Sync HotStuff: Synchronous SMR with 2Δ Latency and Optimistic Responsiveness

Ittai Abraham[1], Dahlia Malkhi[1], Kartik Nayak[1,2], Ling Ren[1,3], and Maofan Yin[4]

[1]VMware Research – {iabraham,dmalkhi,nkartik,lingren}@vmware.com
[2]Duke University
[3]University of Illinois at Urbana-Champaign
[4]Cornell University – my428@cornell.edu

Abstract

Synchronous solutions for building Byzantine Fault Tolerance (BFT) replication can be safe when $< 1/2$ of the replicas fail. Assuming Δ is an upper bound on the time for messages to arrive, these solutions must incur at least Δ latency for consensus on a single value. In this work, we show a consensus protocol named Sync HotStuff designed to achieve consensus on a sequence of values with a latency of 2Δ in the common mode when less than half of the replicas are Byzantine. Thus, in the common mode, Sync HotStuff is within a factor of 2 of the optimal latency. Moreover, Sync HotStuff has *responsiveness*, i.e., it advances at network speed, when $< 1/4$ of the replicas are not responding, a small sacrifice in availability compared with optimal asynchronous solutions. Borrowing from practical BFT solutions in the asynchronous arena, Sync HotStuff has an extremely simple, two-phase leader-based structure, that easily fits in one frame of pseudo-code.

1 Introduction

The synchrony assumption in distributed computing states that there is a globally known upper bound Δ such that any message sent by one replica reaches another replica within Δ time. While Byzantine agreement (BA) protocols assuming asynchrony or partial synchrony require that fewer than $n/3$ replicas are Byzantine, a synchrony assumption allows tolerating $< n/2$ Byzantine replicas.

Early synchronous solutions work in a lock step execution model, and focus on the number of rounds to reach a single consensus decision. Dolev and Strong introduced in [4] a solution optimized for no (or a small number f of) faults, achieving a consensus decision in $f + 2$ lock step rounds. Katz and Koo improved in [7] in the same model to asymptotic latency of expected $O(1)$ rounds, albeit with a non-negligible constant of ~ 30 rounds. Abraham et al. recently improved in [1] to 10 rounds in expectation. It is known that if the maximum network delay is Δ, then a lock step protocol requires 2Δ time per round [1, 3]. Thus, the effective latency of these lock step protocols is at least 20Δ. Recent solutions by Hanke et al. [6] move away from the lock step model. This allows pipelining and improved latency to 8Δ in expectation [6, 2].

In this paper, we explore the latency of synchronous consensus protocols in the context of the state machine replication (SMR) problem. This affects protocol design choices as well as how latency should be defined in two crucial aspects.

First, Byzantine agreement considers consensus on a single value. The aforementioned leader-based protocols [7, 1] reach consensus on this single value and terminate after an honest party is elected as the leader. Thus, the preferred approach is to randomize the leader until an honest leader emerges. In contrast, SMR considers consensus on a sequence of values. In such a setting, Paxos and PBFT observed that it is beneficial to have a so-called "common mode" in which an honest replica remains the leader to commit many values. As a result, the metric of interest for SMR is the latency to commit once such an honest leader emerges.

Second, the input values to SMR come from external clients rather than the replicas themselves. Thus, the latency of a commit should be calculated from a client's perspective, that is, the time difference between when a client sends a request and when it receives a response. If a client's request arrives between two consensus instances, it incurs an additional delay before being processed by a consensus instance. We call this extra delay a *queue-up delay*. Revisiting the state-of-the-art solution, assuming uniform arrival of client requests, Hanke et al. [6] incurs expected Δ queue-up delay.

In this work, we present Sync HotStuff, a synchronous SMR protocol that commits client requests within 2Δ latency in the common mode. We adopt the "common mode" stable leader approach and design the protocol to reduce queue-up delay to be independent of Δ.

Sync HotStuff and lower bounds for synchrony. The only known lower bound on the latency of synchronous Byzantine agreement stems from adapting the proof in Dwork et al. [5]: a Δ latency is necessary for protocols tolerating more than one-third Byzantine faults. This should not be surprising because a protocol that commits faster than Δ, in a way, does not take advantage of synchrony and is thus subject to the one-third partial synchrony bound.

