Hydrangea: Optimistic Two-Round Partial Synchrony

Nibesh Shrestha[1], Aniket Kate[2], and Kartik Nayak[3]

[1]Supra Research, n.shrestha@supra.com
[2]Supra Research/Purdue University, aniket@purdue.edu
[3]Duke University/Espresso Systems*, kartik@cs.duke.edu

Abstract

We introduce Hydrangea, a partially synchronous Byzantine fault-tolerant state machine replication protocol that offers strong fault tolerance and achieves a fast, two-round commit in optimistic scenarios. Specifically, for a system of $n = 3f + 2c + k + 1$ parties, Hydrangea achieves an optimistic good-case latency[1] of two rounds when the number of faulty parties (Byzantine or crash) is at most $p = \lfloor \frac{c+k}{2} \rfloor$ for a parameter $k \geq 0$. In more adversarial settings with up to f Byzantine faults and c crash faults, Hydrangea obtains a good-case latency of three rounds. Furthermore, we prove a matching lower bound: no protocol can achieve two-round optimistic commit under this fault model if $p > \lfloor \frac{c+k+2}{2} \rfloor$.

1 Introduction

Byzantine fault-tolerant state machine replication (BFT SMR) protocols [17, 20, 21], also referred to as consensus protocol, form the foundational backbone of blockchain systems. The primary objective of a BFT SMR protocol is to ensure agreement on a sequence of values (i.e., a log) among a group of distributed parties, even in the presence of faulty or malicious participants. To achieve this, the protocol must satisfy two essential properties: safety and liveness. Safety ensures consistency across honest parties, stipulating that no two non-faulty parties commit different values at the same position in the log. Liveness, on the other hand, guarantees progress in the system, requiring that non-faulty parties continue to append valid values to the log over time.

Most blockchain protocols today operate under the partially synchronous network model [10], in which the network is assumed to become synchronous after some unknown Global Stabilization Time (GST). In this model, consensus protocols typically rely on a "leader" (that may change over time) to drive progress. These protocols ensure safety unconditionally regardless of network conditions, but guarantee liveness only once the network remains synchronous for a sufficiently long period after GST.

A key performance metric in SMR protocols is the good-case latency[1], i.e., the number of rounds needed to commit a transaction under favorable conditions. When defined as the time from when the leader proposes a block to when it is committed by all honest parties, state-of-the-art protocols such as PBFT [7], Tendermint [6], Simplex [8, 23], and Moonshot [9] achieve a good-case latency of three rounds. These protocols tolerate up to $f < n/3$ Byzantine faults, and this combination of latency and resilience is known to be optimal under the partially synchronous model [3, 10].

Can we improve latency by reducing the number of Byzantine faults we tolerate? This question has been explored in a line of work including FaB [18], SBFT [12], Kudzu [24], and Alpenglow [13]. Among these, the state-of-the-art, Kudzu, considers a system of $n = 3f + 2p + 1$ parties, where n is the total number of parties, f is the maximum number of tolerated Byzantine faults, and p is a tunable parameter. Kudzu

*Kartik Nayak is an advisor at Espresso Systems.

[1]Informally, good-case latency refers to the number of rounds required to reach a decision when the broadcaster or leader is honest and the network is synchronous [10].

always guarantees a good-case latency of three rounds (i.e., slow commit). However, when up to $p \leq f$ parties are faulty, the protocol achieves fast commit in just two rounds—thereby exhibiting an *optimistic good-case latency*[2]. For example, when $f = p < n/5$ (i.e., less than 20% of parties are faulty), the protocol can achieve a fast commit as long as at least $n - p$ (i.e., more than 80%) parties behave correctly. However, this gain in latency comes at the cost of fault tolerance—in this setting, the protocol tolerates fewer than $n/5$ faults overall.

In contrast, another line of work explores protocols operating under the setting $n = 3f + 2c + 1$, where f and c denote the bounds on Byzantine and crash faults, respectively. These protocols offer higher resilience to faults, but do not support optimistic two-round commit.

> *Can we support optimistic two-round latency while still achieving high resilience towards Byzantine and crash faults?*

This is the key question we explore in this paper, and we answer it affirmatively. We present Hydrangea, a partially synchronous protocol with $n = 3f + 2c + k + 1$ parties, where $k \geq 0$ is a tunable parameter. The protocol achieves an *optimistic good-case latency* of two rounds when up to $p = \lfloor \frac{c+k}{2} \rfloor$ parties are faulty (either Byzantine or crash) i.e., when at least $n - p$ parties behave correctly. Otherwise, Hydrangea guarantees a good-case latency of three rounds while simultaneously tolerating up to f Byzantine faults and c crash faults. In particular, we obtain the following result:

Theorem 1. *There exists an authenticated, partially synchronous Byzantine broadcast protocol for*

$$n = 3f + 2c + k + 1, \quad \textit{where} \quad \begin{cases} 0 \leq k \leq 2f + c - 4 & \textit{if } c \textit{ is even,} \\ 0 \leq k \leq 2f + c - 3 & \textit{if } c \textit{ is odd,} \end{cases}$$

that simultaneously satisfies the following properties:

(i) an optimistic good-case latency of 2 rounds while tolerating $p = \lfloor \frac{c+k}{2} \rfloor$ faults,

(ii) a good-case latency of 3 rounds while tolerating f Byzantine faults and c crash faults.

As an example, when $f = 10\%$ and $c = 34\%$, our protocol obtains a fast commit when the number of correct parties is 83%, but at the same time, it tolerates up to 10% Byzantine faults and up to 34% crash faults. Prior works that improve latency cannot obtain such a combination of 44% faults.

We note that f and c are independent parameters for Hydrangea, and considering any combination of these can be useful in practice. In comparison, for prior work such as Kudzu [24], given that the protocol tolerates only f Byzantine faults, it is only meaningful to consider $p \leq f$. Moreover, since our fault model considers a combination of Byzantine and crash faults, which is weaker than considering Byzantine faults only, this allows us to use quorums that are smaller than $\frac{2n}{3}$ when $c + k \geq 1$.

A lower bound on resilience for optimistic good-case latency. We also establish a lower bound on the resilience for achieving optimistic good-case latency. Specifically, we show that if more than $p = \lfloor \frac{c+k+2}{2} \rfloor$ are faulty, then no protocol can achieve an optimistic good-case latency of two rounds under $n = 3f + 2c + k + 1$ while simultaneously tolerating f Byzantine faults and c crash faults. In particular, we establish the following result:

Theorem 2. *There exists no authenticated, partially synchronous Byzantine broadcast protocol that simultaneously tolerates f Byzantine faults and c crash faults under the constraint $n = 3f + 2c + k + 1$, for*

$$\begin{cases} 0 \leq k \leq 2f + c - 4 & \textit{if } c \textit{ is even,} \\ 0 \leq k \leq 2f + c - 3 & \textit{if } c \textit{ is odd,} \end{cases}$$

and achieves an optimistic good-case latency of two rounds while tolerating more than $p = \lfloor \frac{c+k+2}{2} \rfloor$ faulty parties (Byzantine or crash).

[2]Informally, optimistic good-case latency refers to the good-case latency achievable under favorable conditions, such as when more parties behave honestly.

Observe that when $c = k = 0$, we have $n = 3f + 1$ and $p = 1$. In this setting, our lower bound implies that it is possible to achieve a two-round optimistic commit while tolerating at least one fault. This suggests that the fast path commit rule in Zyzzyva [14], which requires $n = 3f + 1$ but does not commit in two rounds under a single fault, is not optimal.

Towards a matching upper bound. Our upper and lower bound results do not match, leaving a small gap. However, this gap can be bridged by adapting techniques from the work of Abraham et al. [3], who present a partially synchronous Byzantine broadcast protocol that achieves two-round good-case latency while tolerating f Byzantine faults under $n \geq 5f - 1$. At a high level, their protocol differentiates between types of messages—particularly those sent by the leader. When an equivocation is detected, the leader is definitively identified as faulty. The protocol then waits for one additional message from an honest party, thereby enabling progress with a smaller honest majority and achieving the stated resilience bound.

