a view-based rotating-leader synchronous validated BFT protocol with $n = 2t + 2p + c + 1$.*

- *Fast path: all correct parties obtain a commit certificate responsively in 2δ in any execution with a correct leader and $\leq p$ Byzantine faults.*
- *Responsive middle path: all correct parties obtain a commit certificate responsively in 3δ in any execution with a correct leader, $\leq \lfloor (t + c)/2 \rfloor + p$ total number of faults, $\leq t$ Byzantine faults, and $\leq c$ crash faults.*
- *Synchronous path: all correct parties obtain a commit certificate in $2\Delta + 2\delta$ time in any execution with a correct leader, $\leq t$ Byzantine faults, and $\leq c$ crash faults.*

- *If all correct parties enter a view by time t, then all correct parties leave the view by time $t + 5\Delta + \delta$ in any execution with $\leq t$ Byzantine faults and $\leq c$ crash faults.*

Compared to the protocol in [35, Section 6], which we denote by the name 2Δ-OptSync, 2Δ-Sync Simplex improves the view length, provides a generalization of the resiliences for the fast path, and achieves provable commit (see Table 1).

We generalize the protocols Δ and 2Δ-Sync Simplex to tolerate t Byzantine, $\min(p, c)$ omission and $\max(c - p, 0)$ crash faults in their respective sections (our crash faults are also 'omission-crashes', *i.e.*, at the round of crash, the adversary can decide to deliver some of the sent messages). Theorems 1 to 3 are formalized and proven in Sections 3, 5.1 and 5.2 respectively. We also provide a tighter characterization of the resilience of the responsive middle path of 2Δ-Sync Simplex in Section 5.2.

Synchronous BFT with fixed view schedules. These protocols impose hard constraints on worst case progress, since the views cannot be shortened even when communication is fast. However, they can achieve smaller view lengths in return, which we demonstrate through new protocols tailored to this setting.

First we extend the work of [18] to obtain fast path and provability:

Theorem 4. *TenderSync in Section 4.3 is a fixed-schedule view-based rotating-leader synchronous validated BFT protocol with $n = 2t + 2p + c + 1$.*

- *All correct parties obtain a commit certificate in 3Δ in any execution with a correct leader, $\leq t$ Byzantine faults, and $\leq c$ crash faults.*
- *Fast path: all correct parties obtain a fast commit certificate in 2Δ (equivalently, within 2δ) in any execution with a correct leader and $\leq p$ Byzantine faults.*
- *The view schedule changes views every 4Δ.*

Second, we show a new trade-off that allows for faster, 3Δ length views at the expence of having a slower normal case commit of 4Δ latency.

Theorem 5. *SyncMint in Section 4.4 is a fixed-schedule view-based rotating-leader synchronous validated BFT protocol with $n = 2t + 2p + c + 1$.*

- *All correct parties obtain a commit certificate in 4Δ time in any execution with a correct leader, $\leq t$ Byzantine faults, and $\leq c$ crash faults.*
- *Fast path: all correct parties obtain a fast commit certificate in 2Δ (equivalently, within 2δ) time in any execution with a correct leader and $\leq p$ Byzantine faults.*
- *The view schedule changes views every 3Δ.*

Assuming that a transaction appears uniformly over time, conditioned on good leaders, there are two cases: If there are at most p Byzantine faults, then the expected time to commit given the 2δ fast path is better for SyncMint relative to TenderSync. However, in executions where there at most t Byzantine and c crash faults, then the expected time to commit is better for TenderSync relative to SyncMint.

Theorems 4 and 5 are formalized and proven in Sections 5.3 and 5.4.

Together, these results clarify the fundamental limits of fast, provable commit in synchronous blockchains and provide new protocol constructions that match these limits across optimistic, fixed schedule, and worst case execution regimes.

Outline. In Section 2, we introduce provable synchronous state machine replication, Byzantine broadcast and agreement, capturing cryptographically certifiable commitments independent of timing assumptions. In Section 3, we present our lower bound. In Section 4, we provide our protocols, and in Section 5, prove their security.

Protocol / Paper	2δ **fast path**	3δ **responsive**	**Correct-leader latency**	**View latency**	**Responsive / Fixed View**
Δ-Sync Simplex	$f \leq p$	no	$\Delta + 3\delta$	$6\Delta + \delta$	R
2Δ-Sync Simplex	$f \leq p$	$f \leq \lfloor (p+t)/2 \rfloor$	$2\Delta + 2\delta$	$5\Delta + \delta$	R
TenderSync	$f \leq p$	no	3Δ	4Δ	F
SyncMint	$f \leq p$	no	4Δ	3Δ	F
Sync HotStuff [4]	no	no	$2\Delta + 2\delta$	$7\Delta + \delta$	R
Δ-OptSync [35]	$f = 0$	no	$\Delta + 3\delta$	$7\Delta + \delta$	R
2Δ-OptSync [35]	no	$f \leq \lfloor t/2 \rfloor$	$2\Delta + 2\delta$	$7\Delta + \delta$	R

Table 1. Comparison table. $n = 2t + 2p + c + 1$, $p < t$, t and c denote the maximum number of Byzantine and crash faulty parties, and f denotes the actual number of faults in the execution (which can be of any type). Correct leader latency is the latency to get a certified commit. View latency is the worst-case latency a malicious leader can elongate its view. Latency numbers are stated for after a time all correct parties enter a view, which happens within δ time of each other.

1.2 Related Work

Authenticated synchronous Byzantine agreement. The classical line of work on synchronous Byzantine agreement and broadcast begins with the foundational Byzantine agreement literature [15, 25]. Several works have been proposed to improve the round complexity of synchronous protocols [1, 19, 22, 27, 33, 36].

Synchronous BFT protocols motivated by blockchains. More recent work synchronous BFT protocols has focused on low-latency state machine replication (SMR) and good-case/optimistic guarantees, including XFT [24], the DFINITY consensus protocol [36], PiLi [14], and Sync HotStuff [4]. There has also been a sequence of work optimizing good-case latency and responsiveness in the synchronous model, including Thunderella [32], OptSync [35], optimizing latency under rotating leaders [7], HybridBFT [28] and the near-optimal good-case latency protocols of Abraham et al. [5].

2 Preliminaries

We consider consensus protocols executed by a set of n parties. There is an adversary, denoted by $\mathcal{A}$, which is a probabilistic polynomial-time algorithm. It corrupts f parties before the protocol execution commences. The corrupted (faulty) parties can one of the following fault types:

- Byzantine faults: they can arbitrarily deviate from the protocol rules as determined by $\mathcal{A}$.
- Omission faults: messages sent from and to the party can be dropped arbitrarily by $\mathcal{A}$.
- Omission-crash faults: the adversary can choose a round r such that the party is non-faulty in all rounds $< r$, completely crashed in all rounds $> r$, and is omission-faulty at round r.

The remaining parties are said to be correct and follow the protocol rules. Time proceeds in discrete rounds, and parties have synchronized clocks. They communicate over point-to-point, authenticated channels. There is public-key infrastructure (PKI), and correct parties sign their messages. In the protocol descriptions, "Send x" implies a message x signed by the

sending party, whereas "Relay x" implies forwarding a received (signed) value x. We denote a message m signed by party p by $\langle m \rangle_p$. The network is *synchronous*: any message sent by an correct party reaches its destination before Δ rounds, where Δ is a known delay upper bound. The actual delivery time is determined by $\mathcal{A}$, subject to the upper bound Δ. We denote the actual network delay by δ. Going forward, we use the shorthand 's.t.' to mean 'such that'.

We next present our definition of provable synchronous state machine replication (SMR), along with provable synchronous Byzantine broadcast and agreement. We denote the set of values input to the parties by $\mathcal{V}$.

In SMR, parties are input values by the environment throughout the protocol execution, and their goal is to ensure that all protocol observers obtain the same sequence of values, called the log and denoted by log. Two logs are said to be consistent if one is a prefix of the other (otherwise, there are conflicting). We refer to a value or log committed by a party, together with its proof of commitment, as a *certificate*, denoted by cr. Let cr.v (cr.log) and $\text{cr}.\pi$ denote the value (respectively, the log) and the proof contained by a certificate cr. A certificate cr is said to be valid if it contains a valid commit proof π for its value or log according to a specified commit proof verification function CV, *i.e.*, if $\text{CV}(\text{cr.v}, \text{cr}.\pi) = \text{true}$.

Definition 1 (Provable Synchronous State Machine Replication)**.** A synchronous provable SMR protocol (PS-SMR) provides a commit proof verification function $\text{CV}(\cdot)$ and satisfies the following properties:

- **Safety:** For any cr and cr′ output by a computationally bounded adversary s.t. $\text{CV}(\text{cr.log}, \text{cr}.\pi) = \text{CV}(\text{cr}'.\text{log}, \text{cr}'.\pi) = \text{true}$, it holds that cr.log and cr.log′ are consistent, *i.e.*, a computationally bounded adversary cannot output two valid certificates with conflicting logs.
- **Liveness:** If a value v is input to a correct party, all correct parties obtain a valid cr after a constant number of rounds s.t. $\text{v} \in \text{cr.log}$.

Let L.v denote the value sent by a correct designated sender/leader L.

Definition 2 (Provable Synchronous Byzantine Broadcast)**.** A provable synchronous Byzantine broadcast protocol (PS-BB) provides a commit proof verification function $\text{CV}(\cdot)$ and satisfies the following properties:

- **Agreement:** For any cr and cr′ output by a computationally bounded adversary s.t. $\text{CV}(\text{cr.v}, \text{cr}.\pi) = \text{CV}(\text{cr}'.\text{v}, \text{cr}'.\pi) = \text{true}$, it holds that $\text{cr.v} = \text{cr}'.\text{v}$, *i.e.*, a computationally bounded adversary cannot output two valid certificates with different values.
- **Validity:** If the designated sender/leader L is correct, then no computationally bounded adversary can output a valid cr such that $\text{cr.v} \neq L.\text{v}$, *i.e.*, a computationally bounded adversary cannot output valid certificates that do not contain the leader's value.
- **Termination:** In a constant number of rounds, each correct party obtains a valid cr with some value $\text{cr.v} \in \mathcal{V} \cup \{\perp\}$.

Definition 3 (Provable Synchronous Byzantine Agreement)**.** A provable synchronous Byzantine agreement protocol (PS-BA) provides a commit proof verification function $\text{CV}(\cdot)$ and satisfies the following properties:

- **Agreement:** For any cr and cr′ output by a computationally bounded adversary s.t. $\text{CV}(\text{cr.v}, \text{cr}.\pi) = \text{CV}(\text{cr}'.\text{v}, \text{cr}'.\pi) = \text{true}$, it holds that $\text{cr.v} = \text{cr}'.\text{v}$, *i.e.*, a computationally bounded adversary cannot output two valid certificates with different values.
- **Validity:** If all correct parties start with the same value v, then no computationally bounded adversary can output a valid cr such that $\text{cr.v} \neq \text{v}$, *i.e.*, a computationally

bounded adversary cannot output valid certificates that do not contain the common input to the correct parties.
- **Termination:** In a constant number of rounds, each correct party obtains a valid cr with some value cr.v $\in \mathcal{V} \cup \{\perp\}$.

By default, all commit rules that satisfy the desiderata for PS-SMR, PS-BB and PS-BA are said to be provable commit rules. These rules trivially satisfy the requirements for vanilla SMR, BB and BA as stated in [35]. In contrast, we say that a commit rule is non-provable if it satisfies only the requirements for vanilla SMR, BB and BA, but not necessarily those stated for PS-SMR, PS-BB and PS-BA.

Definition 4 (T-Good Case Latency, Definition 6 of [6])**.** A provable Byzantine broadcast protocol has T-good case latency, if all correct parties obtain a valid certificate within time T (over all executions and adversary strategies) when the leader is correct.

A provable Byzantine agreement protocol has T-good case latency, if all correct parties obtain a valid certificate within time T (over all executions and adversary strategies) when they start with the same value.

3 Lower Bound

For the lower bound we define an *omission-crash* fault that is weaker than omission and somewhat stronger than vanilla crash.

Definition 5. If a party is *omission-crash* faulty, then the adversary can choose a round r such that the party is non-faulty in all rounds $< r$, completely crashed in all rounds $> r$, and can send omission faults in round r.

Said differently, the adversary can choose which of the messages sent by that party in round r are delivered and which are dropped.

Theorem 6 (Lower bound for $< 3\Delta$ provable commit)**.** *For any integers $c \geq 0$ and $t, p \geq 1$, let $n = 2t + 2p + c - 1$. There does not exist a provable Byzantine broadcast protocol that can tolerate t Byzantine faults and c omission-crash faults and achieve T-good case latency with $T < 3\Delta$ when the leader is correct and there are at most p crash faults.*

Proof. For contradiction, suppose there exists such a protocol.

Partition the parties into 5 sets A, B, C, D, S and a singleton L (the leader) as follows: A and D are of size p each, B and C are of size $t - 1$ each, and S is of size c. The total size is $2p + 2(t - 1) + c + 1 = 2t + c + 2p - 1$.

Validity world (correct leader, A is silent, $< 3\Delta$). Consider an execution where a correct leader sends 1 and the parties in A crash at the beginning. All messages take Δ time to reach their recipients.

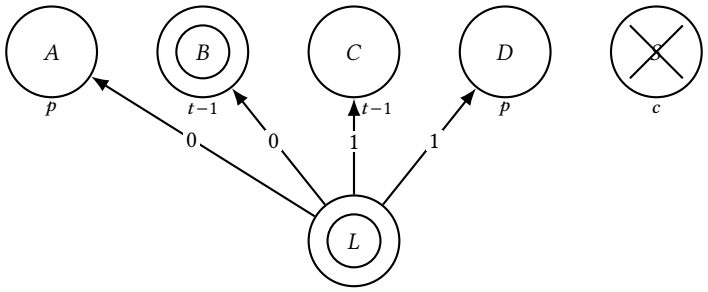
Since the protocol has good-case latency $T < 3\Delta$ in the presence of p crash faults, a commit certificate for 1 is formed before time 3Δ from the $n - p$ round-two messages sent by the leader and all parties outside A.

Split world (malicious leader, S is silent). In this world, a malicious leader sends 0 to the parties in A, B and 1 to the parties in C, D. In addition, the parties in S crash at the beginning. The leader crashes after its first round.