While Sync HotStuff does have a latency of 2Δ, we conjecture that 2Δ is a lower bound on synchronous BA latency, hence, our solution may be optimal in the common mode. Our intuition is that replicas can be out-of-sync by Δ, so one Δ is needed for lagging replicas to catch up and another Δ is needed for messages to reach (the lower bound only captures the latter Δ). Proving this stronger lower bound remains an intriguing open question.

Simplicity of Sync HotStuff. A less tangible benefit of our solution is simplicity. Our solution is presented in Figure 1 in just two phases. A leader sends a proposal, everyone broadcasts votes for it and decides within 2Δ time after this vote if they do not see a conflicting proposal.

This succinct structure has several tangible benefits. First, it can be easily pipelined into the HotStuff framework [10], where proposals and certificates are embedded into chains of blocks. Second, we show how to incorporate Sync HotStuff into an optimistic responsive mode proposed by Thunderella [9]. In this mode, Sync HotStuff has responsiveness when less than $n/4$ of the replicas are not responding, a small sacrifice in availability compared with optimal asynchronous solutions. That is, if the actual number of faulty replicas is less than one-quarter, then the protocol achieves *network speed latency*, i.e., latency does not depend on Δ.

1.1 Related Work

Several decades of solutions in the Byzantine Agreement space brought a myriad of techniques on which Sync HotStuff rests, while producing a combination with unique characteristics that is not manifested in any previous work. More specifically, Sync HotStuff borrows the leader-based approach of Abraham et al. [1], the HotStuff pipelining framework [10], optimistic responsiveness from Thunderella [9], and the non-lock-step approach from Dfinity [6]. We review these works below (except for HotStuff, whose framework is utilized directly in the body of this paper).

Dfinity. The Dfinity Consensus protocol described in [6] is a replication protocol in the synchrony model that tolerates $f < n/2$ Byzantine faults. It makes a key observation that a synchronous replication protocol can start processing the next client request without waiting for the previous one to commit. While a standard practice in partially synchronous protocols, this was not obvious for synchronous protocols. However, as observed in a recent technical report [2], Dfinity Consensus suffers from unbounded message complexity due to an unnecessary requirement that honest replicas vote for all leader proposals, including conflicting ones. Latency-wise, it has an expected latency of 9Δ after taking random leaders and client queue-up delay into account.

Thunderella. The idea of optimistic responsiveness (under one-quarter faults) was introduced in Thunderella [9]. When more than $3n/4$ replicas are correct, Thunderella can commit a decision in one phase. However, the decision cannot be conveyed to external clients, hence it does not support SMR in the traditional sense. Thunderella uses the synchronous mode to monitor the progress of the responsive mode and, if it does not make progress quickly, falls back to the synchronous mode. The fallback mechanism is presented in a black-box fashion but it is unclear how it works in a non-Nakamoto-style protocol.

In comparison, Sync HotStuff uses two-phase commit in the responsive path, and hence provides safety for SMR. Moreover, we take the conventional approach of having replicas monitor the progress of the responsive mode, and using the view change protocol to perform the fallback.

XFT. A different type of protocol with optimistic responsiveness is XFT [8]. XFT guarantees responsiveness when a group of $f+1$ honest replicas is determined. Thus, when the actual number of faults is t, it may take $\binom{n}{f+1}/\binom{n-t}{f+1}$ view changes for an honest group of $f+1$ to emerge; after that time, the protocol is responsive. Such a solution is practical when t is a small constant but for $t=\Theta(n)$, it requires an exponential number of view changes to find an honest group. In comparison, Sync HotStuff is responsive under $t<n/4$ faults after at most t view changes.

2 Definitions and Model

State Machine Replication (SMR). A state machine replication protocol is used for building a fault-tolerant service to process client requests. The service consists of n replicas, up to f of which may be Byzantine faulty. The service commits client requests into a linearizable log and produces a consistent view of the log akin to the presence of a single non-faulty server. More formally, a state machine replication service provides the following two guarantees

- **Safety:** When a value is committed by any non-faulty replica at a log position, all non-faulty replicas commit that value.
- **Liveness:** Each client request is eventually committed by non-faulty replicas.