A similar approach can be incorporated into our setting to design an upper bound protocol that tolerates up to $p = \lfloor \frac{c+k+2}{2} \rfloor$ faults during optimistic commit, thereby closing the gap between the lower and upper bounds in our model. However, Hydrangea opts for slightly reduced optimistic resilience of $p = \lfloor \frac{c+k}{2} \rfloor$ in exchange for a significantly simpler protocol design.

1.1 Technical Overview

For simplicity, in this overview, we will consider parties agreeing on a single value, although our protocol description considers the more general version for committing a sequence of values.

Protocols in partial synchrony. Partially synchronous consensus protocols typically tolerate $< n/3$ Byzantine faults. At a high level, these protocols run in a sequence of views, where each view v has a designated leader L_v. The leader proposes a value, say B, and the parties hope to *certify* and *commit* the value proposed by the leader. Typically, this is performed using two phases of voting. The first phase requires parties to vote on up to one proposal from the leader; votes from a quorum of $\frac{2n}{3}$ parties forms a *certificate* $C_v(B)$ on value B. When a party collects such a certificate, it *locks* onto this value and sends a second phase of votes. If a party collects $\frac{2n}{3}$ such second round of votes for a value B, then it can commit B.

Intuitively, the first round of votes guarantees uniqueness (non-equivocation) within the view, i.e, if there is a certificate $C_v(B)$, then there cannot exist a certificate for $C_v(B')$ where $B \neq B'$. This is because any two quorums of parties voting for B and B' in view v intersect in at least one honest party, and this party will only vote for up to one value. The second round of votes guarantees safety and liveness across views. In particular, if a party has committed B, then it must be the case that at least $\frac{2n}{3}$ parties, and thus at least $\frac{n}{3}$ honest parties (say the quorum is denoted by Q1), would have locked on to B. Thus, in the next view $v + 1$, if a leader L_{v+1} learns about the locks of at least $\frac{2n}{3}$ parties (say quorum Q2), then this quorum will intersect with at least one of the locked honest parties (i.e., $\mid \text{Q1} \cap \text{Q2} \mid \geq 1$). This ensures that in view $v + 1$, an honest leader can propose B and a Byzantine leader cannot propose any value $\neq B$.

Why does a single phase of votes not suffice to commit? Performing two rounds of votes will incur at least three rounds of communication — one round for proposing a value and two rounds for each of the phases of votes. To improve latency, it may be better to use only one round of votes. What can go wrong if a party commits as soon as it receives a certificate based on $\frac{2n}{3}$ votes? With such a protocol, in the presence of $< n/3$ Byzantine faults, it turns out that we may not have safety across views. In particular, suppose party P_i is the only party that receives such a certificate $C_v(B)$ and it commits the value B; other parties may not commit any value, perhaps because the network is asynchronous and parties do not receive sufficiently many votes. In the next view $v + 1$, the set of $\frac{2n}{3}$ parties reporting to the leader L_{v+1} may not have locked onto any value; in particular, this may not include party P_i. Note that the leader cannot wait for more than $\frac{2n}{3}$ messages since $< n/3$ parties can be Byzantine. Thus, an honest leader L_{v+1} can propose $B' \neq B$, causing an equivocating value B' to be committed by other honest parties (or a lack of liveness if honest parties choose not to vote for B'). To improve latency, our solution relies on two key observations.

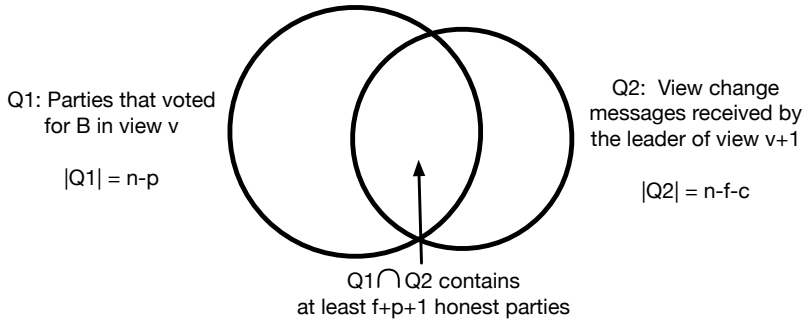

Figure 1: Ensuring safety and liveness across views.

Key observation 1. Observe that in the above example, while no party other than P_i has locked on or committed B, $\frac{2n}{3}$ have voted for B. Perhaps this information can be sent to leader L_{v+1} in addition to the locks. The leader can rely on this additional information in making a proposal in view $v+1$. In particular, in the example described above, since the leader did not obtain any locks, if it observes that sufficiently many parties have voted for B in view v, it can choose to re-propose B in view $v+1$.

While this works for the example described above, it does not work under all situations when the number of Byzantine parties is $< n/3$. Consider the following example. The leader in view v proposes two equivocating values B and B'. Suppose a quorum Q1 of $\frac{2n}{3}$ parties (which include all Byzantine parties) vote for B and the remaining $\frac{n}{3}$ honest parties (say quorum Q2) vote for B'. Again, some party P_i (in Q1) may obtain a certificate $C_v(B)$ and commit B, whereas the rest of the parties, do not receive any of the votes (perhaps due to asynchrony) and perform a view change. In view $v+1$, L_{v+1} receives messages from a quorum Q3 of $\frac{2n}{3}$ parties. In the worst case, Q1 and Q3 intersect in only one honest party (i.e., if Q3 comprises of Q2, all of the Byzantine parties, and one honest party from Q1). Given $\frac{n}{3}$ parties from Q2 voted for B', if Byzantine parties report to L_{v+1} that they voted for B' too (or they report that they did not vote for any value), then L_{v+1} observes up to $\frac{2n}{3}$ votes for B' and only 1 vote for B. Given the abundance of support for B', an honest L_{v+1} can plausibly re-propose B' even if P_i committed B. If sufficiently many parties choose to vote for B' in view $v+1$, we will have a safety violation; if not, the protocol may not have liveness.

Key observation 2. Observe that the above attack requires the Byzantine parties to equivocate: in view v, they vote for B and during view change they report that they either voted for B' or did not vote. What if not all of the faulty parties we tolerate are Byzantine?

In Hydrangea, compared to traditional partially synchronous protocols that support only three round commit latency, we consider a slightly weaker model combining Byzantine and crash faults. The resulting protocol with $n = 3f + 2c + k + 1$ provides two types of commits – a fast commit where parties commit within one round of votes while tolerating up to $p = \lfloor \frac{c+k}{2} \rfloor$ faults (Byzantine or crash), and a slow commit where parties require two rounds of votes and tolerates f Byzantine faults and c crash faults.

Let us understand how the view change progresses under this model. Recall that our goal is to allow parties to commit within one round of votes. When there are up to p faulty parties (Byzantine or crash), observe that a party P_i can obtain $n - p$ votes for a value B proposed in view v. This will constitute a fast commit for B. Suppose the set of votes is denoted by quorum Q1 (cf. Figure 1). Now, even if we have a view change, under the assumption of c crash faults and f Byzantine faults, the leader L_{v+1} receives at least $n - f - c$ responses (say quorum Q2) from parties where parties will indicate their current lock and the value they have voted for. In our example, no party other than P_i has locked or committed any value in view v, but quorum Q1 has voted for B. Observe that based on the intersection between Q1 and Q2, at least $f + p + 1$ responses would indicate having voted for B in view v and no other value will receive that many votes in view v, and thus, the leader can (and with appropriate proposal validation checks by the parties in view $v+1$, is forced) to propose B in view $v+1$. This ensures both safety and liveness across views.

