Hydrangea++: Enhancing Hydrangea with Optimistic Proposals

Nibesh Shrestha[1] and Aniket Kate[2]

[1]Supra Research, n.shrestha@supra.com
[2]Supra Research/Purdue University, aniket@purdue.edu

Abstract

Achieving both low latency and strong fault tolerance remains a fundamental challenge in distributed consensus. Recent efforts toward low-latency consensus have focused on optimistic protocols that commit blocks in two rounds when the number of faults remains below a defined threshold. The state-of-the-art protocol Hydrangea demonstrated that such optimistic two-round commits are possible while also improving fault resilience. However, existing approaches (including Hydrangea) primarily optimize commit latency while retaining a 2δ proposal delay, where δ represents a single network delay. This delay increases queuing time for client transactions and ultimately worsens end-to-end finality.

In this work, we introduce Hydrangea++, which addresses this limitation by enabling proposals every network delay—cutting proposal latency to the theoretical minimum δ. In a system of $n = 3f + 2c + m + 1$ parties (where m is a tunable parameter), under synchrony and an honest leader, Hydrangea++ additionally achieves two-round commits when at most $p = \lfloor \frac{c+m}{2} \rfloor$ parties are faulty, and guarantees three-round commits even with up to f Byzantine and c crash faults. Our geo-distributed evaluation shows that Hydrangea++ delivers up to 11% lower end-to-end latency and nearly 2× higher throughput compared to state-of-the-art protocols.

1 Introduction

Consensus lies at the core of state machine replication (SMR), which is a foundational abstraction for building fault-tolerant distributed systems, replicated databases, and blockchains. A consensus protocol must ensure safety (no two honest parties commit conflicting blocks) and liveness (honest parties eventually commit) in the presence of both network asynchrony and process faults. In this work, we are interested in consensus under the partially synchronous model [9], where the network delay is unbounded until some unknown Global Stabilization Time (GST) after which message delays are bounded by a known constant Δ.

The latency–resilience trade-off. A major goal consensus protocol design is to minimize commit latency, defined as the number of communication rounds required for honest parties to reach a decision in the good-case case, that is, after GST has passed and the designated leader is honest. It is known that protocols tolerating the optimal number of Byzantine faults ($f < n/3$) in this model require at least three rounds to commit in the good case [3]. A long line of optimistic consensus protocols [1, 12, 15] can achieve commits in fewer than three rounds, but only by sacrificing resilience. Specifically, these protocols operate in a system of $n = 3f + 2p + 1$ parties, where f denotes the maximum number of Byzantine faults and p is a tunable parameter. They achieve optimistic two-round commits when the number of Byzantine faults does not exceed $p \leq f$, and guarantee three-round commits when up to f Byzantine faults are present.

Hydrangea: closing the gap. Hydrangea [18] recently closed this gap by introducing the first protocol that simultaneously achieves low-latency optimistic commits and strong fault tolerance. In a system of $n = 3f + 2c + m + 1$ parties, where $m \geq 0$ is a tunable parameter, Hydrangea achieves optimistic two-round commits under synchrony when at most $p = \lfloor \frac{c+m}{2} \rfloor$ parties are faulty (Byzantine or crash), and three-round commits in the good-case even with f Byzantine faults and c crash faults. Hydrangea thus tolerates strictly more faults than prior optimistic protocols while retaining fast-path commit latency. By combining high

fault resilience with low-latency finality in favorable settings, Hydrangea resolves a long-standing tension between latency and resilience in partially synchronous consensus.

The remaining bottleneck: proposal delay. Most modern consensus protocols operate in a rotating-leader model, where each view assigns a unique leader responsible for making a single proposal. This design helps ensure fairness, censorship resistance, and balanced workload distribution across the system. However, in this setting, most protocols (including Hydrangea) still incur a minimum block period of 2δ between proposals from consecutive honest leaders, where δ denotes network delay. The reason is that the leader of view $v+1$ must first obtain a block certificate from view v before issuing its own proposal, even in fully synchronous conditions.

This conservative timing creates a performance bottleneck: client transactions experience an extra δ delay before they can be included in the next proposal, increasing end-to-end latency. At the same time, the strict 2δ spacing restricts pipelining and reduces block production rates in geo-distributed environments. In short, while Hydrangea optimizes commit latency, it does not optimize proposal latency (the time between proposals of consecutive honest leaders) leaving noticeable performance improvements unexploited when the network is operating well.

Our contribution: Hydrangea++

In this work, we introduce Hydrangea++, a new variant of Hydrangea that eliminates proposal delays by enabling leaders to make optimistic proposals, following ideas from Moonshot [8]. Hydrangea++ contributes in three key ways:

1. **Optimistic proposals.** We introduce a mechanism that enables the leader of view $v+1$ to optimistically propose a block as soon as it receives the proposal for view v, rather than waiting for that block to be certified. This reduces the proposal delay from 2δ to the theoretical minimum of δ, effectively matching the underlying network delay.

2. **Preservation of latency guarantees.** We prove that Hydrangea++ preserves Hydrangea's optimistic two-round and good-case three-round commit latencies, along with its enhanced fault resilience. Our results demonstrate that reducing proposal delay does not compromise either safety or liveness.

3. **Performance benefits** By shortening the proposal delay, Hydrangea++ improves both throughput and end-to-end decision latency under synchrony. This makes it particularly well-suited for high-performance deployments such as geo-replicated databases and permissioned blockchains.

2 Preliminaries

We consider a system $\mathcal{P} = P_1, \ldots, P_n$ of $n = 3f + 2c + m + 1$ parties, where up to f parties may be Byzantine and up to c parties may experience crash faults. Byzantine parties may behave arbitrarily, while crashed parties simply halt and cease communication. Any party that does not fail and follows the protocol is referred to as *honest*. The parameter m satisfies $0 \leq m \leq 2f + c - 4$ when c is even and $0 \leq m \leq 2f + c - 3$ when c is odd. Let $p = \lfloor \frac{c+m+2}{2} \rfloor$. For these values of m, it holds that $p + 1 \leq f + c$, ensuring that the number of faults tolerable on the fast path never exceeds the total fault budget of the system, consistent with the lower bound established in [18].

We assume the classical partial synchrony model of Dwork et al. [9]. Initially, the system may be asynchronous, allowing the adversary to arbitrarily delay messages between honest parties. After some unknown Global Stabilization Time (GST), the system becomes partially synchronous: every message sent by an honest party is delivered within a known upper bound Δ. Let δ denote the actual (and possibly variable) transmission delay, where $\delta \leq \Delta$ holds after GST. We assume that local clocks have no drift but may be arbitrarily skewed.