The network consists of pairwise, authenticated communication channels between replicas. In addition, we assume digital signatures and a public-key infrastructure (PKI), and use $\langle x\rangle_p$ to denote a message x signed by replica p. For efficiency, it is customary to sign the hash digest of a message.

We assume a synchronous network, i.e., a message sent at time t by a replica arrives at another replica by time $t+\Delta$ where Δ is a known maximum network delay. We use δ to denote the actual message delay in the network.

Responsiveness. A protocol is said to be *responsive* [9] if the time to process a client request only depends on the actual network δ but not the maximum network delay Δ. A protocol is said to be *optimistically responsive* if it achieves responsiveness when some additional constraints are met. As we will see in subsequent sections, our protocol provides safety and liveness when $f<n/2$. In addition, it is optimistically responsive when round messages from $>3n/4$ replicas arrive within time δ. This happens if $f<n/4$, or $f<n/2$ and $n/4$ Byzantine replicas are (potentially temporarily) responding promptly.

3 Sync HotStuff: Synchronous SMR with 2Δ-Latency

In this section, we present Sync HotStuff, a synchronous state machine replication protocol that requires $n=2f+1$ replicas. Sync HotStuff takes the Paxos/PBFT's approach of having a *stable leader* in a steady state. The reign of a leader is called a *view*, numbered by integers. The leader of view v can simply be replica ($v \mod n$), i.e., leaders are decided in a round-robin order. The leader is expected to keep making progress by committing blocks at increasing *heights*. If replicas detect Byzantine behavior by the leader or lack of progress in a view, they *blame* the leader and engage in a *view-change* protocol to replace the leader. Figures 1 and 2 describe the steady state and view-change protocols, respectively.

Blocks and block format. As commonly done in practice, client requests are batched into *blocks* ordered by the SMR engine. A key ingredient of BFT solutions is a *Quorum Certificate* (or in short, a QC), a set of signatures on a block by a quorum of replicas. Here, quorums consist of $n/2+1$ replicas. We denote by $\mathcal{C}(B)$ a *certificate* on a block B. We call a block B *certified* iff $\mathcal{C}(B)$ exists.

Let v be the current view number and replica L be the leader of the current view. The steady state protocol runs the following steps in iterations.

1. **Propose.** The leader L broadcasts $B_k := \langle b, \mathcal{C}(B_{k-1})\rangle_L$ where b is a batch of new client requests and B_{k-1} is the highest certified block known to L.

2. **Vote.** Upon receiving the first valid height-k block B_k from L or through a vote by some other replica, if it is extending a highest certified block, broadcast a vote by sending $\langle \mathsf{vote}, B_k\rangle$. Set $\mathsf{commit\text{-}timer}_k$ to 2Δ, $\mathsf{blame\text{-}timer}$ to 3Δ, and start counting down.

3. **(Non-blocking) Blame.** If $\mathsf{blame\text{-}timer}$ reaches 0 and the next block is not received, broadcast $\langle \mathsf{blame}, v\rangle$. If more than one block is proposed by a leader at any height (leader has equivocated), broadcast $\langle \mathsf{blame}, v\rangle$ and the two equivocated blocks. These equivocating blocks may be received directly from the leader or through a vote from some other replicas.

4. **(Non-blocking) Commit.** For any k, when $\mathsf{commit\text{-}timer}_k$ reaches 0, if B_k is the only height-k block received, commit B_k and all its predecessors.

Figure 1: The steady state protocol.

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

i **New-view.** Upon gathering $n/2+1$ $\langle \mathsf{blame}, v\rangle$ messages, broadcast them along with the highest certified block, stop all $\mathsf{commit\text{-}timer}$(s), and stop voting in view v.

ii **Status.** Wait for 2Δ time and enter view $v+1$. Upon entering view $v+1$, send a highest certified block to L', set $\mathsf{blame\text{-}timer}$ to 3Δ and transition back to steady state.

Figure 2: The view-change protocol.

The leader is responsible for sequencing blocks in the log. Whenever the leader proposes a block, it is *chained* by including in a block certificate for the preceding block in the log. We refer to the position in the chain as *height.* A block B_k proposed (by a leader L) at height k has the following format:

$$B_k = \langle b, \mathcal{C}(B_{k-1})\rangle_L$$

where b denotes a batch of client requests, $\mathcal{C}(B_{k-1})$ is a *certificate* for its predecessor block B_{k-1}. The first block contains no certificate since it has no predecessor. If B_l is an ancestor of B_k ($l < k$), we say B_k *extends* B_l.