We note that, for simplicity, in this overview, we made several simplifying assumptions. First, we

considered agreement on a single value, whereas our protocol considers agreement on a sequence of values. Thus, in subsequent sections, a block will be denoted by B_m where m denotes its position in the log. Second, the overview only considered specific scenarios where no other parties are locked in view v, and only a single view change to view $v+1$ to explain the intuition. The protocol considers all of the general cases. Finally, we did not discuss the quorum sizes related to slow commits; as we will see, our protocol requires quorums that are smaller than $\frac{2n}{3}$ and thus more efficient compared to traditional partial synchrony protocols.

2 Preliminaries

We consider a system $\mathcal{P} := P_1, \ldots, P_n$ consisting of $n = 3f + 2c + k + 1$ parties, where up to f parties may be Byzantine and up to c parties may suffer crash faults. The parameter k satisfies $0 \leq k \leq 2f + c - 4$ when c is even, and $0 \leq k \leq 2f + c - 3$ when c is odd. Byzantine parties can behave arbitrarily, while crashed parties cease communication after crashing. A party that remains fault-free throughout the execution and follows the protocol correctly is referred to as *honest*.

We assume the partial synchrony model introduced by Dwork et al. [10]. In this model, the system begins in an asynchronous state during which the adversary can arbitrarily delay messages between honest parties. After an unknown point in time, known as the *Global Stabilization Time* (GST), the network becomes partially synchronous: all messages sent by honest parties are guaranteed to be delivered within a fixed delay Δ. We use δ to denote the actual (and possibly variable) message transmission delays, and note that $\delta \leq \Delta$ after GST. Furthermore, we assume that parties' local clocks exhibit *no drift* but may have *arbitrary skew*.

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.

2.1 Property Definitions

Definition 1 (Partially Synchronous Byzantine Broadcast [3])**.** *A partially synchronous Byzantine broadcast protocol provides the following properties.*

- ***Agreement.*** *If two honest parties commit values m and m' respectively, then $m = m'$. .*
- ***Validity.*** *If the designated broadcaster is honest and GST$= 0$, then all honest parties commit the broadcaster's value. .*
- ***Termination.*** *All honest parties commit and terminate after GST.*

Definition 2 (Good-case Latency [3])**.** *A partially synchronous Byzantine broadcast protocol has good-case latency of R rounds under partial synchrony, if all honest parties commit within R synchronous rounds (over all executions and adversarial strategies), given the designated broadcaster is honest and $GST = 0$.*

Definition 3 (Optimistic Good-case Latency)**.** *A partially synchronous Byzantine broadcast protocol has an optimistic good-case latency of R rounds if, under partial synchrony, all honest parties commit within R synchronous rounds provided that: (i) the designated broadcaster is honest, (ii) at least n_o parties behave honestly for some parameter $n_o > n - f - c$, and (iii) GST$= 0$.*

Our upper bound protocol achieves an optimistic good-case latency of 2 rounds when the designated broadcaster is honest, $n_o = n - p$ parties behave honestly for $p = \lfloor \frac{c+k}{2} \rfloor$, and GST $= 0$.

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.

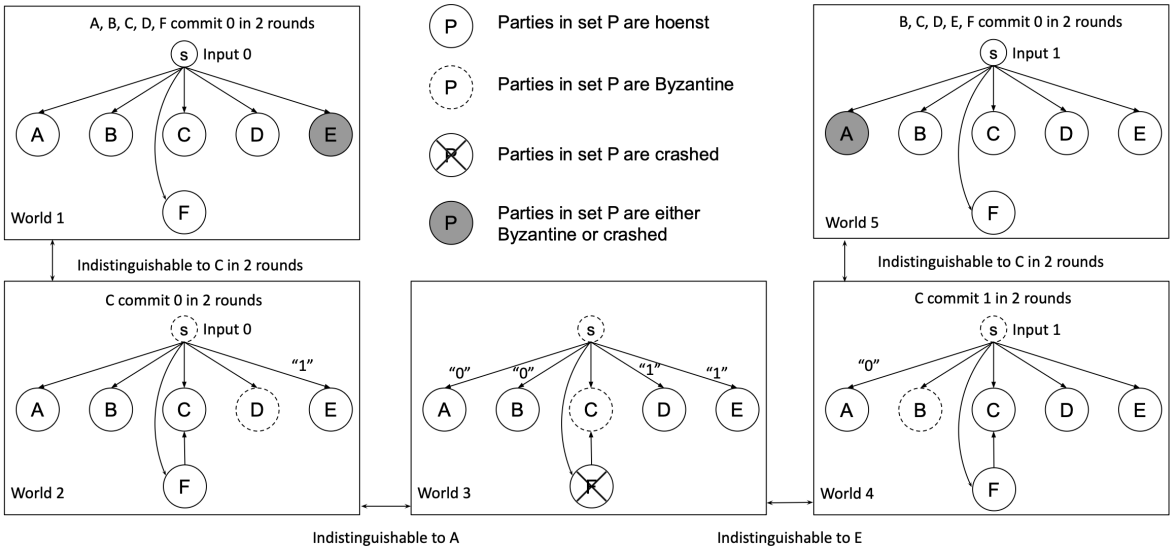

Figure 2: Resilience lower bound for optimistic good-case latency of psync Byzantine broadcast under $n = 3f + 2c + k + 1$.

Definition 4 (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 A Lower Bound on Resilience for Optimistic Good-case Latency

In this section, we establish a lower bound on the resilience required to achieve optimistic good-case latency in partially synchronous Byzantine broadcast protocols, under the constraint $n = 3f + 2c + k + 1$, where $0 \leq k \leq 2f + c - 4$ if c is even, and $0 \leq k \leq 2f + c - 3$ if c is odd. The proof of the lower bound is illustrated in Figure 2.

Intuition. The key intuition behind the lower bound is the ability of a set of $f + c$ parties to split the others into two groups of $f + p$ (for $p = \lfloor \frac{c+k+2}{2} \rfloor$) parties that disagree (cf. Figure 2). The $f + c$ parties include the sender s, a set C of size $f - 1$ and the set F of size c and the two groups include sets $A \cup B$ and $D \cup E$ (World 3). The disagreement occurs despite the ability of the two sets $A \cup B$ and $D \cup E$ to communicate with each other. This is because parties in set A (of size p) need to be able to commit some value when it is unable to receive messages from the $f + c$ parties. This is the situation in a world (World 2) where it does not receive messages from C and F, and a different set D and the sender are Byzantine. In this world, despite E receiving an input 1 from the sender (and Byzantine D stating that it received 1 from the sender), we show that A will still output 0. This is to agree with the set C, who in the same world (World 2) can output 0 in two-rounds based on messages it receives from all parties except from the ones in set E. This can happen if the Byzantine set D behaves with C as if it received 0 from the sender. In this situation, C outputs 0 to be consistent with a world where all parties are honest except E (containing parties that are either crashed or Byzantine), and the sender sends 0 (World 1). A similar argument applies for why the set E will output 1 in World 3 causing a disagreement with set A.

Theorem 3. *There does not exist an authenticated, partially synchronous Byzantine broadcast protocol that tolerates f Byzantine faults and c crash faults simultaneously under $n = 3f + 2c + k + 1$, for*

$$\begin{cases} 0 \leq k \leq 2f + c - 4 & \text{if } c \text{ is even,} \\ 0 \leq k \leq 2f + c - 3 & \text{if } c \text{ is odd,} \end{cases}$$

and achieves an optimistic good-case latency of two rounds while tolerating more than $p = \lfloor \frac{c+k+2}{2} \rfloor$ faults.