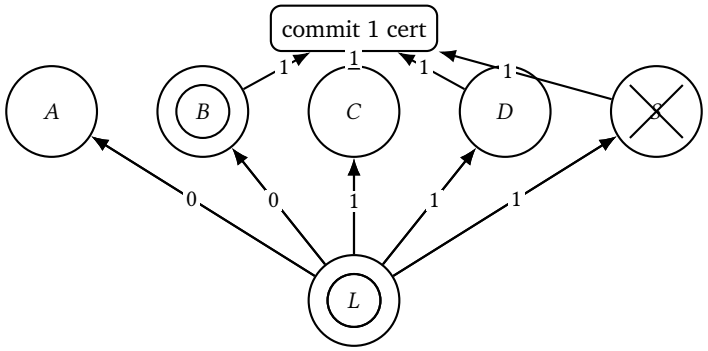
Without loss of generality, assume that the protocol "prefers" 0, *i.e.*, $\Pr[\text{commit } 0] \geq 1/2$. All messages take exactly Δ time to reach their recipients.

Attack world (S gets 1, then crashes). In this world, a malicious leader sends 0 to A, B and 1 to C, D. The adversary corrupts the $t - 1$ parties in B. In addition, the adversary

Split world: Leader sends 0 to A and B, and sends 1 to C and D; S crashes. WLOG, assume 0 is committed most of the time



Attack world: Leader and B and S secretly help commit to 1. Towards the honest in A, C, D, the parties in L, B, S act just like in the split world.



Validity world: Leader sends 1 and A crashes. A commit certificate for 1 is created before 3Δ

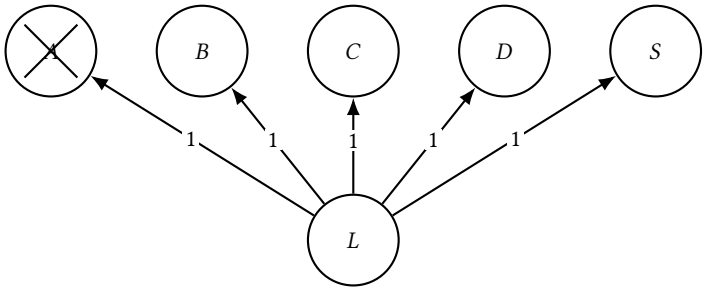


Fig. 1. Three executions used in the proof of Theorem 6. The node placement (geometry) is identical across worlds.

omission-crash corrupts the parteis in S. It then sends 1 to the parties in S, after which they send their responses to the adversary and then crash. Note that this is exactly the round

where the adversary can decide which messages sent by a omission-crashed party arrive at their recipients.

The Attack world and the Split world are indistinguishable to the correct parties in A and D, and in the Split world, a commit for 0 is created at least half the time. However, within the Attack world, the adversary can also *simulate* an execution, in which the leader, the $t-1$ malicious parties in B, and the c omission-crashed parties in S all received 1 and sent their round-two messages to the adversary. This is in addition to the $t+p-1$ correct parties in C, D that receive 1, giving a total of $t+c+(t+p-1) = 2t+p+c-1 = n-p$ round-two messages for the value 1, including the leader's messages.

Finally, the execution simulated by the adversary is indistinguishable from the execution in the Validity world, where there is a certificate for 1. This implies that the adversary holds a commit certificate for 1 in the Attack world. This is a contradiction because the correct parties form a commit certificate for 0 in at least half of the executions. □

For $p = 0$, the mixed-fault lower bound $n > 2t + c$ (equivalently, impossibility when $n \leq 2t + c$) is proved in [2, 3].

The following theorem states a lower bound on the synchronous commit latency in an optimistically responsive protocol in the presence of many faulty parties.

Theorem 7 (Theorem 6 [35])**.** *For $0 < \alpha < \Delta$, there does not exist a Byzantine broadcast protocol in the unsynchronized start model that can tolerate $t > n/3$ faults and simultaneously achieve the following when the leader is correct, messages sent by the correct parties arrive instantaneously, and all correct parties start at time 0:*

(1) Fast Commit: all correct parties commit before round α when there are $\max\{1, n-2t\}$ crash faults; and

(2) Synchronous Commit: all correct parties commit before time $2\Delta - \alpha$ when there are t crash faults.

Unsynchronized start means that the correct parties can start the protocol execution at different times decided by an adversary as long as they all start by round Δ and before receiving any message from the other parties.

Discussion. Theorem 6 implies that no view-based PS-SMR protocol with $n = 2t + 2p + c - 1$ can tolerate t Byzantine and c omission-crash faults and achieve T-good case latency for any $T < 3\Delta$ when there are p or more crash faults. In Section 4.1, we show a *resilience-optimal* PS-SMR protocol, namely Δ-Sync Simplex with Fast Path, that matches this converse.

Note that neither Theorem 6 nor Theorem 7 rule out a *non-provable* fast commit rule that allows all correct parties to commit before round $T \leq 3\Delta$ when there are more than p crash faults. However, Theorem 7 implies that Δ-Sync Simplex with its latency of $(\Delta + 3\delta)$ against t Byzantine parties cannot achieve $O(\delta)$-good case latency against $n - 2t > p$ crash faults, even through a non-provable fast commit rule. Therefore, to tightly match the converse results for resiliences in both theorems, in Section 4.2, we provide a *resilience-optimal* PS-SMR protocol with over 2Δ latency against t faulty parties, namely 2Δ-Sync Simplex with Fast Path, which has a non-provable fast commit rule that ensures a good-case latency of 3δ when there are at most $\lfloor (n+c)/2 \rfloor + p$ crash faults.

Going forward, we will refer to omission-crash faulty parties as simply crash-faulty to better separate them from fully omission-faulty parties.

4 Protocols

All of the protocols proceed in *views*, denoted by $v \in \mathbb{Z}^+$. Each view is associated with a unique leader $\mathsf{Leader}(v)$. For simplicity, we assume that the leaders are determined via round-robin, but it is also possible to select them using a verifiable random function (VRF).

Blocks and chains. Values are batched into blocks, denoted by B. Except a special *genesis block*, each block refers to a parent block via a hash pointer. Therefore, each block B defines a unique chain that starts at the genesis block and ends at B. A block B is represented by the tuple $\mathsf{B} = (v, k, h, \mathsf{v})$, where $\mathsf{B}.v$ is a view number, $\mathsf{B}.k$ is the block's height within its chain, $\mathsf{B}.h$ is the hash of its parent block, and $\mathsf{B}.\mathsf{v}$ is a value included by the block. We denote the parent block of B by $\mathsf{B}.\mathsf{parent}$. For the genesis block, $\mathsf{B}_0 = (0, 0, h_0, \mathsf{v}_0)$, where h_0 and v_0 are special values. As such, any given block also uniquely determines a log that consists of the values within its chain.

Upon receiving a new value, correct parties broadcast the value to all other parties. At the start of each view v, the view leader proposes a new block B, where $\mathsf{B} = (v, k+1, \mathsf{hash}(\mathsf{B}'), \mathsf{v})$, if it extends a blockchain of height k that ends at a block B'. Here, v implicitly contains all values that were input to the leader or received from other parties, yet have not been included in the chain defined by B'.

For the protocols in Sections 4.1 and 4.2, two blocks B and B' of height k and k' are said to conflict with each other if their chains have a different block at some height in $\{1, \ldots, \min(k, k')\}$. A block B is said to extend another block B' if the chain of B' is a prefix of the chain of B.

In the protocol descriptions, 'relay' implies sending a received message to all other parties.

4.1 Δ-Sync Simplex with Fast Path

Let $n = 2t + 2p + c + 1$, $p < t$.

Messages and certificates. The protocol uses four types of messages: *propose*, *echo*, *vote* and *final*, denoted respectively by $(\mathsf{Propose}, v, \mathsf{B})$, $(\mathsf{Echo}, v, \mathsf{B})$, $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{isEarly}(\mathsf{B}))$ and $(\mathsf{Final}, v, \mathsf{B})$. All messages are signed by their senders, and the parties consider only the messages received from those identified by the PKI. The vote messages contain a boolean value called $\mathsf{isEarly} \in \{\mathsf{True}, \mathsf{False}\}$, which denotes if the block B was observed early in the view by the party sending the vote.

There are also five types of certificates built from signed protocol messages.

- A *fast commit certificate* $\mathsf{Fast\text{-}Cert}(v, \mathsf{B})$ consists of $2t + p + c + 1$ vote messages of the form $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$. It acts as a commit proof for the block B and its chain.
- An *echo certificate* $\mathsf{Echo\text{-}Cert}(v, \mathsf{B})$ consists of $t + p + 1$ echo messages of the form $(\mathsf{Echo}, v, \mathsf{B})$, where $\mathsf{B} \neq \perp$.
- A *vote certificate* $\mathsf{Vote\text{-}Cert}(v, \mathsf{B})$ consists of $t + p + 1$ vote messages of the form $(\mathsf{Vote}, v, \mathsf{B}, *)$, which can have different $\mathsf{isEarly}$ values.
- A *slow commit certificate* $\mathsf{Slow\text{-}Cert}(v, \mathsf{B})$ consists of $t + p + 1$ final messages of the form $(\mathsf{Final}, v, \mathsf{B})$. It acts as a commit proof for the block B and its chain.
- A *skip echo certificate* $\mathsf{Skip\text{-}Echo\text{-}Cert}(v)$ consists of $t + p + 1$ echo messages of the form $(\mathsf{Echo}, v, \perp)$.
- A *skip vote certificate* $\mathsf{Skip\text{-}Vote\text{-}Cert}(v)$ consists of $t + p + 1$ vote messages of the form $(\mathsf{Vote}, v, \perp, \mathsf{False})$.

Δ-Sync Simplex with Fast Path

View v: ▷ The genesis block is assumed to have a vote certificate: Vote-Cert$(0, \mathsf{B}_{\text{genesis}})$.
▷ Proposal:
  Upon start view v:
    isEarly$(*) \leftarrow$ False;
    Start timer $T_{v,1}$; ▷ B_{inp} below denotes view leader's block.
    Leader(v) sends (Propose, v, B_{inp}) bundled with cert. that imply isValid$(v-1, \mathsf{B}.\text{parent})$;
▷ Echo step:
  Upon first (Propose, v, B) by Leader(v) **s.t.** isValid$(v-1, \mathsf{B}.\text{parent})$:
    Start timer $T_{v,2}$;
    Relay (Propose, v, B) by the leader;
    If $T_{v,1} < 2\Delta$:
      Send (Echo, v, B) bundled with all cert. that imply isValid$(v-1, \mathsf{B}.\text{parent})$;
      isEarly(B) $\leftarrow$ True;
  Upon equivocation:
    Send equivocating proposals;
▷ Fast commit:
  Upon $2t+p+c+1$ messages (Echo, v, B): ▷ This is a Fast-Cert(v, B).
    Commit B and all its ancestors and send Fast-Cert(v, B);
    Enter view $v+1$;
▷ Vote step:
  Once $T_{v,2} = \Delta$:
    If no equivocation detected:
      Send (Vote, v, B, isEarly(B)) bundled with all cert. that imply isValid$(v-1, \mathsf{B}.\text{parent})$;
▷ Slow commit:
  While $T_{v,1} < 4\Delta$: ▷ This can happen during a future view as well.
    If there are $t+p+1$ (Vote, v, B, True):
      Relay the $t+p+1$ (Vote, v, B, True);
      Send (Final, v, B);
  Upon $t+p+1$ (Final, v, B): ▷ This is a Slow-Cert(v, B).
    Commit B and all its ancestors and send Slow-Cert(v, B);
▷ View change:
  Upon $t+p+1$ (Vote, v, B, $*$): ▷ This is a Vote-Cert(v, B).
    Enter view $v+1$
  Upon $t+p+1$ (Vote, v, $\perp$, False) and ▷ This is a Skip-Vote-Cert(v).
    (Skip-Echo-Cert(v) or Echo-Cert(v, B) s.t. isValid$(v-1, \mathsf{B}.\text{parent})$ = True):
    Enter view $v+1$
▷ Skip votes:
  Upon $T_{v,1} = 6\Delta$:
    If no Vote-Cert$(v, \cdot)$:
      Send (Vote, v, $\perp$, False) ▷ $t+p+1$ (Vote, v, $\perp$, False) is a Skip-Vote-Cert(v).
    If no Echo-Cert$(v, \cdot)$:
      Send (Echo, v, $\perp$) ▷ $t+p+1$ (Echo, v, $\perp$) is a Skip-Echo-Cert(v).

Predicate isValid(v, B): True if and only if
(i) B defines a blockchain of increasing views ending at v, and
(ii) For every block B′ of view v' in the chain defined by B,
  (a) there is a Vote-Cert(v', B'), or
  (b) there is a Fast-Cert(v', B'),
  (c) there are Echo-Cert(v', B') and Skip-Vote-Cert(v'),
(iii) For every view $v' = \mathsf{B}.v, \ldots, v$, and views v' s.t. there is no view v' block in B's prefix,
  There are both Skip-Echo-Cert(v') and Skip-Vote-Cert(v').

Intuitively, the skip vote certificates are used to signal that no block has been committed in a view. Whenever a party forms or observes any certificate or proposal, it broadcasts that certificate or proposal to all parties.

Proposal. Upon entering view v, each party starts a timer $T_{v,1}$. The view leader $\mathsf{Leader}(v)$ selects a chain ending at some block B such that $\mathsf{isValid}(v, \mathsf{B}) = \mathsf{True}$, and attaches its block to the tip of this chain. It then broadcasts its proposal $(\mathsf{Propose}, v, \mathsf{B}_{\mathsf{inp}})$.

Echo step. Upon receiving the *first* proposal $(\mathsf{Propose}, v, \mathsf{B})$ from $\mathsf{Leader}(v)$ s.t. $\mathsf{isValid}(v - 1, \mathsf{B.parent}) = \mathsf{True}$, each party relays the proposal and then starts a second timer $T_{v,2}$. If the proposal arrived *early* in the view, *i.e.*, before $T_{v,1} = 2\Delta$, it broadcasts an echo $(\mathsf{Echo}, v, \mathsf{B})$ for the proposal and sets $\mathsf{isEarly}(x) := \mathsf{True}$. The echo messages enable fast-committing the proposed block via $\mathsf{Fast\text{-}Cert}(v, \mathsf{B})$. If the party detects equivocation by $\mathsf{Leader}(v)$, *i.e.*, observes different blocks proposed for view v, it broadcasts the equivocating proposals to notify the other parties.