We assume the existence of digital signatures and a public-key infrastructure (PKI) to prevent spoofing, protect against replay attacks, and enable message authentication. We use $\langle x \rangle_i$ to denote a message x digitally

signed by party P_i using its private key. We use $H(x)$ to denote the application of the cryptographic hash function H on input x.

State Machine Replication. A state machine replication (SMR) protocol run by a network $\mathcal{P}$ of n nodes receives requests (transactions) from external parties, called *clients*, as input, and outputs a totally ordered log of these requests. We recall the definition of SMR given in [2], below.

Definition 1 (Byzantine Fault-Tolerant State Machine Replication [2])**.** *A Byzantine fault-tolerant state machine replication protocol commits client requests as a linearizable log to provide a consistent view of the log akin to a single non-faulty node, providing the following two guarantees.*

- ***Safety****. Honest parties do not commit different values at the same log position.*
- ***Liveness****. Each client request is eventually committed by all honest parties.*

3 The Hydrangea++ Protocol

3.1 Protocol Definitions

View-based execution. Our protocol progresses through a sequence of numbered *views*, beginning with all parties in view 1 and advancing to higher views as execution progresses. Each view v has a designated leader L_v, responsible for proposing a new block. Leader selection is rotated across views and is independent of the progress achieved within any single view.

Blocks and block format. Client requests are batched into blocks, where each block (except the genesis block) references its immediate predecessor. The position of a block within the chain is called its height. A block B_k at height k is represented as $B_k := (b_k, H(B_{k-1}))$, where b_k is the block's payload and $H(B_{k-1})$ is the hash of its predecessor B_{k-1}. The genesis block has $\perp$ as its predecessor.

A block B_k is considered valid if:

1. its predecessor is valid (or is $\perp$ when $k = 1$), and
2. the client requests it contains satisfy application-level validity conditions and are consistent with the sequence of requests in its ancestor blocks.

We say that a block B_k extends a block B_l ($k \geq l$) if B_l is an ancestor of B_k (noting that a block trivially extends itself). Two blocks B_k and $B'_{k'}$ are said to equivocate (or conflict) if they are distinct and neither extends the other.

Block certificate and weak block certificate. Following Moonshot [8], we employ three types of signed vote messages: the optimistic vote (opt-vote), the normal vote (vote), and the fallback vote (fb-vote). Crucially, vote messages of different types cannot be aggregated. Consequently, we define three corresponding types of block certificates. An *optimistic certificate* $\mathcal{C}^o_v(B_h)$ for a block B_h consists of $\lceil \frac{n+f+1}{2} \rceil$ distinct opt-vote messages for B_h in view v. Similarly, a *normal certificate* $\mathcal{C}^n_v(B_h)$ comprises $\lceil \frac{n+f+1}{2} \rceil$ distinct vote messages for B_h in view v, and a *fallback certificate* $\mathcal{C}^f v(B_h)$ comprises $\lceil \frac{n+f+1}{2} \rceil$ distinct fb-vote messages for B_h in view v. When the type is irrelevant, we simply denote a block certificate as $\mathcal{C}_v(B_h)$. Block certificates are ordered by view number, i.e., $\mathcal{C}_v \leq \mathcal{C}_{v'}$ whenever $v \leq v'$.

Additionally, we also define a weak block certificate $\mathcal{WC}_v(B_k)$ for view v as a collection of $f + p + 1$ distinct signed vote messages for a block B_k. We also differentiate between weak block certificate based on the vote type and define three corresponding types of weak block certificates. An *optimistic weak block certificate* $\mathcal{WC}^o_v(B_k)$ for a block B_k consists of $f+p+1$ distinct opt-vote messages for B_k in view v. Similarly, a *normal weak block certificate* $\mathcal{WC}^n_v(B_k)$ comprises $f + p + 1$ distinct vote messages for B_h in view v, and a *fallback weak block certificate* $\mathcal{WC}^f_v(B_k)$ comprises $f + p + 1$ distinct fb-vote messages for B_k in view v. When the type is irrelevant, we simply denote a weak block certificate as $\mathcal{WC}_v(B_k)$.

Weak block certificates are first ordered by their view number, such that $\mathcal{WC}_v(B_k) \leq \mathcal{WC}_{v'}(B_h)$ whenever $v \leq v'$. Within the same view v, fallback weak block certificates take the highest priority, followed by normal weak block certificates, and finally optimistic weak block certificates, formally: $\mathcal{WC}_v^f > \mathcal{WC}_v^n > \mathcal{WC}_v^o$. To compare a block certificate $\mathcal{C}_v(B_k)$ and a weak block certificate $\mathcal{WC}_{v'}(B_h)$, we rank $\mathcal{C}_v(B_k)$ higher if $v \geq v'$, and otherwise $\mathcal{WC}_{v'}(B_h)$ ranks higher.

Timeout messages and timeout certificates. To guarantee liveness, our protocol requires parties to trigger a leader change if they detect a lack of progress in their current view beyond a specified timeout threshold. Each party signals this by broadcasting a signed timeout message for the current view. A timeout message contains (i) the party's most recent votes in a view, and (ii) the highest weak block certificate or the highest-ranked block certificate observed by that party. The votes included in a timeout message must comply with the protocol's voting rules. For instance, a party is prohibited from casting both a vote and a fb-vote within the same view. Likewise, if a party has already cast an opt-vote for a block, it is forbidden from later casting a vote for an equivocating block in the same view.

A timeout certificate for view v, denoted $\mathcal{TC}_v$, is formed by collecting $n - f - c$ distinct, valid timeout messages for view v. The certificate serves as cryptographic evidence that a quorum of parties agrees to advance from view v to view $v + 1$.

Progress certificate and safe block selection. We use the notion of a progress certificate to justify that a leader in a given view can safely propose a block. Each progress certificate is associated with a view v and is denoted by $\mathcal{PC}_v$; it is used by the leader of view $v + 1$ to determine a safe block to extend. A progress certificate in view v is either a block certificate $\mathcal{C}_v$ or a timeout certificate $\mathcal{TC}_v$. The purpose of $\mathcal{PC}_v$ is to identify the safe block in view v: a block such that no conflicting (equivocating) block could have been, or can still be, committed—ensuring it is safe to extend.

A $\mathcal{TC}_v$ aggregates block certificates, weak block certificates, and the highest votes contained in the included timeout messages. If the votes in $\mathcal{PC}_v$ provide sufficient support for a block, a weak block certificate can be constructed, and we select the highest-ranked weak block certificate that can be formed. The safe block is then chosen as the block associated with the highest-ranked certificate—either a block certificate or a weak block certificate.