Proof. Suppose, for the sake of contradiction, that there exists a protocol which achieves an optimistic good-case latency of two rounds under the constraint $n = 3f+2c+k+1$ even when tolerating $p+1$ i.e., $\lfloor \frac{c+k+2}{2} \rfloor + 1$ faults. Observe that for the given range of k, we have $p+1 \leq f+c$. We will construct a sequence of executions (worlds) and use an indistinguishability argument to demonstrate a violation of the agreement property of such a protocol. To construct our argument, we divide the n parties into the designated broadcaster s and six mutually disjoint sets: A, B, C, D, E, and F, such that $|B| = |C| = |D| = f - 1$ and $|A| = |E| = \lfloor \frac{c+k+4}{2} \rfloor$. When $c + k$ is odd, we set $|F| = c$; otherwise we set $|F| = c - 1$. This ensures that the total number of parties satisfies $1 + |A| + |B| + |C| + |D| + |E| + |F| = n$.

World 1 **(W**1**).** The network is synchronous, i.e., GST $= 0$. The designated broadcaster s is honest, while the parties in set E are fault (either Byzantine or crashed). Since $|E| = p+1$ and $p+1 \leq f+c$, it follows that $|E| \leq f + c$, thereby allowing all parties in E to be faulty (either Byzantine or crashed). The broadcaster sends the value 0 to all parties.

Since the broadcaster is honest and there are at most $p + 1$ faults, the optimistic good-case latency guarantee and the validity property ensure that parties in C commit 0 within 2 rounds, after they deliver all round-0 and round-1 messages. By termination, parties in A, B, D, and F commit 0.

World 5 **(W**5**).** The network is synchronous, i.e., GST $= 0$. The designated broadcaster s is honest, while the parties in set E are faulty (either Byzantine or crashed). Since $|A| = p + 1$ and $p + 1 \leq f + c$, it follows that $|A| \leq f+c$, thereby allowing all parties in A to be faulty (either Byzantine or crashed). The broadcaster sends the value 1 to all parties.

Since the broadcaster is honest and there are at most $p + 1$ faults, the optimistic good-case latency guarantee and the validity property ensure that parties in C commit 1 within 2 rounds, after they deliver all round-0 and round-1 messages. By termination, parties in B, D, E, and F commit 1.

World 2 **(W**2**).** The network remains asynchronous until after parties in sets A, C, and E commit—that is, GST occurs after parties in A, C, and E commit. The broadcaster and parties in set D are Byzantine. The broadcaster sends the value 0 to parties in sets A, B, C, and F and the value 1 to parties in E. It emulates the behavior of **W**1 when interacting with parties in A, B, C, and F, and the behavior of **W**5 when interacting with E during the first two rounds, and then remains silent thereafter. The Byzantine parties in D behave according to **W**1 when interacting with parties in C, and act as honest parties with input 1 from the broadcaster when communicating with all other parties. Furthermore:

- All messages from parties in C, except their round-0 messages, are delayed indefinitely until after GST.
- All messages from E to C are also delayed indefinitely until GST.
- Round-1 messages from F are delivered only to C; their delivery to all other parties is delayed until GST.

Claim 1. *The parties in C cannot distinguish between **W**1 and **W**2 prior to receiving round-2 messages.*

Proof. The round-0 messages sent by a party depend solely on its initial state. Therefore, the round-0 messages received by parties in A, B, C, and F, along with their senders, are identical in both **W**1 and **W**2. The broadcaster behaves identically in both worlds toward parties in A, B, C, and F during the first two rounds, leading these parties to send identical round-1 messages in both executions. Furthermore, the Byzantine parties in D behave identically in both **W**1 and **W**2 toward parties in C. In both worlds, the parties in C do not receive any messages from E prior to GST. As a result, the view of the protocol execution from the perspective of parties in C remains indistinguishable between **W**1 and **W**2 until they receive round-2 messages. ■

World 4 **(W**4**).** The network remains asynchronous until after parties in sets A, C, and E commit—that is, GST occurs after parties in A, C, and E commit. The broadcaster and parties in set B are Byzantine. The broadcaster sends the value 1 to parties in sets C, D, E and F and the value 1 to parties in A. It then emulates the behavior of **W**5 when interacting with parties in C D, E and F, and the behavior of **W**1 when interacting with A during the first two rounds, and then remains silent thereafter. The Byzantine parties in B behave according to **W**5 when interacting with parties in C, and act as honest parties with input 0 from the broadcaster when communicating with all other parties. Furthermore:

- All messages from parties in C, except their round-0 messages, are delayed indefinitely until after GST.

- All messages from A to C are also delayed indefinitely until GST.

- Round-1 messages from F are delivered only to C; their delivery to all other parties is delayed until GST.

Claim 2. *The parties in C cannot distinguish between **W**4 and **W**5 prior to receiving round-2 messages.*

Proof. The round-0 messages sent by a party depend solely on its initial state. Therefore, the round-0 messages received by parties in C, D, E, and F, along with their senders, are identical in both **W**4 and **W**5. The broadcaster behaves identically in both worlds toward parties in C, D, E, and F during the first two rounds, resulting in identical round-1 messages sent by these parties in the two executions. Furthermore, the Byzantine parties in B behave identically in both **W**4 and **W**5 toward parties in C. In both worlds, the parties in C do not receive any messages from A prior to GST. As a result, the view of the protocol execution from the perspective of parties in C remains indistinguishable between **W**1 and **W**2 until they receive round-2 messages. ■

World 3 **(W**3**).** The network remains asynchronous until after parties in A and E commits (i.e., GST occurs after parties in A and E commit).

The broadcaster and parties in C are Byzantine. The broadcaster sends 0 to parties in A, B and 1 to parties in D, E. Then the broadcaster behaves to parties in A and B similar to **W**2 and to parties in D and E similar to **W**4. The Byzantine parties in C sends round-0 messages to parties in A, B, D and E, but do not send any messages thereafter. The parties in F sends only round-0 messages and become crashed thereafter.

Claim 3. *The parties in A cannot distinguish between **W**2 and **W**3 prior to GST.*

Proof. The parties in A receive round-0 messages from parties in B, C, D, E, and F in both **W**2 and **W**3, and these messages are identical since they depend only on the parties' initial states. The broadcaster behaves identically in both **W**2 and **W**3 toward parties in A and B during the first two rounds. The Byzantine parties in C do not send any messages (beyond round-0) to parties in A in **W**3, which is identical to **W**2, where messages from C to A are indefinitely delayed until GST.

Similarly, the Byzantine parties in D in **W**2 behave identically to those in **W**3. The parties in E also behave identically in both executions, as their input from the broadcaster and their received messages are the same. Moreover, parties in A do not receive any messages from parties in F beyond round-0 messages in either execution.

Thus, the parties in A cannot distinguish between **W**2 and **W**3 prior to GST. ■

Claim 4. *The parties in E cannot distinguish between **W**3 and **W**4 prior to GST.*

Proof. The parties in E receive round-0 messages from parties in A, B, C, D, and F in both **W**3 and **W**4, and these messages are identical since they depend only on the parties' initial states. The broadcaster behaves identically in both **W**3 and **W**4 toward parties in D and E during the first two rounds. The Byzantine parties in C do not send any messages (beyond round-0) to parties in E in **W**3, which is identical to **W**4, where messages from C to E are indefinitely delayed until GST.

Similarly, the Byzantine parties in B in **W**4 behave identically to those in **W**3. The parties in A also behave identically in both executions, as their input from the broadcaster and their received messages are

the same. Moreover, parties in E do not receive any messages from parties in F beyond round-0 messages in either execution.

Thus, the parties in E cannot distinguish between **W**3 and **W**4 prior to GST. ■

By Claim 1, parties in C cannot distinguish between **W**1 and **W**2 prior to receiving round-2 messages. Thus, parties in C commit 0 in **W**2. By termination, parties in A and E also commit 0 in **W**2. Similarly, by Claim 2 parties in C cannot distinguish between **W**4 and **W**5 prior to receiving round-2 messages. Thus, parties in C commit 1 in **W**4. By termination, parties in A and E also commit 1 in **W**4.