Vote step. Once $T_{v,2} = \Delta$, *i.e.*, Δ time after observing a valid proposal, each party that has *not* detected equivocation broadcasts a vote $(\mathsf{Vote}, v, x, \mathsf{isEarly}(x))$. The timer $T_{v,2}$ prevents the parties from voting for conflicting blocks, as the first party to observe a proposal B communicates this to all other parties within Δ time, *i.e.*, before the other parties vote for a conflicting block.

Each echo and vote message also relays the leader's signature on the proposal, thus, in effect, relaying the proposal.

Slow and fast commit. Each party waits until $T_{v,1} = 4\Delta$ to see if they receive $t + p + 1$ votes such that $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$, *i.e.*, with $\mathsf{isEarly} = \mathsf{True}$ for all votes. In this case, the party relays these votes and then sends a final message $(\mathsf{Final}, v, \mathsf{B})$ for B. Note that the party can proceed to the next view without sending a final or committing the block: it might send a final for view v while in view $v + 1$, and slow commit may occur later when the party observes a $\mathsf{Slow\text{-}Cert}(v, \mathsf{B})$ containing $t + p + 1$ final messages.

Parties use two commit rules *concurrently*: (i) a responsive fast commit triggered by a $\mathsf{Fast\text{-}Cert}(v, \mathsf{B})$, and (ii) a synchronous slow commit triggered by a $\mathsf{Slow\text{-}Cert}(v, \mathsf{B})$. Crucially, they do not have to "decide" which rule to follow: whichever certificate is formed first causes the corresponding commit.

View change. Upon collecting $t + p + 1$ votes $(\mathsf{Vote}, v, \mathsf{B}, *)$ for a block $\mathsf{B} \neq \perp$ (where the last bit can vary), each party forms a $\mathsf{Vote\text{-}Cert}(v, \mathsf{B})$ and advances to the next view. Similarly, upon receiving $t + p + 1$ $(\mathsf{Vote}, v, \perp, \mathsf{False})$, each party forms a $\mathsf{Skip\text{-}Vote\text{-}Cert}(v)$ and advances to the next view. However, in this case, before advancing views, the parties make sure to obtain either a $\mathsf{Skip\text{-}Echo\text{-}Cert}(v)$, or an $\mathsf{Echo\text{-}Cert}(v, \mathsf{B})$.

Skip votes. Once $T_{v,1} = 6\Delta$, if the party has not observed any $\mathsf{Vote\text{-}Cert}(v, \mathsf{B})$ for some block B, it sends a vote $(\mathsf{Vote}, v, \perp, \mathsf{False})$ with the special value $\perp$.

Proposal validity predicate $\mathsf{isValid}(v, \mathsf{B})$. The predicate checks for the following conditions:

(1) *Well-formed chain.* The block B must define a blockchain whose blocks have strictly increasing view numbers.
(2) *Per-view justification.* For each block B' with view v' in this chain,
   - there is a $\mathsf{Vote\text{-}Cert}(v', \mathsf{B}')$, or
   - there is a $\mathsf{Fast\text{-}Cert}(v', \mathsf{B}')$.
   - there are both an $\mathsf{Echo\text{-}Cert}(v', \mathsf{B}')$ and a $\mathsf{Skip\text{-}Vote\text{-}Cert}(v')$.

(3) *Skipped views.* For all views $v' = \mathsf{B}.v, \ldots, v$, and views v' such that there is no view v' block in the prefix of B, there is a Skip-Echo-Cert(v') and a Skip-Vote-Cert(v').

In other words, isValid forces the leader to present a chain that is consistent with what the parties could have committed in the earlier views, and to account for any gaps via skip certificates. Note that block B may not be of view v; in this case, isValid(v, B) requires Skip-Echo-Cert(v') and Skip-Vote-Cert(v') for every view $v' \leq v$ that is greater than the view of B.

Remark 1. It is technically not necessary for isValid(v, B) to check for the certificates Skip-Vote-Cert(v') and Skip-Echo-Cert(v') for the views $v' < \mathsf{B}.v$ with a missing block in the chain of B (it must still check the presence of these certificates for every view v' such that $\mathsf{B}.v < v' \leq v$). This is because for any block B' in the chain of B, a Fast-Cert(v', B'), Vote-Cert(v', B') or an Echo-Cert(v', B') can exist only if a correct party sent an echo or vote message for that block, *i.e.*, only if a correct party observes that isValid$(v', \mathsf{B}'.\mathsf{parent}) = \mathsf{True}$. This in turn implies that this correct party must have observed Skip-Vote-Cert(v'') and Skip-Echo-Cert(v'') for the views v'' such that $\mathsf{B}'.\mathsf{parent}.v < v'' \leq v'$, *i.e.*, there are in fact Skip-Vote-Cert(v') and Skip-Echo-Cert(v') for the views in between those of the blocks in B.

Remark 2. For the views v' with missing blocks in the chain of B, isValid(v, B) requires not only a Skip-Vote-Cert(v'), but also a Skip-Echo-Cert(v'). This is to ensure safety when the adversary creates a 'hidden' Fast-Cert(v', B') for some block B' with the echo messages of t Byzantine parties, when there are already $t + p + c + 1$ $(\mathsf{Echo}, v', \mathsf{B}')$. In this case, all correct parties observe an Echo-Cert(v', B') by 5Δ into the view v', and thus no correct party sends an $(\mathsf{Echo}, v', \perp)$, implying that a Skip-Echo-Cert(v') cannot exist, and the predicate must ask to see an Echo-Cert(v', B').

Remark 3. Bundled with a vote or echo for a block B at some view v, each party conveys the certificates that imply isValid$(v - 1, \mathsf{B}.\mathsf{parent}) = \mathsf{True}$. This is to guarantee that the parties observe all certificates that imply isValid$(v, \mathsf{B}) = \mathsf{True}$ for some block B right before exiting any view v, and thus the next leader can propose a block immediately upon entering view $v + 1$. In a practical implementation, this step can be avoided by stipulating that upon observing a certificate that enable advancing to some view $v + 1$, the parties also wait to hear all other certificates that imply isValid$(v - 1, \mathsf{B}.\mathsf{parent}) = \mathsf{True}$ before advancing the view.

Remark 4. While sending a $(\mathsf{Final}, v, \mathsf{B})$, the order of messages matters: the $t + p + 1$ $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$ must be relayed before the $(\mathsf{Final}, v, \mathsf{B})$. This is to guarantee that if there is a Slow-Cert(v, B) which contains a $(\mathsf{Final}, v, \mathsf{B})$ by a crash-faulty party, this party has successfully relayed all votes, and thus prevented any correct or crash-faulty party from sending a $(\mathsf{Vote}, v, \perp, \mathsf{False})$, thus the formation of a Skip-Vote-Cert(v). A similar observation is used for the order of votes and proposals: a $(\mathsf{Vote}, v, \mathsf{B},)$ by a crash-faulty party implies that the party has successfully relayed the corresponding proposal for B to all other parties.

4.2 2Δ-Sync Simplex with Fast Path

Let $n = 2t + 2p + c + 1$, $p < t$.

Messages and certificates. The protocol uses three types of messages: *propose*, *vote* and *final*, denoted respectively by $(\mathsf{Propose}, v, \mathsf{B})$, $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{isEarly}(\mathsf{B}))$ and $(\mathsf{Final}, v, \mathsf{B})$. All messages are signed by their senders, and the parties consider only the messages received from those identified by the PKI. The vote messages contain a boolean value called $\mathsf{isEarly} \in \{\mathsf{True}, \mathsf{False}\}$, which denotes if the block B was observed early in the view by the party sending the vote.

2Δ-Sync Simplex with Fast Path

View k: ▷ The genesis block is assumed to have a vote certificate: Vote-Cert$(0, \mathsf{B}_{\text{genesis}})$.
▷ Proposal:
  Upon start view v:
    Start timer $T_{v,1}$;
    Leader(v) sends $(\text{Propose}, v, \mathsf{B}_{\text{inp}})$ bundled with cert. that imply isValid$(v-1, \mathsf{B}.\text{parent})$;
▷ Vote step:
  Upon first $(\text{Propose}, v, \mathsf{B})$ by Leader(v) **s.t.** isValid$(v-1, \mathsf{B}.\text{parent})$:
    Relay $(\text{Propose}, v, \mathsf{B})$ by the leader;
    If $T_{v,1} < 2\Delta$:
      Start timer $T_{v,2}$;
      Send $(\text{Vote}, v, \mathsf{B}, \text{True})$ bundled with cert. that imply isValid$(v-1, \mathsf{B}.\text{parent})$;
    If $2\Delta \le T_{v,1} < 4\Delta$:
      Send $(\text{Vote}, v, \mathsf{B}, \text{False})$ bundled with all cert. that imply isValid$(v-1, \mathsf{B}.\text{parent})$;
  Upon equivocation:
    Send equivocating proposals;
▷ Fast commit path:
  Upon $2t+p+c+1$ messages $(\text{Vote}, v, \mathsf{B}, \text{True})$: ▷ This is a Fast-Cert$(v, \mathsf{B})$.
    Commit B and all its ancestors;
    Enter view $v+1$;
▷ Mid commit path:
  Upon $\lfloor (3t+c)/2 \rfloor + p + 1$ messages $(\text{Vote}, v, \mathsf{B}, \text{True})$: ▷ This is a Mid-Cert$(v, \mathsf{B})$.
    Relay the $\lfloor (3t+c)/2 \rfloor + p + 1$ $(\text{Vote}, v, \mathsf{B}, \text{True})$;
    If $T_{v,1} < 3\Delta$:
      Send $(\text{Final}, v, \mathsf{B})$;
    Enter view $v+1$;
▷ Slow commit path:
  Once $T_{v,2} = 2\Delta$:
    If no equivocation detected:
      Send $(\text{Final}, v, \mathsf{B})$ ▷ where B is the block from the early proposal
▷ Commit:
  Upon $t+p+1$ messages $(\text{Final}, v, \mathsf{B})$: ▷ This is a Final-Cert$(v, \mathsf{B})$.
    Commit B and all its ancestors;
▷ View change:
  Upon $t+p+1$ $(\text{Vote}, v, \mathsf{B}, *)$ and Skip-Mid-Cert(v): ▷ The former is a Vote-Cert(v, B).
    Enter view $v+1$
  Upon $t+p+1$ $(\text{Vote}, v, \perp, \text{False})$ and Skip-Mid-Cert(v) ▷ The former is a Skip-Vote-Cert(v).
    Enter view $v+1$
▷ Skip votes:
  Upon $T_{v,1} = 5\Delta$:
    If no Vote-Cert$(v, \cdot)$:
      Send $(\text{Vote}, v, \perp, 0)$; ▷ $t+p+1$ $(\text{Vote}, v, \perp, 0)$ is a Skip-Vote-Cert(v).
    If no Mid-Cert$(v, \cdot)$:
      Send $(\text{Mid-Skip}, v)$ ▷ $t+p+1$ $(\text{Mid-Skip}, v)$ is a Skip-Mid-Cert(v).

Predicate isValid(v, B): True if and only if
(i) B defines a blockchain of increasing views, and
(ii) for every block B' of view v' in the chain defined by B, either
  (a) there is a Mid-Cert(v', B'), or
  (b) there are Vote-Cert(v', B') and Skip-Mid-Cert(v'), and
(iii) For every view $v' = \mathsf{B}.v, \ldots, v$, and views v' s.t. there is no view v' block in B's prefix,
  There are both Skip-Vote-Cert(v') and Skip-Mid-Cert(v').

There are also six types of certificates built from signed protocol messages.

- A *fast commit certificate* Fast-Cert(v, B) consists of $2t + p + c + 1$ vote messages of the form $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$. It acts as a commit proof for the block B and its chain.
- A *mid vote certificate* Mid-Cert(v, B) consists of $\lfloor (3t + c)/2 \rfloor + p + 1$ vote messages of the form $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$.
- A *vote certificate* Vote-Cert(v, B) consists of $t + p + 1$ vote messages of the form $(\mathsf{Vote}, v, \mathsf{B}, *)$, which can have different isEarly values.
- A *slow commit certificate* Slow-Cert(v, B) consists of $t + p + 1$ final messages of the form $(\mathsf{Final}, v, \mathsf{B})$. It acts as a commit proof for the block B and its chain.
- A *skip mid certificate* Skip-Mid-Cert(v) consists of $t + p + 1$ mid-skip messages of the form $(\mathsf{Mid\text{-}Skip}, v)$.
- A *skip vote certificate* Skip-Vote-Cert(v) consists of $t + p + 1$ vote messages of the form $(\mathsf{Vote}, v, \perp, \mathsf{False})$.

Intuitively, the skip mid and skip vote certificates are used to signal that no block has been committed in a view. Whenever a party forms or observes any certificate, it broadcasts that certificate to all parties.

Proposal. Upon entering view v, each party starts a timer $T_{v,1}$, and the view leader Leader(v) broadcasts its proposal $(\mathsf{Propose}, v, \mathsf{B}_{\mathsf{inp}})$.

Vote step. Upon receiving the first proposal $(\mathsf{Propose}, v, \mathsf{B}_{\mathsf{inp}})$ from Leader(v) such that isValid$(v - 1, \mathsf{B}_{\mathsf{inp}}.\mathsf{parent}) = \mathsf{True}$, each party does as follows:

- It relays the proposal.
- *Early proposal:* If $T_{v,1} < 2\Delta$, it starts a second timer $T_{v,2}$ and broadcasts a vote $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$, signalling that the proposal was observed early in the view.
- *Late proposal:* If $2\Delta \leq T_{v,1} < 4\Delta$, it broadcasts a vote $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{False})$, signalling that the proposal was observed late.

If the party detects equivocation by Leader(v) (*i.e.*, multiple proposals for view v with different blocks), it broadcasts the equivocating proposals to notify the other parties.

Fast commit path. Upon collecting $2t + p + c + 1$ votes of the form $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$, each party forms a fast commit certificate Fast-Cert(v, B), commits B and its ancestors, and advances to view $v + 1$.