3.1 Steady State Protocol

The steady state protocol runs in iterations. In each iteration, the leader proposes a block $B_k := \langle b, \mathcal{C}(B_{k-1})\rangle_L$ containing a set of client requests b and a certificate for a previous block (Step 1). If a leader has just entered the steady state after a view-change, it should extend the highest certified block it learned during the view-change. If there are multiple such blocks, the leader can pick one arbitrarily. If the leader has been in the steady state, it should extend the previous block it has proposed.

Every replica records locally the highest certified block(s) is has received. Each replica r, upon receiving a valid block B_k from the leader, broadcasts a vote $\langle \mathsf{vote}, B_k\rangle_r$ for it (Step 2). The replica deems the block as valid only if it extends a highest certified block known to it.

Although we assume synchrony in the protocol, the replicas do not progress in lock steps. Thus, a replica r may first hear a block through another replica's vote. A vote for a block can thus be considered a *re-proposal* of the block. In this case, if the block is valid, replica r also broadcasts a vote for B_k. Once replica r votes, it starts two timers:

- A 2Δ-$\mathsf{commit\text{-}timer}_k$: a timer used to ensure safety of block B_k. Block B_k is committed if replica r does not observe any equivocating blocks at height-k within 2Δ time after it votes for B_k.

- A 3Δ-blame-timer: a timer used to ensure progress in case a Byzantine leader stalls.

Note that the commit timers (Step 4) do not affect the critical path of progress. A replica starts the next iteration "concurrently" without waiting for the previous height to be committed. In fact, a replica can potentially have many previous heights whose commit timers are still running.

Why does absence of an equivocating block within 2Δ time suffice to ensure safety? Consider an honest replica r that votes for a block B_k at time t. First, note that replicas vote on blocks obtained through votes from other replicas. Every vote message essentially re-proposes the block and r's vote reaches all honest replicas before $t + \Delta$. If the leader has not equivocated, r should receive votes for B_k from all honest replicas, which form a certificate for B_k, before $t + 2\Delta$. Note that the above holds even if the leader did not propose B_k to all replicas.

Second, recall that honest replicas vote for the first valid block received at each height. Thus, if any honest replica votes for a conflicting block B'_k, it must do so before $t + \Delta$; otherwise, it should receive B_k from r first and vote for B_k. This implies that r would have known about B'_k before $t+2\Delta$, which was when its commit-timer expired, and would not have committed. On the other hand, if r receives B'_k after $t + 2\Delta$, it must have come from a Byzantine replica. r can be sure that all honest replicas have voted for B_k, and hence B'_k will not get certified. In summary, if a block B_k is committed by some honest replica, it will be the only certified block for that height.

Remark. A commit by some honest replica at height k does not imply a commit by all honest replicas at that height. This is because a Byzantine leader can send an equivocating block to a subset of honest replicas before their commit timers expires, causing them to not commit.

How does Sync HotStuff ensure progress? The leader can prevent progress through two mechanisms – stalling and equivocating. The two blame conditions, based on blame-timer and equivocation, defend against these two attacks, respectively. The blame-timer is set to 3Δ in Step 2 to ensure that a leader has enough time to make progress: the slowest replica may send its vote up to Δ time later, it takes at most Δ time for the vote to reach the leader, and another Δ time for leader's next proposed block to reach all replicas. This forces a Byzantine leader to propose a block every 3Δ time to avoid being overthrown. If a leader equivocates by proposing conflicting blocks, the two equivocated blocks serve as a proof of Byzantine behavior and are sent together with the blame message. All honest replicas will blame the leader within Δ time and a view change ensues.

Certificate chaining and HotStuff. Note that blocks across heights are chained by certificates and committing a block commits all its predecessors. This is a key idea of the HotStuff framework [10]: having each voting step simultaneously play different roles at multiple heights greatly simplifies the protocol. In Sync HotStuff, the vote step of height k serves to also notify the certificate of height $k - 1$.