By Claim 3, the parties in A cannot distinguish between **W**2 and **W**3 prior to GST. Moreover, there are at most f Byzantine and c crash faults in **W**3. Since, the parties in A committed 0 in **W**2, they will also commit 0 in **W**3. Similarly, by Claim 4, the parties in E cannot distinguish between **W**3 and **W**4 prior to GST. Since, the parties in E committed 1 in **W**4, they will also commit 1 in **W**3. Thus, agreement is violated. A contradiction. □

4 The Hydrangea Protocol

In this section, we present Hydrangea, a protocol that achieves an optimistic good-case latency of two rounds under the condition $n = 3f + 2c + k + 1$, provided that there are at most $p = \lfloor \frac{c+k}{2} \rfloor$ faulty parties (either Byzantine or crash). In the presence of up to f Byzantine faults and c crash faults, Hydrangea guarantees a good-case latency of three rounds.

4.1 Protocol Definitions

View-based execution. Our protocol progresses through a sequence of numbered *views*, starting with all parties in view 1 and advancing to higher views as execution continues. Each view v has a designated leader L_v responsible for proposing a new block. The leader for each view is rotated, independent of the progress made within that view.

Blocks and block format. Client requests are grouped into *blocks*, each of which references its predecessor, except for the genesis block, which has no predecessor. The position of a block in the chain is referred to as its *height*. A block B_m at height m is represented as $B_m := (b_m, H(B_{m-1}))$, where b_m is the block's content and $H(B_{m-1})$ is the hash of its predecessor block B_{m-1}. The genesis block has $\perp$ as its predecessor. A block B_m is considered *valid* if: (1) its predecessor is valid (or is $\perp$ when $m = 1$), and (2) the client requests it contains satisfy application-level validity conditions and are consistent with the sequence of requests in its ancestor blocks. Finally, we say that a block B_m *extends* a block B_l ($m \geq l$) if B_l is an ancestor of B_m. Note that a block trivially extends itself. Two blocks B_m and $B'_{m'}$ *equivocate* (or conflict) if they are distinct and neither extends the other.

Block certificate and weak block certificate. In our protocol, a party sends a signed vote message to indicate its acceptance of a block. A block certificate $\mathcal{C}_v(B_m)$ for view v consists of a $\lceil \frac{n+f+1}{2} \rceil$ distinct signed vote messages for B_m for v. We use $\mathcal{C}_v$ to denote a block certificate for view v when the specific block it certifies is not relevant to the context. Block certificates are ranked by views, i.e., $\mathcal{C}_v \leq \mathcal{C}_{v'}$ if $v \leq v'$.

Additionally, we define a weak block certificate $\mathcal{WC}_v(B_m)$ for view v as a collection of $f + p + 1$ distinct signed vote messages for block B_m in view v. Weak block certificates are also ranked by view number, with $\mathcal{WC}_v(B_m) \leq \mathcal{WC}_{v'}(B_h)$ if $v \leq v'$. To compare a block certificate $\mathcal{C}_v(B_m)$ and a weak block certificate $\mathcal{WC}_{v'}(B_h)$, we rank $\mathcal{C}_v(B_m)$ higher if $v \geq v'$, and otherwise $\mathcal{WC}_{v'}(B_h)$ ranks higher.

Timeout messages and timeout certificates. To ensure liveness, our protocol require parties to initiate a leader change if they observe no progress in their current view within a certain time threshold. Parties signal this by sending signed timeout messages for the view, which include their most recently voted block and the highest weak block certificate or the highest-ranked block certificate they have seen. A timeout certificate for view v, denoted $\mathcal{TC}_v$, is formed by collecting $n - f - c$ distinct signed timeout messages for view v.

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

Note that $\mathcal{PC}_v$ comprises block certificates, weak block certificates, and the highest votes included in the timeout messages (in the case where $\mathcal{PC}_v = \mathcal{TC}_v$). If the highest vote messages included in $\mathcal{PC}_v$ provide sufficient support for a common block, a weak block certificate can be constructed from them. The safe block is then selected as the block associated with the highest-ranked certificate—either a block certificate or a weak block certificate.

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

1. **Propose.** Upon entering view v, the leader L_v multicasts a proposal of the form $\langle \mathsf{propose}, B_m, \mathcal{PC}_{v-1}, v \rangle_{L_v}$, where B_m extends a block B_{m-1} that was proposed in a previous view $v' < v$, and B_{m-1} is considered a safe block according to the accompanying $\mathcal{PC}_{v-1}$. Specifically, $\mathcal{PC}_{v-1}$ is set to $\mathcal{C}_{v-1}(B_{m-1})$ if L_v entered view v upon receiving this certificate; otherwise, $\mathcal{PC}_{v-1}$ is set to $\mathcal{TC}_{v-1}$.
2. **Vote.** Upon receiving the first proposal $\langle \mathsf{propose}, B_m, \mathcal{PC}_{v-1}, v \rangle_{L_v}$ from leader L_v, such the following conditions hold: (i) $\mathsf{timeout\_view}_i < v$, (ii) B_{m-1} is a safe block according to $\mathcal{PC}_{v-1}$, and (iii) B_m extends B_{m-1}, then party P_i sets $\mathsf{hv}_i := \langle \mathsf{vote}, H(B_m), v \rangle_{P_i}$ and multicasts hv_i to all parties while still in view v. If B_{m-1} is deemed safe due to a weak block certificate $\mathcal{WC}_{v'}(B_{m-1})$ included in $\mathcal{PC}_{v-1}$, then $\mathsf{hwc}_i := \mathcal{WC}_{v'}(B_{m-1})$.
3. **Direct Pre-commit.** Upon receiving $\mathcal{C}_v(B_m)$ whilst in any view $v' \leq v$ if $\mathsf{timeout\_view}_i < v$, multicast $\langle \mathsf{commit}, H(B_m), v \rangle_{P_i}$.
4. **Indirect Pre-commit.** Upon receiving $\mathcal{C}_v(B_m)$ in any view, if a party has previously multicast a commit for any descendant of B_m and its $\mathsf{timeout\_view}_i < v$, then it multicasts $\langle \mathsf{commit}, H(B_m), v \rangle_{P_i}$ if it has not already done so.
5. **Timeout.** When $\mathsf{view\text{-}timer}_i$ expires, party P_i multicasts $\langle \mathsf{timeout}, v, \mathsf{h\text{-}cert}_i, \mathsf{hv}_i \rangle_{P_i}$ (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 $\langle \mathsf{timeout}, v', \mathsf{h\text{-}cert}_i, \mathsf{hv}_i \rangle_{P_i}$ 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.
6. **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 $\mathcal{C}_v(B_m)$ in any protocol message whilst having $\mathsf{lock}_i = \mathcal{C}_{v'}(B_{m'})$ such that $v > v'$, set lock_i to $\mathcal{C}_v(B_m)$.
2. **Direct Commit.** P_i directly commits B_m whilst in any view using one of the following rules:
   - **(Fast Commit.)** Upon receiving $n-p$ distinct $\langle \mathsf{vote}, H(B_m), v \rangle_*$ messages,
   - **(Slow Commit.)** Upon receiving $2f+c+1$ distinct $\langle \mathsf{commit}, H(B_m), v \rangle_*$ messages.
3. **Indirect Commit.** Upon directly committing B_m, commit all of its uncommitted ancestors.

Figure 3: Hydrangea: Optimistic Two-Round BFT SMR under $n = 3f + 2c + k + 1$

4.2 Protocol Details

Our protocol is outlined in Figure 3, with the key steps described below for view v.

Advance View and Timeout. Party P_i enters view v from some earlier view $v' < v$ upon receiving either a view $v-1$ block certificate or a view $v-1$ timeout certificate. In the former case, it updates its lock lock_i and then multicasts $\mathcal{C}_{v-1}$ to assist with view synchronization. In the latter case, P_i unicasts $\mathcal{TC}_{v-1}$ to L_v to ensure that L_v enters view v within Δ time of the first honest party doing so after GST. Finally, P_i enters view v, resets $\mathsf{view\text{-}timer}_i$ to 3Δ, and begins the countdown.