4 Security Analysis

Claim 1. *Any two quorums $\mathcal{Q}_1$ and $\mathcal{Q}_2$ of size $\lceil \frac{n+f+1}{2} \rceil$ intersect in at least one honest party.*

Proof. Each quorum contains at least $\lceil \frac{n+f+1}{2} \rceil$ parties. Hence, the intersection of two quorums $\mathcal{Q}_1$ and $\mathcal{Q}_2$ is of size at least $\lceil \frac{n+f+1}{2} \rceil + \lceil \frac{n+f+1}{2} \rceil - n \geq f + 1$. Since at most f parties can be Byzantine, this guarantees that $\mathcal{Q}_1$ and $\mathcal{Q}_2$ share at least one honest party. □

Claim 2. *If $\mathcal{C}_v^o(B_k)$ exists then $\mathcal{TC}_{v-1}$ does not exist, and vice-versa.*

Proof. Suppose, for the sake of contradiction, that both certificates exist. Let $\mathcal{Q}_1$ denote the set of parties that contributed to $\mathcal{C}_v^o(B_k)$ and $\mathcal{Q}_2$ the set of parties that contributed to $\mathcal{TC}_{v-1}$. By definition, $|\mathcal{Q}_1| = \lceil \frac{n+f+1}{2} \rceil$ and $|\mathcal{Q}_2| = n - f - c$. Hence, their intersection satisfies $|\mathcal{Q}_1 \cap \mathcal{Q}_2| \geq \lceil \frac{n+f+1}{2} \rceil + (n - f - c) - n = f + 1 + \lceil \frac{k}{2} \rceil$. Since at most f parties can be Byzantine, there must exist at least one honest party (say P_i) that appears in both $\mathcal{Q}_1$ and $\mathcal{Q}_2$. Thus, P_i must have sent both $\langle \text{opt-vote}, H(B_k), v \rangle_i$ and a view $v - 1$ timeout message.

By the optimistic vote rule, P_i must have had timeout_view $< v - 1$ when casting its optimistic vote for B_k, which implies that its timeout message for view $v - 1$ was sent after voting for B_k. However, the optimistic vote rule also requires that P_i be in view v when voting for B_k. By the timeout rule, a party in view v cannot multicast $\mathcal{T}$ messages for view $v - 1$ or any lower view after casting a vote in view v. This yields a contradiction, completing the proof. □

Claim 3. *If $\mathcal{C}_v^o(B_k)$ and $\mathcal{C}_v^n(B_l)$ exist then $B_k = B_l$.*

Each party P_i runs the following protocol while in view v:

1. **Propose.** Upon entering v and after executing *Advance View* and *Lock*, if P_i is L_v, propose using one of the following rules:
   (a) **Normal Propose.** If L_v entered v by receiving $\mathcal{C}_{v-1}(B_{k-1})$, multicast $\langle \mathsf{propose}, B_k, \mathcal{C}_{v-1}(B_{k-1}), v\rangle$ such that B_k extends B_{k-1}.
   (b) **Fallback Propose.** If L_v entered v by receiving $\mathcal{TC}_{v-1}$, multicast $\langle \mathsf{fb\text{-}propose}, B_k, \mathcal{TC}_{v-1}, v\rangle$, where B_k extends B_{k-1} that is proposed in some view $v' < v$, and is considered a safe block according to the accompanying $\mathcal{TC}_{v-1}$.
2. **Vote.** P_i votes at most twice in view v when the following conditions are met:
   (a) **Optimistic Vote.** Upon receiving $\langle \mathsf{opt\text{-}propose}, B_k, v\rangle$ such that B_k extends B_{k-1}, if (i) $\mathsf{timeout\_view}_i < v-1$, (ii) $\mathsf{lock}_i = \mathcal{C}_{v-1}(B_{k-1})$ and (iii) P_i has not voted in v, set $\mathsf{last\text{-}opt\text{-}vote}_i := \langle \mathsf{opt\text{-}vote}, H(B_k), v\rangle_i$ multicast $\mathsf{last\text{-}opt\text{-}vote}_i$ to all parties.
   (b) After executing *Advance View* and *Lock* with all embedded certificates, vote once when one of the following conditions are satisfied:
      i. **Normal Vote.** Upon receiving $\langle \mathsf{propose}, B_k, \mathcal{C}_{v-1}(B_{k-1}), v\rangle$, if (i) $\mathsf{timeout\_view}_i < v$, (ii) B_k directly extends B_{k-1} and (iii) P_i has not sent an optimistic vote for an equivocating block $B'_{k'}$ in v, set $\mathsf{last\text{-}vote}_i := \langle \mathsf{vote}, H(B_k), v\rangle_i$ and multicast $\mathsf{last\text{-}vote}_i$ to all parties.
      ii. **Fallback Vote.** Upon receiving $\langle \mathsf{fb\text{-}propose}, B_k, \mathcal{TC}_{v-1}, v\rangle$ if (i) $\mathsf{timeout\_view}_i < v$, (ii) B_k directly extends B_{k-1} that was proposed in a previous view $v' < v$, and B_{k-1} is considered a safe block according to the accompanying $\mathcal{TC}_{v-1}$, set $\mathsf{last\text{-}fb\text{-}vote}_i := \langle \mathsf{fb\text{-}vote}, H(B_k), v\rangle_i$ and multicast $\mathsf{last\text{-}fb\text{-}vote}_i$ to all parties. If B_{k-1} is deemed safe due to a weak block certificate $\mathcal{WC}_{v'}(B_{k-1})$ included in $\mathcal{TC}_{v-1}$, then $\mathsf{hwc}_i := \mathcal{WC}_{v'}(B_{k-1})$.
3. **Optimistic Propose.** Upon voting for B_k in view v, if P_i is L_{v+1}, multicast $\langle \mathsf{opt\text{-}propose}, B_{k+1}, v+1\rangle$ such that B_{k+1} extends B_k.
4. **Direct Pre-commit.** Upon receiving $\mathcal{C}_v(B_k)$ whilst in any view $v' \leq v$ if $\mathsf{timeout\_view}_i < v$, multicast $\langle \mathsf{commit}, H(B_k), v\rangle_i$.
5. **Indirect Pre-commit.** Upon receiving $\mathcal{C}_v(B_k)$ in any view, if a party has previously multicast a commit for any descendant of B_k and its $\mathsf{timeout\_view}_i < v$, then it multicasts $\langle \mathsf{commit}, H(B_k), v\rangle_i$ if it has not already done so.
6. **Timeout.** When $\mathsf{view\text{-}timer}_i$ expires, party P_i multicasts $\langle \mathsf{timeout}, v, \mathsf{h\text{-}cert}_i, \mathsf{last\text{-}opt\text{-}vote}_i, \mathsf{last\text{-}vote}_i, \mathsf{last\text{-}fb\text{-}vote}_i\rangle_{P_i}$ (if it has not already done so) and updates $\mathsf{timeout\_view}_i$ to $\max(\mathsf{timeout\_view}_i, v)$. Furthermore, upon receiving either $f+1$ distinct messages of the form $\langle \mathsf{timeout}, v', \_, \_\rangle_*$ or a valid $\mathcal{TC}_{v'}$ for some $v' \geq v$, it multicasts $\langle \mathsf{timeout}, v', \mathsf{h\text{-}cert}_i, \mathsf{last\text{-}opt\text{-}vote}_i, \mathsf{last\text{-}vote}_i, \mathsf{last\text{-}fb\text{-}vote}_i\rangle_{P_i}$ and updates $\mathsf{timeout\_view}_i$ to $\max(\mathsf{timeout\_view}_i, v')$. Here, $\mathsf{h\text{-}cert}_i$ refers to the higher-ranked certificate between lock_i and hwc_i.
7. **Advance View.** P_i enters v' where $v' > v$ using one of the following rules:
   - Upon receiving $\mathcal{C}_{v'-1}(B_h)$. Also, multicast $\mathcal{C}_{v'-1}(B_h)$.
   - Upon receiving $\mathcal{TC}_{v'-1}$. Also, unicast $\mathcal{TC}_{v'-1}$ to $L_{v'}$.