3.2 View Change Protocol

As mentioned earlier, if a replica observes lack of progress due to an expiring blame-timer or observes an equivocation, it broadcasts a $\langle \mathsf{blame}, v\rangle$ message where v is the view number. If a replica gathers $n/2 + 1$ $\langle \mathsf{blame}, v\rangle$ messages, it starts a view-change process by broadcasting the $n/2+1$ blame messages. In addition, it broadcasts its highest certified block (Step i).

The replica then waits for 2Δ time to learn the highest certified block held among honest replicas, and then reports one such block to L' (Step ii). The 2Δ wait before sending status gives the following guarantee (formalized in Lemma 1): an honest replica will learn all blocks committed by all honest replicas in previous views before sending status in a new view.

After that, a replica sets a blame timer of 3Δ, transitions back to steady state, and waits for the leader to propose a block that extends the block it reports in status. Similar to the reasoning for the steady state, 3Δ suffices for the new leader to receive all honest statuses and propose such a new block.

3.3 Safety and Liveness

We say a block B_k is committed *directly* if an honest replica commits it by observing $\mathcal{C}(B_k)$ and no equivocating block B'_k at height k within 2Δ after it votes. We say a block B_k is committed *indirectly* if it is a result of directly committing a block extending B_k.

Lemma 1. *If an honest replica directly commits a block B_l in a view, then every honest replica receives $\mathcal{C}(B_l)$ by the end of that view.*

Proof. Consider any two honest replicas r and r': r directly commits B_l in view v at time t and r' quits view v at time t'. According to the view change protocol, r' sends $n/2+1$ blame messages at time $t'-2\Delta$. At time no later than $t'-\Delta$, r will receive $n/2+1$ blame messages. (r at least receives them from r' by that time, but may also receive them earlier from other replicas.) If $t > t'-\Delta$, then r cannot have committed B_l in view v at time t due to the blame messages. Thus, $t < t'-\Delta$. r broadcasts the highest certified block it has at time $t'-\Delta$ on receiving the blame certificate. If r has already voted for block extending B_l prior to this point, then it has already broadcast $\mathcal{C}(B_l)$ in that vote. Otherwise, B_l will be the highest certified block known to r, and r broadcasts $\mathcal{C}(B_l)$. r' will receive the above information no later than time t'. □

Lemma 2. *If an honest replica directly commits a block B_l, then there does not exist $\mathcal{C}(B'_l)$ where $B'_l \neq B_l$.*

Proof. Suppose an honest replica r votes for B_l at time t and commits B_l at time $t+2\Delta$. We first prove that a conflicting certificate does not exist in views up to v (included). The vote from r for B_l reaches all honest replicas by time $t+\Delta$. If another honest replica has not voted at this height by then, it will vote for B_l. In order for $\mathcal{C}(B'_l)$ to exist in this view or prior views, some honest replica must vote for B'_l, and it must do so before time $t+\Delta$. But in that case, r would have received this equivocating vote before $t+2\Delta$, which contradicts the hypothesis of r committing B_l.

Next, we prove that a conflicting certificate does not exist in views greater than v. By Lemma 1, every honest replica receives $\mathcal{C}(B_l)$ by the end of view v, and there was no conflicting certificate at height l up until then. From then on, the highest certified block of every honest replica is at least B_l, and no honest replica will vote for $B'_l \neq B_l$. Thus, no $\mathcal{C}(B'_l)$ where $B'_l \neq B_l$ can come into existence in future views. □

Theorem 3 (Safety)**.** *Honest replicas always commit the same block B_k for each height k.*

Proof. Suppose for contradiction that two distinct blocks B_k and B'_k are committed at height k. Suppose B_k is committed as a result of B_l being directly committed and B'_k is committed as a result of $B'_{l'}$ being directly committed. $B_l = B_k$ if $l = k$ and B_l extends B_k if $l > k$; similarly, $B'_{l'} = B'_k$ if $l' = k$ and $B'_{l'}$ extends B'_k if $l' > k$. Without loss of generality, assume $l \leq l'$. By Lemma 2, there exists no other certified block at height l. If $l = l'$, then $B'_{l'} = B_l$ and $B'_{k'} = B_k$. If $l < l'$, then $B'_{l'}$ extends B_l, which extends B_k. □