Mid commit path. Upon collecting $\lfloor (3t + c)/2 \rfloor + p + 1$ votes of the form $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$, each party forms and relays a mid vote certificate Mid-Cert(v, B) and advances to view $v + 1$. If this happens before $T_{v,1} = 3\Delta$, it also broadcasts a final message $(\mathsf{Final}, v, \mathsf{B})$.

Slow commit path. If a party started the timer $T_{v,2}$ for some proposed block B before 2Δ and has not detected equivocation, then once $T_{v,2} = 2\Delta$, it broadcasts $(\mathsf{Final}, v, \mathsf{B})$.

Commit. Upon collecting $t + p + 1$ $(\mathsf{Final}, v, \mathsf{B})$, each party forms a slow commit certificate Slow-Cert(v, B) and commits B along with its ancestors.

View change. Once a party collects $t + p + 1$ vote messages $(\mathsf{Vote}, v, \mathsf{B}, *)$ for some block B (where the last bit can vary) and obtains a Skip-Mid-Cert(v), it forms a Vote-Cert(v, B) with the vote messages and advances to view $v + 1$. Similarly, upon receiving $t + p + 1$ $(\mathsf{Vote}, v, \perp, \mathsf{False})$, each party forms a Skip-Vote-Cert(v) and advances to the next view.

Skip votes. Once $T_{v,1} = 5\Delta$:

- If the party has not observed any Vote-Cert(v, B), it broadcasts $(\text{Vote}, v, \perp, 0)$. A quorum of $t + p + 1$ such messages constitutes a skip vote certificate Skip-Vote-Cert(v) indicating that no block was committed in view v.
- If the party has not observed any Mid-Cert(v, B), it broadcasts $(\text{Mid-Skip}, v)$. A quorum of $t + p + 1$ such messages constitutes a skip mid certificate Skip-Mid-Cert(v).

Proposal validity predicate isValid(v, B). The predicate checks for the following conditions:

(1) *Well-formed chain.* The block B must define a blockchain whose blocks have strictly increasing view numbers.
(2) **Per-view justification.** For each block B' with view v' in this chain, either:
   (a) there is a Mid-Cert(v', B'), or
   (b) there is a Vote-Cert(v', B') and a Skip-Mid-Cert(v').
(3) **Skipped views.** For every view $v' = \mathsf{B}.v, \ldots, v$, and views v' such that there is no view v' block in the prefix of B, there exist both Skip-Vote-Cert(v') and Skip-Mid-Cert(v').

Remark 5. While sending a $(\text{Final}, v, \mathsf{B})$, the order of messages matters: the $\lfloor (3t + c)/2 \rfloor + p + 1$ $(\text{Vote}, v, \mathsf{B}, \text{True})$ must be relayed before the $(\text{Final}, v, \mathsf{B})$. This is to guarantee that if there is a Slow-Cert(v, B) which contains a $(\text{Final}, v, \mathsf{B})$ by a crash-faulty party, this party has successfully relayed all votes, and thus prevented any correct or crash-faulty party from sending a $(\text{Mid-Skip}, v)$, thus the formation of a Skip-Mid-Cert(v).

A similar reasoning applies to the order of votes and proposals: the proposal $(\text{Propose}, v, \mathsf{B})$ must be relayed before $(\text{Vote}, v, \mathsf{B},)$. This is to ensure that if there is a Vote-Cert(v, B) which contains a $(\text{Vote}, v, \mathsf{B}, \text{True})$ by a crash-faulty party, this party has sucessfully relayed the proposal, and thus prevented any correct or crash-faulty party from sending a conflicting vote.

4.3 TenderSync with 4Δ Views, 3Δ Commit, and 2Δ Fast Commit

We present a fixed-view-schedule (lock-step) protocol, called TenderSync, with view length 4Δ, good-case commit latency of 3Δ for $n = 2t + y + 2p + 1$ tolerating t Byzantine faults, $o = \min(y, p)$ omission faults and $c = \max(y - p, 0)$ crash faults, such that it provides fast path commit latency of 2Δ against p Byzantine faults. The protocol follows a similar approach as that of Abraham et al. [1] but additionally allows for provable commit certificates, a fast path commit in the presence of p faults, and support for additional crash and omission fault tolerance. Achieving this fast path requires introducing the notion of a "weak lock" to ensure both safety and liveness properties of the protocol.

We number views starting from $v = 1$. Each view v starts at global time $4\Delta \cdot v$. Within view v, Propose/Vote/Lock/$\text{Lock}_{\text{high}}$ are sent at times $4\Delta v$, $4\Delta v{+}\Delta$, $4\Delta v{+}2\Delta$, and $4\Delta v{+}3\Delta$ respectively. Whenever convenient, we refer to these as rounds 1, 2, 3, and 4 of view v. Commit is done at time $4\Delta v{+}3\Delta$. In the best case (at most p faults), the value can be fast-committed at time $4\Delta v{+}2\Delta$.

Messages and certificates. We use the following message types (all signed): Propose(v, B, Π), Vote(v, B), WeakLock(v, B), and Lock(v, B).

There are four types of certificates built from signed protocol messages:

- A *Weak lock certificate* for (v, B) is a set of $t{+}p{+}1$ votes Vote(v, B).
- A *Lock certificate* for (v, B) is a set of (i) $n - t - y$ votes Vote(v, B), or (ii) a set of $n - t - y$ weak locks WeakLock(v, B).
- A *Fast commit certificate* for (v, B) is a set of $n{-}p$ votes Vote(v, B).
- A *Commit certificate* for (v, B) is a set of $n - t - y$ locks Lock(v, B).

TenderSync with 4Δ Views, 3Δ Commit, and 2Δ Fast Commit

Time $4\Delta v$**:** ▷ Round 1
  If one or more valid $\mathsf{Lock}(\max(v-1,1),*)$ exists, then update Π with one of them
  $\mathsf{Leader}(v)$ sends $\mathsf{Propose}(v, \mathsf{B}, \Pi)$ where B extends $\Pi.\mathsf{value}$;
Time $4\Delta v{+}\Delta$**:** ▷ Round 2
  If received unique valid $\mathsf{Propose}(v, \mathsf{B}, \Pi)$ **s.t.** $\Pi.\mathsf{view} \geq \mathsf{Lock}_{\mathsf{high}}.\mathsf{view}$:
    Relay $\mathsf{Propose}(v, \mathsf{B}, \Pi)$
    Send $\mathsf{Vote}(v, \mathsf{B})$;
  Upon equivocation:
    Send equivocating proposals; ▷ Receiving multiple proposal value signed by the leader
Time $4\Delta v{+}2\Delta$**:** ▷ Round 3
  If received $n-p$ distinct valid $\mathsf{Vote}(v, \mathsf{B})$:
    Form a fast commit certificate for B; ▷ fast commit at 2Δ
  If received $n-t-y$ distinct valid $\mathsf{Vote}(v, x)$ and no equivocation detected:
    Send $\mathsf{Lock}(v, \mathsf{B})$; ▷ lock
    Send $\mathsf{Final}(v, \mathsf{B})$; ▷ final
  Else if received $t{+}p{+}1$ distinct valid $\mathsf{Vote}(v, \mathsf{B})$:
    Send $\mathsf{WeakLock}(v, \mathsf{B})$; ▷ weak lock
Time $4\Delta v{+}3\Delta$**:** ▷ Round 4
  If received $n-t-y$ distinct valid $\mathsf{Final}(v, \mathsf{B})$:
    Form a commit certificate and commit B and all its ancestors; ▷ commit at 3Δ
  If received a valid $\mathsf{Lock}(v, \mathsf{B})$ then
    Set $\mathsf{Lock}_{\mathsf{high}} = \mathsf{Lock}(v, \mathsf{B})$;
  Else if received $n-t-y$ distinct valid $\mathsf{WeakLock}(v, \mathsf{B})$:
    Set $\mathsf{Lock}_{\mathsf{high}} = \mathsf{Lock}(v, \mathsf{B})$;
  Send $\mathsf{Lock}_{\mathsf{high}}$

Local state. Each party maintains (i) Π, the highest view/value pair supported by a lock/weak lock it has observed. Initially, parties set $\Pi = (0, \mathsf{B}_{\mathsf{genesis}})$; and (ii) $\mathsf{Lock}_{\mathsf{high}}$, the highest locked view/value pair it has observed (initially $(-1, \perp)$).

A party commits a block B in view v once it forms a commit certificate or fast commit certificate for (v, B) for either itself or one of its descendents.

Protocol description. The protocol runs views, where each view has four rounds. In the first round of view v, the leader receives the highest lock $\mathsf{Lock}_{\mathsf{high}}$ from all non-faulty parties sent in view $v-1$; it updates its Π using this lock. The leader creates a proposal based on the value of Π; in particular, it creates a block extending the block contained in Π.

In the second round, a party first checks if the received proposal $\mathsf{Propose}(v, \mathsf{B}, \Pi)$ is *valid* by checking (i) whether Π is a valid lock certificate, and (ii) B extends $\Pi.\mathsf{value}$. On receiving such a unique valid proposal from a leader in view v, if the key corresponding to this proposal is from a view higher than or equal to $\mathsf{Lock}_{\mathsf{high}}.\mathsf{view}$ maintained by the party, then the party first relays the proposal and votes for it. The ordering of the messages as completing the relay of the proposal to all parties before voting ensures that a potentially crashing party will forward the received proposal to all parites first before starting to vote on the block; thus, even if it crashes in the middle of sending the vote messages, the proposal sending has succeeded. If the party observes more than one proposal, it forwards each of these proposals to all parties.

SyncMint with 3Δ Views, 4Δ Commit, and 2Δ Fast Commit

Time $3\Delta v$**:** ▷ Round 1
  If received $t{+}p{+}1$ distinct valid $\mathsf{Key}(v-1, \mathsf{B})$
    Send $\mathsf{Lock}(v-1, \mathsf{B})$;
    Send $\mathsf{Final}(v-1, \mathsf{B})$;
  Else if received $n-y-t = t{+}2p{+}1$ distinct valid mesg. in $\{\mathsf{Key}(v-1, \mathsf{B}), \mathsf{WeakKey}(v-1, \mathsf{B})\}$
    Send $\mathsf{WeakLock}(v-1, \mathsf{B})$;
  Set Π to be an element in $\{\mathsf{Key}(v-1, \cdot), \mathsf{WeakKey}(v-1, \cdot)\}$ (break ties arbitrarily)
  $\mathsf{Leader}(v)$ sends $\mathsf{Propose}(v, \mathsf{B}, \Pi)$ where B extends $\Pi.\mathsf{value}$; ▷ view v leader proposes
Time $3\Delta v{+}\Delta$**:** ▷ Round 2
  If received $n-y-t = t{+}2p{+}1$ distinct valid $\mathsf{Final}(v-1, \mathsf{B})$:
    Form a commit certificate for B;
  If received valid $\mathsf{Lock}(v-1, \mathsf{B})$: ▷ view $v-1$: update highest lock
    $\mathsf{Lock}_{\mathsf{high}} = \mathsf{Lock}(v-1, \mathsf{B})$;
  If received $n-y-t = t{+}2p{+}1$ distinct valid $\mathsf{WeakLock}(v-1, \mathsf{B})$
    $\mathsf{Lock}_{\mathsf{high}} = \mathsf{Lock}(v-1, \mathsf{B})$;
  If received first valid $\mathsf{Propose}(v, \mathsf{B}, \Pi)$ **s.t.** $\Pi.\mathsf{view} \geq \mathsf{Lock}_{\mathsf{high}}.\mathsf{view}$: ▷ view v: vote
    Relay $\mathsf{Propose}(v, \mathsf{B}, \Pi)$
    Send $\mathsf{Vote}(v, \mathsf{B})$;
  Upon equivocation:
    Send equivocating proposals; ▷ Receiving multiple proposal values signed by the leader
Time $3\Delta v{+}2\Delta$**:** ▷ Round 3
  If received $n-p = 2t{+}p+y{+}1$ distinct valid $\mathsf{Vote}(v, \mathsf{B})$:
    Form a fast commit certificate for B;
  If received $n-y-t = t{+}2p{+}1$ distinct valid $\mathsf{Vote}(v, \mathsf{B})$ and no equivocation:
    Send $\mathsf{Key}(v, \mathsf{B})$;
  Else if received $t{+}p{+}1$ distinct valid $\mathsf{Vote}(v, \mathsf{B})$:
    Send $\mathsf{WeakKey}(v, \mathsf{B})$ for unique value B (even if multiple values exist);

In the third round, if $n-p = 2t+2p+1$ votes are generated for this block, then that forms a fast commit certificate. If some correct party observers $n-t-y$ votes for some block B, and does not observe any equivocation, it sends a $\mathsf{Lock}(v, \mathsf{B})$. Otherwise, if it receives $t+p+1$ valid votes for B, it sends $\mathsf{WeakLock}(v, \mathsf{B})$ independent of whether it observed an equivocation. If two or more values satisfy this condition, then the party could potentially send a weak lock for $O(1)$ of them.

In the fourth round, if there are $n-t-y$ distinct valid locks sent for B, then it forms a commit certifiate for B. Moreover, every party updates their lock based on receiving Lock messages; if they receive no locks, then they update their locks based on $n-t-y$ $\mathsf{WeakLock}$ if they exist. If there are locks of $n-t-y$ weak locks for several values, one can pick any of them. Finally, they send their highest lock to all parties.

4.4 SyncMint with 3Δ Views, 4Δ Commit, and 2Δ Fast Commit

For $n = 2t+y+2p+1$, we present a fixed-view-schedule (lock-step) protocol, called SyncMint, with view length 3Δ, good-case commit latency of 4Δ against t Byzantine, $o = \min(p, y)$ omission faults, and $c = \max(y-p, 0)$ crash faults, and best-case commit latency of 2Δ against

p faults. To the best of our knowledge, this is the first synchronous protocol where the view duration is 3Δ time.

We number views starting from $v = 1$. Each view v starts at global time $3\Delta \cdot v$. Within view v, propose, vote, key/weakkey are sent at times $3\Delta v$, $3\Delta v{+}\Delta$, and $3\Delta v{+}2\Delta$ respectively. Wherever appropriate, we also call them rounds 1, 2, and 3 of view v. Lock, weak lock and final are sent at time $3\Delta v{+}3\Delta = 3\Delta(v{+}1)$ and commit is done at time $3\Delta v{+}4\Delta = 3\Delta(v{+}1){+}\Delta$ (at the same time that votes for view $v{+}1$ are sent).

