On the Optimality of Optimistic Responsiveness

Ittai Abraham[1], Kartik Nayak[2], Ling Ren[3], and Nibesh Shrestha[*4]

[1]VMware Research, iabraham@vmware.com
[2]Duke University, kartik@cs.duke.edu
[3]University of Illinois at Urbana-Champaign, renling@illinois.edu
[4]Rochester Institute of Technology, nxs4564@rit.edu

Abstract

Synchronous consensus protocols, by definition, have a worst-case commit latency that depends on the bounded network delay. The notion of optimistic responsiveness was recently introduced to allow synchronous protocols to commit instantaneously when some optimistic conditions are met. In this work, we revisit this notion of optimistic responsiveness and present optimal latency results.

We present a lower bound for Byzantine Broadcast that relates the latencies of optimistic and synchronous commits when the designated sender is honest and while the optimistic commit can tolerate some faults. We then present two matching upper bounds for tolerating f faults out of $n = 2f+1$ parties. Our first upper bound result achieves optimal optimistic and synchronous commit latencies when the designated sender is honest and the optimistic commit can tolerate some faults. Our second upper bound result achieves optimal optimistic and synchronous commit latencies when the designated sender is honest but the optimistic commit does not tolerate any faults. The presence of matching lower and upper bound results make both of the results tight for $n = 2f + 1$. Our upper bound results are presented in a state machine replication setting with a steady state leader who is replaced with a view-change protocol when they do not make progress. For this setting, we also present an optimistically responsive protocol where the view-change protocol is optimistically responsive too.

1 Introduction

Byzantine fault-tolerant (BFT) protocols based on a synchronous network have a high resilience of up to one-half Byzantine faults. In comparison, BFT protocols under asynchronous or partially synchronous networks can tolerate only one-third Byzantine faults. Although partially synchronous protocols have a lower tolerance for Byzantine faults, they have an advantage in terms of the latency to commit – they can commit in $O(\delta)$ time where δ is the actual latency of the network. On the other hand, the latency for synchronous protocols depends on Δ, where Δ is a pessimistic bound on the network delay.

A recent work, Hybrid Consensus [21], introduced the notion of *responsiveness* to describe a commit whose latency depends only on the actual network delay δ. In this regard, asynchronous and partially synchronous protocols are responsive by design, whereas synchronous protocols are not. Another recent work, Thunderalla [22], introduced the notion of *optimistic responsiveness* that allows a synchronous consensus protocol to commit responsively when some *optimistic* conditions are met. Their protocol is always safe against up to one-half Byzantine faults. Moreover, if a "leader" and $> 3n/4$ replicas are honest, and if they are on a "fast-path", then replicas can commit responsively with a commit latency independent of Δ. Otherwise, the protocol falls back to a "slow-path", and the commit latency depends on Δ.

The Thunderella paradigm of optimistic responsiveness requires replicas in the protocol to know which of the two paths they are in, and perform an explicit switch between them. If they start in the slow-path,

*Lead author

a switch to the fast-path is performed when the optimistic conditions are met. The fast-path allows for responsive commits. If, at some point, the conditions are not met, then the replicas switch to the slow-path again. Their slow-path protocols are Nakamoto consensus and Dolev-Strong. Thus, the slow-path, as well as the switch between the two paths, is extremely slow, requiring $O(\kappa\Delta)$ and $O(n\Delta)$ latency respectively. Their slow-path–fast-path paradigm, however, holds generically for other synchronous protocols. Sync HotStuff [2] adopted this paradigm and presented a protocol where the slow-path, as well as the switch, is considerably faster, requiring only a 2Δ latency. A fundamental question then is,

What is the optimal latency of an optimistically responsive synchronous protocol?

Observe that if all the replicas know whether the optimistic conditions are met, i.e., in the case of Sync HotStuff, whether or not fewer than 1/4 replicas are Byzantine, then we can use a protocol with optimal latency for that setting. For instance, if the optimistic conditions are met, then we can rely on partially synchronous protocols [26, 18, 6, 5] to commit responsively. Otherwise, we can use an optimal synchronous protocol with Δ latency [3]. Observe that Thunderella or Sync HotStuff with optimistic responsiveness will be inferior to either of the above options; if we start off in the wrong path, i.e., slow-path when optimistic conditions are met and vice versa, then we incur an additional path switching latency making the latency worse than either of the options. Thus, with optimistically responsive synchronous protocols, we are interested in the optimal latency when *it is not known whether the optimistic conditions are met* (or if during the execution of the protocol, the adversarial conditions can switch to-and-from meeting the optimistic conditions). In such a scenario, using a partially synchronous protocol, or a non-optimistically responsive synchronous protocol leads to either a safety violation or sub-optimal latency respectively.

A lower bound on the latency of an optimistically responsive synchronous protocol. Our first result presents a lower bound on the latency of such optimistically responsive synchronous protocols. Specifically, we show the following result:

Theorem 1 (Lower bound on the latency of an optimistically responsive synchronous protocol, informal)**.** *For some $0 < \alpha < \Delta$, there does not exist a Byzantine Broadcast protocol that can tolerate $n/3 < f < n/2$ faults and when all messages between non-faulty parties arrive instantaneously, achieves the following simultaneously under an honest designated sender:*

(i) (optimistic commit) a commit latency of $< \alpha$ in the presence of $n - 2f$ crash faults, and

(ii) (synchronous commit) a commit latency of $< 2\Delta - \alpha$ in the presence of f omission faults.

The lower bound says that if a Byzantine Broadcast protocol tolerating $n/3 < f < n/2$ corruption has an optimistic (fast) commit with latency $< \alpha$ time while still being able to tolerate $n - 2f$ faults, then the synchronous (slow) commit should have a latency $\geq 2\Delta - \alpha$ time when tolerating f faults. For instance, if the optimistic commit requires 0.5Δ time for a commit, then the synchronous path cannot commit in time $< 1.5\Delta$. Known optimistically responsive protocols such as Thunderella and Sync HotStuff use $\alpha = O(\delta)$, but require a slow-path latency of $O(\kappa\Delta)/O(n\Delta)$ for Thunderella and 2Δ for Sync HotStuff. Thus, none of these protocols obtain the desired optimal latency.

Our next two results present matching upper bounds for this lower bound for $n = 2f + 1$. Thus, both our lower bounds and upper bounds are tight. In the process of seeking an optimal latency protocol in this setting, we depart from the back-and-forth slow-path–fast-path paradigm for optimistic responsiveness and present an optimal one-half resilient synchronous protocols where replicas do not need to know which path they are on. If the conditions for a *optimistic commit* are met, they commit optimistically. Otherwise, they commit using the *synchronous commit* rule. Thus, intuitively, they exist in both paths *simultaneously* without requiring an explicit switch. Since all of our upper bounds assume a value of $\alpha = O(\delta)$, whenever appropriate, we also call the optimistic commit a *responsive commit.*

Optimal optimistic responsiveness with 2Δ-synchronous latency and $> 3n/4$-sized responsive quorum. Our first result obtains optimistic responsiveness where the synchronous commit has a commit latency of 2Δ, while the responsive commit has a latency of 2δ using quorums of size $> 3n/4$. Specifically, we show the following:

Theorem 2 (Optimistic responsiveness with 2Δ-synchronous latency and $> 3n/4$-sized responsive quorum, informal)**.** *There exists a Byzantine Broadcast protocol that can tolerate $< n/2$ faults, and under an honest designated sender achieves the following simultaneously:*

(i) (responsive commit) a commit latency of 2δ when $> 3n/4$ replicas are honest, and

(ii) (synchronous commit) a commit latency of $2\Delta + O(\delta)$ otherwise.

Intuitively, the fundamental property that this upperbound provides in comparison to Thunderella or Sync HotStuff is *simultaneity*, i.e., replicas do not need to on agree on specific paths for performing a responsive commit or a synchronous commit. Moreover, the parameters obtained in this result are optimal. First, the early stopping lower bound due to Dolev, Reischuk, and Strong [9] states that when the number of faults is f, and the maximum number of faults is t, there will exist an execution in a Byzantine Broadcast protocol that requires $\min(t + 1, f + 2)$ rounds. Hence, in the absence of any faults, no protocol can have a latency of less than 2δ. Second, the $> 3n/4$ quorum size is tight due to a lower bound presented in Thunderella [22]. Specifically, it says that no protocol can have a worst-case resilience of one-half Byzantine replicas while being optimistically responsive when there are more than $n/4$ Byzantine replicas. Finally, due to the lower bound from our first result (Theorem 1), latency for the synchronous commit is optimal for $\alpha = O(\delta)$, ignoring $O(\delta)$ delays.

Optimal optimistic responsiveness with Δ-synchronous latency and n-sized responsive quorum. In Theorem 1, the $2\Delta - \alpha$ latency for a synchronous commit is applicable only when the optimistic commit can tolerate $n - 2f$ faults. In this upperbound result, we show that the synchronous latency can be improved if the optimistic commit guarantees hold only when all $n = 2f + 1$ replicas are honest. Specifically, we show the following:

Theorem 3 (Optimistic responsiveness with Δ-synchronous latency and n-sized responsive quorum, informal)**.** *There exists a Byzantine Broadcast protocol that can tolerate $< n/2$ faults, and under an honest designated sender achieves the following simultaneously:*

(i) (responsive commit) a commit latency of 2δ when all n replicas are honest, and

(ii) (synchronous commit) a commit latency of $\Delta + O(\delta)$ otherwise.

The responsive commit latency is optimal due to Dolev et al. [9] while the synchronous commit latency Δ is optimal (ignoring $O(\delta)$ delays) due to the lower bound in [2].

Comparison with works having simultaneity in commits. We note that our upperbound results are not the first results to use simultaneous paths. There are works such as Zyzzyva [17], SBFT [14] and FaB [20] which have considered the notion of simultaneous paths under partial synchrony. Similarly, a recent work called PiLi [7] achieves simultaneity under a synchronous assumption. This is the first work that achieves simultaneity under a synchrony assumption while obtaining optimal latency.

Byzantine Broadcast vs. State Machine Replication. We note that although we present our theorem statements for Byzantine Broadcast in the presence of an honest designated sender, we directly describe our protocols in a state machine replication setting with a steady state leader proposing a sequence of values. Whenever the leader does not make progress, it is replaced using a view-change protocol. An honest designated sender in Byzantine Broadcast is thus equivalent to having an honest leader in a state machine replication setting. Thus, when the leader is honest, our protocol from Theorem 2 (resp. Theorem 3) can commit every value optimistically in 2δ time and synchronously in $2\Delta + O(\delta)$ (resp. $\Delta + O(\delta)$). Moreover, the honest leader can propose consecutive values as fast as 2δ (resp. $\Delta + 2\delta$) time independent of whether commits are performed responsively or synchronously.

Optimistic responsiveness with responsive view-change. The upperbound protocols that we just mentioned can commit responsively when the leader is honest and their respective optimistic conditions are met. However, when executing on a sequence of values, for reasons such as fairness or distribution of work, we may want to change leaders every block, or every few blocks. Indeed, several recent protocols have been designed with this goal in mind [2, 6, 8, 13, 15, 25]. However, for the upper bounds described earlier, the view-change protocols, although efficient, still require $4\Delta + O(\delta)$ time; such a latency is reasonable if a view-change happens only occasionally. However, the incurred latency maybe high if we need to change views after every block. Moreover, the latency is incurred even when the optimistic conditions are met.

Our final result addresses this concern and presents a protocol which has an optimistically responsive view-change as well. Thus, when rotating among honest leaders and if $> 3n/4$ replicas are honest, this protocol can commit a block every $O(\delta)$ time without incurring $O(\Delta)$ time in the steady state or the view-change. On the other hand, even if the optimistic conditions are not met, the protocol requires 2Δ time to elect a leader and $3\Delta + O(\delta)$ time to commit a block after an honest leader is elected.

Organization. The rest of the paper is organized as follows. In Section 2, we define state machine replication and Byzantine Broadcast. In Section 3, we present a lower bound on latency for optimistic responsiveness. Section 4 presents an optimal optimistically responsive protocol with 2Δ-synchronous latency and $> 3n/4$-sized responsive quorum for $n = 2f + 1$. Section 5 presents an optimal optimistically responsive protocol with Δ-synchronous latency and n-sized responsive quorum for $n = 2f + 1$. In Section 6, we present an optimistically responsive protocol that includes optimistically responsive view-change. Finally, Section 7 compares with closely related work.

2 Model and Definitions

We consider a standard State Machine Replication (SMR) problem used for building a fault tolerant service to process client requests. The system consists n replicas out of which $f < n/2$ replicas are Byzantine faulty. The Byzantine replicas may behave arbitrarily. The aim is to build a consistent linearizable log across all non-faulty (honest) replicas such that the system as a whole behaves like a single non-faulty server in the presence of $f < n/2$ Byzantine replicas.

Definition 2.1 (Byzantine Fault-tolerant State Machine Replication [24])**.** *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 server, providing the following two guarantees.*

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

We assume the network between replicas includes a standard synchronous communication channel with point-to-point, authenticated links between them. Messages between replicas may take at most Δ time before they arrive, where Δ is a known maximum network delay. To provide safety under adversarial conditions, we assume that the adversary is capable of delaying the message for an arbitray time upper bounded by Δ. The actual message delay in the network is denoted by δ. We make use of digital signatures and a public-key infrastructure (PKI) to prevent spoofing and replays and to validate messages. Message x sent by a replica p is digitally signed by p's private key and is denoted by $\langle x \rangle_p$.

Byzantine Broadcast. Our lower bound is presented for a Byzantine Broadcast setting with a designated sender.