Theorem 4 (Liveness)**.** *(i) A view change will not happen if the current leader is honest; (ii) A Byzantine leader must commit a block every 3Δ time to avoid a view change; and (iii) If k is the highest height at which some honest replica has committed a block in view v, then leaders in subsequent views must propose blocks at heights higher than k.*

Proof. Parts (i) and (ii) follow from the choices of blame-timer. Simply note that an honest leader has sufficient time and does not equivocate, so it will not be blamed by any other honest replica. On the other hand, if a Byzantine leader delays beyond 3Δ, it will be blamed by all honest replicas. For part (iii), observe that all honest receive $\mathcal{C}(B_k)$ due to Lemma 1. Hence, their highest certified block has height at least k. If the leader does not propose a block with a height higher than k within 3Δ, it will be blamed by all honest replicas. □

3.4 Efficiency Analysis

From an honest leader's perspective, each block incurs a latency of $\delta + 2\Delta$ after being proposed. (Step 1 proceeds at the actual network speed.) If we consider the queue-up time, it takes δ for the client to send its request to the replicas. On average, this request will arrive half way between two leader proposals, which are

2δ time apart. It takes an additional δ time for replicas to reply to the client. So the total average latency is $2\Delta + 4\delta$. Without queue-up delay, the best prior result is $8\Delta + 9\delta$ [2]. With queue-up delay, it is $9\Delta + 11.5\delta$, following a similar analysis.

The protocol requires all-to-all communication. Each message contains a single signature (using threshold signatures or multisignature for certificates). Thus, the communication complexity is $O(n^2)$ per block. We remark that our protocol also achieves what may be called network throughput: the throughput is determined by the network bandwidth, communication complexity and block size — it does not depend on Δ. Theoretically, any protocol can achieve network speed throughput through batching, but naively do so further hurts latency.

4 Optimistic Responsiveness

In this section, we incorporate the Thunderella [9] optimistic responsive mode into Sync HotStuff. In Section 3, a certificate/quorum required only $f + 1$ votes. Thus, a vote from a single honest replica *can* result in a certificate if all f Byzantine replicas vote on the same block. Therefore, before committing a block, a replica needs to wait long enough to hear all honest replicas' votes and make sure none of them voted for a conflicting block. The commit latency thus inherently depends on the maximum network delay Δ.

Partially synchronous protocols rule out the existence of a conflicting certificate with larger quorums. For instance, PBFT requires $> 2n/3$ votes and tolerates $f < n/3$ Byzantine replicas. A simple quorum intersection argument shows that two conflicting blocks cannot both receive $> 2n/3$ votes. Since our protocol tolerates up to minority corruption, similar to Thunderella [9], to achieve responsiveness we will use a quorum size of $> 3n/4$. Because honest replicas no longer rely on ruling out honest conflicting votes to commit, this obviates the need for waiting, achieving the desired responsiveness, and also obviates the need for broadcasting the vote.

The responsive mode makes progress only when $> 3n/4$ replicas respond to a leader proposal. Put differently, if $< n/4$ replicas or more are faulty, they can prevent responsiveness (but they cannot cause safety violation). In case there is no responsive progress, we fall back to the synchronous mode in Section 3.

4.1 Protocol

Since the responsive mode requires a larger quorum, we introduce the notion of a *fast quorum certificate*. A block B is said to have a fast quorum certificate (or fast QC in short) if it has $> 3n/4$ votes. It is denoted by $\mathcal{C}_{q_f}(B)$. In the responsive mode, honest replicas only vote for blocks that extend blocks with a fast QC, namely, in the form of $B_k = \langle x, \mathcal{C}_{q_f}(B_{k-1})\rangle_L$.

Steady state. The protocol for the responsive mode is described in Figure 3. It differs from the synchronous mode in two ways. First, as mentioned, a vote on the block is only sent to the leader and not broadcast to all replicas. Second, block B_{k-2} is committed when B_k is received, i.e., blocks that are two blocks deep are committed. Finally, the rule to blame the leader for lack of progress does not need to depend on Δ (Step 3). It can be based on any rule used by partially synchronous protocols. For instance, if the actual delay $\delta << \Delta$, then the leader can be blamed more aggressively.