Messages and certificates. We use the following message types (all signed): $\mathsf{Propose}(v, \mathsf{B}, \Pi)$, $\mathsf{Vote}(v, \mathsf{B})$, $\mathsf{Key}(v, \mathsf{B})$, $\mathsf{Lock}(v, \mathsf{B})$, $\mathsf{WeakKey}(v, \mathsf{B})$, and $\mathsf{WeakLock}(v, \mathsf{B})$.

There are six types of certificates built from signed protocol messages:

- A *Key certificate* for (v, B) is a set of $n - y - t = t{+}2p{+}1$ votes for (v, B).
- A *Weak key certificate* for (v, B) is a set of $t{+}p{+}1$ votes for (v, B).
- A *Lock certificate* for (v, B) is a set of $t{+}p{+}1$ keys for (v, B).
- A *Weak lock certificate* for (v, B) is a set of $n - y - t = t{+}2p + 1$ messages among $\{$ $\mathsf{Key}(v, \mathsf{B})$, $\mathsf{WeakKey}(v, \mathsf{B})$ $\}$.
- A *Commit certificate* for (v, B) is a set of $n - y - t = t{+}2p{+}1$ $\mathsf{Final}(v, \mathsf{B})$,
- A *Fast Commit certificate* for (v, B) is a set of $n - p = 2t{+}2p{+}1$ $\mathsf{Vote}(v, \mathsf{B})$.

Local state. Each party maintains (i) Π, the highest valid key certificate it has observed. Initially, parties set $\Pi = (0, \mathsf{B}_{\mathsf{genesis}})$ and (ii) $\mathsf{Lock}_{\mathsf{high}}$, the highest valid lock certificate it has observed. Initially, $\mathsf{Lock}_{\mathsf{high}} = (-1, \perp)$.[2]

A party commits a block B in view v once it forms a commit certificate or fast commit certificate for (v, B) for either itself or one of its descendents

Protocol description. The protocol runs in views, where each view has three rounds. In the first round of view v, the leader receives the highest keys and weak keys from the previous view. The leader updates its Π using this information, and creates a proposal containing a block that extends block contained in Π.

In round 2 of view v, on receiving a valid proposal from a leader in view v, if the proof corresponding to this proposal is from a view higher than or equal to $\mathsf{Lock}_{\mathsf{high}}$.view maintained by the party, then the party first relays the proposal and then votes for it. The ordering of the messages as completing the relay of the proposal to all parties before voting ensures that a potentially crashing party will forward the received proposal to all parites first before starting to vote on the block; thus, even if it crashes in the middle of sending the vote messages, the proposal sending has succeeded. If the party observes more than one proposal, it forwards each of these proposals to all parties.

In round 3 of view v, if $n - p = 2t + y + p + 1$ votes are generated for this value, then that forms a fast certificate. If some correct party receives $n - y - t = t + 2p + 1$ valid votes and does not observe any equivocating proposals/votes, then it sends a $\mathsf{Key}(v, \mathsf{B})$. Otherwise, if it receives $t + p + 1$ valid votes for B, it sends a $\mathsf{WeakKey}(v, \mathsf{B})$ independent of whether it observed an equivocation. If two are more values satisfy this condition, then the party sends it for a unique value B.

In the next round (round 1 of view $v + 1$), if a party observes $t + p + 1$ valid keys, it sends $\mathsf{Lock}(v, \mathsf{B})$ to all parties followed by $\mathsf{Final}(v, \mathsf{B})$ to all parties. Otherwise, if it observes $n{-}y{-}y = t{+}2p{+}1$ valid messages in $\{\mathsf{Key}(v, \mathsf{B}), \mathsf{WeakKey}(v, \mathsf{B})\}$, then it sends a $\mathsf{WeakLock}(v, \mathsf{B})$.

[2]Strictly speaking, there is no certificate associated with the initial proof or lock, but we assume it is valid.

In round 2 of view $v+1$, if there are $t+2p+1$ valid Final(v, B), it forms a commit certificate. Moreover, if a party receives a Lock(v, x) or $t+2p+1$ WeakLock(v, B), it sets $\mathsf{Lock}_{\mathsf{high}}$ to (v, B). This will be used as a constraint when voting for view $v+1$.

5 Security Proofs

5.1 Δ-Sync Simplex with Fast Path

Definition 6 (Commit proof verification function for Δ-Sync Simplex)**.** Given a certificate $\mathsf{cr} = (\mathsf{B}, \pi)$, CV$(\mathsf{B}, \pi)$ outputs true if and only if π includes a Fast-Cert(v, B) or a Slow-Cert(v, B), where $v = \mathsf{B}.v$ is the view of block B.

Lemma 1. *All correct parties enter a view within $\delta \leq \Delta$ time of each other, where δ denotes the actual network delay. If a crash-faulty party enters a view, it does so within $\delta \leq \Delta$ time of all correct and other crash-faulty parties that enter the view.*

Proof. Before a correct or crash-faulty party proceeds to a new view, it broadcasts the set of messages that triggered the view change. Therefore, all correct parties receive these messages and advance their view within δ time. Similarly, all crash-faulty parties receive these messages and advance their view within δ time if they have not crashed by then. □

Hereafter, by Lemma 1, we will take it as a given that the correct and crash-faulty parties' $T_{v,1}$ timers are at most Δ apart at any point in time (as long as the party is not crashed).

5.1.1 *Safety Proof.*

Theorem 8 (Safety of Δ-Sync Simplex with Fast Path)**.** *Let $n = 2t+2p+c+1$ and suppose there are at most f_b Byzantine, f_o omission and f_c crash faults. Given the commit proof verification function in Definition 6, Δ-Sync Simplex with Fast Path satisfies safety, if $f_b \leq t$, $f_o \leq \min(p, c)$, and $f_o + f_c \leq c$.*

Proof. By the protocol rules, a non-Byzantine party never sends echo and vote messages for conflicting blocks in the same view.

Lemma 2. *There cannot exist both a* Vote-Cert(v, B) *and a* Vote-Cert(v, B') *for some v,* B*, and* $\mathsf{B}' \neq \mathsf{B}$ *(except with negligible probability).*

Proof. For contradiction, suppose there are both a Vote-Cert(v, B) and a Vote-Cert(v, B') for some v, B, and $\mathsf{B}' \neq \mathsf{B}$. Then, at least one correct or crash-faulty party P must have sent $(\mathsf{Vote}, v, \mathsf{B}, *)$, and similarly, at least one correct or crash-faulty party P' (potentially the same as P) must have sent $(\mathsf{Vote}, v, \mathsf{B}', *)$, except with negligible probability. Without loss of generality, suppose P sends its vote before P'. As P sends a $(\mathsf{Vote}, v, \mathsf{B}, *)$, it could not have been crashed at the time of relaying the proposal $(\mathsf{Propose}, v, \mathsf{B})$. Then, since P' waits for Δ time after observing $(\mathsf{Propose}, v, \mathsf{B}')$ and before sending its vote, it must have received both $(\mathsf{Propose}, v, \mathsf{B})$ and $(\mathsf{Propose}, v, \mathsf{B}')$, *i.e.* noticed an equivocation, before it sends $(\mathsf{Vote}, v, \mathsf{B}', *)$. This is a contradiction. □

Lemma 3. *There cannot exist both a* Fast-Cert(v, B) *and an* Echo-Cert(v, B') *for some v,* B*, and* $\mathsf{B}' \neq \mathsf{B}$ *(except with negligible probability). Similarly, there cannot exist both a* Fast-Cert(v, B) *and a* Skip-Echo-Cert(v) *for some v and* B *(except with negligible probability).*

Proof. Suppose there is a Fast-Cert(v, B) for some v and B, *i.e.*, $2t+p+c+1$ $(\mathsf{Echo}, v, \mathsf{B})$ messages, and an Echo-Cert(v, B'), *i.e.*, $t+p+1$ $(\mathsf{Echo}, v, \mathsf{B}')$ messages for some $\mathsf{B}' \neq \mathsf{B}$. By quorum intersection, at least one non-Byzantine party must have sent both an Echo-Cert(v, B) and an Echo-Cert(v, B') except with negligible probability, which is a contradiction. Moreover,

when there is a Fast-Cert(v, B), at least $t + p + 1$ correct parties must have sent an $(\text{Echo}, v, \mathsf{B})$ within 2Δ time after entering view v, except with negligible probability. Therefore, all correct and crash-faulty parties (that have not crashed yet) receive $t + p + 1$ $(\text{Echo}, v, \mathsf{B})$ messages, *i.e.*, an Echo-Cert(v, B), within 4Δ time after entering view v. Hence, no correct or crash-faulty party sends an $(\text{Echo}, v, \perp)$, and there cannot exist a Skip-Echo-Cert(v). □

Lemma 4. *There cannot exist both a* Fast-Cert(v, B) *and a* Vote-Cert(v, B') *for some* v, $\mathsf{B}, \mathsf{B}' \neq \mathsf{B}$ *(except with negligible probability). Similarly, there cannot exist both a* Fast-Cert(v, B) *and a* Slow-Cert(v, B') *for some* v, $\mathsf{B}, \mathsf{B}' \neq \mathsf{B}$ *(except with negligible probability).*

Proof. Suppose there is a Fast-Cert(v, B) for some v and B, *i.e.*, $2t + p + c + 1$ $(\text{Echo}, v, \mathsf{B})$ messages, and a Vote-Cert(v, B), *i.e.*, $t + p + 1$ $(\text{Vote}, v, \mathsf{B},)$ messages for some $\mathsf{B}' \neq \mathsf{B}$. By quorum intersection, at least one non-Byzantine party must have sent both an Echo-Cert(v, B) and a Vote-Cert(v, B') except with negligible probability, which is a contradiction. Now, a Slow-Cert(v, B') implies that there is at least one $(\text{Final}, v, \mathsf{B}')$ message by a correct or crash-faulty party P, which must have entered view v and observed at least $t + p + 1$ $(\text{Vote}, v, \mathsf{B}, \text{True})$ messages, *i.e.*, a Vote-Cert(v, B'). Hence, by the first part of the lemma, there cannot exist both a Fast-Cert(v, B) and a Slow-Cert(v, B') for some v, B, $\mathsf{B}' \neq \mathsf{B}$ except with negligible probability. □

Lemma 5. *There cannot exist both a* Slow-Cert(v, B) *and a* Vote-Cert(v, B') *for some* v, B, $\mathsf{B}' \neq \mathsf{B}$ *(except with negligible probability). Similarly, there cannot exist both a* Slow-Cert(v, B) *and a* Skip-Vote-Cert(v) *for some* v *and* B *(except with negligible probability).*

Proof. Suppose there is a Slow-Cert(v, B) for some v and B. In other words, there is at least one $(\text{Final}, v, \mathsf{B})$ message by a correct or crash-faulty party P, which must have entered view v and observed at least $t + p + 1$ $(\text{Vote}, v, \mathsf{B}, \text{True})$ messages, *i.e.*, a Vote-Cert(v, B), before $T_{v,1} = 4\Delta$ in its view. Therefore, by Lemma 2, there cannot exist a Vote-Cert(v, B') for some $\mathsf{B}' \neq \mathsf{B}$ (except with negligible probability). Moreover, as P broadcasts a $(\text{Final}, v, \mathsf{B})$, it could not have been crashed at the time of relaying the corresponding $t + p + 1$ $(\text{Vote}, v, \mathsf{B}, \text{True})$ messages, which will be sent before $T_{v,1} = 4\Delta$ in its view. Then, these votes reach all correct and crash-faulty parties (that have not crashed yet) before $T_{v,1} = 6\Delta$ in their views. Consequently, no correct or crash-faulty party sends a $(\text{Vote}, v, \perp, \text{False})$, implying that there cannot exist a Skip-Vote-Cert(v). □

We next consider all possible cases, in which two conflicting blocks B and B' with views v and $v' \geq v$ acquire a valid commit certificate.

Case 1: There exist a Fast-Cert(v, B) and a Fast-Cert(v, B'), which implies an Echo-Cert(v, B'). By Lemma 3, this cannot happen (except with negligible probability).

Case 2: There exist a Fast-Cert(v, B) and a Slow-Cert(v, B'). By Lemma 4, this cannot happen (except with negligible probability).

Case 3: There exist a Slow-Cert(v, B) and Slow-Cert(v, B'), *i.e.*, there is at least one $(\text{Final}, v, \mathsf{B}')$ message by some correct or crash-faulty party, which must have observed at least $t + p + 1$ $(\text{Vote}, v, \mathsf{B}', \text{True})$ messages, *i.e.*, a Vote-Cert(v, B'). By Lemma 5, this cannot happen (except with negligible probability).

Case 4: $v < v'$, and there exist a Fast-Cert(v, B). Since there is a commit proof for B', at least one correct or crash-faulty party must have observed $\text{isValid}(v', \mathsf{B}'.\text{parent}) = \text{True}$. Now, consider the following sub-cases:

- **Case 4-a:** There is a view-v block B'' in the prefix of B' with a Vote-Cert(v, B''). By Lemma 4, this cannot happen (except with negligible probability).
- **Case 4-b:** There is a view-v block B'' in the prefix of B' with a Fast-Cert(v, B''), which implies an Echo-Cert(v, B''). By Lemma 3, this cannot happen (except with negligible probability).
- **Case 4-c:** There is a Skip-Vote-Cert(v) together with a view-v block B'' in the prefix of B' with an Echo-Cert(v, B''). By Lemma 3, these cases cannot happen (except with negligible probability).
- **Case 4-d:** There are both Skip-Echo-Cert(v) and Skip-Vote-Cert(v). By Lemma 3, this cannot happen (except with negligible probability).

Case 5: $v < v'$, and there exist a Slow-Cert(v, B). Since there is a commit proof for B', at least one correct or crash-faulty party must have observed isValid$(v', \mathsf{B}'.\text{parent}) = \text{True}$. Now, consider the following sub-cases:

- **Case 5-a:** There is a view-v block B'' in the prefix of B' with a Vote-Cert(v, B''). By Lemma 5, this cannot happen (except with negligible probability).
- **Case 5-b:** There is a view-v block B'' in the prefix of B' with a Fast-Cert(v, B''). By Lemma 4, this cannot happen (except with negligible probability).
- **Case 5-c:** There is a Skip-Vote-Cert(v), and a view-v block B'' in the prefix of B' with an Echo-Cert(v, B''). By Lemma 5, this cannot happen (except with negligible probability).
- **Case 5-d:** There are both Skip-Echo-Cert(v) and Skip-Vote-Cert(v). By Lemma 5, this cannot happen (except with negligible probability).