Definition 2.2 (Byzantine Broadcast)**.** *A Byzantine broadcast protocol provides the following three guarantees.*

- *Agreement. If two honest replicas commit values b and b' respectively, then $b = b'$.*
- *Termination. All honest replicas eventually commit.*
- *Validity. If the designated sender is honest, then all honest replicas commit on the same value it proposes.*

3 A Lower Bound on the Latency of Optimistic Responsiveness

An optimistically responsive synchronous protocol has two commit rules – an optimistic commit rule and a synchronous commit rule. This lower bound captures the relationship between the latencies of the two commit rules. Essentially, it says that if the optimistic commit rule is too fast, then the synchronous commit rule should be correspondingly slower. Specifically, the sum of the latencies of the two commit rules should be at least 2Δ time.

In this lower bound, we assume that an optimistic commit is capable of tolerating $n-2f$ crash faults and has a commit latency of $< \alpha$ time for some $0 < \alpha < \Delta$. The lower bound then proves that if an optimistically responsive protocol can tolerate $n/3 < f < n/2$ Byzantine faults, then its synchronous commit rule cannot have a latency $< 2\Delta - \alpha$.

Intuition. The intuition behind the lower bound is to show a *split-brain* attack that can be performed by a minority of Byzantine replicas if a protocol has sum of latencies for the two commit rules to be less than 2Δ. For simplicity, we present intuition for $\alpha = O(\delta)$ and $n = 2f + 1$. First, observe that any protocol tolerating minority Byzantine faults cannot use quorum sizes larger than $n - f = f + 1$ in the worst case. Hence, it is always possible that a single honest replica R commits to a value due to a quorum of messages received from only the Byzantine replicas if it does not wait long enough before committing. Second, since the optimistic commit rule can tolerate at least one crash fault ($n - 2f = 1$), replicas (set P) committing through the optimistic rule may commit without receiving any messages from replica R. Thus, to avoid a safety violation through a split-brain attack, it is required for the replicas in P and R to communicate about protocol instance specific messages with each other. Using the fact that the adversary can delay any message by up to Δ time, replicas in P may not receive the sender's message in the first Δ time. Moreover, it takes Δ time for messages from P to arrive at R. So, R cannot be stopped from committing a different value unless it waits for 2Δ time. On the other hand, since P is performing an optimistic commit, it may not wait for more than $O(\delta)$ (in general, α time) before committing (which is not sufficient to receive messages from R).

Before presenting the formal lower bound, we define the notion of a (β, f_c)-time-shift-invariant protocol.

Definition 3.1 ((β, f_c)-time-shift-invariant Byzantine Broadcast protocol)**.** *We say a Byzantine Broadcast protocol among n parties tolerating f Byzantine faults is (β, f_c)-time-shift-invariant if the following two executions are such that, for every non-faulty party if it takes an action at time t in execution (i), it will take the same action at time $\leq t + \beta$ in execution (ii):*

(i) All f_c crash faults (not including the designated sender), crash at time 0. All messages sent among non-faulty parties arrive instantaneously.

(ii) All f_c crash faults (not including the designated sender), crash at time 0. An execution where the first message from the designated sender received by any party is delayed by β time and all other messages sent among non-faulty parties arrive instantaneously.

Intuitively, parties in protocols may behave differently depending on the schedule of messages in executions. For instance, protocols may have time-outs for receiving some messages, which if triggered, parties take a different set of actions (e.g., blame a leader) than if the time-outs did not trigger. A (β, f_c)-time-shift-invariant Byzantine Broadcast protocol tolerating f-out-of-n Byzantine faults defines a sub-class of Byzantine Broadcast protocols which under f_c crashes, even if the arrival of the first message from the designated sender is delayed by β time and all other messages arrive instantaneously, the parties take the same actions compared to a setting where all messages arrive instantaneously; however, each action may be taken up to β time later corresponding to the all-instantaneous world. We note that for $\beta \leq \Delta$, all known synchronous Byzantine Broadcast protocols, to the best of our knowledge, satisfy (β, f_c)-time-shift-invariance. This is because under a synchrony assumption, any message can be delayed by up to Δ time, and this could have happened to the sender's message. Thus, any delay of $< \Delta$ for the first message is not handled differently by known synchronous Byzantine Broadcast protocols.

We now present the formal lower bound below.

Theorem 4 (Lower bound on the latency of an optimistically responsive synchronous protocol)**.** *For $0 < \alpha < \Delta$, there does not exist a $(\Delta - \alpha, n - 2f)$-time-shift-invariant Byzantine Broadcast protocol that can tolerate $n/3 < f < n/2$ faults and when all messages between non-faulty parties arrive instantaneously, achieves the following simultaneously under an honest designated sender:*

(i) the commit latency is $< \alpha$ in the presence of $n - 2f$ crash faults, and

(ii) the commit latency is $< 2\Delta - \alpha$ in the presence of f omission faults.

Proof. Suppose there exists a protocol that can simultaneously achieves both of these properties for some $0 < \alpha < \Delta$. We will show a sequence of worlds, and through an indistinguishability argument prove a violation in the agreement property of such a protocol. Consider parties being split into three groups P, Q, and R. P contains f parties, Q contains f parties, and R contains the remaining $n - 2f$ parties. Observe that under a synchrony assumption, the messages can take up to Δ time to arrive at its destination but within this duration the adversary can deliver it at an arbitrary time. We consider four worlds as follows. In all four worlds, we suppose a party from set Q is the designated sender.

World 1**.**

- *Setup.* Parties in P and Q are honest while the parties in R have crashed. Suppose the honest sender sends input value b at time 0.
- *Adversarial schedule.* The adversary follows the following message schedule (schedule (i)): All messages sent among parties in $P \cup Q$ are delivered instantaneously.
- *Execution and views of honest players.* Since there are $2f$ honest parties in $P \cup Q$ and the remaining $n - 2f$ have crashed, the protocol will commit at time $< \alpha$. By the validity property of Byzantine Broadcast, all parties in $P \cup Q$ commit b at time $< \alpha$.

World $1'$**.**

- *Setup.* Parties in P and Q are honest while the parties in R have crashed. Suppose the honest sender sends input value b at time 0.
- *Adversarial schedule.* The adversary follows the following message schedule (schedule (ii)): The first message received by parties in $P \cup Q$ from the designated sender is delayed by time $\Delta - \alpha$. All other messages sent among parties in $P \cup Q$ are delivered instantaneously.
- *Execution and views of honest players.* Given the protocol is $(\Delta - \alpha, n - 2f)$-time-shift-invariant, the replicas in $P \cup Q$ will take the same actions as the replicas in World 1; however each action may be taken up to $\Delta - \alpha$ time later. Hence, all parties in $P \cup Q$ commit b at time $< (\Delta - \alpha) + \alpha = \Delta$.

World 2**.**

- *Setup.* Parties in Q and R are honest while parties in P are omission faults. Suppose the honest sender sends input value $b' \neq b$ at time 0.
- *Adversarial schedule.* The first message received by any party in P is delivered at time $\Delta - \alpha$. Also, messages sent by parties in P at time $< \Delta - \alpha$ are not omitted. Messages sent by parties in P at time $\geq \Delta - \alpha$ are omitted.

  The adversary follows the following message schedule (schedule (iii)) for parties in $Q \cup R$. All messages sent by P are delivered at a delay of Δ to R and vice versa. All the messages between all honest replicas in $Q \cup R$ (including that of the designated sender) are delivered instantaneously.
- *Execution and views of honest players.* Since $Q \cup R$ contains $n - f$ honest parties, by the validity property of Byzantine Broadcast, all parties in $Q \cup R$ commit b' at time $< 2\Delta - \alpha$.

World 3.

- *Setup.* Parties in P and R are honest while parties in Q (which contains the designated sender) are Byzantine. The designated sender sends input value b to P and b' to R.
- *Adversarial schedule.* The parties in Q behave exactly the same as honest Q in the execution of World 2 with parties in R. Parties in Q also behave exactly the same way as in execution of World $1'$ with parties in P.

  Messages sent to P follow schedule (ii) described in World $1'$. This can be performed by delaying the first message from the designated sender in Q to $P \cup Q$ by $\Delta - \alpha$ time. All other messages among parties in $P \cup Q$ are delivered instantaneously. In addition, all messages sent from R to P are delayed by Δ.

  Messages sent to R follow schedule (iii) described in World 2, i.e., messages within $Q \cup R$ are delivered instantaneously whereas messages received from P to R are delivered at a delay of Δ.
- *Execution and views of honest parties.* Since messages from R are delayed to the maximum Δ time, parties in P do not receive any message from R during the first $< \Delta$ time period. Hence, the view of parties in P is exactly the same as in World $1'$. Hence, parties in P commit b at time $< \Delta$.

  For honest R, all the messages delivered to it from P are delayed by Δ. All the messages sent by parties in P at time $< \Delta - \alpha$ are not related to the sender's input (since the sender's first message to parties in P is delayed by $\Delta - \alpha$ time). So, until time $< 2\Delta - \alpha$, the view of R is exactly the same as that in World 2. Hence, it commits to b' at time $< 2\Delta - \alpha$.

□

Remark 1. Our lower bound has a restriction $n/3 < f < n/2$ due to the way the proof is constructed. We believe this should be easily modifiable to the setting where $f > n/3$ by setting the sizes of P, Q, and R differently.

Remark 2. Our lower bound holds only for $(\Delta - \alpha, n - 2f)$-time-shift-invariant Byzantine Broadcast protocols. We do not know of any existing synchronous Byzantine Broadcast protocols that does not satisfy this requirement. It is an interesting open problem to come up with a protocol that is not $(\Delta - \alpha, n - 2f)$-time-shift-invariant and can circumvent the lower bound to obtain better latency results.

4 Optimal Optimistic Responsiveness with 2Δ-synchronous Latency and $> 3n/4$-sized Quorum

We first present a simple synchronous consensus protocol that achieves optimal optimistic responsiveness when the optimistic commit does not require a quorum of all replicas. In a *synchronous commit*, a replica commits 2Δ time after voting (recall that Δ is an upper bound on the maximum network delay) if an equivocating proposal has not been detected. In a *responsive commit*, a replica can commit immediately, i.e., without waiting for the 2Δ time period, if a sufficient number of replicas have voted for the block and no equivocation has been detected. For every block, a replica opportunistically waits to commit using either of the commit rules.

Recall that $\delta \leq \Delta$ is the actual network delay. If a "leader" is honest then no matter what the adversary does, the system can commit a block in time $2\Delta + O(\delta)$. But if there are $> 3n/4$ honest replicas along with an honest leader, then the system can commit in time $O(\delta)$ (in an optimistically responsive manner).

Why does our protocol perform better than protocols in the slow-path–fast-path paradigm? The general strategy employed in the protocols with back-and-forth slow-path–fast-path paradigm is to start in one of the two paths, say, the slow path. When the optimistic conditions are met, an explicit switch is performed to move to the fast path. Similarly, when a lack of progress is detected in the fast path, the protocol makes another switch to the slow path. The explicit switch between the paths incurs a latency of at least Δ in all of these protocols. The slow path and the fast path in Sync HotStuff has a commit latency of 2Δ and 2δ respectively. Similarly, the slow path to fast path switch incurs a latency of Δ and fast path to slow path switch incurs 2Δ latency is Sync HotStuff.

Under minority Byzantine failures, the adversary can attack the above strategy to worsen the commit latency compared to a protocol with a single slow path. For example, when the protocol is in slow path, the adversary responds promptly and the replicas receive $> 3n/4$ responses thereby triggering fast path by performing an explicit switch. Once in the fast path, the adversary stops responding completely thus preventing progress in the fast path. This forces an explicit switch to the slow path again. Under this attack, a single decision can incur a latency of 4Δ if the replicas are in the fast path and switch to the slow path to commit. Replicas may never commit in the fast path if the adversary does not respond in the fast path.

Our protocol avoids this concern by avoiding an explicit switch between the responsive and synchronous conditions. Instead, both paths are active simultaneously. Replicas commit responsively when $> 3n/4$ replicas respond and synchronously otherwise. As a result, when the leader is honest, the commit latency is 2δ during optimistic executions and 2Δ in the worst-case (during non-optimistic executions).

View-based execution. Like PBFT [5], our protocol progresses through a series of numbered *views* with each *view* coordinated by a distinct leader. Views are represented by non-negative integers with 0 being the first view. The leader of the current view v is determined by ($v \bmod n$). Within each view, also called the steady state, the leader is expected to propose values and keep making progress by committing client requests at increasing *heights*. An honest replica participates in any one view at a time and moves to a higher numbered view when the current view fails to make progress. If the replicas detect equivocation or lack of progress in a view, they initiate a *view-change* by blaming the current leader. When a quorum of replicas have blamed the current leader, they perform a view-change and replace the faulty leader.

Blocks and block format. Client requests are batched into *blocks*. Each block references its predecessor with the exception of the genesis block which has no predecessor. We call a block's position in the chain of blocks as its height. A block B_k at height k has the format,

$$B_k := (b_k, H(B_{k-1}))$$

where B_{k-1} is the block at height $k-1$ and $H(B_{k-1})$ is the hash digest of B_{k-1}. The predecessor for the genesis block is $\bot$. A block B_k is said to be *valid* if (i) its predecessor block is valid, or if $k = 1$, predecessor is $\bot$, and (ii) client requests in the block meet application-level validity conditions and are consistent with its chain of requests in ancestor blocks.

Block extension and equivocation. A block B_k *extends* a block B_l ($k \geq l$) if B_l is an ancestor of B_k. Note that a block B_k extends itself. Two blocks B_k and $B'_{k'}$ proposed in the same view *equivocate* one another if they are not equal to *and* do not extend one another.