If the leader L_v is honest and the network is synchronous, then party P_i is expected to enter view $v+1$ within 3Δ of entering view v. If this does not happen, P_i assumes the leader has failed and multicasts $\langle \mathsf{timeout}, v, \mathsf{h\text{-}cert}_i, \mathsf{hv}_i \rangle_{P_i}$ to trigger a view change and ensure the protocol makes progress. Additionally, P_i performs the same action upon observing that any other honest party has initiated a view change for some view $v' \geq v$. Here, $\mathsf{h\text{-}cert}_i$ denotes the highest-ranked certificate (either a block certificate or a weak block certificate), and hv_i is the highest vote previously sent by P_i.

Propose. The leader L_v proposes a block B_m by multicasting a proposal of the form $\langle \mathsf{propose}, B_m, \mathcal{PC}_{v-1}, v \rangle_{L_v}$, where B_m extends a block B_{m-1} proposed in a prior view $v' < v$, and B_{m-1} is the safe block determined from $\mathcal{PC}_{v-1}$ using the rule defined above. Note that $\mathcal{PC}_{v-1}$ is set to $\mathcal{C}_{v-1}(B_{m-1})$ if L_v has received it; otherwise, it is set to $\mathcal{TC}_{v-1}$.

Vote. When a party P_i receives the first proposal $\langle \mathsf{propose}, B_m, \mathcal{PC}_{v-1}, v \rangle_{L_v}$ from L_v in view v, it proceeds as follows. If P_i has not yet sent a vote in view v or a $\mathsf{timeout}$ message for view v or higher, and if B_m extends the safe block B_{m-1} determined by $\mathcal{PC}_{v-1}$, then it updates its hv_i to $\langle \mathsf{vote}, H(B_m), v \rangle_{P_i}$ and multicasts hv_i to all parties. Furthermore, if B_{m-1} is considered safe due to a weak block certificate $\mathcal{WC}_{v'}(B_{m-1})$ contained in $\mathcal{PC}_{v-1}$, then P_i also sets $\mathsf{hwc}_i := \mathcal{WC}_{v'}(B_{m-1})$.

Pre-commit. Our protocol includes two pre-commit rules for sending explicit commit messages. Under the direct pre-commit rule, a party P_i sends $\langle \mathsf{commit}, H(B_m), v \rangle_{P_i}$ upon receiving $\mathcal{C}_v(B_m)$ while in any view $v' \leq v$, provided it has not sent a $\mathsf{timeout}$ message in view v or higher. Note that our protocol allows parties to jump to a higher view v' upon receiving $\mathcal{C}_{v'-1}$. Since the direct pre-commit rule requires that parties be in view $v'' \leq v$ in order to send a commit message for B_m, this may prevent a commit from being sent when a party jumps to a higher view. To ensure that blocks are still committed even when parties jump to higher views, our protocol includes an indirect pre-commit rule: a party P_i may send a commit message for block B_m upon receiving $\mathcal{C}_v(B_m)$ in any view if it has already sent a commit message for one of its descendants, provided it has not sent a $\mathsf{timeout}$ message for view v or higher.

Commit. Our protocol includes two direct commit rules: (i) the *fast commit rule*, and (ii) the *slow commit rule*. An honest party P_i directly commits a block B_m using whichever rule is triggered first during protocol execution. Specifically, P_i commits B_m via the fast commit rule when it receives $n-p$ distinct vote messages for B_m in view v. Alternatively, P_i directly commits B_m via the slow commit rule when it receives $2f+c+1$ distinct messages of the form $\langle \mathsf{commit}, H(B_m), v \rangle_*$. Note that when P_i directly commits B_m, it also *indirectly commits* all of B_m's uncommitted ancestor blocks.

Intuition behind the safe block selection in $\mathcal{PC}_v$. This intuition is the crux of the safety and liveness of our protocol; it generalizes the description in Section 1.1 to a sequence of blocks. In our protocol, once a block certificate for block B_ℓ exists in some view v', no conflicting block certificate can exist for the same view due to standard quorum intersection guarantees. Furthermore, an equivocating block cannot be committed via the fast commit rule in view v'. This follows from the fact that certifying B_ℓ requires votes from a quorum $\mathcal{Q}$ of at least $\lceil \frac{n-f+1}{2} \rceil$ honest parties. For a conflicting block $B'_{\ell'}$ proposed in the same view v' to be fast-committed, it must receive votes from a set $\mathcal{Q}'$ of at least $n-p-f$ honest parties. However, since $|\mathcal{Q} \cap \mathcal{Q}'| > 1$, at least one honest party would need to vote for both B_ℓ and $B'_{\ell'}$, which is disallowed by protocol rules. Therefore, the only block that could have possibly been committed by any honest party is B_ℓ. Consequently, we prioritize a block certificate (when available) in the same or higher view when selecting the safe block in the progress certificate.

On the other hand, suppose a block B_h proposed in view v' is committed via the fast commit rule. Then there exists a set $\mathcal{S}$ of at least $n-f-p$ honest parties that voted for B_h in view v'. Since honest parties include their highest vote and any weak block certificate they observe in subsequent $\mathsf{timeout}$ messages, and

a $\mathcal{TC}_{v''}$ (for some $v'' \geq v'$) consists of at least $n - f - c$ such messages, the intersection between the senders of these timeout messages and the set $\mathcal{S}$ contains at least $f + p + 1$ honest parties. By protocol design, we ensure that block certificate for a conflicting block can exist in any view $v'' \geq v'$. Consequently, $\mathcal{PC}_v$ must contain $\mathcal{WC}_{v'}(B_h)$ or a higher-ranked certificate.

Furthermore, no equivocating block proposed in view v' or earlier can have gathered more than $f + p$ votes, since honest parties vote only once per view. Although a block certificate $\mathcal{C}_{v^*}(B_\ell)$ for some view $v^* < v'$ may also be present in $\mathcal{PC}_v$, the presence of at least $f + 1$ votes for B_h in view v' implies one of the following must hold: either B_h extends B_ℓ, or B_ℓ was not committed. Our protocol ensures that if a block B_ℓ is committed (via either commit rule), then all honest parties will only vote for blocks extending B_ℓ in subsequent views. Therefore, if B_ℓ were committed, B_h must extend it. Hence, it is safe to ignore any block certificate for view $v'' < v'$, and the only safe block to extend in $\mathcal{PC}_v$ is B_h or one of its descendants.

Communication complexity. Hydrangea requires parties to include their lock or hwc and their highest vote in timeout messages. Consequently, timeout certificates—and any proposal that uses a timeout certificate as its progress certificate—are naturally of size $O(n)$. This results in an $O(n^2)$ communication cost for the proposal action in each view.

Similarly, the all-to-all multicasting of $O(1)$-sized timeout messages (when threshold signatures are employed), vote messages, commit messages, and block certificates, along with the forwarding of timeout certificates to the next leader, each incurs $O(n^2)$ communication. Therefore, the overall communication complexity of our protocol is $O(n^2)$ per view when threshold signatures are used.