□

5.1.2 Liveness Proof. In this section, we make the actual network delay δ explicit in the expressions. Hence, we will take it as a given that the correct parties' $T_{v,1}$ timers are at most δ apart at any point in time.

Lemma 6. *Let $n = 2t + 2p + c + 1$, and suppose there are at most f_b Byzantine, f_o omission and f_c crash faults such that $f_b \leq t$, $f_o \leq \min(p, c)$, and $f_o + f_c \leq c$. Then, all correct parties advance views within $6\Delta + 2\delta$ time, and observe a blockchain B s.t. $\text{isValid}(v, \mathsf{B}) = \text{True}$ upon entering view $v + 1$ for any view v.*

PROOF. We first prove that all correct parties advance views within $6\Delta + 2\delta$ time. Consider the following cases (which cover all possible events) for a correct party P and view v.

Case 1: A correct party P' observes a Vote-Cert(v, B) or a Fast-Cert(v, B), or a tuple of Skip-Vote-Cert(v) and Echo-Cert(v, B) before $T_{v,1} = 6\Delta$ in its view. Hence, P enters view $v + 1$ before $T_{v,1} = 6\Delta + 2\delta$ in its view.

Case 2: No correct party observes a Vote-Cert(v, B) or an Echo-Cert(v, B) before $T_{v,1} = 6\Delta$ in its view. In this case, all correct parties send a $(\text{Vote}, v, \perp, \text{False})$ and an $(\text{Echo}, v, \perp)$ at $T_{v,1} = 6\Delta$ in their views. Thus, P obtains both a Skip-Vote-Cert(v) and a Skip-Echo-Cert(v) before $T_{v,1} = 6\Delta + 2\delta$ in its view and enters view v.

Case 3: No correct party observes a Vote-Cert(v, B) or a Fast-Cert(v, B) before $T_{v,1} = 6\Delta$ in its view, but a correct party P' observes an Echo-Cert(v, B) before $T_{v,1} = 6\Delta$ in its view. In this case, all correct parties send a $(\text{Vote}, v, \perp, \text{False})$ at $T_{v,1} = 6\Delta$ in their views. Thus, P obtains both a Skip-Vote-Cert(v) and the Echo-Cert(v, B) before $T_{v,1} = 6\Delta + 2\delta$ in its view and enters view v.

Next, we prove by induction that all correct parties observe a blockchain B s.t. isValid(v, B) = True upon entering view $v+1$ for any view v. Suppose all correct parties observed a blockchain B (not necessarily the same blockchain) s.t. isValid$(v-1, \mathsf{B})$ = True upon entering view v. By definition, the induction hypthesis holds for $v = 1$. Now, a correct party P enters view $v+1$ upon observing either of (i) a Fast-Cert(v, B'), (ii) a Vote-Cert(v, B'), (iii) a Skip-Vote-Cert(v) and some Echo-Cert(v, B'), or (iv) a Skip-Vote-Cert(v) and a Skip-Echo-Cert(v). In cases (i)-(ii)-(iii), there must be a correct or crash-faulty party P' who sent an $(\mathsf{Echo}, v, \mathsf{B}')$ or a $(\mathsf{Vote}, v, \mathsf{B}',)$, and thus isValid$(v-1, \mathsf{B}'.\mathsf{parent})$ = True in the view of P'. As P' sends all certificates in its view that imply isValid$(v-1, \mathsf{B}'.\mathsf{parent})$ = True, before sending its echo or vote, these certificates are delivered to P, which observes a blockchain B′ s.t. isValid(v, B') = True upon entering view $v+1$. In case (iv), by the induction hypothesis, P observes a blockchain B s.t. isValid(v, B) = True upon entering view $v+1$. □

Theorem 9 (Liveness of Δ-Sync Simplex with Fast Path)**.** *Let $n = 2t + 2p + c + 1$, and suppose there are most f_b Byzantine, f_o omission, and f_c crash faults such that $f_b \leq t$, $f_o \leq p$, and $f_o + f_c \leq c$. Given the commit proof verification function in Definition 6, Δ-Sync Simplex with Fast Path satisfies liveness with latency 3δ if in addition $f_b + f_o + f_c \leq p$, and with latency $4\delta + \Delta$ otherwise.*

Proof. Consider a view v with a correct leader, denoted by Leader(v). By Lemma 6, Leader(v) eventually enters view v, and all other correct parties do so within δ time. By Lemma 6, Leader(v) observes a blockchain B s.t. isValid(v, B) = True upon entering view v and proposes its block $\mathsf{B}_{\mathsf{inp}}$ on top of that blockchain. Therefore, all correct parties receive the leader's proposal $(\mathsf{Propose}, v, \mathsf{B}_{\mathsf{inp}})$ and see that isValid$(v-1, \mathsf{B}.\mathsf{parent})$ = True within 2δ time after entering view v. Then, they all set isEarly$(\mathsf{B}_{\mathsf{inp}})$ = True, and send an $(\mathsf{Echo}, v, \mathsf{B}_{\mathsf{inp}})$ within 2δ time, all of which are received by all other correct parties within 3δ time into the view. They also send a $(\mathsf{Vote}, v, \mathsf{B}_{\mathsf{inp}}, \mathsf{True})$ within $2\delta + \Delta$ time, all of which are received by all other correct parties within $3\delta + \Delta$ time into the view. Now, if $f_b + f_o + f_c \leq p$ (in addition), $\mathsf{B}_{\mathsf{inp}}$ gets committed in the view of all correct parties within 3δ time (fast-commit). Otherwise, each correct party observes $t + p + 1$ $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$ messages without any conflicts within $3\delta + \Delta \leq 4\Delta$ time, and sends a $(\mathsf{Final}, v, \mathsf{B})$ for B. In this case, B gets committed in all views within $4\delta + \Delta$ time (slow-commit). □

Remark 6. Assuming synchronized start for a view, the worst-case view length becomes $6\Delta + \delta$, and the protocol satisfies liveness with latency 2δ if $f \leq p$, and $3\delta + \Delta$ if $f \leq t$ and $f_c \leq c$.

Remark 7. Note that a Vote-Cert(v, B) consisting of $t+p+1$ messages $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$ does not constitute a commit certificate. The reason is that there might exist views, where $p+1$ correct parties sent $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$, yet the remaining correct parties observed an equivocation and did not do so. In this case, the correct parties eventually create a Skip-Vote-Cert(v), and in a later view, commit a block conflicting with B. However, now the adversary can send t $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$ messages, making B committed as well.

Remark 8. It is possible to ensure safety and liveness for $n = 2t + 2p + c$ with the same resilience bounds as above for Byzantine, omission and crash faults. For this purpose, we must adjust the quorum sizes for Vote-Cert(v, B), Skip-Echo-Cert(v), Skip-Vote-Cert(v), and Slow-Cert(v, B) to be $t + p$ messages for their respective message types, and the quorum size for Fast-Cert(v, B) to be $2t + p + c$. We must also use an explicit 'adversary' detection subroutine for the leader. Formally, the leader's messages $(\mathsf{Echo}, v, \mathsf{B}')$ and $(\mathsf{Vote}, v, \mathsf{B}', *)$ will

not be considered as part of a quorum, if it is detected to have sent a proposal $(\text{Propose}, v, \text{B})$ for $\text{B} \neq \text{B}'$. For simplicity, we keep this optimization separate from the protocol and state it as an add-on to achieve the lower bound.

5.2 Proof of 2Δ-Sync Simplex with Responsive View Change

Definition 7 (Commit proof verification function for 2Δ-Sync Simplex)**.** Given a certificate $\text{cr} = (\text{B}, \pi)$, $\text{CV}(\text{B}, \pi)$ outputs true if and only if π includes a Fast-Cert(v, B) or a Slow-Cert(v, B), where $v = \text{B}.v$ is the view of block B.

Lemma 7. *All correct parties enter a view within $\delta \leq \Delta$ time of each other, where δ denotes the actual network delay. If a crash-faulty party enters a view, it does so within $\delta \leq \Delta$ time of all correct and other crash-faulty parties that enter the view.*

Proof. Before a correct or crash-faulty party proceeds to a new view, it broadcasts the set of messages that triggered the view change. Therefore, all correct parties receive these messages and advance their view within δ time. Similarly, all crash-faulty parties receive these messages and advance their view within δ time if they have not crashed by then. □

Hereafter, by Lemma 7, we will take it as a given that the correct and crash-faulty parties' $T_{v,1}$ timers are at most Δ apart at any point in time (as long as the party is not crashed).

5.2.1 *Safety Proof.*

Theorem 10 (Safety of 2Δ-Sync Simplex with Fast Path)**.** *Let $n = 2t + 2p + c + 1$, and suppose there are at most f_b Byzantine, f_o omission and f_c crash faults. Given the commit proof verification function in Definition 7, 2Δ-Sync Simplex with Fast Path satisfies safety, if $f_b \leq t$, $f_o \leq \min(p, c)$, and $f_o + f_c \leq c$.*

By the protocol rules, a non-Byzantine party never sends vote messages for conflicting blocks in the same view.

Lemma 8. *Except with negligible probability,*

- *there cannot exist both a* Fast-Cert(v, B) *and a* Fast-Cert(v, B') *for some v,* B, $\text{B}' \neq \text{B}$*,*
- *there cannot exist both a* Fast-Cert(v, B) *and a* Mid-Cert(v, B') *for some v,* B, $\text{B}' \neq \text{B}$*,*
- *there cannot exist both a* Fast-Cert(v, B) *and a* Vote-Cert(v, B') *for some v,* B, $\text{B}' \neq \text{B}$*,*
- *there cannot exist both a* Mid-Cert(v, B) *and a* Mid-Cert(v, B') *for some v,* B, $\text{B}' \neq \text{B}$*.*

Proof. A Fast-Cert(v, B) consists of $2t + p + c + 1$ $(\text{Vote}, v, \text{B}, \text{True})$, a Mid-Cert$(v, \text{B})$ consists of $\lfloor (3t + c)/2 \rfloor + p + 1$ $(\text{Vote}, v, \text{B}, \text{True})$, and a Vote-Cert$(v, \text{B})$ consists of $t + p + 1$ $(\text{Vote}, v, \text{B}, \text{True})$, whereas $n = 2t + 2p + c + 1$. By quorum intersection, there must be conflicting votes by a non-Byzantine party in each case, which happens with negligible probability. □

Lemma 9. *Suppose a correct or crash-faulty party P sends a* $(\text{Final}, v, \text{B})$ *upon its timer $T_{v,2}$ reaching 2Δ. Then, all correct and crash-faulty parties receive a* Vote-Cert(v, B) *before $T_{v,1} = 5\Delta$ in their views (if they have not crashed yet), and none of them sends a* $(\text{Vote}, v, \text{B}', *)$ *for some* $\text{B}' \neq \text{B}$*, except with negligible probability.*

Proof. Let t denote the absolute time when P voted for B and started its timer $T_{v,2}$. Since P sends a $(\text{Final}, v, \text{B})$ upon its timer $T_{v,2}$ reaching 2Δ, it could not have been crashed by time t. For contradiction, suppose a correct or crash-faulty party P' sends a $(\text{Vote}, v, \text{B}', *)$ for some $\text{B}' \neq \text{B}$ at some absolute time t'. This implies that P' must have observed a view v proposal for the first time at time t', which must be before $T_{v,1} = 4\Delta$ in its view, and that P' could not have crashed at the time it relays this proposal for B', which will be delivered to P by

time $t' + \Delta$. Now, for P to send $(\mathsf{Final}, v, \mathsf{B})$ at time $t + 2\Delta$ without detecting equivocation, P must not have received the proposal for B' relayed by P', which implies $t' + \Delta > t + 2\Delta$, *i.e.*, $t' > t + \Delta$. Moreover, since P is not crashed by time t, its vote will be delivered to P' by time $t + \Delta$. Hence, for the first proposal observed by P' to be for the block B', it must also be that $t' < t + \Delta$. Thus, we have reached a contradiction: except with negligible probability, no correct or crash-faulty party sends a $(\mathsf{Vote}, v, \mathsf{B}', *)$ for some $\mathsf{B}' \neq \mathsf{B}$. The reasoning above also implies that the first (valid) view v proposal observed by any correct party cannot be for any block $\mathsf{B}' \neq \mathsf{B}$.

Finally, note that the time t must be before $T_{v,1}$ reaches 2Δ in P's view, which is before $T_{v,1}$ reaches 3Δ in any correct party's view. Therefore, all correct parties observe P's $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$ by time $t + \Delta$, *i.e.*, before $T_{v,1} = 4\Delta$ in their views, and this is the first (valid) proposal in their views. Hence, all correct parties send $(\mathsf{Vote}, v, \mathsf{B}, *)$ for B by time $t + \Delta$. These votes reach all correct and crash-faulty parties (that have not crashed by then) by time $t + 2\Delta$, *i.e.*, all of them observe a $\mathsf{Vote\text{-}Cert}(v, \mathsf{B})$ before $T_{v,1} = 5\Delta$ in their views. □

Lemma 10. *There cannot exist both a* $\mathsf{Slow\text{-}Cert}(v, \mathsf{B})$*, and one of* $\mathsf{Mid\text{-}Cert}(v, \mathsf{B}')$*,* $\mathsf{Fast\text{-}Cert}(v, \mathsf{B}')$*, or* $\mathsf{Slow\text{-}Cert}(v, \mathsf{B})$ *for some* v*,* B*,* $\mathsf{B}' \neq \mathsf{B}$ *(except with negligible probability).*

Proof. A $\mathsf{Slow\text{-}Cert}(v, \mathsf{B})$ implies that least one correct or crash-faulty party P sent $(\mathsf{Final}, v, \mathsf{B})$, either (1) after observing a $\mathsf{Mid\text{-}Cert}(v, \mathsf{B})$ before $T_{v,1} = 3\Delta$, or (2) at time $T_{v,2} = 2\Delta$, after voting for B and without detecting equivocation.