Block certificates. A *block certificate* represents a set of signatures on a block by a quorum of replicas. Given a ratio $0 \leq \alpha < 1$, a block B_k and a view v we denote by $\mathcal{C}_v^\alpha(B_k)$ a set of $\lfloor \alpha n \rfloor + 1$ signatures from different replicas on block B_k signed in view v. In this section, we will use *synchronous certificate* where $\alpha = 1/2$, and *responsive certificate* where $\alpha = 3/4$. Whenever the distinction between the two is not important, we will represent the certificates by $\mathcal{C}_v(B_k)$ and ignore the superscript α. In later sections, we will also use *full certificates* which require all n replicas to sign.

Chain certificates. We use the notion of *chain certificates* to compare different chains when replicas receive many of them. Most earlier protocols (e.g., HotStuff [26] or Sync HotStuff [2]) compared certified chains using just the views and heights. However, in our protocol, there are two types of certificates, a responsive certificate and a synchronous certificate and hence, comparing them is more subtle. However, as we will see the rank of a chain will be completely determined by the highest synchronous certificate from a view and the highest responsive certificate of its ancestor. A chain certificate comprises of a pair of certificates $\mathcal{C}_v^{3/4}(B_k)$ and $\mathcal{C}_v^{1/2}(B_\ell)$. Each element in the pair is either a block certificate or $\bot$ such that (i) if either of them are not $\perp$, both certificates are from the same view, (ii) if not $\perp$'s, the first certificate has threshold 3/4, the second has threshold 1/2, and (iii) block B_ℓ extends block B_k, if $\mathcal{C}_v^{3/4}(B_k)$ is not $\perp$.

Ranking chain certificates. Given two chain certificates $\mathcal{CC} = (\mathcal{C}_v^{3/4}(B_k),\mathcal{C}_v^{1/2}(B_\ell))$ and $\mathcal{CC}' = (\mathcal{C}_{v'}^{3/4}(B_{k'}),\mathcal{C}_{v'}^{1/2}(B_{\ell'}))$, the chain certificates are first ranked by views, i.e., $\mathcal{CC} < \mathcal{CC}'$ if $v < v'$. While moving from view v to any higher view, our protocol ensures that if a certified block B_k is committed in view v, then all honest replicas lock on a chain certificate that extends B_k. Hence, a certificate chain produced in a higher view will always include B_k. Said another way, a certificate chain $\mathcal{CC}'$ in a higher view will extend B_k; if it does not, it must be the case that B_k was not committed by any honest replica in view v. Thus, it is safe to extend $\mathcal{CC}'$.

For chain certificates in the same view v, the chain certificates are first ranked based on the height of the responsive certificate, i.e., $\mathcal{CC} < \mathcal{CC}'$ if $k < k'$. In our protocol, we ensure that if there exists a responsive certificate for a block $B_{k'}$ in view v, i.e., $\mathcal{C}_v^{3/4}(B_{k'})$ exists, there cannot exists a responsive certificate for a conflicting block at any height in view v. Thus, if there is a responsive certificate for B_k in view v, then $B_{k'}$ must extend B_k. Moreover, we also ensure that if $\mathcal{C}_v^{3/4}(B_k)$ exists, no replica will have synchronously committed on an equivocating block B_ℓ with certificate $\mathcal{C}_v(B_\ell)$. Thus, any equivocating chain with chain certificate $\mathcal{CC}$ will not contain committed blocks that are not extended by $\mathcal{CC}'$.

Finally, if both chain certificates are in the same view v and have a common responsive certificate in the view (or both do not have a responsive certificate), the chain certificates are ranked by the heights of synchronous certificates, i.e., $\mathcal{CC} < \mathcal{CC}'$ if $\ell < \ell'$. Our protocol ensures that if B_k is committed synchronously in view v, then there does not exist an equivocating certified block. Thus, if equivocating $\mathcal{C}_v^{1/2}(B_\ell)$ and $\mathcal{C}_v^{1/2}(B_{\ell'})$ exist, both B_ℓ and $B_{\ell'}$ could not have been committed. To ease the rule in the case where they do not equivocate and one chain certificate extends the other, we select the higher of the two.

Thus, Given two chain certificates $\mathcal{CC} = (\mathcal{C}_v^{3/4}(B_k),\mathcal{C}_v^{1/2}(B_\ell))$ and $\mathcal{CC}' = (\mathcal{C}_{v'}^{3/4}(B_{k'}),\mathcal{C}_{v'}^{1/2}(B_{\ell'}))$, we say $\mathcal{CC} < \mathcal{CC}'$ if:

1. $v < v'$ (the chain certificates are first ranked by view),
2. $v = v'$ and $k < k'$ (secondly ranked by responsive certificates), or
3. $v = v'$ and $k = k'$ and $\ell < \ell'$ (finally ranked by synchronous certificates).

In this above comparison, we use the numerical value -1 to represent a $\perp$.

Tip of a chain certificate. The tip of a chain certificate is the highest block in the chain. Given a $\mathcal{CC} = (\mathcal{C}^{3/4}(B_k), \mathcal{C}^{1/2}(B_\ell))$, if $\mathcal{C}^{1/2}(B_\ell) \neq \perp$ then define $\mathsf{tip}(\mathcal{CC}) = B_\ell$, otherwise define $\mathsf{tip}(\mathcal{CC}) = B_k$.

Updating chain certificates. Each replica stores $\mathcal{CC}$, the highest chain certificate it has ever received. Any time a new block certificate is received, the replica updates its highest ranked chain certificate using the comparison rule described earlier.

4.1 Steady State Protocol

Our protocol executes the following steps in iterations within a view v.

Propose. The leader L of view v proposes a block $B_k := (b_k, H(B_{k-1}))$ by broadcasting $\langle \mathsf{propose}, B_k, v, \mathcal{C}_v(B_{k-1})\rangle_L$. The proposal contains a block at height-k extending a block B_{k-1} at height $k-1$, the view number v, and a view-v certificate for B_{k-1}. The leader makes such a proposal as soon as it receives a view-v certificate for B_{k-1}. The first view-v certificate is obtained during the view-change process as will be described in the next subsection.

Vote. When a replica r receives the first proposal for B_k either from L or through some other replica, if r hasn't received a proposal for an equivocating block, i.e., it has not detected a leader equivocation in view v, it broadcasts a vote for B_k in the form of $\langle \mathsf{vote}, B_k, v\rangle_r$, and forwards the proposal to all replicas. It also starts a synchronous $\mathsf{commit\text{-}timer}_{k,v}$ and sets it to 2Δ.

Let v be the view number and replica L be the leader of view v. While in view v, a replica r runs the following protocol:

1. **Propose.** If replica r is the leader L, upon receiving $\mathcal{C}_v(B_{k-1})$, it broadcasts $\langle \mathsf{propose}, B_k, v, \mathcal{C}_v(B_{k-1})\rangle_L$ where B_k extends B_{k-1}.

2. **Vote.** Upon receiving the first proposal $\langle \mathsf{propose}, B_k, v, \mathcal{C}_v(B_{k-1})\rangle_L$ with a valid view v certificate for a block at height $k-1$ (not necessarily from L) where B_k extends B_{k-1}, if no leader equivocation is detected, forward the proposal to all replicas, broadcast a vote in the form of $\langle \mathsf{vote}, B_k, v\rangle_r$, set $\mathsf{commit\text{-}timer}_{v,k}$ to 2Δ, and start counting down.

3. **(Non-blocking) Commit rules.** Replica r commits block B_k using either of the following rules if r is still in view v:

   (a) **Responsive commit.** On receiving $\lfloor 3n/4 \rfloor + 1$ votes for B_k, i.e., $\mathcal{C}_v^{3/4}(B_k)$, commit B_k and all its ancestors immediately. Broadcast $\langle \mathsf{notify}, \mathcal{C}_v^{3/4}(B_k), v\rangle_r$, and abort $\mathsf{commit\text{-}timer}_{v,k}$.

   (b) **Synchronous commit.** If $\mathsf{commit\text{-}timer}_{v,k}$ reaches 0, commit B_k and all its ancestors.

4. **(Non-blocking) Blame and quit view.**

   - **Blame if no progress.** For $p > 0$, if fewer than p proposals trigger r's votes in $(2p+4)\Delta$ time in view v, broadcast $\langle \mathsf{blame}, v\rangle_r$.
   - **Quit view on $f+1$ blame messages.** Upon gathering $f+1$ distinct $\langle \mathsf{blame}, v\rangle_r$ messages, broadcast them, abort all view v timers, and quit view v.
   - **Quit view on detecting equivocation.** If leader equivocation is detected, broadcast the equivocating proposals signed by L, abort all view v timers, and quit view v.

Figure 1: Steady state protocol for optimal optimistic responsiveness with 2Δ-synchronous latency and $> 3n/4$-sized quorum.

Observe that the certificate in the proposal need not be the same as the certificate that replica r has obtained. Specifically, replica r can vote for a proposal containing a synchronous certificate for the previous block even if it holds a responsive certificate for the same block, and vice versa.

Commit. The protocol includes two commit rules and the replica commits using the rule that is triggered first. In a responsive commit, a replica commits block B_k and its ancestors immediately if the replica receives $> 3n/4$ votes for B_k in view v. Note that a responsive commit doesn't depend on the $\mathsf{commit\text{-}timer}$ and Δ, and a replica can commit at the actual speed of the network (δ). In addition, the replica immediately notifies the strong certificate $\mathcal{C}_v^{3/4}(B_k)$ to all replicas. This is critical to maintain safety. When a replica's $\mathsf{commit\text{-}timer}_{v,k}$ for B_k expires in view v, the replica synchronously commits B_k and all its ancestors. When a replica commits B_k, it aborts commit-timers for all its ancestors.

The commit step is non-blocking and it does not affect the critical path of progress. The leader can make a proposal for the next block as soon as it receives a certificate for the previous block independent of whether replicas have committed blocks for previous heights.

Note that if an honest replica commits a block B_k in view v using one of the rules, it is not necessary that all honest replicas commit B_k in view v using the same rule, or commit B_k at all. Some Byzantine replicas may decide to send votes to only a few honest replicas causing some honest replicas to commit using a responsive rule whereas some others using a synchronous rule. A Byzantine leader could send an equivocating block to some honest replicas and prevent them from committing. The protocol ensures safety despite all inconsistencies introduced by Byzantine replicas.

Blame and quit view. A view-change is triggered when replicas observe lack of progress or an equivocating proposal from the current leader. If an honest replica learns an equivocation, it broadcasts $\langle \mathsf{blame}, v\rangle_r$

Let L and L' be the leaders of view v and $v+1$, respectively. Each replica r runs the following steps.

i) **Status.** Wait for 2Δ time. Until this time, if a replica receives any chain certificates, the replica updates its chain certificate $\mathcal{CC}$ to the highest possible rank. Set lock_{v+1} to be the highest ranked chain certificate at the end of the 2Δ wait. Send $\langle \mathsf{status}, \mathsf{lock}_{v+1}\rangle_r$ to L'. Enter view $v+1$.

ii) **New-view.** The new leader L' waits for 2Δ time after entering view $v+1$. L' broadcasts $\langle \mathsf{new\text{-}view}, v+1, \mathsf{lock}_{v+1}\rangle_{L'}$, where lock_{v+1} is the highest ranked chain certificate known to L' after this wait.

iii) **First vote.** Upon receiving the first $\langle \mathsf{new\text{-}view}, v+1, \mathsf{lock}'\rangle_{L'}$, if $\mathsf{lock}_{v+1} \leq \mathsf{lock}'$, then broadcast $\langle \mathsf{new\text{-}view}, v+1, \mathsf{lock}'\rangle_{L'}$ and $\langle \mathsf{vote}, \mathsf{tip}(\mathsf{lock}'), v+1\rangle_r$.

Figure 2: View-change protocol for optimal optimistic responsiveness with 2Δ-synchronous latency and $> 3n/4$-sized quorum.

message along with the equivocating proposals and quits view v. The equivocating proposals serve as a proof of misbehavior and all honest replicas blame the leader to trigger a view-change. To ensure progress, the leader is expected to propose at least one block every 2Δ time that trigger's the replica's vote. Otherwise, replicas blame the current leader. When an honest replica receives a blame certificate ($f+1$ blame messages), it broadcasts the blame certificate, quits current view v and stops participating in view v. All replicas receive the blame certificate within Δ time due to synchrony assumption and quit view v.

We now provide some intuition on why either of these commit rules are safe within a view. We discuss safety across views in the subsequent section.

Why does a responsive commit ensure safety within a view? Consider an honest replica r that responsively commits a block B_k at time t. This is because it received $\lfloor 3n/4 \rfloor + 1$ votes for B_k by time t and it did not observe any equivocation until then. It is easy to see that if there exists $\lfloor 3n/4 \rfloor + 1$ votes for B_k, no other equivocating block $B'_{k'}$ at any height k' can be committed responsively due to a simple quorum intersection argument. Under a minority corruption, any two quorums of size $\lfloor 3n/4 \rfloor + 1$ intersect in $f+1$ replicas out of which at least one replica is honest. This honest replica will not vote for two equivocating blocks.

A synchronous commit of an equivocating block cannot happen due to the following reason. Since replica r hasn't received an equivocation until time t, no replica has voted for an equivocating proposal until time $t-\Delta$. Hence, their synchronous 2Δ window for committing an equivocating block ends at time $> t+\Delta$. A commit for B_k at time t implies that some honest replica must have voted and forwarded the corresponding proposal before t and this will arrive by time $t+\Delta$ at all honest replicas. This will prevent any other replica from committing an equivocating block. Observe that a responsive commit does not imply that an equivocating block $B'_{k'}$ will not be certified; hence, during a view-change, we need to be able to carefully extend the chain that contains a block that has been committed by some other replica.