Switching between the modes. We now specify how to switch between the responsive mode and synchronous mode. Whenever the responsive mode fails due to an equivocating leader or lack of participation from $> 3n/4$ participants, we engage in a view-change (Figure 2). After the view-change, the replicas move to the Sync HotStuff steady state protocol which requires participation from only $> n/2$ replicas. If the steady state leader recognizes that $> 3n/4$ replicas are participating, it signals a switch to the responsive mode. The Sync HotStuff steady state protocol is identical to the one in Figure 1 except for a couple of differences highlighted in Figure 4.

Although the responsive mode itself is rather straightforward, much care is needed when switching between the two modes. Switching from the responsive mode to synchronous mode is performed via a view change protocol. Similar to before, a replica requests a view change if it detects leader equivocation or lack of progress. When a replica gathers $> n/2$ blame messages (Figure 3 step 3), it executes the view change

Let v be the current view number and replica L be the leader of the current view.

1. **Propose.** The leader L broadcasts $\langle B_k \rangle_L := \langle v, \mathcal{C}_{q_f}(B_{k-1}) \rangle_L$ where v is a batch of new transactions and B_{k-1} is the previous block L proposed.
2. **Vote and commit.** Upon receiving the first valid B_k from L, send a vote to L. Commit B_{k-2}.
3. **Blame.** If an equivocating block at height-k is received, or if there is no progress in the responsive mode (the next height block is not received), broadcast $\langle \mathsf{blame}, v \rangle$ and the equivocating blocks.
4. **View-change.** Upon receiving $> n/2$ $\langle \mathsf{blame}, v \rangle$ messages, invoke the view change protocol in Figure 2 and transition to the steady state of Sync HotStuff in the next view.

Figure 3: Protocol of the responsive mode.

Let v be the current view number and replica L be the leader of the current view. The responsive mode runs the following steps in iterations.

1. **Propose.** The leader L broadcasts $B_k := \langle b, \mathcal{C}(B_{k-1}) \rangle_L$ where b is a batch of new transactions and B_{k-1} is the highest certified block known to L. If the leader wants to switch to the responsive mode, it sets $b :=$ switch-to-responsive.
2. **Vote.** Upon receiving the first valid height-k block B_k from L or through a vote by some other replica, if it is extending a highest certified block, broadcast a vote by sending $\langle \mathsf{vote}, B_k \rangle$. Set $\mathsf{commit\text{-}timer}_k$ to 2Δ, blame-timer to 3Δ, and start counting down. If block B_k contains a switch-to-responsive command, do not vote for subsequent blocks in the synchronous mode.
3. **(Asynchronous) Blame.** If blame-timer reaches 0 and the next block is not received, broadcast $\langle \mathsf{blame}, v \rangle$. If more than one height-k blocks have been received from the same leader (leader has equivocated), broadcast $\langle \mathsf{blame}, v \rangle$ and the two equivocated blocks.
4. **(Asynchronous) Commit.** When $\mathsf{commit\text{-}timer}_k$ reaches 0, if B_k is the only height-k block received, commit B_k and all its predecessors. If block B_k is committed and contains the switch-to-responsive command, transition to the responsive mode (Figure 3)

Figure 4: Augmented steady state protocol.

protocol in Figure 2. Intuitively, the view change protocol guarantees that, if a block B_k is committed by any honest replica in a view (in either mode), then all honest replicas receive the relevant certificates by the end of that view (after the 2Δ wait) and will only vote for blocks extending B_k in future views. This preserves the safety of all blocks committed by honest replicas.

To switch from the synchronous mode to the responsive, the leader proposes a block that contains a special command switch-to-responsive (Figure 4 step 1). After voting for the switch command, an honest replica temporarily stops voting for subsequent blocks. When this block is committed in the synchronous mode, the replica transitions to the responsive mode and starts voting according to the rule in Figure 3.

4.2 Safety with Optimistic Responsiveness

We now amend the proofs for safety to account for the addition of the responsive mode. Lemma 1 and 2 only apply to directly committed blocks in the synchronous mode. We provide two similar lemmas for directly committed blocks in the responsive mode.