A $\mathsf{Slow\text{-}Cert}(v, \mathsf{B}')$ implies that least one correct or crash-faulty party P' sent $(\mathsf{Final}, v, \mathsf{B}')$, either (a) after observing a $\mathsf{Mid\text{-}Cert}(v, \mathsf{B}')$ before $T_{v,1} = 3\Delta$, or (b) at time $T_{v,2} = 2\Delta$, after voting for B' and without detecting equivocation.

Now, suppose there is a $\mathsf{Mid\text{-}Cert}(v, \mathsf{B}')$ or a $\mathsf{Fast\text{-}Cert}(v, \mathsf{B}')$. Then, case (1) cannot happen by Lemma 8, except with negligible probability. Moreover, both $\mathsf{Mid\text{-}Cert}(v, \mathsf{B}')$ and $\mathsf{Fast\text{-}Cert}(v, \mathsf{B}')$ imply that at least one correct or crash-faulty party P' must have sent $(\mathsf{Vote}, v, \mathsf{B}', \mathsf{True})$. Thus, case (2) cannot happen by Lemma 9, except with negligible probability.

Finally, suppose there is a $\mathsf{Slow\text{-}Cert}(v, \mathsf{B}')$. Then, case (1-a) cannot happen by Lemma 8, and the cases (1-b), (2-a), and (2-b) cannot happen by Lemma 9, except with negligible probability. □

Lemma 11. *There cannot exist both a* $\mathsf{Fast\text{-}Cert}(v, \mathsf{B})$ *and a* $\mathsf{Skip\text{-}Vote\text{-}Cert}(v)$ *for some* v*,* B *(except with negligible probability).*

Proof. Suppose there is a $\mathsf{Fast\text{-}Cert}(v, \mathsf{B})$, which implies that at least $t + p + 1$ correct parties sent a $(\mathsf{Vote}, v, \mathsf{B}, \mathsf{True})$ before $T_{v,1} = 2\Delta$ in their views. Therefore, all correct and crash-faulty parties (that have not crashed by then) obtain a $\mathsf{Vote\text{-}Cert}(v, \mathsf{B})$ before $T_{v,1} = 4\Delta$ in their views, and none of them sends a $(\mathsf{Vote}, v, \perp, \mathsf{False})$. Hence, $\mathsf{Skip\text{-}Vote\text{-}Cert}(v)$ cannot exist except with negligible probability. □

Proof of Theorem 10. We next consider all possible cases in which two conflicting blocks B and B' with views v and $v' \geq v$ acquire a valid commit certificate.

Case 1: There exist a $\mathsf{Fast\text{-}Cert}(v, \mathsf{B})$ and a $\mathsf{Fast\text{-}Cert}(v, \mathsf{B}')$. By Lemma 8, this cannot happen except (with negligible probability).

Case 2: There exist a $\mathsf{Fast\text{-}Cert}(v, \mathsf{B})$ and $\mathsf{Slow\text{-}Cert}(v, \mathsf{B}')$. By Lemma 10, this cannot happen except (with negligible probability).

Case 3: There exist a $\mathsf{Slow\text{-}Cert}(v, \mathsf{B})$ and $\mathsf{Slow\text{-}Cert}(v, \mathsf{B}')$. By Lemma 10, this cannot happen except (with negligible probability).

Case 4: $v < v'$, and there exist a Fast-Cert(v, B). Since there is a commit proof for B', at least one correct or crash-faulty party must have voted for B' in view v', *i.e.*, isValid$(v', \mathsf{B}'.\text{parent}) =$ True in the view of at least one correct or crash-faulty party. Consider the following sub-cases:

- **Case 4-a:** There is a view-v block B'' in the prefix of B' with a Mid-Cert(v, B''). By Lemma 8, this cannot happen except with negligible probability.
- **Case 4-b:** There is a Skip-Mid-Cert(v), together with a view-v block B'' in the prefix of B' with a Vote-Cert(v, B''). By Lemma 8, this cannot happen except with negligible probability.
- **Case 4-c:** There are both Skip-Mid-Cert(v) and Skip-Vote-Cert(v). By Lemma 11, this cannot happen except with negligible probability.

Case 5: $v < v'$, and there exist a Slow-Cert(v, B). Since there is a commit proof for B', at least one correct or crash-faulty party must have voted for B' in view v', *i.e.*, isValid$(v', \mathsf{B}'.\text{parent}) =$ True in the view of at least one correct or crash-faulty party. Recall that a Slow-Cert(v, B) implies at least one correct or crash-faulty party P sent $(\text{Final}, v, \mathsf{B})$, either (i) after observing a Mid-Cert(v, B) before $T_{v,1} = 3\Delta$ in its view, or (ii) at time $T_{v,2} = 2\Delta$, after voting for B and without detecting equivocation. Since P is not crashed before sending $(\text{Final}, v, \mathsf{B})$, it relays the Mid-Cert$(v, \mathsf{B})$ in case (i). Next, consider the following sub-cases:

- **Case 5-a:** There is a view-v block B'' in the prefix of B' with a Mid-Cert(v, B''). By Lemma 10, this cannot happen except with negligible probability.
- **Case 5-b:** There is a Skip-Mid-Cert(v), and a view-v block B'' in the prefix of B' with a Vote-Cert(v, B''). Now, if case (i) happens, all correct and crash-faulty parties observe Mid-Cert(v, B) before $T_{v,1} = 5\Delta$ in their views. Thus, none of them sends a $(\text{Mid-Skip}, v)$, and there cannot exist a Skip-Mid-Cert(v), except with negligible probability. If case (ii) happens, by Lemma 9, no correct or crash-faulty party would have sent a vote $(\text{Vote}, v, \mathsf{B}'', *)$ for $\mathsf{B}'' \neq \mathsf{B}$, and thus there cannot be a Vote-Cert(v, B''), except with negligible probability.
- **Case 5-c:** There are both Skip-Vote-Cert(v) and Skip-Mid-Cert(v). If case (i) happens, by the reasoning above, no correct or crash-faulty party sends a $(\text{Mid-Skip}, v)$, and there cannot exist a Skip-Mid-Cert(v), except with negligible probability. If case (ii) happens, again by Lemma 9, all correct and crash-faulty parties observe a Vote-Cert(v, B) before $T_{v,1} = 5\Delta$ in their views, implying that none of them sends a $(\text{Vote}, v, \perp, \text{False})$. Thus, there cannot be a Skip-Vote-Cert(v), except with negligible probability.

□

5.2.2 Liveness Proof. In this section, we make the actual network delay δ explicit in the expressions. Hence, we will take it as a given that the correct parties' $T_{v,1}$ timers are at most δ apart at any point in time.

Lemma 12. *Let $n = 2t + 2p + c + 1$, and suppose there are at most f_b Byzantine, f_o omission and f_c crash faults such that $f_b \leq t$, $f_o \leq \min(p, c)$, and $f_o + f_c \leq c$. Then, all correct parties advance views within $5\Delta + 2\delta$ time, and observe a blockchain B s.t. isValid$(v, \mathsf{B}) =$ True upon entering view $v + 1$ for any view v.*

PROOF. We first prove that all correct parties advance views within $5\Delta + 2\delta$ time. Consider the following cases (which cover all possible events) for a correct party P and view v.

Case 1: A correct party P' observes a Mid-Cert(v, B), or a Fast-Cert(v, B), or a tuple of Vote-Cert(v, B) and Skip-Mid-Cert(v) before $T_{v,1} = 5\Delta$ in its view. Then, P enters view $v + 1$ before $T_{v,1} = 5\Delta + 2\delta$ in its view.

Case 2: No correct party P observes a Vote-Cert(v, B) before $T_{v,1} = 5\Delta$ in its view. In this case, all correct parties send a $(\mathsf{Vote}, v, \perp, \mathsf{False})$ and a $(\mathsf{Mid\text{-}Skip}, v)$ at $T_{v,1} = 5\Delta$ in their views. Thus, P obtains both a Skip-Vote-Cert(v) and a Skip-Mid-Cert(v) before $T_{v,1} = 5\Delta + 2\delta$ in its view and enters view v.

Case 3: No correct party P observes any of Mid-Cert(v, B), Fast-Cert(v, B), or Skip-Mid-Cert(v), but a correct party P' observes a Vote-Cert(v, B), before $T_{v,1} = 5\Delta$ in the respective views. In this case, all correct parties send a $(\mathsf{Mid\text{-}Skip}, v)$ at $T_{v,1} = 5\Delta$ in their views. Thus, P obtains both a Vote-Cert$(\mathsf{B},)$ and a Skip-Mid-Cert(v) before $T_{v,1} = 5\Delta + 2\delta$ in its view and enters view v.

Next, we prove by induction that all correct parties observe a blockchain B s.t. isValid$(v, \mathsf{B}) =$ True upon entering view $v+1$ for any view v. Suppose all correct parties observed a blockchain B (not necessarily the same blockchain) s.t. isValid$(v-1, \mathsf{B}) =$ True upon entering view v. By definition, the induction hypthesis holds for $v = 1$. Now, a correct party P enters view $v + 1$ upon observing either of (i) a Fast-Cert(v, B'), (ii) a Mid-Cert(v, B'), (iii) a Skip-Mid-Cert(v) and some Vote-Cert(v, B'), or (iv) a Skip-Vote-Cert(v) and Skip-Mid-Cert(v). In cases (i)-(ii)-(iii), there must be a correct or crash-faulty party P' who sent a $(\mathsf{Vote}, v, \mathsf{B}',)$, and isValid$(v-1, \mathsf{B}'.\mathsf{parent}) =$ True in the view of P'. As P' sends all certificates in its view that imply isValid$(v-1, \mathsf{B}'.\mathsf{parent}) =$ True, along with its vote, P observes a blockchain B' s.t. isValid$(v, \mathsf{B}') =$ True upon entering view $v + 1$. In case (iv), by the induction hypothesis, P observes a blockchain B s.t. isValid$(v, \mathsf{B}) =$ True upon entering view $v + 1$. □

Theorem 11 (Liveness of 2Δ-Sync Simplex with Fast Path)**.** *Let $n = 2t + 2p + c + 1$, and suppose there are at most f_b Byzantine, f_o omission and f_c crash faults such that $f_b \le t$, $f_o \le p$, and $f_o + f_c \le c$. Given the commit proof verification function in Definition 7, 2Δ-Sync Simplex with Fast Path satisfies liveness with latency 3δ if in addition, $f_b + f_o + f_c \le p$ (fast path), with latency 4δ if in addition $f_b + f_o + f_c \le \lfloor (t + c)/2 \rfloor + p$, and with latency $3\delta + 2\Delta$ otherwise (slow path).*

Proof. Consider a view v with a correct leader, denoted by Leader(v). By Lemma 12, Leader(v) eventually enters view v, and all other correct parties do so within δ time. By Lemma 12, Leader(v) observes a blockchain B s.t. isValid$(v, \mathsf{B}) =$ True upon entering view v and proposes its block $\mathsf{B}_{\mathsf{inp}}$ on top of that blockchain. Therefore, all correct parties receive the leader's proposal $(\mathsf{Propose}, v, \mathsf{B}_{\mathsf{inp}})$ and see that isValid$(v-1, \mathsf{B}.\mathsf{parent}) =$ True within 2δ time after entering view v. Then, they all send a $(\mathsf{Vote}, v, \mathsf{B}_{\mathsf{inp}}, \mathsf{True})$ within 2δ time, which are received by all other correct parties within 3δ time into the view.

Now, if $f_b + f_o + f_c \le p$ (in addition), $\mathsf{B}_{\mathsf{inp}}$ gets committed in all views within 3δ time (fast-commit). If $f_b + f_o + f_c \le \lfloor (t + c)/2 \rfloor + p$ (in addition), then all correct parties obtain a Mid-Cert(v, B) within $3\delta \le 3\Delta$ time and send a $(\mathsf{Final}, v, \mathsf{B})$ for B. In this case, B gets committed in all views within 4δ time (mid-commit). Otherwise, as there cannot be any equivocation in this view, all correct parties send a $(\mathsf{Final}, v, \mathsf{B})$ for B once their $T_{v,2}$ timer expires, by time $2\delta + 2\Delta$. In this case, B gets committed in all views within $3\delta + 2\Delta$ time (slow-commit). □

Remark 9. Assuming synchronized start for a view, the worst-case view length becomes $5\Delta + \delta$, and the protocol satisfies liveness with latency 2δ if there are at most p Byzantine parties (fast path), latency 3δ if there are at most $\lfloor (t + c)/2 \rfloor + p$ Byzantine parties (mid path), and latency $2\delta + 2\Delta$ when there are at most t Byzantine parties and c crash faults (slow path).

Remark 10. It is possible to ensure safety and liveness for $n = 2t + 2p + c$ with the same resilience bounds as above for Byzantine, omission and crash faults. For this purpose, we must adjust the quorum sizes for Vote-Cert(v, B), Skip-Vote-Cert(v), Skip-Mid-Cert(v), and Slow-Cert(v, B) to be $t + p$ messages for their respective message types, and the quorum size for Fast-Cert(v, B) to be $2t + p + c$, and for Mid-Cert(v, B) to be $\lfloor (3t + c)/2 \rfloor + p$. We must also again use an explicit 'adversary' detection subroutine for the leader. More specifically, the leader's $(\mathsf{Vote}, v, \mathsf{B}', *)$ will not be considered as part of a quorum, if it is detected to have sent a proposal $(\mathsf{Propose}, v, \mathsf{B})$ for $\mathsf{B} \neq \mathsf{B}'$. For simplicity, we keep this optimization separate from the protocol and state it as an add-on to achieve the lower bound.

5.3 TenderSync with 4Δ Views, 3Δ Commit, and 2Δ Fast Commit

Lemma 13. *If there is a fast commit certificate for* B *in view* v*, then (i)* $\mathsf{Lock}(v, \mathsf{B}')$ *or* $\mathsf{WeakLock}(v, \mathsf{B}')$ *for* $\mathsf{B} \neq \mathsf{B}'$ *does not exist, and (ii) all correct parties and crash faulty parties that have not crashed yet will set* $\mathsf{Lock}(v, \mathsf{B})$ *by the start of round 4 of view* v*.*

Proof. (i) Since there is a fast commit for B in view v, there are $n - p = 2t + y + p + 1$ votes for it. Thus, at least $t + p + 1$ correct parties must have voted for it. Any block $\mathsf{B}' \neq \mathsf{B}$ can be voted upon by only $t + p$ parties. This includes potentially t Byzantine parties and the p parties who are not a part of the fast commit certificate. In particular, the potentially crashed or omission parties in the fast commit certificate will not equivocate. Thus, no party sends a weak lock (or lock) on B', and a $\mathsf{WeakLock}(v, \mathsf{B}')$ does not exist.