Why does a synchronous commit ensure safety within a view? Consider an honest replica r that votes for a block B_k at time t and commits at time $t+2\Delta$ because it did not observe an equivocation until then. This implies (i) all honest replicas have received B_k by time $t+\Delta$, and (ii) no honest replica has voted for an equivocating block by time $t+\Delta$. Due to the rules of voting, no honest replica will vote for an equivocating block in this view after time $t+\Delta$ ruling out an equivocating commit through either of the two rules.

4.2 View-change

The view-change protocol is responsible for replacing a possibly faulty leader with a new leader to maintain liveness. In the process, it needs to maintain safety of any commit that may have happened in the previous views.

Status. After quitting view v, a replica waits for 2Δ time before entering view $v+1$. The 2Δ wait ensures that all honest replicas receive a certificate for a block B_k before entering view $v+1$ if some honest replica committed B_k in view v. This is critical to maintain the safety of the commit in view v. The replica updates its chain certificate $\mathcal{CC}$ to the highest possible rank and sets lock_{v+1} to $\mathcal{CC}$. It then sends lock_{v+1} to the next leader L' via a $\langle \mathsf{status}, \mathsf{lock}_{v+1} \rangle_r$.

New-view. Leader L' waits 2Δ time after entering view $v+1$ to receive a status message from all honest replicas. Based on these status messages, L' picks the highest ranked chain certificate lock'. It creates a $\mathsf{new\text{-}view}$ message $\langle \mathsf{new\text{-}view}, v+1, \mathsf{lock}' \rangle_{L'}$ and sends it to all honest replicas. The highest ranked chain certificate across all honest replicas at the end of view v helps an honest leader to appropriately send a new-view message that will be voted upon by all honest replicas and maintain the liveness of the protocol.

First vote. Upon receiving a $\langle \mathsf{new\text{-}view}, v+1, \mathsf{lock}' \rangle_{L'}$ message, if the certified block lock' has a rank no lower than r's locked chain certificate lock_{v+1}, then it forwards the $\mathsf{new\text{-}view}$ message to all replicas and broadcasts a vote for it.

Next, we provide some intuition on how the view-change protocol ensures safety across views and liveness.

Why do replicas lock on chains extending committed blocks before entering the next view? In this protocol, we use locks to ensure safety. The protocol guarantees that if an honest replica commits a block (through either rules), then at the end of the view all honest replicas will lock on a chain certificate that extends the committed block. At the start of the next view, when the leader sends a lock through the new-view message, by testing whether this lock is higher than the lock stored locally, an honest replica ensures that only committed blocks are extended.

What ensures that replicas lock on chains extending committed blocks before entering the next view? Suppose an honest replica r responsively commits a block B_k in view v at time t. Notice that no honest replica has entered view $v+1$ by time $t+\Delta$; otherwise, replica r must have received blame certificate by time t due to the 2Δ wait in the status step. Replica r's notify containing $\mathcal{C}_v^{3/4}(B_k)$ reaches all honest replicas by time $t+\Delta$. As noted earlier, there may exist a synchronous certificate for an equivocating block $B'_{k'}$ in view v i.e., $\mathcal{C}_v^{1/2}(B'_{k'})$ (for any value of k', e.g., $k' > k$). However, a chain certificate $\mathcal{CC}$ containing $\mathcal{C}_v^{3/4}(B_k)$ ranks higher than a chain certificate $\mathcal{CC}'$ containing $\mathcal{C}_v^{1/2}(B'_{k'})$ as per our ranking rule and all honest replicas lock on $\mathcal{CC}$ or higher before entering view $v+1$.

If replica r synchronously commits B_k in view v at time t then, replica r voted for B_k at time $t-2\Delta$ didn't detect an equivocation or blame certificate by time t. This implies all honest replicas will vote for B_k at time $t-\Delta$ and receive $\mathcal{C}_v^{1/2}(B_k)$ by time t. As noted earlier, there doesn't exist an equivocating certificate in view v during synchronous commit. Hence, all honest replicas will lock on a chain certificate $\mathcal{CC}$ containing $\mathcal{C}_v^{1/2}(B_k)$ before entering view $v+1$.

How does the protocol ensure liveness? The protocol ensures liveness by allowing a new honest leader to *always* propose a block that will be voted for by all honest replicas. All honest replicas send their locked chain certificate to the next leader L' at the start of the new view in a status message. L' could be lagging and enter $v+1$ Δ time after other replicas. Thus, it waits 2Δ time to collect chain certificates from all honest replicas. If L' is honest, it extends the highest ranked chain certificate lock'. This suffices to ensure that all honest replicas vote on its proposal, in turn, ensuring liveness when the leader is honest. In the new view, as long as the leader keeps proposing valid blocks, honest replicas will vote and keep committing new blocks.

4.3 Safety and Liveness

We say a *block B_k is committed directly in view v* if an honest replica successfully runs the responsive commit rule 3(a), or the synchronous commit rule 3(b) on block B_k. Similarly, we say a *block B_k is committed indirectly* if it is a result of directly committing a block B_ℓ ($\ell > k$) that extends B_k but is not equal to B_k.

We say that a replica is *in view v at time t* if the replica executes the *Enter view v* of Step (i) in Fig 2 by time t and did not execute any *Quit view* of Step 5 in Fig 1 for view v at time t or earlier.

Claim 5. *If a block B_k is committed directly in view v, using the responsive commit rule, then a responsive certificate for an equivocating block $B'_{k'}$ in view v does not exist.*

Proof. If replica r commits B_k due to the responsive commit rule in view v, then r must have received $\lfloor 3n/4 \rfloor + 1$ votes, i.e., $\mathcal{C}_v^{3/4}(B_k)$, forming a quorum Q for B_k in view v. Since a responsive block certificate requires $\lfloor 3n/4 \rfloor + 1$ votes, a quorum intersection argument shows that a responsive certificate for equivocating block $B'_{k'}$ cannot exist. Specifically, if an equivocating block receives $\lfloor 3n/4 \rfloor + 1$ votes forming a quorum Q', then we have $|Q \cap Q'| \geq 2\lfloor 3n/4 \rfloor + 2 - n \geq \lfloor n/2 \rfloor + 1$. Since at least one replica in $Q \cap Q'$ must be honest, it will not vote for two equivocating blocks. □

Claim 6. *If a block B_k is committed directly in view v, using the responsive commit rule, then there does not exists a chain certificate $\mathcal{CC}$ in view v, such that $\mathcal{CC} > (\mathcal{C}_v^{3/4}(B_k), \perp)$ where a block in $\mathcal{CC}$ equivocates B_k.*

Proof. By Claim 5, no equivocating block can have a responsive block certificate. So all responsive block certificates must extend B_k. Since we assume that $\mathcal{CC} > (\mathcal{C}_v^{3/4}(B_k), \perp)$ then it must be that either $\mathcal{CC}$ is of the form $(\mathcal{C}_v^{3/4}(B_k), \mathcal{C}_v^{1/2}(B_\ell))$ and by definition B_ℓ extends B_k, or $\mathcal{CC}$ is of the form $(\mathcal{C}_v^{3/4}(B_{k'}), \mathcal{C}_v^{1/2}(B_{\ell'}))$ where $B_{k'}$ extends B_k and again by definition and transitivity $B_{\ell'}$ must extend B_k. □

Claim 7. *If a block B_k is committed directly in view v using the synchronous commit rule, then a block certificate for an equivocating block $B'_{k'}$ does not exist in view v.*

Proof. Suppose replica r directly commits block B_k at time t using the synchronous commit rule. So replica r voted and forwarded the proposal for B_k at time $t - 2\Delta$ and its $\textsf{commit-timer}_{v,k}$ expired without detecting equivocation. By synchrony assumption, all replicas receive the forwarded proposal for B_k by time $t - \Delta$. Since they do not vote for equivocating blocks, they will not vote for $B'_{k'}$ received at time $> t - \Delta$. Moreover, no honest replica must have voted for an equivocating block at time $\leq t - \Delta$. Otherwise, replica r would have received the equivocating proposal by time t and it wouldn't have committed. Since no honest replica votes for an equivocating block, $B'_{k'}$ will not be certified. □

Claim 8. *If a block B_k is committed directly in view v using the responsive commit rule, then all honest replicas receive $\mathcal{C}_v^{3/4}(B_k)$ before entering view $v + 1$.*

Proof. Suppose replica r directly commits block B_k at time t using the responsive commit rule. So replica r notifies the certificate $C_v^{3/4}(B_k)$ which arrives at all honest replicas at time $\leq t + \Delta$. No honest replica r' is at view $v + 1$ at time $\leq t + \Delta$. Because if r' was at view $v + 1$ at time $\leq t + \Delta$, then due to the 2Δ wait in the status step, r' would have quit view v at time $\leq t - \Delta$. Hence, replica r must have received a blame certificate or equivocation at time $\leq t$ and it wouldn't commit. Hence, all honest replicas receive $\mathcal{C}_v^{3/4}(B_k)$ before entering view $v + 1$. □

Claim 9. *If a block B_k is directly committed in view v at time t using the synchronous commit rule, then all honest replicas receive $\mathcal{C}_v(B_k)$ before entering view $v + 1$.*

Proof. We will prove that if a block B_k is directly committed in view v at time t using the synchronous commit rule, then (i) all honest replicas are in view v at time $t - \Delta$, (ii) all honest replicas vote for B_k at time $\leq t - \Delta$, and (iii) all honest replicas receive $\mathcal{C}_v(B_k)$ before entering view $v + 1$. Part (iii) is the desired claim.

Suppose honest replica r synchronously commits B_k at time t in view v. It votes for block B_k at time $t - 2\Delta$. Thus, replica r entered view v at time $\leq t - 2\Delta$. Due to the 2Δ wait before sending a status message, replica r must have sent a blame certificate or equivocating blocks at time $\leq t - 4\Delta$ which arrives all honest replicas at time $\leq t - 3\Delta$. Hence, all honest replicas enter view v at time $\leq t - \Delta$ (again due to 2Δ wait in the status step). Also, observe that no honest replica has quit view v at time $\leq t - \Delta$. Otherwise, replica r hears of blame certificate or equivocation at time $\leq t$. This proves part (i).

Replica r received a proposal for B_k which contains $\mathcal{C}_v(B_{k-1})$ at time $t - 2\Delta$. Thus, replica r's vote and forwarded proposal for B_k arrives all honest replicas by time $t - \Delta$. No honest replica has voted for an equivocating block at time $\leq t - \Delta$; otherwise replica r would have received an equivocation at time $\leq t$. Thus, all honest replicas will vote for B_k at time $\leq t - \Delta$. This proves part (ii).

The votes from all honest replicas will arrive at all honest replicas by time $\leq t$. By part(i) of the claim and 2Δ wait in the status step, honest replicas do not enter view $v + 1$ at time $\leq t + \Delta$. Thus, all honest replicas receive $\mathcal{C}_v(B_k)$ before entering view $v + 1$.

□

Lemma 10. *If an honest replica directly commits a block B_k in view v, then: (i) all honest replicas have lock_{v+1} such that $\mathsf{tip}(\mathsf{lock}_{v+1})$ extends B_k, (ii) for any chain certificate $\mathcal{CC}$ that the adversary can create and any honest lock lock_{v+1}, either $\mathcal{CC} < \mathsf{lock}_{v+1}$ or $\mathsf{tip}(\mathcal{CC})$ extends B_k.*

Proof. By Claim 8 and Claim 9, all honest replicas will receive $\mathcal{C}_v(B_k)$ before entering view $v + 1$. If B_k is committed using the synchronous commit rule, then by Claim 7, there doesn't exists a view v certificate that equivocates B_k. Since, honest replicas lock on highest ranked chain certificate, $\mathsf{tip}(\mathsf{lock}_{v+1})$ must extend B_k. Similarly, if B_k is committed using the responsive commit rule, then by Claim 6 there doesn't exist chain certificate $\mathcal{CC}$ such that $\mathcal{CC} > (\mathcal{C}_v(B_k), \perp)$ and $\mathcal{CC}$ equivocates B_k. Thus, $\mathsf{tip}(\mathsf{lock}_{v+1})$ must extend B_k. By similar argument, any $\mathcal{CC}$ that an adversary creates either has $\mathcal{CC} < \mathsf{lock}_{v+1}$ or $\mathsf{tip}(\mathcal{CC})$ extends B_k.

□

The following lemma considers safety of directly committed blocks across views.

Lemma 11 (Unique Extensibility)**.** *If an honest replica directly commits a block B_k in view v, and $\mathcal{C}_{v'}(B_{k'})$ is a view $v' > v$ block certificate, then $B_{k'}$ extends B_k. Moreover, all honest replicas have $\mathsf{lock}_{v'}$ such that $\mathsf{tip}(\mathsf{lock}_{v+1})$ extends B_k.*

Proof. The proof is by induction on the view $v' > v$. For a view v', we prove that if $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}'))$ exists then it must extend B_k. A simple induction then shows that all later block certificates must also extend $\mathsf{tip}(\mathsf{lock}')$, this follows directly from the Vote rule in line 2.

For the base case, where $v' = v + 1$, the proof that $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}'))$ extends B_k follows from Lemma 10 because the only way such a block can be certified is some honest votes for it. However, all honest are locked on a block that extends B_k and a chain certificate with a higher rank for an equivocating block does not exist. Thus, no honest replica will first vote (Figure 2 step iii) for a block that does not extend B_k. The second part follows directly from Lemma 10.