   Finally, reset $\mathsf{view\text{-}timer}_i$ to 3Δ and start counting down.

P_i additionally performs the following actions in any view:

1. **Lock.** Upon receiving $\mathcal{C}_v(B_k)$ in any protocol message whilst having $\mathsf{lock}_i = \mathcal{C}_{v'}(B_{k'})$ such that $v > v'$, set lock_i to $\mathcal{C}_v(B_k)$.
2. **Direct Commit.** P_i directly commits B_k whilst in any view using one of the following rules:
   - **(Fast Commit.)** Upon receiving $n-p$ distinct vote messages of the same type (i.e., opt-vote, vote, or fb-vote) for B_k in view v,
   - **(Slow Commit.)** Upon receiving $2f+c+1$ distinct $\langle \mathsf{commit}, H(B_k), v\rangle_*$ messages.
3. **Indirect Commit.** Upon directly committing B_k, commit all of its uncommitted ancestors.

Figure 1: The Hydrangea++ protocol

Proof. By Claim 1 and the requirement that a block certificate be formed from a quorum of votes of the same type for the same block, there must exist at least one honest node (say P_i) that voted for both B_k and B_l. By the optimistic vote rule, P_i could have voted for B_k only if it had not yet voted in view v, which implies that its vote for B_l must have occurred after its vote for B_k. Furthermore, the normal vote rule permits P_i to vote for B_l only if it has not already issued an optimistic vote for a conflicting block. Consequently, P_i could have voted for B_l after voting for B_k only if $B_l = B_k$. Since P_i must have voted for both blocks, it

follows that $B_l = B_k$. □

Claim 4. *If $\mathcal{C}_v(B_k)$ and $\mathcal{C}_v(B_l)$ exist then $B_k = B_l$.*

Proof. By Claim 1, there must exist at least one honest node (say P_i) that contributed votes to both certificates. We analyze four possible cases:

1. Both certificates have the same type. Since each vote rule can be triggered at most once in a given view, P_i could have voted for both certificates only if $B_k = B_l$.
2. $\mathcal{C}_v(B_k) = \mathcal{C}_v^n(B_k)$ and $\mathcal{C}_v(B_l) = \mathcal{C}_v^f(B_l)$ (or vice versa). The respective vote rules prohibit a party from voting for a proposal of one type if it has already voted for a proposal of the other type. Hence, P_i could not have voted for both certificates—contradicting the earlier conclusion.
3. $\mathcal{C}_v(B_k) = \mathcal{C}_v^o(B_k)$ and $\mathcal{C}_v(B_l) = \mathcal{C}_v^f(B_l)$ (or vice versa). By the fallback vote rule, P_i could have voted for B_l only if it had a justification from $\mathcal{TC}_{v-1}$. By Claim 2, this would make $\mathcal{C}_v^o(B_k)$ impossible, contradicting the assumption that it exists.
4. $\mathcal{C}_v(B_k) = \mathcal{C}_v^o(B_k)$ and $\mathcal{C}_v(B_l) = \mathcal{C}_v^n(B_l)$ (or vice versa). This case is handled by Claim 3.

Since all cases either force $B_k = B_l$ or lead to a contradiction, it follows that $B_k = B_l$. □

Claim 5. *A set $\mathcal{Q}$ of $n - f - p$ honest parties and a set $\mathcal{Q}'$ of $n - 2f - c$ honest parties must intersect in at least $f + p + 1$ honest parties.*

Proof. The intersection of $\mathcal{Q}$ and $\mathcal{Q}'$ has size at least $n - f - p + n - 2f - c - (n - f) = f + c + k + 1 - p$. We now analyze this quantity based on the parity of $c + k$:

- When $c + k$ is even, $p = \frac{c+k}{2}$; so, $f + c + k + 1 - p = f + \frac{c+k}{2} + 1 = f + p + 1$.
- When $c + k$ is odd, $p = \frac{c+k-1}{2}$; so $f + c + k + 1 - p = f + \frac{c+k+1}{2} + 1 > f + p + 1$.

Thus, in both cases, the intersection has size at least $f + p + 1$ honest parties. □

Claim 6. *If a block B_k proposed in view v is directly committed via the fast commit rule and a corresponding $\mathcal{TC}_v$ exists, then $\mathcal{WC}_v(B_k)$ must appear in $\mathcal{TC}_v$, and no weak block certificate for a conflicting block with rank higher than $\mathcal{WC}_v(B_k)$ can appear in $\mathcal{TC}_v$.*

Proof. For a block B_k to be directly committed via the fast commit rule, it must collect at least $n - p$ votes of the same type (either opt-vote, vote, or fb-vote) in view v. This voting set contains a subset $\mathcal{Q}$ of at least $n - f - p$ honest parties. The corresponding $\mathcal{TC}_v$ contains timeout messages from a set $\mathcal{Q}'$ of at least $n - 2f - c$ honest parties. By Claim 5, the intersection $\mathcal{Q} \cap \mathcal{Q}'$ contains a set $\mathcal{S}$ of at least $f + p + 1$ honest parties. Then every party in $\mathcal{S}$ must have cast its last vote for B_k in view v and subsequently contributed to $\mathcal{TC}_v$.