4.3 Security Analysis

Claim 5. *If $\mathcal{C}_v(B_m)$ and $\mathcal{C}_v(B_\ell)$ exist, then $B_m = B_\ell$.*

Proof. Suppose, for the sake of contradiction, that two conflicting certificates $\mathcal{C}_v(B_m)$ and $\mathcal{C}_v(B_\ell)$ exist for view v, where $B_m \neq B_\ell$. Each certificate must contain votes from at least $\lceil \frac{n+f+1}{2} \rceil$ parties in view v. Therefore, the overlap between the two voting sets is at least $\lceil \frac{n+f+1}{2} \rceil + \lceil \frac{n+f+1}{2} \rceil - n \geq f + 1$, implying at least one honest party must have voted for both B_m and B_ℓ in the same view, which contradicts the condition that honest parties vote for at most one block per view. □

Claim 6. *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 7. *If a block B_m proposed in view v is directly committed using the fast commit rule and a corresponding $\mathcal{TC}_v$ exists, then v must be the highest view in $\mathcal{TC}_v$ with at least $f+p+1$ votes for some block—and that block must be B_m. Furthermore, no other block in view v has more than $f + p$ votes in $\mathcal{TC}_v$.*

Proof. For a block B_m to be directly committed using the fast commit rule, it must receive at least $n - p$ votes in view v, which includes 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 6, $\mathcal{Q}$ and $\mathcal{Q}'$ intersects in at least $f + p + 1$ honest parties. Thus, there are at least $f + p + 1$ honest parties in $\mathcal{TC}_v$ that must have voted for B_m in view v. Since v is the highest view represented in $\mathcal{TC}_v$, and no other block in view v could have received more than the remaining $n - f - c - (f + p + 1) = f + p$ votes, B_m is the unique block with sufficient support. The claim follows. □

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

Proof. Suppose an honest party P_i directly commits block B_m proposed in view v using the fast commit rule, having received at least $n - p$ vote messages for B_m. This ensures that a set $\mathcal{Q}$ of at least $n - f - p$ honest parties voted for B_m 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 5, no conflicting block certificate can exist in the same view. Hence, $\mathcal{PC}_v = \mathcal{C}_v(B_m)$, and B_m 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 $\mathcal{C}_v(B_m)$. In this case, by Claim 7, B_m remains the only block that receives $f + p + 1$ votes in view v. Since v is the highest view in $\mathcal{PC}_v$, only B_m 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 $B_{m'}$ does not extend B_m. Note that $\mathcal{PC}_{v'}$ could either be $\mathcal{C}_{v'}(B_{m'})$ or $\mathcal{TC}_{v'}$. First, consider the case where $\mathcal{PC}_{v'} = \mathcal{C}_{v'}(B_{m'})$. For this certificate to exist, at least $\lceil \frac{n-f+1}{2} \rceil$ honest parties must have voted for $B_{m'}$ in view v'. However, by assumption, in every intermediate view $v \leq v'' < v'$, the safe block determined by $\mathcal{PC}_{v''}$ extends B_m. Since honest parties vote only for blocks that extend the safe block from the previous view, they would not have voted for $B_{m'}$ in view v' unless it also extended B_m. This contradicts the assumption that $B_{m'}$ does not extend B_m. 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 6, $\mathcal{Q}$ 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_m. 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_m. 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 $\mathcal{WC}_v(B_m)$. If instead they voted in a higher view $v'' > v$, they would either update their hwc to $\mathcal{WC}_v(B_m)$ (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_m. 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_m. Moreover, since at most $f + p$ high votes from views v or earlier in $\mathcal{TC}_{v'}$ can support a conflicting block (i.e., one that does not extend B_m), the safe block computed from $\mathcal{TC}_{v'}$ must necessarily extend B_m. This contradicts the assumption that the safe block determined by $\mathcal{PC}_{v'}$ does not extend B_m. □

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

Proof. Suppose an honest party P_i directly commits block B_m proposed in view v using the slow commit rule, having received at least $2f + c + 1$ commit messages for B_m. This implies that a set $\mathcal{Q}$ of at least $f + c + 1$ honest parties must have received $\mathcal{C}_v(B_m)$ before sending a timeout message for view v or exiting the view. Consequently, the progress certificate $\mathcal{PC}_v$ must be the block certificate $\mathcal{C}_v(B_m)$, 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 5, no conflicting block certificate can exist in view v. Hence, $\mathcal{PC}_v = \mathcal{C}_v(B_m)$, and B_m 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 $B_{m'}$ does not extend B_m. Note that $\mathcal{PC}_{v'}$ could either be a block certificate $\mathcal{C}_{v'}(B_{m'})$ or a timeout certificate $\mathcal{TC}_{v'}$. First, consider the case where $\mathcal{PC}_{v'} = \mathcal{C}_{v'}(B_{m'})$. For such a certificate to exist, at least $\lceil \frac{n-f+1}{2} \rceil$ honest parties must have voted for $B_{m'}$ 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_m, and honest parties vote only for blocks that extend the safe block of the previous view. Therefore, honest parties would not have voted for $B_{m'}$ in view v' if it did not extend B_m. This leads to a contradiction. This implies that no block certificate for a conflicting block (i.e., one that does not extend B_m) 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_m. 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_m. As a result, the weak block certificate for a block from view v^* must also extend B_m. Furthermore, if $v^* = v'$, then the weak block certificate must still extend B_m, since the safe block identified by $\mathcal{PC}_{v^*-1}$ also extends B_m. Alternatively, if $v^* \leq v$, then $\mathcal{PC}_{v'}$ must include either the block certificate $\mathcal{C}_v(B_m)$ 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_m. Therefore, in either case, the safe block determined from $\mathcal{PC}_{v'}$ must extend B_m, 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_m proposed in view v, and $B_{m'}$ is the safe block determined by $\mathcal{PC}_{v'}$ for some view $v' \geq v$, then $B_{m'}$ must extend B_m.*

Theorem 4 (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_m and B'_m, then $B_m = B'_m$. Suppose, for the sake of contradiction, that P_i and P_j commit B_m and B'_m but $B_m \neq B'_m$. 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_m with $\ell \geq m$, and P_j committed block B_o in view v' such that B_o extends B'_m with $o \geq m$. 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, $B_m = B'_m$. □

Claim 8. *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 9. *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 8, 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 9 and the requirement that timeout certificates be constructed from $n - f - p$ 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 10. *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 10, 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 10, 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 11. *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 $\mathcal{C}_v(B_m)$ for some block B_m proposed by L_v by time $t + 3\Delta$.*

Proof. By Claim 5, only one block can be certified in a given view. Thus, if $\mathcal{C}_v(B_m)$ exists, any party that receives a view-v block certificate must receive $\mathcal{C}_v(B_m)$. Additionally, by Claim 10 and Claim 11, 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 8, some honest party must have already entered $v + 1$ via $\mathcal{C}_v(B_m)$. By the view advancement rule, this party would have multicast $\mathcal{C}_v(B_m)$, so all honest parties receive this certificate by $t+3\Delta$, completing the proof. Alternatively, if no honest party receives $\mathcal{C}_v(B_m)$ before $t + 2\Delta$, then all honest parties will enter v before $t + 2\Delta$.

Since L_v is honest, it will create a proposal that extends the safe block determined by $\mathcal{PC}_{v-1}$ by time $t+\Delta$, and all honest parties will receive this proposal before $t+2\Delta$. Moreover, they cannot have $\mathsf{timeout\_view}_i \geq v$ by this time. Thus, by the voting rules, they will all vote for the proposed block, enabling all honest parties to construct $\mathcal{C}_v(B_m)$ before $t + 3\Delta$. □

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_m proposed by L_v by time $t + 4\Delta$.*

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

Otherwise, suppose for contradiction that at least one honest party, say P_i, fails to send $\langle \mathsf{commit}, H(B_m), v\rangle_*$ by $t + 3\Delta$. However, by Claim 9, no honest party's view timer can expire before this time, so P_i must have $\mathsf{timeout\_view}_i < v$ upon receiving $\mathcal{C}_v(B_m)$.

Thus, by the pre-commit rules, upon receiving $\mathcal{C}_v(B_m)$, P_i must neither be in view v or lower nor have previously multicast a commit message for any descendant of B_m. Let v' be the view P_i is in upon receiving $\mathcal{C}_v(B_m)$. 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_m. However, because $v' > v + 1$ and the only way to transition between these views (before $t + 3\Delta$) is via block certificates, and since $\mathcal{C}_v(B_m)$ exists and, by Claim 5, no conflicting certificate exists in view v, it follows that honest parties only vote for blocks in view $v + 1$ that extend B_m. By induction, this implies B_l must be a descendant of B_m, contradicting the earlier conclusion.