Given that the statement is true for all views below v', the proof that $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}'))$ extends B_k follows from the induction hypothesis because the only way such a block can be certified is if some honest votes for it. An honest party with a lock lock will vote only if $\mathsf{tip}(\mathsf{lock}_{v'})$ has a valid block certificate and $\mathsf{lock} \geq \mathsf{lock}_{v'}$. Due to Lemma 10 and the induction hypothesis on all block certificates of view $v < v'' < v'$ is must be that $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}))$ extends B_k.

□

Theorem 12 (Safety)**.** *Honest replicas do not commit conflicting blocks for any height ℓ.*

Proof. Suppose for contradiction that two distinct blocks B_ℓ and B'_ℓ are committed at height ℓ. Suppose B_ℓ is committed as a result of B_k being directly committed in view v and B'_ℓ is committed as a result of $B'_{k'}$ being directly committed in view v'. This implies B_k extends B_ℓ and $B'_{k'}$ extends B'_ℓ. Without loss of generality, assume $v \leq v'$; if $v = v'$, further assume $k \leq k'$. If $v = v'$ and $k \leq k'$, by Claim 6 and Claim 7, $B'_{k'}$ extends B_k. Similarly, if $v < v'$, by Lemma 11, $B'_{k'}$ extends B_k. Thus, $B'_\ell = B_\ell$. □

Theorem 13 (Liveness)**.** *All honest replicas keep committing new blocks.*

Proof. In a view, a leader has to propose at least p blocks that trigger honest replicas votes in $(2p + 4)\Delta$ time. As long as the leader proposes at least p valid blocks, honest replicas will keep voting for the blocks and keep committing the proposed blocks. If the Byzantine leader equivocates or proposes less than p blocks, a view-change will occur. Eventually, there will be an honest leader due to round-robin leader election.

Next, by Lemma 10, all honest replicas lock on a highest certified chain before entering a new view. The leader may enter the new view Δ time later than others; hence need to wait for 2Δ before proposing. Due to 2Δ wait, the new leader receives the highest locked certified chains from all honest replicas. If the leader is honest, the leader will extend upon the tip of the highest ranked certified chain. Honest replicas will vote for the new block since the lock sent by the leader is at least as large as their lock. Moreover, the honest leader doesn't equivocate and keeps proposing at least p blocks. This prevents forming a blame certificate to cause view-change and all honest replicas will keep committing new blocks. □

5 Optimal Optimistic Responsiveness with Δ-synchronous Latency and n-sized Quorum

Recall that our lower bound in Section 3 showed that we cannot have the following two commit latencies simultaneously: (i) a responsive commit with $O(\delta)$ latency where $\geq n - 2f$ faults are tolerated in the responsive mode, and (ii) a synchronous commit with $< 2\Delta$ latency simultaneously. The previous section showed a protocol for $n = 2f + 1$ with optimal synchronous latency of 2Δ when fewer than n replicas participate in the responsive mode. In this section, we will present an optimal synchronous latency of $\Delta + O(\delta)$ when all $n = 2f + 1$ replicas participate in a responsive commit. For a synchronous commit, an honest replica commits a block in $\Delta + O(\delta)$ time after receiving a valid proposal for the block if no equivocating proposals are received and $f + 1$ replicas have voted for the block. A responsive commit completes immediately when a replica receives *acknowledgments* for a block from *all* replicas and no equivocation has been detected. The responsive commit has a commit latency of 2δ. The protocol maintains a commit latency of 2δ as long as all replicas are behaving honestly and responding promptly.

Unlike the protocol in the previous section where a replica immediately votes for a valid proposal, in this protocol a replica sends an acknowledgements (ack) for the proposed block immediately and votes only if it does not detect any equivocation Δ time after its acknowledgement. We define a set of $2f + 1$ signed ack messages from the same view for a block B_k as *full certificate* for B_k and represent it as $\mathcal{C}_v^f(B_k)$. As before, we call a set of $f + 1$ signed vote messages for B_k as *synchronous certificate* and represent it as $\mathcal{C}_v^{1/2}(B_k)$. Whenever the distinction is not important, we represent certificates as $\mathcal{C}_v(B_k)$. Later in the section, we show that if there exists a certificate (either full or synchronous) for a block B_k in a view v, there cannot exists a certificate for an equivocating block in view v. For this reason, we define a simple certificate ranking rule. Certified blocks are first ranked by views and then by height, i.e., (i) blocks certified in a higher view have a higher rank, and (ii) for blocks certified in the same view, a higher height implies a higher rank.

5.1 Steady State Protocol

The steady state protocol runs following steps in iterations with a view v.

Propose. The Leader L of view v proposes a block B_k by extending a highest certified block $\mathcal{C}_{v'}(B_{k-1})$ known to L. If the leader has just entered steady state after view-change, it waits for 2Δ time to receive the highest certified blocks from all honest replicas in which case $v' < v$. Otherwise, the leader proposes a new block as soon as it learns a certificate for the previous block proposed in the same view.

Ack. The protocol includes an additional ack step before voting. A replica r broadcasts an ack $\langle \mathsf{ack}, B_k, v\rangle$ for a proposed block B_k if (i) it hasn't detected any equivocation in view v, and (ii) $\mathcal{C}_{v'}(B_{k-1})$ has rank equal to or higher than its own locked block. Once replica r sends an ack, it starts a $\mathsf{vote\text{-}timer}_{v,k}$ initialized to Δ time and starts counting down. Replica r also forwards the received proposal for B_k.

Vote. When $\mathsf{vote\text{-}timer}_{v,k}$ for block B_k expires, if replica r hasn't heard of any equivocation in view v, it broadcasts a vote $\langle \mathsf{vote}, B_k, v\rangle$ for B_k.

Commit. Replica r can commit either responsively or synchronously based on which rule is triggered first. A responsive commit is triggered when r receives $2f + 1$ ack messages for B_k, i.e., $\mathcal{C}_v^f(B_k)$ and r commits B_k and all its ancestors immediately. Replica r stops $\mathsf{vote\text{-}timer}_{v,k}$ and notifies $\mathcal{C}_v^f(B_k)$ to all honest

Let v be the view number and replica L be the leader of the current view. While in view v, a replica r runs the following protocol in steady state.

1. **Propose.** If replica r is the leader L, upon receiving $\mathcal{C}_{v'}(B_{k-1})$, it broadcasts $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1})\rangle_L$ where B_k extends B_{k-1}. If this is the first block in this view, i.e., $v' < v$, then it waits for an additional 2Δ time after entering the view before proposing.

2. **Ack.** Upon receiving the first proposal $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1})\rangle_L$ (not necessarily from L) at height k in view v, if $\mathcal{C}_{v'}(B_{k-1})$ is ranked greater than or equal to its locked block, forward the proposal to all replicas and broadcast an acknowledgment in the form of $\langle \mathsf{ack}, B_k, v\rangle$. Set $\mathsf{vote\text{-}timer}_{v,k}$ to Δ and start counting down.

3. **Vote**. If $\mathsf{vote\text{-}timer}_{v,k}$ reaches 0, send a vote for B_k in the form of $\langle \mathsf{vote}, B_k, v\rangle$.

4. **(Non-blocking) Commit.** Replicas can commit block B_k using either of the following rules:

   (a) **Responsive commit.** On receiving $2f+1$ acks for B_k, i.e., $\mathcal{C}_v^f(B_k)$ in view v, commit B_k and all its ancestors immediately. Stop $\mathsf{vote\text{-}timer}_{v,k}$ and notify the certificate $\mathcal{C}_v^f(B_k)$.

   (b) **Synchronous commit.** On receiving $f+1$ votes for B_k, i.e., $\mathcal{C}_v^{1/2}(B_k)$ in view v, commit B_k and all its ancestors immediately. Notify the certificate $\mathcal{C}_v^{1/2}(B_k)$ to all replicas.

5. **(Non-blocking) Blame and quit view.**

   - **Blame if no progress.** For $p > 0$, if fewer than p proposals trigger r's votes in $(3p+4)\Delta$ time in view v, broadcast $\langle \mathsf{blame},\ v\rangle_r$.
   - **Quit view on $f+1$ blame messages.** Upon gathering $f+1$ distinct $\langle \mathsf{blame}, v\rangle_r$ messages, broadcast them, abort all view v timers, and quit view v.
   - **Quit view on detecting equivocation.** If leader equivocation is detected, broadcast the equivocating proposals signed by L, abort all view v timers, and quit view v.

Figure 3: Steady state protocol for optimal optimistic responsiveness with Δ-synchronous latency and n-sized quorum.

replicas. Similarly, replica r synchronously commits B_k along with its all ancestors when it receives $f+1$ vote messages for B_k, i.e., $\mathcal{C}_v^{1/2}(B_k)$. r also notifies $\mathcal{C}_v^{1/2}(B_k)$ to all replicas. Like before, both the commit paths are non-blocking and the leader can keep proposing as soon as it learns a certificate for previous block.

Blame and quit view step remains identical to the one in Figure 2.

Next, we provide some intuition on why either of these commit rules are safe within a view.

Why does a responsive commit ensure safety within a view? A replica commits a block B_k responsively only when it receives acks from all replicas which includes all honest replicas. This implies no honest replicas will either ack or vote for an equivocating block $B'_{k'}$ at any height k'. Hence, an equivocating block $B'_{k'}$ will neither receive $2f+1$ acks nor $f+1$ votes required for a block to be committed.

Why does a synchronous commit ensure safety within a view? An honest replica r synchronously commits a block B_k at time t when it receives $f+1$ votes for B_k and hears no equivocation by time t. This implies no honest replica has voted for an equivocating block $B'_{k'}$ by time $t-\Delta$. At least one honest replica r' sent an ack for B_k by time $t-\Delta$. r's ack arrives all honest replicas by time t. Hence, honest replicas will neither ack nor vote for an equivocating block $B'_{k'}$ after time t. This also prevents honest replicas from committing an equivocating block after time t.

Let L and L' be the leader of view v and $v+1$, respectively. Each replica r runs the following steps.

i) **Status.** Wait for 2Δ time. Pick the highest certified block $B_{k'}$ with certificate $\mathcal{C}_{v'}(B_{k'})$. Lock on $\mathcal{C}_{v'}(B_{k'})$, and send $\mathcal{C}_{v'}(B_{k'})$ to the new leader L'. Enter view $v+1$.

Figure 4: View-change protocol for optimal optimistic responsiveness with Δ-synchronous latency and n-sized quorum.

5.2 View Change Protocol

The view-change protocol involves only sending status message. During this step, a replica r waits for 2Δ time and locks on the highest certified block $\mathcal{C}_{v'}(B_{k'})$ known to r. It forwards $\mathcal{C}_{v'}(B_{k'})$ to the next leader and enters next view. As shown in Lemma 17, the 2Δ wait ensures that all honest replicas lock on the highest-certified block corresponding to a commit at the end of the view, which, in turn, is essential to maintain the safety of the protocol. The status message along with the accompanying 2Δ wait in the propose step ensures liveness, i.e., it ensures that an honest leader proposes a block that extends locks held by all honest replicas and hence will be voted upon by all honest replicas.

5.3 Safety and Liveness

We say a block B_k is committed *directly* in view v if any of the two commit rules are triggered for B_k. Similarly, a block B_k is committed *indirectly* if it is a result of directly committing a block B_ℓ ($\ell > k$) that extends B_k but is not equal to B_k.

Claim 14. *If an honest replica directly commits a block B_k in view v using the responsive commit rule, then there does not exist a certificate for an equivocating block in view v.*

Proof. If replica r commits B_k in view v using responsive commit rule, r must have received $2f+1$ acks, i.e., $\mathcal{C}_v^f(B_k)$. This implies all honest replicas have sent ack for B_k and no honest replica would send ack or vote for an equivocating block $B'_{k'}$ in view v. Since, a certificate for $B'_{k'}$ requires either $2f+1$ acks for full certificate or at least one vote from an honest replica for synchronous certificate, a certificate for an equivocating block cannot exist. ☐

Claim 15. *If an honest replica directly commits a block B_k in view v using the synchronous commit rule, then there does not exist a certificate for an equivocating block in view v.*

Proof. Suppose replica r synchronously commits B_k in view v at time t without detecting an equivocation. Observe that an equivocating responsive certificate does not exist since replica r would not ack two equivocating blocks. Hence, we need to only show that a synchronous equivocating certificate does not exist. We show it with the following two arguments. First, r votes for B_k at time $leqt$ and sends an ack for B_k at time $\leq t-\Delta$. r's ack for B_k arrives all honest replicas by time t. Hence, no honest replica will vote for an equivocating block $B'_{k'}$ at time $\geq t$. Second, no honest replica must have sent an equivocating ack at time $\leq t-\Delta$. Otherwise, replica r would not have committed. This also implies that no honest replica will vote for an equivocating block at time $\leq t$ (due to the Δ wait between ack and vote).

☐

Lemma 16. *If an honest replica directly commits a block B_k in view v then, (i) there doesn't exist an equivocating certificate in view v, and (ii) all honest replicas receive $\mathcal{C}_v(B_k)$ before entering view $v+1$.*

Proof. Part(i) follows immediately from Claim 14 and Claim 15.