We now analyze each possible case based on the type of votes that triggered the fast commit rule:

- **Fast commit triggered by $n - p$ opt-vote for B_k.** Since $n - p > \lceil \frac{n+f+1}{2} \rceil$, by Claim 3, $\mathcal{TC}_{v-1}$ cannot exist. Hence, no honest party can send a fb-vote in view v. Similarly, by the normal vote rule, an honest party that has sent opt-vote for B_k in view v cannot later send a vote for a conflicting block. Therefore, at most $f + p$ parties could have sent either vote or opt-vote for a conflicting block in view v, which is insufficient to form a weak block certificate.
- **Fast commit triggered by $n - p$ vote for B_k.** As above, at most $f + p$ parties could have cast any vote for a conflicting block in view v, which is not enough to form a weak block certificate.
- **Fast commit triggered by $n - p$ fb-vote for B_k.** In this case, there must be at least $f + p + 1$ fb-vote for B_k included in $\mathcal{TC}_v$. Since $\mathcal{WC}_v^f(B_k)$ has the highest rank among all weak block certificates in view v, no weak block certificate of higher rank can exist in $\mathcal{TC}_v$.

Hence, in all cases, $\mathcal{WC}_v(B_k)$ appears in $\mathcal{TC}_v$, and no weak block certificate for a conflicting block of rank higher than $\mathcal{WC}_v(B_k)$ can exist in $\mathcal{TC}_v$. □

Lemma 1. *If an honest party directly commits block B_k proposed in view v using the fast commit rule, and $B_{k'}$ is the safe block as per $\mathcal{PC}_{v'}$ such that $v' \geq v$ then $B_{k'}$ extends B_k.*

Proof. Suppose an honest party P_i directly commits block B_k proposed in view v using the fast commit rule, having received at least $n - p$ votes of the same type for B_k. This ensures that a set $\mathcal{Q}$ of at least $n - f - p$ honest parties voted for B_k in view v. We first consider the case where $v' = v$. If $\mathcal{PC}_v$ is a block certificate for view v, then by Claim 4, no conflicting block certificate can exist in the same view. Hence, $\mathcal{PC}_v = \mathcal{C}_v(B_k)$, and B_k is the only safe block. Next, we consider the case where $\mathcal{PC}_v = \mathcal{TC}_v$, and specifically when $\mathcal{TC}_v$ does not contain $\mathcal{C}_v(B_k)$. In this case, by Claim 6, $\mathcal{WC}_v(B_k)$ is the highest ranked weak block certificate in $\mathcal{PC}_v$ and only B_k qualifies as a safe block under $\mathcal{PC}_v$.

We now consider the case where $v' > v$. Suppose, for contradiction, that there exists $\mathcal{PC}_{v'}$ at the lowest such view $v' > v$, where the associated safe block $B_{k'}$ does not extend B_k. Note that $\mathcal{PC}_{v'}$ could either be $\mathcal{C}_{v'}(B_{k'})$ or $\mathcal{TC}_{v'}$. First, consider the case where $\mathcal{PC}_{v'} = \mathcal{C}_{v'}(B_{k'})$. For this certificate to exist, at least $\lceil \frac{n-f+1}{2} \rceil$ honest parties must have voted for $B_{k'}$ in view v'. However, by assumption, in every intermediate view $v \leq v'' < v'$, the safe block determined by $\mathcal{PC}_{v''}$ extends B_k. Since honest parties vote only for blocks that extend the safe block from the previous view, they would not have voted for $B_{k'}$ in view v' unless it also extended B_k. This contradicts the assumption that $B_{k'}$ does not extend B_k. A similar argument rules out the existence of a weak block certificate or block certificate for a conflicting block in any view v'' such that $v \leq v'' \leq v'$.

Now, consider the case where $\mathcal{PC}_{v'} = \mathcal{TC}_{v'}$, which consists of timeout messages from view v' sent by $n - f - c$ parties, among which a subset $\mathcal{Q}'$ of at least $n - 2f - c$ are honest. By Claim 5, $\mathcal{Q}$ and $\mathcal{Q}'$ intersect in set $\mathcal{S}$ of at least $f + p + 1$ honest parties. As mentioned before, honest parties do not vote for blocks in view v' if it does not extend B_k. Therefore, all honest parties in $\mathcal{S}$ must have last voted in views v'' such that $v \leq v'' \leq v'$, and these votes must extend B_k. If these honest parties did not vote in any view higher than v, then their high votes in view v will be included in $\mathcal{TC}_{v'}$, collectively forming $\mathcal{WC}_v(B_k)$. If instead they voted in a higher view $v'' > v$, they would either update their hwc to $\mathcal{WC}_v(B_k)$ (or a higher ranked one), or update their lock to a block certificate from some view in the range $[v, v' - 1]$. As shown earlier, all weak block certificates and block certificates in views $v \leq v'' \leq v'$ must extend B_k. Therefore, $\mathcal{TC}_{v'}$ must include either a block certificate for some view in the range $[v, v' - 1]$ or a weak block certificate from some view v'' such that it extends B_k. Thus, the safe block computed from $\mathcal{TC}_{v'}$ must necessarily extend B_k, contradicting the assumption that the safe block determined by $\mathcal{PC}_{v'}$ does not extend B_k. □

Lemma 2. *If an honest party directly commits block B_k proposed in view v using the slow commit rule, and $B_{k'}$ is the safe block determined by $\mathcal{PC}_{v'}$ for some view $v' \geq v$, then $B_{k'}$ must extend B_k.*

Proof. Suppose an honest party P_i directly commits block B_k proposed in view v using the slow commit rule, having received at least $2f + c + 1$ commit messages for B_k. This implies that a set $\mathcal{Q}$ of at least $f + c + 1$ honest parties must have received $\mathcal{C}_v(B_k)$ before sending a timeout message for view v or exiting the view. Consequently, the progress certificate $\mathcal{PC}_v$ must be the block certificate $\mathcal{C}_v(B_k)$, since at least $f + c + 1$ honest parties observed it and this intersects with the $n - f - c$ messages in the timeout certificate $\mathcal{TC}_v$. Moreover, by Claim 4, no conflicting block certificate can exist in view v. Hence, $\mathcal{PC}_v = \mathcal{C}_v(B_k)$, and B_k extends itself.

We now consider the case where $v' > v$. Suppose, for the sake of contradiction, that there exists a progress certificate $\mathcal{PC}_{v'}$ for some lowest view $v' > v$ such that the corresponding safe block $B_{k'}$ does not extend B_k. Note that $\mathcal{PC}_{v'}$ could either be a block certificate $\mathcal{C}_{v'}(B_{k'})$ or a timeout certificate $\mathcal{TC}_{v'}$. First, consider the case where $\mathcal{PC}_{v'} = \mathcal{C}_{v'}(B_{k'})$. For such a certificate to exist, at least $\lceil \frac{n-f+1}{2} \rceil$ honest parties must have voted for $B_{k'}$ in view v'. However, by assumption, in every view v'' such that $v \leq v'' < v'$, the safe block identified by $\mathcal{PC}_{v''}$ extends B_k, and honest parties vote only for blocks that extend the safe block of the previous view. Therefore, honest parties would not have voted for $B_{k'}$ in view v' if it did not extend B_k. This leads to a contradiction. This implies that no block certificate for a conflicting block (i.e., one that does not extend B_k) can exist in the range of views $[v, v']$.