(ii) Moreover, since at least $t + p + 1$ correct parties voted for B and there does not exist $t + p + 1$ votes for B', all correct parties will send a weak lock for (v, B) (they could also send a lock which implies a weak lock). Thus, at the end of the view, either parties (correct or crash faulty that have not crashed yet) receive a lock on (v, B), or if not, they receive weak lock on (v, B) from all $(n - t - y)$ correct parties. In either case, they set their $\mathsf{Lock}_{\mathsf{high}}$ to (v, B). □

Lemma 14. *If there is a commit certificate for* B *in view* v*, then (i) then* $\mathsf{Lock}(v, \mathsf{B}')$ *or* $\mathsf{WeakLock}(v, \mathsf{B}')$ *for* $\mathsf{B} \neq \mathsf{B}'$ *does not exist, and (ii) all correct parties and crash faulty parties that have not crashed yet will set* $\mathsf{Lock}(v, \mathsf{B})$ *by the start of round 4 of view* v*.*

Proof. (i) We will prove by contradiction. Suppose an adversary holds a weak lock for (v, B'). This implies at least $t + p + 1$ votes were received by the adversary. Out of these, t parties could be Byzantine and $\leq p$ parties could be omission faulty (since $o = \min(y, p)$). Thus, there is at least one sender that is either correct or crash faulty. Even if this party is crash faulty, since it has sent a vote, it must have relayed the proposal to all correct and crash faulty parties. Thus, due to equivocation detection, no correct or crash faulty parties will send a final for (v, B). Consequently, the number of final messages received by any party for (v, B) at the start of round 4 is at most $o + t \leq t + p < n - t - y$. Recall that a commit certificate requires $n - t - y$ $\mathsf{Final}(v, \mathsf{B})$ messages. Thus, no certificate for B from view v exists, a contradiction.

(ii) Forming a commit certificate implies receiving $n - t - y = t + 2p + 1$ distinct Final messages. Out of these up to t of the finals could be sent by Byzantine parties, $\leq p$ final could be sent by omission parties (since $o = \min(y, p)$), the remaining $p + 1$ are sent by crash faulty that have not yet crashed or correct parties. Since Final messages are sent only after sending the Lock message, all correct parties and crash faulty parties that have not crashed yet must have received this lock at the start of round 4. Moreover, due to part (i) no party could obtain a lock certificte or weak lock certificate for any equivocating value, and thus they would set their $\mathsf{Lock}_{\mathsf{high}}$ to $\mathsf{Lock}(v, \mathsf{B})$. □

Theorem 12 (Agreement)**.** *Suppose there are commit certificates for* B *and* B′*. Then either* B *extends* B′ *or vice versa.*

PROOF. First, observe that there are cannot be commit certificates for both B and B′ for B ≠ B′ in the same view v. By Lemmas 13 and 14 if there is a commit certificate for B, no party can lock on B′ and thus commit B′.

WLOG, suppose B is committed in a lower view v. By Lemmas 13 and 14, all correct parties and crash faulty parties that have not crashed yet will set their lock to (v, B), and there does not exist an equal or higher lock for any value ≠ B at the end of the view. Thus, in view $v + 1$, a correct party or a crash faulty party that has not crashed yet will not vote for any value B″ that does not extend B accompanied by a key from a lower view, and thus a lock or weak lock for any other value cannot be formed. Inductively, in any view $> v$, no value B″ that does not extend B will be voted upon by any correct party or crash faulty party that has not crashed yet. Thus, B′ extends B. The vice versa case holds when B′ was committed in a lower view. □

Theorem 13 (Liveness)**.** *If the leader of view v is correct, then a commit certificate is formed by round 4 of view v.*

PROOF. Observe that a correct leader obtains the highest lock from all correct parties and crash faulty parties that have not crashed yet. Thus, it will create a proposal that extending the key corresponding to the locked block. Thus, all correct parties and crash faulty parties that have not crashed yet will vote on the leader's proposal. Since, there will be no equivocation, all correct parties will be locked on this block and send the lock in round 3. Thus, a commit certificate will be formed by round 4. □

5.4 Proof of SyncMint with 3Δ Views, 4Δ Commit, and 2Δ Fast Commit

Lemma 15. *If there is a fast commit certificate for* B *in view v, then (i) no equivocating locks, weak locks, keys, or weak keys from view v exist, (ii) all correct parties and crash faulty parties that have not crashed yet will set lock on (v, B) by the start of round 2 of view $v + 1$.*

PROOF. If there is a fast commit certificate for B in view v, then $n - p = 2t + y + p + 1$ parties voted for it, and thus, at least $t + p + y + 1$ non-equivocating parties (a combination of correct, omission, and crashed parties that have not crashed yet) voted for B. Thus, any block B′ will not receive more than $t + p$ votes (t of these are Byzantine and p parties that did not vote for x). Hence, there will be no weak keys (or keys or weak locks or locks) corresponding to an equivocating value.

Moreover, since $t + p + 1$ correct parties voted for B, all correct parties will send a $\mathsf{WeakKey}(v, \mathsf{B})$ (or $\mathsf{Key}(v, \mathsf{B})$). This ensures that all correct parties send a lock or weak lock for (v, B) in round 1 of view $v + 1$ due to which all correct parties will be locked on (v, B) in round 2 of view $v + 1$. □

Lemma 16. *If there is a regular commit certificate for* B *from view v, then (i) no equivocating locks, weak locks, keys, or weak keys from view v exist, (ii) all correct parties and crash faulty parties that have not crashed yet will set lock on (v, B) by the start of round 2 of view $v + 1$.*

PROOF. Suppose a weak key certificate exists for B′ in view v. Then, $t + p + 1$ parties voted for B′. t of these parties could be Byzantine, and up to p parties could be omission faulty (since $o = \min(p, y)$). Thus, at least one of these parties is either correct or a crash faulty party that has not crashed yet. Since every party first relays the received proposal before voting, every correct or crash faulty party must have received the relayed proposal

containing B′. Thus, due to the no equivocation check, no correct or crash faulty party that has not crashed yet will send Key(v, B). Thus, only $o = \min(y, p)$ omission faulty parties and t Byzantine parties can send Key(v, B). Since $t + o < t + p + 1$, no party can send a Final(v, B), and thus a regular commit certificate for B cannot be formed in view v, a contradiction.

We know that B has a commit certificate in view v. Thus, there exist $n - y - t$ distinct finals Final(v, B). Thus, $n - y - t = t + 2p + 1$ parties must have sent this Final. Out of these t parties are Byzantine, p parties are omission faulty, and thus, at least $p + 1$ parties must be correct or crash faulty party that has not crashed yet and has sent the lock. Since an equivocating weak key or key from view v does not exist, all correct or crash faulty parties that have not crashed yet will be locked on (v, B) at the start of round 2 of view $v + 1$.

□

Theorem 14 (Agreement)**.** *Suppose there are commit certificates for* B *and* B′*, then either* B *extends* B′ *or vice versa.*

Proof. Observe that there cannot be commit certificates for both B and B′ for B ≠ B′ in the same view v. By Lemmas 15 and 16, if there is a commit certificate for B, no party can lock on B′ and thus commit B′.

WLOG, suppose B is committed in a lower view v. By Lemmas 15 and 16, all correct and crash faulty parties that have not crashed yet will set their lock to (v, B) by the start of round 2 of view $v + 1$. There does not exist a higher lock or higher weak lock or higher key or higher weak key for any block not extending B. Thus, in view $v + 1$, a correct party or crash faulty party that has not crashed yet will not vote for any value B″ that does not extend B accompanied by a key from a lower view. Inductively, in any view $> v$, no value B″ that does not extend B will be voted upon by any correct party or crash faulty party that has not crashed yet. Thus, B extends B′. The vice versa case holds when B′ was committed in a lower view. □

Theorem 15 (Liveness)**.** *If the leader of view v is correct, then a commit certificate is formed by round 2 of view $v + 1$.*

Proof. Observe that a correct party can receive a lock from view $v - 1$ through two paths, based on receiving a (i) lock or (ii) several weak locks.

If a correct party P receives a lock, it must consist of at least $t + p + 1$ keys. At least one of these keys will be sent by a correct party or a crashed party that has not crashed yet. Thus, this key arrives at the correct leader. Thus, the correct leader will propose a value with a key that is at least as high as P's lock.

If a correct party P locks because of receiving several weak locks, then each weak lock contains at least $t + 2p + 1$ keys or weak keys. Again, observe that at least one of the senders of these keys is correct or a crashed party that has not crashed yet due to which the correct leader would obtain this key; which is at least as high as P's lock. □

References

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected o (1) rounds, expected communication, and optimal resilience. In *International Conference on Financial Cryptography and Data Security*, pages 320–334. Springer, 2019.

[2] Ittai Abraham, Danny Dolev, Alon Kagan, and Gilad Stern. Authenticated consensus in synchronous systems with mixed faults. Cryptology ePrint Archive, Paper 2022/805, 2022.

[3] Ittai Abraham, Danny Dolev, Alon Kagan, and Gilad Stern. Brief announcement: Authenticated consensus in synchronous systems with mixed faults. In *36th International Symposium on Distributed Computing*

(DISC 2022), Leibniz International Proceedings in Informatics (LIPIcs), pages 38:1–38:3. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

[4] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync HotStuff: Simple and practical synchronous state machine replication. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 106–118. IEEE, 2020.

[5] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Byzantine agreement, broadcast and state machine replication with near-optimal good-case latency. *CoRR*, abs/2003.13155, 2020.

[6] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: a complete categorization. In *PODC*, pages 331–341. ACM, 2021.

[7] Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. Optimal good-case latency for rotating leader synchronous bft. In *Proceedings of the 25th International Conference on Principles of Distributed Systems (OPODIS 2021)*, volume 217 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:19, 2021.

[8] Kaya Alpturer, Kushal Babel, and Aditya Saraf. Timing games in responsive consensus protocols. *CoRR*, abs/2510.25144, 2025.

[9] Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. Mysticeti: Reaching the latency limits with uncertified dags. In *NDSS*. The Internet Society, 2025.

[10] Jeb Bearer, Benedikt Bünz, Philippe Camacho, Binyi Chen, Ellie Davidson, Ben Fisch, Brendon Fish, Gus Gutoski, Fernando Krell, Chengyu Lin, Dahlia Malkhi, Kartik Nayak, Keyao Shen, Alex Luoyuan Xiong, and Nathan Yospe. The espresso sequencing network: Hotshot consensus, tiramisu data-availability, and builder-exchange. *IACR Cryptol. ePrint Arch.*, page 1189, 2024.

[11] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018.

[12] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X. Zhang. Combining GHOST and casper. *CoRR*, abs/2003.03052, 2020.

[13] Benjamin Y. Chan and Rafael Pass. Simplex consensus: A simple and fast consensus protocol. In Guy N. Rothblum and Hoeteck Wee, editors, *Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part IV*, volume 14372 of *Lecture Notes in Computer Science*, pages 452–479. Springer, 2023.

[14] T-H. Hubert Chan, Rafael Pass, and Elaine Shi. PiLi: An extremely simple synchronous blockchain. Cryptology ePrint Archive, Paper 2018/980, 2018.

[15] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[16] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[17] Espresso systems. https://www.espressosys.com/, n.d.

[18] TenderSync (citation details tbd), n.d. Placeholder entry to resolve `\cite{FCpaper}`; please replace with the correct bibliographic metadata.

[19] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.

[20] Flow. https://flow.com/, n.d.

[21] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *Financial Cryptography*, volume 13411 of *Lecture Notes in Computer Science*, pages 296–315. Springer, 2022.

[22] Jonathan Katz and Chiu-Yuen Koo. Expected-constant-round protocols for byzantine agreement. *Journal of Computer and System Sciences*, 75(2):91–112, 2009.

[23] Quentin Kniep, Kobi Sliwinski, and Roger Wattenhofer. Alpenglow: A new consensus for solana. Technical report, Anza Technology, 2025. White paper and SIMD-0326 proposal.

[24] Vladimir Kostic, Lili Alvisi, and Michael Dahlin. Xft: Practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, 2016.

[25] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[26] Layerzero. https://layerzero.network/, n.d.

[27] Silvio Micali and Vinod Vaikuntanathan. Optimal and player-replaceable consensus with an honest majority. Technical report, MIT, 2017.

[28] Atsuki Momose, Jason Paul Cruz, and Yuichi Kaji. Hybrid-bft: Optimistically responsive synchronous consensus with optimal latency or resilience. Technical Report 2020/406, IACR Cryptology ePrint Archive, 2020.

[29] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.
[30] Kartik Nayak, Ittai Abraham, and Ling Ren. Authenticated synchronous BFT. https://decentralizedthoughts.github.io/2019-11-11-authenticated-synchronous-bft/, 2019.
[31] Burak Öz, Benjamin Kraner, Nicolò Vallarano, Bingle Stegmann Kruger, Florian Matthes, and Claudio Juan Tessone. Time moves faster when there is nothing you anticipate: The role of time in MEV rewards. In *DeFi@CCS*, pages 1–8. ACM, 2023.
[32] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, volume 10821 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2018.
[33] Michael O Rabin. Randomized byzantine generals. In *24th annual symposium on foundations of computer science (sfcs 1983)*, pages 403–409. IEEE, 1983.
[34] Caspar Schwarz-Schilling, Fahad Saleh, Thomas Thiery, Jennifer Pan, Nihar Shah, and Barnabé Monnot. Time is money: Strategic timing games in proof-of-stake protocols. In *AFT*, volume 282 of *LIPIcs*, pages 30:1–30:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
[35] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. On the optimality of optimistic responsiveness. In *CCS*, pages 839–857. ACM, 2020.
[36] The DFINITY Team. The DFINITY consensus system. https://dfinity.org/pdf-viewer/pdfs/viewer?file=../library/dfinity-consensus.pdf, 2018.
[37] Andrei Tonkikh, Balaji Arun, Zhuolun Xiang, Zekun Li, and Alexander Spiegelman. Raptr: Prefix consensus for robust high-performance BFT. *CoRR*, abs/2504.18649, 2025.