Suppose replica r commits B_k at time t either responsively or synchronously. r notifies the certificate ($\mathcal{C}_v^f(B_k)$ or $\mathcal{C}_v^{1/2}(B_k)$) which arrives at all honest replicas at time $\leq t+\Delta$. Observe that no honest replica r' has entered view $v+1$ at time $\leq t+\Delta$. Otherwise, due to 2Δ wait before entering the new view, r' must have sent either equivocating or a blame certificate at time $\leq t-\Delta$; r must have received the blame certificate at time $\leq t$. It would have quit view and not committed. Hence, all honest replicas receive $\mathcal{C}_v(B_k)$ before entering view $v+1$. □

Lemma 17. *If an honest replica directly commits a block B_k in view v, then all honest replicas lock on a certified block that ranks higher than or equal to $\mathcal{C}_v(B_k)$ before entering view $v+1$.*

Proof. By Lemma 16 part (ii), all honest replicas will receive $\mathcal{C}_v(B_k)$ before entering view $v+1$. By Lemma 16 part (i), no equivocating certificate exists in view v. Since replicas lock on the highest certified block as soon as they enter the next view, all honest replicas lock on a certified block that ranks higher than or equal to $\mathcal{C}_v(B_k)$ before entering view $v+1$. □

Lemma 18 (Unique Extensibility)**.** *If an honest replica directly commits a block B_k in view v, then any certified block that ranks equal to or higher than $\mathcal{C}_v(B_k)$ must extend B_k.*

Proof. Any certified block $B'_{k'}$ in view v of rank equal to or higher than $\mathcal{C}_v(B_k)$ must extend B_k. Otherwise, $B'_{k'}$ equivocates B_k and by Lemma 16, $B'_{k'}$ cannot be certified in view v. For views higher than v, we prove the lemma by contradiction. Let S be the set of certified blocks that rank higher than $\mathcal{C}_v(B_k)$, but do not extend B_k. Suppose for contradiction $S \neq \emptyset$. Let $\mathcal{C}_{v^*}(B_{\ell^*})$ be a lowest ranked block in S. Also, note that if B_{ℓ^*} does not extend B_k, then B_{ℓ^*-1} does not extend B_k either.

For $\mathcal{C}_{v^*}(B_{\ell^*})$ to exist, some honest replica must vote for B_{ℓ^*} in view v either upon receiving a proposal $\langle \mathsf{propose}, B_{\ell^*}, v^*, \mathcal{C}_{v'}(B_{\ell^*-1})\rangle$ for $v' < v$ or $\langle \mathsf{propose}, B_{\ell^*}, v^*, \mathcal{C}_{v^*}(B_{\ell^*-1})\rangle$. If it is the former, then $\mathcal{C}_{v'}(B_{\ell^*-1})$ must rank higher than or equal to $\mathcal{C}_v(B_k)$. This is because due to Lemma 17 all honest replicas will have received a certified block that ranks higher than or equal to $\mathcal{C}_v(B_k)$ before entering view $v+1$. Moreover, replicas only lock on blocks of monotonically increasing ranks. However, since $v' < v^*$, the rank of $\mathcal{C}_{v'}(B_{\ell^*-1})$ is less than $\mathcal{C}_{v^*}(B_{\ell^*})$ by our certificate ranking rule. This contradicts the fact that $\mathcal{C}_{v^*}(B_{\ell^*})$ is a lowest ranked block in S. If it is the latter, then observe that $\mathcal{C}_{v^*}(B_{\ell^*-1})$ exists in view v^*. Again, this certificate is ranked higher than $\mathcal{C}_v(B_k)$ since $v^* > v$. Also, this certificate is ranked lower than $\mathcal{C}_{v^*}(B_{\ell^*})$ due to its height. Hence, this contradicts the fact that $\mathcal{C}_{v^*}(B_{\ell^*})$ is a lowest ranked block in S. □

Safety. The safety proof is identical to that of Theorem 12 except Lemma 18 needs to be invoked.

Liveness. The liveness proof is similar to that of Theorem 13.

6 Optimistic Responsiveness with Optimistically Responsive View-Change

The protocols in Section 4 and Section 5 are optimistically responsive in the steady-state. However, whenever a leader needs to be replaced, the view-change protocol must always incur a synchronous wait. This suffices if leaders are replaced occasionally, e.g., when a leader replica crashes. However, in a democracy-favoring approach it may be beneficial to replace leaders after every block, or every few blocks. In such a scenario, the synchronous wait during view-change will increase the latency of the protocol. For example, the protocol in Section 4 waits at least 4Δ time during view-change to ensure that the new leader collects status from all honest replicas. Thus, in an execution where leaders are changed after every block, even when the leader is honest, this protocol requires at least $4\Delta + O(\delta)$ for one block to be committed even during optimistic executions, and requires at least 6Δ when $< 3n/4$ replicas are honest.

In this section, we present a protocol that is optimistically responsive in both the steady state as well as view-change. In a world with rotating honest leaders, when $> 3n/4$ replicas are honest, this protocol can commit blocks in $O(\delta)$ time and replace leaders in $O(\delta)$ time. When more than $n/4$ replicas are malicious under the rotating honest leader setting, the protocol still commits in $5\Delta + O(\delta)$ time.

6.1 Steady State Protocol

We make following modifications to the steady state protocol in Section 4 to support a responsive view-change. In a synchronous commit, a replica commits within 3Δ time after voting if no equivocation or blame certificate has been received. The additional Δ wait in the synchronous commit accounts for the responsive view-change that may occur before all honest replicas receive a certificate for committed blocks. The $\mathsf{propose}$ and vote steps remain identical. However, after voting for B_k, the $\mathsf{commit\text{-}timer}_{v,k}$ is set to 3Δ time.

Pre-commit. The protocol includes an additional pre-commit step with two pre-commit rules active simultaneously. The pre-commit is identical to the commit step in the previous protocol. A replica pre-commits using the rule that is triggered first. In a responsive pre-commit, a replica r pre-commits a block B_k immediately when it receives $\lfloor 3n/4 \rfloor + 1$ votes for B_k, i.e., $\mathcal{C}_v^{3/4}(B_k)$ in view v and notifies $\mathcal{C}_v^{3/4}(B_k)$ to all replicas via $\langle \mathsf{notify}, \mathcal{C}_v^{3/4}(B_k), v\rangle_r$. If replica r hasn't blamed in view v, it broadcasts commit message via $\langle \mathsf{commit}, B_k, v\rangle_r$. Note that replica r doesn't send commit message if it has already blamed in view v. This is critical to ensure safety during responsive view-change. After responsive pre-commit, replica r resets its $\mathsf{commit\text{-}timer}_{v,k}$ to $\min(2\Delta, \mathsf{commit\text{-}timer}_{v,k})$.

In a synchronous pre-commit, a replica pre-commits B_k when its $\mathsf{commit\text{-}timer}_{v,k}$ reaches Δ. After pre-commit, replica r broadcasts $\langle \mathsf{commit}, B_k, v\rangle_r$ only if it hasn't blamed in view v.

Commit. In a responsive commit, a replica commits a block B_k immediately along with its ancestors when it receives $\lfloor 3n/4 \rfloor + 1$ commit messages for B_k. In a synchronous commit, a replica commits B_k and all its ancestors when its $\mathsf{commit\text{-}timer}_{v,k}$ expires and it doesn't detect an equivocation or blame certificate. We note that a replica can commit even if it has sent a blame message in view v as long as it hasn't detected an equivocation or blame certificate in the view. As before, the commit rules are non-blocking to rest of the execution.

Yield. When leader L wants to relinquish his leadership in view v, L broadcasts $\langle \mathsf{yield}, v\rangle_L$. The yield message forces an explicit view-change and useful for *democracy-favoring* leader policy and change leader after every block. Ideally, an honest leader issues yield after committing at least one block itself in view v.

Blame and quit view. The conditions for blaming and quitting a view are identical to earlier protocols. Additionally, replicas blame and quit view v on receiving a yield message from the current leader and sets $\mathsf{blame\text{-}timer}_v$ to 2Δ. We modify the blame message to include the highest ranked chain certificate $\mathcal{CC}$ known to a replica. The blame message for a replica r has the form $\langle \mathsf{blame}, v, \mathcal{CC}\rangle_r$. After quitting a view v, replica r sets $\mathsf{blame\text{-}timer}_v$ to 2Δ and starts counting down.

The requirements for a pre-commit in this protocol is identical to the requirements for a commit in the protocol in Section 4. Hence, a similar intuition for those steps apply here as well. The key differences are (i) if a replica r has blamed the leader, it does not send a commit message after pre-committing, and (ii) when a blame message is sent, the replica also broadcasts its highest chain certificate $\mathcal{CC}$ with it. Both of these steps are related to the safety of performing a responsive view-change and hence, we explain the intuition after we describe the view-change protocol.

6.2 View-change Protocol

Unlike a synchronous view-change as shown in Figure 2 that waits 2Δ before entering a new view, a responsive view-change allows replicas to quit current view and immediately transition to the next view without any delay. In the new view, a leader can also propose blocks without waiting for an additional 2Δ time. We make the following modifications to the view-change protocol to accommodate the responsive view-change.

Status. The status step includes two rules for entering into the new view. A replica r enters into view $v+1$ based on which rule is triggered first. A responsive rule is triggered when replica r receives a responsive blame certificate with a set $Q_B^{3/4}$ of $\lfloor 3n/4 \rfloor + 1$ blame messages in view v and enters view $v+1$ immediately. Replica r broadcasts the blame certificate to all replicas, updates its lock, lock_{v+1} to a highest ranked chain certificate and sends lock_{v+1} to the new leader L' via a status message. The responsive status rule ensures

Let v be the view number and replica L be the leader of the current view. While in view v, a replica r runs the following steps in iterations:

1. **Propose.** If replica r is the leader L, upon receiving $\mathcal{C}_v(B_{k-1})$, it broadcasts $\langle \textsf{propose}, B_k, v, \mathcal{C}_v(B_{k-1})\rangle_L$ where B_k extends B_{k-1}.

2. **Vote.** Upon receiving the first proposal $\langle \textsf{propose}, B_k, v, \mathcal{C}_v(B_{k-1})\rangle_L$ with a valid view v certificate for B_{k-1} (not necessarily from L) where B_k extends B_{k-1}, forward the proposal to all replicas, broadcast a vote in the form of $\langle \textsf{vote}, B_k, v\rangle_r$. Set $\textsf{commit-timer}_{v,k}$ to 3Δ and start counting down.

3. **Pre-commit.** Replica r pre-commits B_k using one of the following rules if r is still in view v:

   (a) **Responsive Pre-commit.** On receiving $\lfloor 3n/4 \rfloor + 1$ votes for B_k, i.e., $\mathcal{C}_v^{3/4}(B_k)$ in view v, pre-commit B_k, and broadcast a notify in the form of $\langle \textsf{notify}, \mathcal{C}_v^{3/4}(B_k), v\rangle_r$. If r hasn't sent a blame message in view v, broadcast $\langle \textsf{commit}, B_k, v\rangle_r$. Reset $\textsf{commit-timer}_{v,k}$ to $\min(2\Delta, \textsf{commit-timer}_{v,k})$ and start counting down.

   (b) **Synchronous Pre-commit.** If $\textsf{commit-timer}_{v,k}$ reaches Δ, pre-commit B_k. If r hasn't sent a blame message in view v, broadcast $\langle \textsf{commit}, B_k, v\rangle_r$ to all replicas.

4. **(Non-blocking) Commit.** If replica r is still in view v, r commits B_k using the following rules:

   (a) **Responsive Commit.** On receiving $\lfloor 3n/4 \rfloor + 1$ commit messages for B_k in view v, commit B_k and all its ancestors. Stop $\textsf{commit-timer}_{v,k}$.

   (b) **Synchronous Commit.** If $\textsf{commit-timer}_{v,k}$ reaches 0, commit B_k and all its ancestors.

5. **Yield.** Upon committing at least a block in view v, Leader L broadcasts $\langle \textsf{yield}, v\rangle_L$ when it wants to renounce leadership.

6. **(Non-blocking) Blame and quit view.**

   - **Blame if no progress.** For $p > 0$, if fewer than p proposals trigger r's votes in $(2p+4)\Delta$ time in view v, broadcast $\langle \textsf{blame}, v, \mathcal{CC}\rangle_r$ where $\mathcal{CC}$ is the highest ranked chain certificate known to r.
   - **Quit view on $f+1$ blame messages.** Upon gathering $f+1$ distinct blame messages, broadcast them. If r hasn't blamed in view v, broadcast $\langle \textsf{blame}, v, \mathcal{CC}\rangle_r$. Abort all view v timers, and quit view v. Set $\textsf{blame-timer}_v$ to 2Δ and start counting down.
   - **Quit view on detecting equivocation.** If leader equivocation is detected, broadcast $\langle \textsf{blame}, v, \mathcal{CC}\rangle_r$ along with the equivocating proposals, abort all view v timers, and quit view v. Set $\textsf{blame-timer}_v$ to 2Δ and start counting down.
   - **Quit view on yield.** Upon receiving yield, broadcast $\langle \textsf{blame}, v, \mathcal{CC}\rangle_r$ message along with yield message, abort all view v timers, and quit view v. Set $\textsf{blame-timer}_v$ to 2Δ and start counting down.

Figure 5: Steady state protocol for optimistically responsive view-change.