Now consider the case where $\mathcal{PC}_{v'} = \mathcal{TC}_{v'}$, which consists of timeout messages from view v' sent by a set $\mathcal{Q}'$ of $n - f - c$ parties. As established earlier, honest parties do not vote for blocks in view v' unless those blocks extend B_k. Let v^* denote the view corresponding to the highest-ranked weak block certificate present in $\mathcal{PC}_{v'}$. If $v < v^* < v'$, then by assumption, all blocks voted for or potentially certified in this range of views must extend B_k. As a result, the weak block certificate for a block from view v^* must also extend B_k. Furthermore, if $v^* = v'$, then the weak block certificate must still extend B_k, since the safe block identified by $\mathcal{PC}_{v^*-1}$ also extends B_k. Alternatively, if $v^* \leq v$, then $\mathcal{PC}_{v'}$ must include either the block certificate $\mathcal{C}_v(B_k)$ or higher-ranked block certificate, since the intersection of the honest voting sets $\mathcal{Q}$ (from view v) and $\mathcal{Q}'$ (from view v') contains at least one honest party. As argued before all block certificates in the range of views $[v, v']$ must extend B_k. Therefore, in either case, the safe block determined from $\mathcal{PC}_{v'}$ must extend B_k, contradicting the assumption that it does not. □

The proof of the following lemma (Lemma 3) follows immediately from Lemma 1 and Lemma 2.

Lemma 3. *If an honest party directly commits a block B_k proposed in view v, and $B_{k'}$ is the safe block determined by $\mathcal{PC}_{v'}$ for some view $v' \geq v$, then $B_{k'}$ must extend B_k.*

Theorem 1 (Safety)**.** *Honest parties do not commit different values at the same log position.*

Proof. We show that if two honest parties P_i and P_j commit B_k and B'_k, then $B_k = B'_k$. Suppose, for the sake of contradiction, that P_i and P_j commit B_k and B'_k but $B_k \neq B'_k$. By the indirect commit rule, both parties must have directly committed descendant blocks—specifically, P_i committed block B_ℓ in view v such that B_ℓ extends B_k with $\ell \geq k$, and P_j committed block B_o in view v' such that B_o extends B'_k with $o \geq k$. By Lemma 3, either $v \leq v'$ and B_o extends B_l, or $v \geq v'$ and B_l extends B_o. Therefore, since B_l and B_o are a part of the same chain and because each block in the chain has exactly one parent, $B_k = B'_k$. □

Claim 7. *If an honest party enters view v then at least one honest party must have already entered $v - 1$.*

Proof. An honest party enters view v only after receiving either a block certificate for view $v - 1$ (i.e., $\mathcal{C}_{v-1}$) or a timeout certificate (i.e., $\mathcal{TC}_{v-1}$). In the case of $\mathcal{C}_{v-1}$, the voting rule mandates that parties must be in view $v - 1$ when casting votes, implying that at least one honest party was already in view $v - 1$. In the case of $\mathcal{TC}_{v-1}$, at least one honest party must have experienced a view timeout while in view $v - 1$ before any honest party can multicast a view $v - 1$ timeout message. Therefore, in either case, an honest party can enter view v only if at least one honest party has already entered view $v - 1$. □

Claim 8. *If the first honest party enters view v at time t then no honest party multicasts* timeout *for view $v' \geq v$ before $t + 3\Delta$.*

Proof. Since t is defined as the time at which the first honest party enters view v, the timeout rule guarantees that no honest party's view timer can expire in view v before time $t + 3\Delta$. At most f timeout messages for view v may exist prior to this point, all from Byzantine parties. Because the timeout rule requires an honest party to either (i) have its own view timer expire in v or (ii) observe at least $f + 1$ timeout messages for v before it can multicast a timeout message for v, no honest party can send a timeout message for view v before time $t + 3\Delta$.

Furthermore, by Claim 7, no honest party can enter any view $v'' > v$ before time t. Applying the same reasoning, it follows that no honest party can send a timeout message for any view $v'' > v$ before time $t + 3\Delta$. □

Corollary 1 follows from Claim 8 and the requirement that timeout certificates be constructed from $n - f - c$ of timeout messages for the same view.

Corollary 1. *If the first honest party enters view v at time t then $\mathcal{TC}_{v'}$ cannot exist for $v' \geq v$ before $t+3\Delta$.*

Claim 9. *Let t_g denote GST. If the first honest party to enter view v, say P_i, does so at time $t \geq t_g$, then every honest party enters view v or a higher view by time $t + 2\Delta$.*

Proof. If an honest party enters view $v' \geq v$ via a certificate $\mathcal{C}_{v'-1}$ before time $t + \Delta$, then it must have multicast $\mathcal{C}_{v'-1}$. As a result, all honest parties will receive this certificate and enter view v' or higher before time $t + 2\Delta$. Now suppose that no honest party enters view v' via $\mathcal{C}_{v'-1}$ before $t + \Delta$. In this case, P_i must have entered view v using a timeout certificate $\mathcal{TC}_{v-1}$. This implies that at least $n - 2f - c$ honest parties must have multicast view $v - 1$ timeout messages before time t, and thus all honest parties will receive these messages by time $t + \Delta$.

Furthermore, since t is defined as the time at which the first honest party enters view v, by Corollary 1, a valid $\mathcal{TC}_{v'}$ for any $v' \geq v$ cannot exist before time $t + 3\Delta$. Consequently, all honest parties must still be in view v or lower when they receive the aforementioned timeout messages for view $v - 1$. By the timeout rule, every honest party in view $v - 1$ or lower that has not yet multicast a view $v - 1$ timeout messages will do so by time $t + \Delta$.

Moreover, every honest party that enters view v before this time must have done so via $\mathcal{TC}_{v-1}$, and by the timeout rule, must also multicast a timeout message for view $v - 1$ before $t + \Delta$. As a result, all honest parties will have multicast view $v - 1$ timeout message by this time, enabling each of them to construct $\mathcal{TC}_{v-1}$ before $t + 2\Delta$. Thus, every honest party will enter view v or higher before time $t + 2\Delta$. □

Lemma 4. *Let t_g denote GST. If the first honest party to enter view v, say P_i, does so at time t, then all honest parties will enter view v or higher by time* $\max(t_g, t) + 3\Delta$.

Proof. If $t \geq t_g$, then by Claim 9, all honest parties will enter view v or higher before time $\max(t_g, t) + 2\Delta$. Now consider the case when $t < t_g$. Let v'' be the highest view reached by any honest party at time t_g, and let P_j be a party in view v'' at this moment. Note that $v'' \geq v$.