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

Theorem 5 (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 Optimistic Commit Tolerating $\frac{n}{3}$ Byzantine Faults

Instead of having a combination of Byzantine and crash faults, suppose all $< \frac{n}{3}$ faulty parties are Byzantine, i.e., we have $n = 3f + 3p + 1$ where we tolerate $f + p$ Byzantine faults. Can a fast commit provide any guarantees?

Observe that if the fast commit relies on votes from a quorum of $\frac{2n}{3}$ parties, then as discussed in Section 1.1, it may be the case that only one party P_i commits on some value B in some view v. Suppose

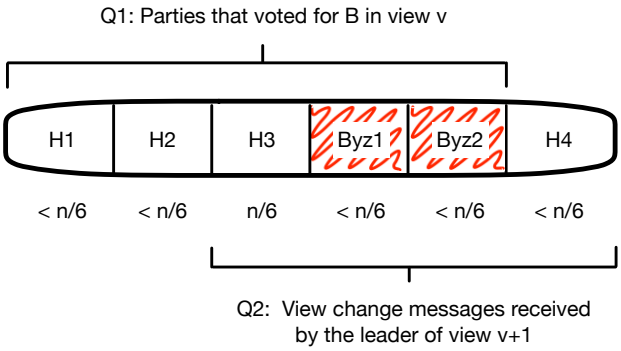

Figure 4: An example quorum intersection. The parties are divided into five groups of size $< \frac{n}{6}$ (H1, H2, H4, Byz1, Byz2) and one group of size $\frac{n}{6}$ (H3). Groups starting with H denote honest parties, and with Byz denote Byzantine parties. Observe that for the leader L_{v+1} to obtain a set of votes such that the majority is for a value $\neq B$, at least $\frac{n}{6}$ parties in Byz1 $\cup$ Byz2 need to report a value $\neq B$.

this voting quorum is denoted by quorum Q1. In the next view $v+1$, the set of $\frac{2n}{3}$ parties reporting to the leader L_{v+1} may intersect with Q1 in only one honest party. If all other parties report having voted for a different value B', then an honest leader can propose B' in view $v+1$.

What if the fast commit requires more votes, say $n-p$ votes? For simplicity, let us consider $f = p < \frac{n}{6}$, and we want to tolerate $f+p$ Byzantine faults. Observe that, as long as total number of Byzantine faulty parties are $< \frac{n}{6}$ Byzantine faulty parties, again, a party P_i can obtain $\frac{5n}{6}$ votes for a value B proposed in view v. This constitutes a fast commit for B. Suppose the set of votes is denoted by quorum Q1. During a view change, we consider two scenarios. First, under the optimistic case where we have $< \frac{n}{6}$ Byzantine faulty parties. In this case, observe that the quorum Q2 of size $\frac{2n}{3}$ that the leader L_{v+1} obtains intersects with Q1 at more than $\frac{n}{3}$ honest parties. Thus, if L_{v+1} proposes the majority value received, then it will propose B. Second, consider the scenario where there are $\geq \frac{n}{6}$ but $< \frac{n}{3}$ Byzantine parties. In this case, the quorum intersection between Q1 and Q2 contains only $\frac{n}{6}$ honest parties (cf. Figure 4). In particular, L_{v+1} can obtain a majority vote for a value $B' \neq B$. Due to this, even an honest leader L_{v+1} can propose B' in view $v+1$, and it could be committed by another honest party P_j, leading to a safety violation. We argue that, in such situations, it must be the case that there exist $\frac{n}{6}$ Byzantine parties at the intersection of Q1 and Q2 who must have equivocated, i.e., voted for B in view v and reported an equivocating value when responding to leader L_{v+1}. Thus, when the number of Byzantine faults is $\geq \frac{n}{6}$, if there is a safety violation, at least $\frac{n}{6}$ faulty parties can be held accountable using ideas from Sheng et al. [22]. We will describe the protocol, including analysis, in the full version of this paper.

6 Closely Related Work

Partially synchronous protocols can tolerate up to $< n/3$ Byzantine faults [10]. Several protocols in the literature obtain this resilience while simultaneously obtaining a latency of three rounds from the time a transaction is proposed to the time it is committed [6–8]. A generalization of this work considers a setting where $n = 3f + 2c + 1$ where f is the number of Byzantine faults and c is the number of crash faults tolerated. Compared to this line of work (Line1), under optimistic conditions, our protocol can commit within two rounds.

A line of work (Line2) under partial synchrony, starting with FaB [18], proposed an optimistic path to commit; at a high level, if fewer parties are faulty, these protocols can commit within two rounds [1,12,15,18, 24]. The state of the art in this line of work works with $n = 3f + 2p + 1$ where f is the number of Byzantine faults tolerated. These protocols guarantee a latency of three rounds; however, if up to $p \leq f$ parties are faulty, then the parties can commit within two rounds. Compared to this line of work, our protocols tolerate

additional crash faults. In fact, our work strictly generalizes the two lines of work (Line1 and Line2).

We note that while Alpenglow [13] claims to tolerate $< 20\%$ Byzantine faults and $< 20\%$ crash faults simultaneously, this does not hold true under partial synchrony. This is acknowledged in their work [13, Section 2.11, Example 44]. In comparison, we obtain the $f + c$ resilience and the optimistic fast commit under partial synchrony.

The notion of an optimistic fast path has also been considered under synchronous and asynchronous networks for SMR and other closely related consensus primitives [4, 5, 11, 16, 19, 25, 26].

7 Acknowledgment

We thank Victor Shoup for insightful discussions and for identifying a subtle liveness issue in an earlier version of this paper. We also thank Ittai Abraham, Yuval Efron, Ling Ren, and Giulia Scaffino for valuable feedback and helpful discussions.

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] Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. Optrand: Optimistically responsive reconfigurable distributed randomness. In *Network and Distributed System Security Symposium (NDSS)*, 2023.

[5] Francisco Brasileiro, Fabíola Greve, Achour Mostéfaoui, and Michel Raynal. Consensus in one communication step. In *Parallel Computing Technologies: 6th International Conference, PaCT 2001 Novosibirsk, Russia, September 3–7, 2001 Proceedings 6*, pages 42–50. Springer, 2001.

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

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

[8] Benjamin Y. Chan and Rafael Pass. Simplex consensus: A simple and fast consensus protocol. In *Theory of Cryptography: 21st International Conference, TCC 2023, Taipei, Taiwan, November 29–December 2, 2023, Proceedings, Part IV*, page 452–479, Berlin, Heidelberg, 2023. Springer-Verlag.

[9] Isaac Doidge, Raghavendra Ramesh, Nibesh Shrestha, and Joshua Tobkin. Moonshot: Optimizing chain-based rotating leader bft via optimistic proposals. *arXiv preprint arXiv:2401.01791*, 2024.

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

[11] Roy Friedman, Achour Mostefaoui, and Michel Raynal. Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46–56, 2005.

[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/1y_7ddr8oNOknTQYHzXeeMD2ProQOWjMs/view`.

[14] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 45–58, 2007.

[15] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comput. Syst.*, 27(4), January 2010.

[16] Klaus Kursawe. Optimistic byzantine agreement. In *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings.*, pages 262–267. IEEE, 2002.

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

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

[19] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37*, pages 3–33. Springer, 2018.

[20] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, April 1980.

[21] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[22] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. Bft protocol forensics. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 1722–1743. ACM, November 2021.

[23] Victor Shoup. Sing a Song of Simplex. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:22, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[24] Victor Shoup, Jakub Sliwinski, and Yann Vonlanthen. Kudzu: Fast and simple high-throughput bft. *arXiv preprint arXiv:2505.08771*, 2025.

[25] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. On the optimality of optimistic responsiveness. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 839–857, 2020.

[26] Nibesh Shrestha, Qianyu Yu, Aniket Kate, Giuliano Losa, Kartik Nayak, and Xuechao Wang. Optimistic, signature-free reliable broadcast and its applications. *arXiv preprint arXiv:2505.02761*, 2025.