Let L and L' be the leader of view v and $v+1$, respectively.

i) **Status.** Replica r can enter view $v+1$ using one of the following rules:

a) **Responsive.** Upon gathering $\lfloor 3n/4 \rfloor + 1$ distinct blame messages, broadcast them. Update its chain certificate $\mathcal{CC}$ to the highest possible rank. Set lock_{v+1} to $\mathcal{CC}$ and send $\langle \mathsf{status}, \mathsf{lock}_{v+1} \rangle_r$ to L'. Enter view $v+1$ immediately. Stop $\mathsf{blame\text{-}timer}_v$.

b) **Synchronous.** When $\mathsf{blame\text{-}timer}_v$ expires, update its chain certificate $\mathcal{CC}$ to the highest possible rank. Set lock_{v+1} to $\mathcal{CC}$ and send $\langle \mathsf{status}, \mathsf{lock}_{v+1} \rangle_r$ to L'. Enter view $v+1$.

ii) **New View.** Upon receiving a set $\mathcal{S}$ of $f+1$ distinct status messages after entering view $v+1$, broadcast $\langle \mathsf{new\text{-}view\text{-}resp}, v+1, \mathsf{lock}_{v+1} \rangle_{L'}$ along with $\mathcal{S}$ where lock_{v+1} is highest ranked chain certificate in $\mathcal{S}$.

iii) **First Vote.** Upon receiving the first $\langle \mathsf{new\text{-}view\text{-}resp}, v+1, \mathsf{lock}' \rangle_{L'}$ along with $\mathcal{S}$, if lock' has a highest rank in $\mathcal{S}$, update lock_{v+1} to lock', broadcast $\langle \mathsf{new\text{-}view\text{-}resp}, v+1, \mathsf{lock}' \rangle_{L'}$, and $\langle \mathsf{vote}, \mathsf{tip}(\mathsf{lock}'), v+1 \rangle_r$.

Figure 6: The optimistically responsive view-change protocol

that a replica receives a responsively committed blocks when making immediate transition to a higher view. This is critical to maintain the safety of protocol (explained later). Due to the synchrony assumption, all other honest replicas receive the responsive blame certificate $\mathcal{B}$ within Δ time and transition immediately to view $v+1$.

The synchronous status rule is triggered when $\mathsf{blame\text{-}timer}_v$ expires. Note that the $\mathsf{blame\text{-}timer}_v$ was set to 2Δ. The 2Δ wait ensures that all honest replicas receive a responsive certificate $\mathcal{C}_v^{3/4}(B_k)$ for some possibly committed block B_k before entering view $v+1$. Replica r enters view $v+1$, and updates its lock, lock_{v+1} to a highest ranked chain certificate and sends lock_{v+1} to the new leader L' via $\langle \mathsf{status}, \mathsf{lock}_{v+1} \rangle_r$.

New-View. Upon entering view $v+1$, the leader waits for a set $\mathcal{S}$ of $f+1$ status messages. We call the set $\mathcal{S}$ of $f+1$ status messages as *status certificate*. Based on the status certificate $\mathcal{S}$, L' picks the highest ranked chain certificate lock_{v+1} and broadcasts new-view message $\langle \mathsf{new\text{-}view\text{-}resp}, v+1, \mathsf{lock}_{v+1} \rangle_{L'}$ along with $\mathcal{S}$. Sending $\mathcal{S}$ along with new-view message justifies that $\mathsf{tip}(\mathsf{lock}_{v+1})$ extends committed blocks in previous view.

First-Vote. Upon receiving a $\langle \mathsf{new\text{-}view\text{-}resp}, v+1, \mathsf{lock}' \rangle_{L'}$ message along with status certificate $\mathcal{S}$, if chain certificate lock' has the highest rank in $\mathcal{S}$, then it forwards the new-view message to all replicas and broadcasts a vote for it. Note that replica r may have lock_{v+1} with rank higher than lock'. A replica votes for lock' as long as lock' is vouched by $\mathcal{S}$. This is critical to ensure safety across views.

Next, we provide some intuition on how the view-change protocol provides liveness and safety across views.

How is the safety of a responsive commit maintained across views? Suppose an honest replica r responsively commits a block B_k at time t. A responsive commit for a block B_k requires a set $Q_C^{3/4}$ of $\lfloor 3n/4 \rfloor + 1$ commit messages. A responsive view-change requires a set $Q_B^{3/4}$ of $\lfloor 3n/4 \rfloor + 1$ blame messages. Due to a quorum intersection argument, $Q_C^{3/4}$ and $Q_B^{3/4}$ intersect in at least one honest replica h which sends chain certificate $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends $\mathcal{C}_v(B_k)$. Observe that this also explains why (i) highest ranked chain certificate issent with a blame message, and (ii) a replica does not send a commit message if it has blamed the leader. For (i), the highest chain certificate $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends $\mathcal{C}_v(B_k)$ from the honest replica h at the intersection allows another replica r' performing a responsive view change to learn about the commit of B_k. For (ii), if the honest replica at the intersection h does send a commit message for a block B_{k+1}, r' would not know of a potential commit of B_{k+1}, and it could vote for blocks that do not extend B_{k+1} and cause a safety violation.

A synchronous view-change waits 2Δ time before moving to a higher view. Note that a replica making a synchronous view-change hasn't entered a higher view by time $t + \Delta$. If an honest replica $r' \in Q_C^{3/4}$ pre-commits responsively by time t, its notify containing $\mathcal{C}_v^{3/4}(B_k)$ must reach all honest replicas by time $t + \Delta$. If replica $r' \in Q_C^{3/4}$ pre-commits synchronously, honest replicas making a synchronous view-change receive $\mathcal{C}_v(B_k)$ by the time r' pre-commits. Thus, all honest replicas lock on chain certificate $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends $\mathcal{C}_v(B_k)$.

How is the safety of a synchronous commit maintained across views? Consider replica r votes for B_k at time t' and synchronously commits at time t. If replica r pre-commits responsively at time s with $s - t' < \Delta$, it waits 2Δ before its $\mathsf{commit\text{-}timer}_{v,k}$ expires. Note that no honest replica has entered a higher view by time $t - \Delta$. Replica $r's$ notify containing $\mathcal{C}_v^{3/4}(B_k)$ arrives all honest replicas by time $s + \Delta$ (i.e., by time $t - \Delta$). In all other cases, replica r waits 3Δ time before its $\mathsf{commit\text{-}timer}_{v,k}$ expires and replica r votes at time $t - 3\Delta$. This implies all honest replicas receive $\mathcal{C}_v(B_k)$ by time $t - \Delta$. Any view-change after $t - \Delta$ will receive $\mathcal{C}_v(B_k)$ or higher and honest replicas will lock on chain certificate $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends $\mathcal{C}_v(B_k)$ before entering a higher view.

Why is it safe to vote for a valid new-view message with a lower ranked lock? The commit rules in the protocol ensure that there does not exist an equivocating chain certificate CC' such that $\mathsf{tip}(CC')$ does not extend committed blocks. This implies honest replicas lock on chain certificates that extend the committed blocks. After entering a higher view, honest replicas send their locked chain certificates via a status message. The new leader collects a status certificate $\mathcal{S}$ of $f + 1$ status messages, extends on the highest ranked certified block in $\mathcal{S}$. Note that an honest replica sends a status message only after entering a higher view and has locked on a chain certificate that extends committed blocks in the previous view. As $\mathcal{S}$ contains status from at least one honest replica, the highest ranked chain certificate lock' in $\mathcal{S}$ will extend committed blocks in the previous view. Thus, it is safe for replicas to unlock a lock with a rank higher than lock'.

In the new view, due to the status certificate, all honest replicas will vote for the new-view message sent by an honest leader. Subsequently, in the steady state, honest replicas will keep committing new blocks.

6.3 Safety and Liveness

Claim 19. *If a block B_k is committed directly in view v using the responsive commit rule, then there does not exist a chain certificate $\mathcal{CC}'$ in view v such that $\mathcal{CC}' > \mathcal{CC}$ where $\mathsf{tip}(\mathcal{CC})$ extends B_k and a block in $\mathcal{CC}'$ equivocates B_k.*

Proof. If a replica r responsively commits a block B_k in view v, then r must have received $\lfloor 3n/4 \rfloor + 1$ distinct commit messages out of which at least a set $\mathcal{R}$ of $\lfloor (n-f)/2 + 1 \rfloor$ are from honest replicas. An honest replica (say, $r' \in \mathcal{R}$) sends commit message only if it pre-commits and has not sent a blame message.

Replica r' can pre-commit in two ways. First, r' received $\lfloor 3n/4 \rfloor + 1$ votes for B_k in view v and pre-committed responsively. This case is identical to responsive commit rule for the protocol in Section 4. By Claim 6, an equivocating chain certificate $\mathcal{CC}'$ of rank higher than $(\mathcal{C}_v^{3/4}(B_k), \perp)$ cannot exist in view v. Second, replica r' voted for B_k at time $t - 2\Delta$ and received no equivocation or blame certificate by time t and synchronously pre-commits at time t. This case is identical to synchronous commit rule for the protocol in Section 4. By Claim 7, there does not exist a block certificate for an equivocating block in view v. Thus, chain certificate $\mathcal{CC}'$ with an equivocating block such that $\mathcal{CC}' > \mathcal{CC}$ cannot exist in view v. □

Claim 20. *If a block B_k is directly committed in view v, using the synchronous commit rule then there does not exist a chain certificate $\mathcal{CC}'$ in view v such that $\mathcal{CC}' > \mathcal{CC}$ where $\mathsf{tip}(\mathcal{CC})$ extends B_k and a block in $\mathcal{CC}'$ equivocates B_k.*

Proof. Replica r synchronously commits a block B_k when its $\mathsf{commit\text{-}timer}_{v,k}$ expires. Replica r could pre-commits in two ways. First, replica r pre-commits responsively and then waits at least 2Δ time. The responsive pre-commit rule is identical to the responsive commit rule for the protocol in Section 4. By Claim 6, an equivocating chain certificate $\mathcal{CC}'$ of rank higher than $(\mathcal{C}_v^{3/4}(B_k), \perp)$ cannot exist in view v.

Second, replica r synchronously pre-commits at time t, i.e., it voted for B_k at time $t - 2\Delta$ and received no equivocation or blame certificate by time t. This case is identical to synchronous commit rule for the protocol in Section 4. By Claim 7, there does not exist a block certificate for an equivocating block in view v. Thus, chain certificate $\mathcal{CC}'$ with an equivocating block cannot exist in view v. □

Lemma 21. *If a block B_k is directly committed in view v, then there does not exist a chain certificate $\mathcal{CC}'$ in view v such that $\mathcal{CC}' > \mathcal{CC}$ where $\mathsf{tip}(\mathcal{CC})$ extends B_k and a block in $\mathcal{CC}'$ equivocates B_k.*

Proof. Straightforward from Claim 19 and Claim 20. □

Claim 22. *Let B_k be a block proposed in view v using Step 1 in Figure 5. If an honest replica votes for B_k at time t in view v and detects no equivocation or blame certificate at time $\leq t + 2\Delta$, then (i) all honest replicas are in view v at time $t + \Delta$ (ii) all honest replicas vote for B_k at time $\leq t + \Delta$.*

Proof. Suppose an honest replica r votes for B_k at time t in view v and detects no equivocation or blame certificate by time $t + 2\Delta$. This implies two facts. First, replica r entered view v at time $\leq t$. If r quit the previous view responsively, i.e., by receiving $\lfloor 3n/4 \rfloor + 1$ blame messages, it must have sent a responsive blame certificate at time $\leq t$. All honest replicas receive the responsive blame certificate and enter view v at time $\leq t + \Delta$. If r quit the previous view due to $f + 1$ blame messages, it must have sent a blame certificate at time $\leq t - 2\Delta$ which arrives all honest replicas at time $\leq t - \Delta$. Due to the 2Δ wait after receiving $f+1$-sized blame certificate, all honest replicas enter view v at time $\leq t + \Delta$. We note that no honest replica has quit view v at time $\leq t + \Delta$; otherwise, replica r receives a blame certificate at time $\leq t + 2\Delta$. This proves part (i) of the claim.

Replica r received a proposal for B_k which contains $\mathcal{C}_v(B_{k-1})$ at time t. Replica r's vote and forwarded proposal for B_k arrives at all honest replicas at time $\leq t + \Delta$. No honest replica has voted for an equivocating block or received a blame certificate at time $\leq t + \Delta$; otherwise replica r would have received an equivocation or blame certificate at time $\leq t + 2\Delta$. Thus, all honest replicas will vote for B_k at time $\leq t + \Delta$. This proves part (ii) of the claim. □

Claim 23. *Let B_k be a block proposed in view v using Step 1 in Figure 5. If an honest replica votes for B_k at time t in view v and detects no equivocation or blame certificate at time $\leq t + 3\Delta$, then (i) all honest replicas are still in view v at time $t + 2\Delta$ (ii) all honest replicas receive $\mathcal{C}_v(B_k)$ at time $\leq t + 2\Delta$.*

Proof. Suppose an honest replica r votes for a block B_k at time t in view v and detects no equivocation or blame certificate by time $t + 3\Delta$. Trivially, replica r has not received an equivocation or blame certificate by time $t + 2\Delta$. By Claim 22 (i), all honest replicas are in view v at time $t + \Delta$. No honest replica has quit view v by time $t + 2\Delta$; otherwise replica r must receive blame certificate by time $t + 3\Delta$ contradicting our hypothesis. Thus, all honest replicas are still in view v at time $t + 2\Delta$. This proves part (i) of the claim.