If any honest party enters a view higher than v'', say v^*, during the interval $[t_g, t_g + \Delta]$, then by Claim 9, all honest parties will enter view $v^* > v$ or higher by time $t_g + 3\Delta = \max(t_g, t) + 3\Delta$.

Otherwise, assume no honest party enters a view higher than v'' before $t_g + \Delta$. If some honest party enters view v'' via a block certificate $\mathcal{C}_{v''-1}$ before this time, then all honest parties will enter v'' before $t_g + 2\Delta < \max(t_g, t) + 3\Delta$.

In the remaining case, P_j (and any other party in v'' at time t_g) must have entered v'' via a timeout certificate $\mathcal{TC}_{v''-1}$ and thus would have sent a timeout message for view $v'' - 1$ no later than the time of entry. Since a valid $\mathcal{TC}_{v''-1}$ requires at least $n - 2f - c$ honest parties to have sent timeout messages before t_g, all honest parties in views lower than v'' will receive these messages by time $t_g + \Delta$. By the timeout rule, they will multicast their own timeout messages for view $v'' - 1$ if they have not already done so.

As a result, all honest parties will multicast a timeout message for view $v'' - 1$ before $t_g + \Delta$, enabling them to construct $\mathcal{TC}_{v''-1}$ and enter view v'' before $t_g + 2\Delta < \max(t_g, t) + 3\Delta$.

Therefore, in all cases, every honest party will enter view v or higher before time $\max(t_g, t) + 3\Delta$. □

Lemma 5. *All honest parties keep entering increasing views.*

Proof. Suppose, for the sake of contradiction, that at least one honest party, say P_i, becomes stuck in view v, and let v' be the highest view reached by any honest party at any time. If $v' > v$, then by Lemma 4, P_i must eventually enter view v' or higher, contradicting the assumption that it remains stuck in v. On the other hand, if $v' = v$, then no honest party ever enters a view higher than v. But since Lemma 4 guarantees that all honest parties will eventually enter view v, this implies that all honest parties become stuck in v. However, by the view advancement and timeout rules, every honest party will eventually multicast a timeout message for view v, allowing them to construct $\mathcal{TC}_v$ and enter view $v + 1$. This again contradicts the assumption that they remain stuck in v. □

Claim 10. *If the first honest party enters view v at time t after GST and the leader L_v is honest, then L_v will propose a block by time $t + \Delta$, and all honest parties will receive this proposal by time $t + 2\Delta$.*

Proof. Let P_i be the first honest party to enter view v at time t. By the view advancement rule, P_i may have entered v either via a block certificate $\mathcal{C}_{v-1}$ or a timeout certificate $\mathcal{TC}_{v-1}$. In the former case, P_i multicasts $\mathcal{C}_{v-1}$ to all parties; in the latter case, it unicasts $\mathcal{TC}_{v-1}$ to the leader L_v. Thus, in either case, the honest leader L_v will receive a valid certificate enabling it to enter view v and propose a block by time

$t + \Delta$, according to the proposal rule. Since L_v is honest, it multicasts the proposal, ensuring that all honest parties receive it by time $t + 2\Delta$. □

Lemma 6. *If the first honest party enters view v at time t after GST and the leader L_v is honest, then all honest parties receive the block certificate $\mathcal{C}_v(B_k)$ for some block B_k proposed by L_v by time $t + 3\Delta$.*

Proof. By Claim 4, only one block can be certified in a given view. Thus, if $\mathcal{C}_v(B_k)$ exists, any party that receives a view-v block certificate must receive $\mathcal{C}_v(B_k)$. Additionally, by Claim 9 and Claim 10, all honest parties enter view v or higher and receive the proposal from L_v by $t + 2\Delta$. Since t is the time when the first honest party enters view v, no honest party will multicast a view v $\mathsf{timeout}$ message before $t+3\Delta$. Therefore, if any honest party enters a view higher than v before $t+2\Delta$, then by Claim 7, some honest party must have already entered $v+1$ via $\mathcal{C}_v(B_k)$. By the view advancement rule, this party would have multicast $\mathcal{C}_v(B_k)$, so all honest parties receive this certificate by $t + 3\Delta$, completing the proof. Alternatively, if no honest party receives $\mathcal{C}_v(B_k)$ before $t + 2\Delta$, then all honest parties will enter v before $t + 2\Delta$.

Since L_v is honest, it will generate either a normal or fallback proposal that extends the safe block determined by $\mathcal{PC}_{v-1}$ at time $t + \Delta$. If L_v created a normal proposal, any equivocal optimistic proposal it might have previously generated must have a different parent than the normal proposal, since honest leaders propose a fixed block payload for a given view. Consequently, by Claim 4, the parent of such an equivocal optimistic proposal cannot be certified, and thus, by the optimistic vote rule, no honest node can vote for it.

Furthermore, all honest parties will receive L_v's proposal by time $t + 2\Delta$, and at this time none of them will have $\mathsf{timeout\_view}_i \geq v$. By the voting rules, they will therefore all vote for the proposed block, allowing every honest party to construct $\mathcal{C}_v(B_k)$ by time $t + 3\Delta$. □

Lemma 7. *If the first honest party to enter view v does so at time t after GST and L_v is honest, then all honest parties commit block B_k proposed by L_v by time $t + 4\Delta$.*

Proof. By Lemma 6, all honest parties obtain $\mathcal{C}_v(B_k)$ for the block B_k proposed by L_v by time $t+3\Delta$. Consequently, following the pre-commit rule, if each honest party multicasts $\langle \mathsf{commit}, H(B_k), v\rangle_*$ upon receiving this certificate, then all honest parties will commit B_k by time $t + 4\Delta$.

Otherwise, suppose for contradiction that at least one honest party, say P_i, fails to send $\langle \mathsf{commit}, H(B_k), v\rangle_*$ by $t + 3\Delta$. However, by Claim 8, no honest party's view timer can expire before this time, so P_i must have $\mathsf{timeout\_view}_i < v$ upon receiving $\mathcal{C}_v(B_k)$.

Thus, by the pre-commit rules, upon receiving $\mathcal{C}_v(B_k)$, P_i must neither be in view v or lower nor have previously multicast a commit message for any descendant of B_k. Let v' be the view P_i is in upon receiving $\mathcal{C}_v(B_k)$. Since, by Corollary 1, no timeout certificate for v or higher can exist before $t + 3\Delta$, P_i must have entered v' via a block certificate $\mathcal{C}_{v'-1}(B_l)$.

By the direct pre-commit rule, it must have multicast $\langle \mathsf{commit}, H(B_l), v' - 1\rangle_i$ upon receiving this certificate, implying B_l is not a descendant of B_k. However, because $v' > v + 1$ and the only way to transition between these views (before $t + 3\Delta$) is via block certificates, and since $\mathcal{C}_v(B_k)$ exists and, by Claim 4, no conflicting certificate exists in view v, it follows that honest parties only vote for blocks in view $v + 1$ that extend B_k. By induction, this implies B_l must be a descendant of B_k, contradicting the earlier conclusion.

Therefore, all honest parties must multicast $\langle \mathsf{commit}, H(B_k), v\rangle_*$ before $t + 3\Delta$, and hence all honest parties commit B_k before $t + 4\Delta$. □

Theorem 2 (Liveness)**.** *Each client request is eventually committed by all honest parties.*

Proof. By Lemma 5, all honest parties continue to make progress by entering higher views. As the leader rotates across views, honest leaders will eventually be elected. Then, by Lemma 7, any block proposed by an honest leader after GST will be committed. □

5 Related Work

Classical three-round BFT protocols. Classical partially synchronous consensus protocols [4, 5, 8, 16] primarily optimize block commit latency, achieving finality in three communication rounds while tolerating up to $f < n/3$ Byzantine faults [9]. This three-round bound has been proven to be optimal [3]. These designs naturally extend to the more general $n \geq 3f + 2c + 1$ setting, where f and c represent Byzantine and crash faults, respectively. However, despite variations in implementation and performance, this entire class of protocols [7] fundamentally remains constrained to three-round block commit latency.

Optimistic two-round BFT protocols. A complementary line of work [1,11,12,14,15,17] explores protocols that can optimistically commit the proposed block in two rounds under favorable conditions—specifically, when the actual number of faulty parties is below a certain threshold. These protocols typically operate in systems of $n = 3f + 2p + 1$ where f is the tolerated Byzantine faults and p is a tunable optimism parameter. They ensure three-round block commits, but achieve two-round block commits when at most $p \leq f$ faulty parties are present, gaining speed at the cost of reduced resilience.

A recent work, Hydrangea [18] introduced a new resilience model capturing both Byzantine and crash faults, operating in systems of $n = 3f + 2c + m + 1$, where $m \geq 0$ is a tunable parameter. Hydrangea can commit blocks in two rounds when the number of faulty parties (Byzantine or crash) is at most $p = \lfloor \frac{c+m}{2} \rfloor$, while still guaranteeing three-round block commit even with up to f Byzantine and c crash faults. Compared to prior optimistic two-round protocols, Hydrangea additionally tolerates crash faults without sacrificing optimistic two-round block commitment.

Alpenglow [13] claims to tolerate up to $< 20\%$ Byzantine and 20% crash faults while committing optimistically; however, this guarantee does not hold under partial synchrony, as acknowledged in their own analysis [13, Section 2.11, Example 44]. In practice, Alpenglow supports a fast two round commit with $< 20\%$ Byzantine faults, or alternatively up to 40% crash faults with no Byzantine faults.

Recent two-round BFT protocols. Both Minimmit [6] and ChonkyBFT [10] operate in the $n \geq 5f + 1$ setting and therefore tolerate only 20% Byzantine faults. These protocols eliminate the three-round fallback path entirely and always attempt to commit in two rounds, requiring $n - f$ votes (approximately 81% of parties) to finalize a block. Minimmit further improves proposal latency by allowing the leader of view $v + 1$ to propose as soon as the previous block in view v $2f + 1$ votes, rather than waiting for $3f + 1$ as in earlier protocols. In geo-distributed deployments, collecting fewer messages can shorten proposal delays and reduce end-to-end latency.

However, while these protocols achieve two-round commit, they rely on large quorum sizes. In practice, a three-round fallback that requires fewer messages (e.g., 61% in the second round and 41% in the third) can outperform a two-round path that requires 81%.

All of these protocols (except Moonshot) focus primarily on reducing block commit latency while still incurring a 2δ proposal delay. Proposal delay directly translates to queuing delay for client transactions, and therefore has a substantial impact on end-to-end latency. Moonshot improves on this by enabling δ-delay proposals, but still requires three rounds to commit a block.

Hydrangea++ advances Hydrangea's design by optimizing both sides of the latency equation. While Hydrangea retains a minimum 2δ gap between honest proposals, Hydrangea++ reduces this to the theoretical minimum of δ by incorporating optimistic proposals, inspired by Moonshot's approach. Crucially, Hydrangea++ does so without sacrificing any correctness guarantees: it preserves Hydrangea's two-round optimistic and three-round block commit paths while significantly improving throughput and end-to-end latency under optimistic network conditions.

References

[1] Ittai Abraham, Guy Gueta, Dahlia Malkhi, and Jean-Philippe Martin. Revisiting fast practical byzantine fault tolerance: Thelma, velma, and zelma. *arXiv preprint arXiv:1801.10022*, 2018.

[2] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *IEEE S&P*, pages 106–118. IEEE, 2020.

[3] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 331–341, 2021.

[4] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018.

[5] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[6] Brendan Kobayashi Chou, Andrew Lewis-Pye, and Patrick O'Grady. Minimmit: Fast finality with even faster blocks. *arXiv preprint arXiv:2508.10862*, 2025.

[7] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. Upright cluster services. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 277–290, 2009.

[8] Isaac Doidge, Raghavendra Ramesh, Nibesh Shrestha, and Joshua Tobkin. Moonshot: Optimizing block period and commit latency in chain-based rotating leader bft. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 470–482. IEEE, 2024.

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

[10] Bruno França, Denis Kolegov, Igor Konnov, and Grzegorz Prusak. Chonkybft: Consensus protocol of zksync. *arXiv preprint arXiv:2503.15380*, 2025.

[11] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. In *Proceedings of the 5th European conference on Computer systems*, pages 363–376, 2010.

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

[13] Quentin Kniep, Jakub Sliwinski, and Roger Wattenhofer. Solana alpenglow consensus. `https://drive.google.com/file/d/1Rlr3PdHsBmPahOInP6-Pl0bMzdayltdV`.

[14] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva. *ACM Transactions on Computer Systems*, 27(4):1–39, 2009.

[15] J-P Martin and Lorenzo Alvisi. Fast byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006.

[16] Victor Shoup. Sing a song of simplex. In *38th International Symposium on Distributed Computing (DISC 2024)*, pages 37–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[17] Victor Shoup, Jakub Sliwinski, and Yann Vonlanthen. Kudzu: Fast and Simple High-Throughput BFT. In *39th International Symposium on Distributed Computing (DISC 2025)*, volume 356, pages 42:1–42:19, 2025.

[18] Nibesh Shrestha, Aniket Kate, and Kartik Nayak. Hydrangea: Optimistic two-round partial synchrony with improved fault resilience. *Cryptology ePrint Archive*, 2025.