If replica r receives no equivocation or blame certificate at time $\leq t + 3\Delta$, it is easy to see that replica r receives no equivocation or blame certificate by time $t + 2\Delta$. By Claim 22, all honest replicas vote at time $\leq t + \Delta$. By synchrony assumption, all honest replicas receive at least $f + 1$ votes for B_k i.e., $\mathcal{C}_v(B_k)$ at time $\leq t + 2\Delta$. This proves part (ii) of the claim. □

Claim 24. *If an honest replica directly commits a block B_k in view v using the responsive commit rule, then all honest replicas receive a chain certificate $\mathcal{CC}$ before entering view $v + 1$ such that $\mathsf{tip}(\mathcal{CC})$ extends B_k.*

Proof. We first discuss the case where some replica performs a view-change due to a responsive blame certificate, and then discuss a view-change due to a synchronous blame certificate. Suppose an honest replica r receives a set $Q_C^{3/4}$ of $\lfloor 3n/4 \rfloor + 1$ commit messages for block B_k in view v and responsively commits B_k at time t. Thus, all honest replicas in $Q_C^{3/4}$ must have received $\mathcal{C}_v(B_k)$ before sending the commit message. By Claim 19, there does not exist a chain certificate $\mathcal{CC}'$ in view v such that $\mathcal{CC}' > \mathcal{CC}$ where $\mathsf{tip}(\mathcal{CC})$ extends B_k and a block in $\mathcal{CC}'$ equivocates B_k. Consider the quorum $Q_B^{3/4}$ that made some honest replica r' quit view v. r' receives a responsive blame certificate of $\lfloor 3n/4 \rfloor + 1$ blame messages each of which contains a chain certificate when the blame message was sent. By quorum intersection argument, $Q_C^{3/4}$ and $Q_B^{3/4}$ must intersect in at least one honest replica. Thus, the intersecting honest replica must include a higher ranked chain certificate $\mathcal{CC}$ where $\mathsf{tip}(\mathcal{CC})$ extends B_k in blame message. This implies any replica that makes a responsive view-change must receive $\mathcal{CC}$ before entering view $v+1$.

Consider a view-change due to a synchronous blame certificate. Observe that any honest replica (say, replica u) that quits view v due to a synchronous blame certificate has not entered view $v+1$ at time $t+\Delta$; otherwise replica u must have sent a blame certificate at time $\leq t-\Delta$ (due to the 2Δ wait in the status step) and r must receive the blame certificate at time $\leq t$ and r wouldn't commit. If some honest replica r' in $Q_C^{3/4}$ pre-committed responsively, r' notifies $\mathcal{C}_v^{3/4}(B_k)$ by time t and replica u receives $\mathcal{C}_v^{3/4}(B_k)$ at time $\leq t+\Delta$. Similarly, if replica r' synchronously pre-commits B_k by time t, it votes for B_k by time $t-2\Delta$ and detects no equivocation or blame certificate by time t. By Claim 22 (ii), all honest replicas vote for B_k by time $t-\Delta$. Hence, replica u receives $C_v(B_k)$ by time t before entering view $v+1$. This implies any replica that makes a synchronous view-change has $\mathcal{CC}$ before entering view $v+1$ such that $\mathsf{tip}(\mathcal{CC})$ extends B_k. □

Claim 25. *If an honest replica directly commits a block B_k in view v using the synchronous commit rule, then all honest replicas receive a chain certificate $\mathcal{CC}$ before entering view $v+1$ such that $\mathsf{tip}(\mathcal{CC})$ extends B_k.*

Proof. Suppose an honest replica r synchronously commits a block B_k at time t in view v. Its $\mathsf{commit\text{-}timer}_{v,k}$ for B_k expires at time t without detecting an equivocation or blame certificate.

Let t' be the time at which replica r votes for B_k. If replica r pre-commits responsively and notifies $\mathcal{C}_v^{3/4}(B_k)$ at time s with $s-t'<\Delta$, it waits at least 2Δ (with $s+2\Delta=t$) before its $\mathsf{commit\text{-}timer}_{v,k}$ expires. It is easy to see that replica r voted at time t' and hasn't detected an equivocation or blame certificate by time $t'+2\Delta$. By Claim 22 (i), all honest replicas are in view v by time $t'+\Delta$. All honest replicas are still in view v at time $t-\Delta$; otherwise, replica r must have received a blame certificate by time t and wouldn't commit. Replica r's notify containing $\mathcal{C}_v^{3/4}(B_k)$ reaches all honest replicas at time $\leq s+\Delta$ i.e., at time $\leq t-\Delta$. Hence, all honest replicas receive $\mathcal{C}_v^{3/4}(B_k)$ before entering view $v+1$.

In all other cases, replica r waits for 3Δ before its $\mathsf{commit\text{-}timer}_{v,k}$ expires. Replica r votes for B_k in view v at time $t-3\Delta$ and detects no equivocation or blame certificate by time t. By Claim 23, all honest replicas are in view v at time $t-\Delta$ and receive $C_v(B_k)$ by time $t-\Delta$. Thus, all honest replicas receive $\mathcal{C}_v(B_k)$ before entering view $v+1$. This implies all honest replicas have a chain certificate $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends B_k. □

Lemma 26. *If an honest replica directly commits a block B_k in view v, then all honest replicas have lock_{v+1} before entering view $v+1$ such that $\mathsf{tip}(\mathsf{lock}_{v+1})$ extends B_k.*

Proof. By Claim 24 and Claim 25, all honest replicas receive a certificate chain $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends B_k. By Lemma 21, there doesn't exists an equivocating chain certificate $\mathcal{CC}'$ in view v such that $\mathcal{CC}'>\mathcal{CC}$. Since, honest replicas lock on highest ranked chain certificate, all honest replicas update lock_{v+1} to $\mathcal{CC}$ with $\mathsf{tip}(\mathsf{lock}_{v+1})$ extending B_k.

□

Claim 27. *If an honest replica directly commits a block B_k in view v, the tip of a highest ranked chain certificate $\mathcal{CC}$ in a view v status certificate, i.e., $\mathsf{tip}(\mathcal{CC})$ must extend B_k.*

Proof. Suppose an honest replica r commits a block B_k in view v. By Lemma 26, all honest replicas lock on $\mathcal{CC}$ before entering view $v+1$ such that $\mathsf{tip}(\mathcal{CC})$ extends B_k. An honest replica sends status message containing their $\mathcal{CC}$ only after entering view $v+1$. A view v status certificate contains a set $\mathcal{S}$ of $f+1$ status messages which includes the status message from at least one honest replica. By Lemma 21, there does not exist a chain certificate $\mathcal{CC}'$ in view v such that $\mathcal{CC}'>\mathcal{CC}$ where $\mathsf{tip}(\mathcal{CC})$ extends B_k and a block in $\mathcal{CC}'$ equivocates B_k. Thus, the tip of highest ranked chain certificate $\mathcal{CC}$ in $\mathcal{S}$, i.e., $\mathsf{tip}(\mathcal{CC})$ must extend B_k. □

Corollary 28. *If the tip of highest ranked chain certificate $\mathcal{CC}$ in a view v status certificate, i.e., $\mathsf{tip}(\mathcal{CC})$ does not extend a block B_k, then B_k has not been committed in view v.*

Lemma 29 (Unique Extensibility)**.** *If an honest replica directly commits a block B_k in view v, and $\mathcal{C}_{v'}(B_{k'})$ is a view $v' > v$ block certificate, then $B_{k'}$ extends B_k. Moreover, all honest replicas have $\mathsf{lock}_{v'}$ such that $\mathsf{tip}(\mathsf{lock}_{v+1})$ extends B_k.*

Proof. The proof is by induction on the view $v' > v$. For a view v', we prove that if $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}'))$ exists then it must extend B_k. A simple induction then shows that all later block certificates must also extend $\mathsf{tip}(\mathsf{lock}')$, this follows directly from the Vote rule in line 2.

For the base case, where $v' = v + 1$, the proof that $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}'))$ extends B_k follows from Lemma 26 because the only way such a block can be certified is if some honest replica votes for it. However, all honest replicas are locked on a block that extends B_k and a chain certificate with a higher rank for an equivocating block does not exist. Although, honest replicas unlock on their locked chain certificates lock_{v+1} and lock on a highest ranked chain certificate lock' in a status certificate $\mathcal{S}$, by Claim 27, $\mathsf{tip}(\mathsf{lock}')$ must extend B_k. Thus, no honest replica will first vote (Figure 2 step iii) for a block that does not extend B_k. The second part follows directly from Lemma 26.

Given that the statement is true for all views below v', the proof that $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}'))$ extends B_k follows from the induction hypothesis because the only way such a block can be certified is if some honest votes for it. An honest party with a lock lock will vote only if $\mathsf{tip}(\mathsf{lock}_{v'})$ has a valid block certificate and $\mathsf{lock} \geq \mathsf{lock}_{v'}$. Due to Lemma 26 and the induction hypothesis on all block certificates of view $v < v'' < v'$ is must be that $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}))$ extends B_k. ☐

Safety. The safety proof remains identical to that of Theorem 12 except Lemma 21 and Lemma 29 needs to be invoked.

Theorem 30 (Liveness)**.** *All honest replicas keep committing new blocks.*

Proof. In a view, a leader has to propose at least p blocks that trigger honest replica's votes in $(2p+4)\Delta$ time. As long as the leader proposes at least p valid blocks, honest replicas will keep voting for the blocks and keep committing the proposed blocks. If the Byzantine leader equivocates or proposes less than p blocks, a view-change will occur. Eventually, there will be an honest leader due to round-robin leader election.

Next, we show that once the leader is honest, a view-change will not occur and all honest replicas keep committing new blocks. If a block B_k has been committed in a previous view, by Lemma 26, all honest replicas lock on a chain certificate lock_{v+1} such that $\mathsf{tip}(\mathcal{CC})$ extends B_k before entering a new view. After entering a new view, honest replicas send their locked $\mathcal{CC}$ to the new leader in status message. The new leader extends on the tip of a highest ranked chain certificate (say, lock') in a status certificate $\mathcal{S}$. Even if some honest replicas are locked on chain certificates (say, $\mathcal{CC}$") that rank higher than lock'), by Corollary 28 it is safe to unlock on $\mathcal{CC}$"). Hence, honest replicas will vote for blocks that extend $\mathsf{tip}(\mathsf{lock}')$. After that, the honest leader can propose at least one block in 2Δ time and keep making progress. Moreover, the honest leader doesn't equivocate. This ensures all honest replicas keep committing new blocks.

☐

7 Related Work

There has been a long line of work on Byzantine agreement starting at the Byzantine Generals Problem by Lamport, Shostak and Pease [19]. Dolev and Strong [10] presented a deterministic solution to the Byzantine Broadcast problem in the synchronous model and tolerates $f < n - 1$ faults. Their protocol achieves $f + 1$ round complexity and $O(n^2 f)$ communication complexity. Several other works [1, 4, 11, 12, 16, 23, 15] have been proposed to improve the round complexity. We review the most recent and closely related works below. In particular, we make comparisons with synchronous BFT protocols with the notion of optimistic and synchronous commit paths. Compared to all of these protocols, our responsive commit incurs an optimal latency of 2δ and our synchronous commit incurs a latency of 2Δ time while tolerating the same number of faults.

Thunderella. The idea of optimistic responsiveness in a back-and-forth slow-path–fast-path paradigm was first introduced in Thunderella [22]. They commit a decision in a single round under optimistic executions but fall back on $O(\kappa\Delta)$ or $O(n\Delta)$ Nakamoto or Dolev-Strong slow paths when optimistic conditions are not met. Moreover, the time to switch between these paths are $O(\kappa\Delta)$ or $O(n\Delta)$ respectively.

Sync HotStuff. Sync HotStuff [2] presents a synchronous SMR protocol with an optimistically responsive commit path. Like Thunderella, it is presented in a back-and-forth slow-path–fast-path paradigm. If started in the wrong path, their responsive commit will incur a latency of $2\Delta + O(\delta)$ time and synchronous commit incurs $4\Delta + O(\delta)$ time. Compared to them, our protocol in Section 6 can also perform an optimistically responsive view change, while their view change always incurs a 2Δ delay.

PiLi. PiLi [7] presents a BFT SMR protocol that progresses through a series of epochs. The protocol assumes lock-step execution in epochs. Each epoch either lasts for 5Δ time under normal conditions or $O(\delta)$ time during optimistic executions. The protocol commits 5 blocks after 13 consecutive epochs. PiLi has a responsive commit latency of at least 16δ-26δ and incurs a synchronous latency of 40Δ-65Δ.

References

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected $O(1)$ rounds, expected $O(n^2)$ communication, and optimal resilience. In *International Conference on Financial Cryptography and Data Security*, pages 320–334. Springer, 2019.

[2] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: simple and practical synchronous state machine replication. In *IEEE Security and Privacy*, 2020.

[3] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Optimal good-case latency for byzantine broadcast and state machine replication. *arXiv preprint arXiv:2003.13155*, 2020.

[4] Michael Ben-Or. Another advantage of free choice (extended abstract) completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30, 1983.

[5] Miguel Castro, Barbara Liskov, et al. Practical Byzantine Fault Tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[6] T-H Hubert Chan, Rafael Pass, and Elaine Shi. Pala: A simple partially synchronous blockchain. *IACR Cryptology ePrint Archive*, 2018:981, 2018.

[7] T-H Hubert Chan, Rafael Pass, and Elaine Shi. Pili: An extremely simple synchronous blockchain. *IACR Cryptology ePrint Archive*, 2018:980, 2018.

[8] TH Hubert Chan, Rafael Pass, and Elaine Shi. Pili: A simple, fast, and robust family of blockchain protocols.

[9] Danny Dolev, Ruediger Reischuk, and H Raymond Strong. Early stopping in byzantine agreement. *Journal of the ACM (JACM)*, 37(4):720–741, 1990.

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

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

[12] Matthias Fitzi and Juan A Garay. Efficient player-optimal protocols for strong and differential consensus. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 211–220, 2003.

[13] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.

[14] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: a scalable decentralized trust infrastructure for blockchains. *arXiv preprint arXiv:1804.01626*, 2018.

[15] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[16] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In *Annual International Cryptology Conference*, pages 445–462. Springer, 2006.

[17] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. *ACM SIGOPS Operating Systems Review*, 41(6):45–58, 2007.

[18] Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1(11), 2014.

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

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

[21] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[22] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2018.

[23] Michael O Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 403–409. IEEE, 1983.

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

[25] Elaine Shi. Streamlined blockchains: A simple and elegant approach (a tutorial and survey). In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–17. Springer, 2019.

[26] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.