Lemma 5. *If an honest replica directly commits a block B_l in the responsive mode of a view, then every honest replica receives $\mathcal{C}_{q_f}(B_j)$ by the end of that view for some $j \geq l + 1$ and B_j extends B_l.*

Proof. Simply recall that committing B_l in the responsive mode requires receiving a successor block B_{l+2} which contains the said certificate $\mathcal{C}_{q_f}(B_{l+1})$. The rest of the Lemma 1 proof applies and we repeat for completeness. Consider any two honest replicas h and h': h directly commits B_l in the responsive mode of view v at time t and h' quits view v at time t'. According to the view change protocol, h' sends $> n/2$ blame messages at time $t' - 2\Delta$. At time no later than $t' - \Delta$, h will receive $> n/2$ blame messages. (h at least receives them from h' by that time, but may also receive them earlier from other replicas.) At this point, h broadcasts the highest certified block $\mathcal{C}(B_j)$ to its knowledge, which either is or extends B_{l+1}. h' will receive the above information no later than t'. □

Lemma 6. *If an honest replica directly commits a block B_l in the responsive mode, then there does not exist $\mathcal{C}(B'_{l+1})$ where B'_{l+1} does not extend B_l.*

Proof. Suppose an honest replica h directly commits B_l in the responsive mode of view v. We first prove that such a $\mathcal{C}(B'_{l+1})$ does not exist in views up to v (included). Suppose for contradiction that $\mathcal{C}(B'_{l+1})$ exists. Then, B'_{l+1} must gather at least one honest vote. There are two cases.

Case 1. An honest replica votes for B'_{l+1} in the responsive mode. This requires the existence of $\mathcal{C}_{q_f}(B'_l)$ where B'_{l+1} extends B'_l. Since h committed B_l in the responsive mode, it must have received $\mathcal{C}_{q_f}(B_l)$. Since B'_{l+1} does not extend B_l, we must have $B'_l \neq B_l$. Both blocks have strong certificates. But this requires at least $(3n/4 + 1) + (3n/4 + 1) - n = n/2 + 1 = f + 1$ replicas to double-vote at height l, a contradiction.

Case 2. An honest replica votes for B'_{l+1} in the synchronous mode. To enter the responsive mode, h must have committed in the synchronous mode a block B_{l_0} containing the switch-to-responsive command at a height $l_0 < l$. By Lemma 2, there does not exist a conflicting certified block $\mathcal{C}(B'_{l_0})$ where $B'_{l_0} \neq B_{l_0}$. Thus, B'_{l+1} must extend B_{l_0}. However, an honest replica will not keep voting in the synchronous mode after receiving a switch-to-responsive command.

Next, we prove that such a conflicting certificate does not exist in views greater than v. This part is similar to the proof of Lemma 2. By Lemma 5, every honest replica receives $\mathcal{C}(B_j)$ where $j \geq l + 1$ and B_j extends B_l by the end of view v, and there was no conflicting certificate at height $l + 1$ up until then. From then on, the most preferred block of every honest replica is at least B_{l+1}, and no honest replica will vote for $B'_{l+1} \neq B_{l+1}$. Thus, no $\mathcal{C}(B'_{l+1})$ where B'_{l+1} does not extend B_l can come into existence in future views. □

Theorem 7 (Safety with Optimistic Responsiveness)**.** *Honest replicas always commit the same block B_k for each height k.*

Proof. Suppose for contradiction that two distinct blocks B_k and B'_k are committed at height k. As before, suppose B_k is committed as a result of B_l being directly committed and B'_k is committed as a result of $B'_{l'}$ being directly committed. Without loss of generality, assume $l \leq l'$. The original proof covers the case where B_l is committed in the synchronous mode. In that case, there exists no other certified block at height l by Lemma 2. Thus, $B'_{l'}$ either is or extends B_l, which extends B_k. For the other case where B_l is directly committed in the responsive mode, there does not exist $\mathcal{C}(B'_{l+1})$ where B'_{l+1} does not extend B_l by Lemma 6. Thus, $B'_{l'}$ either is or extends B_l, which extends B_k. □

5 Bound on Responsiveness

Our protocol commits with a 2Δ latency in the steady state when $f < n/2$. For completeness, in this section, we show a lower bound on the latency when $f > n/3$. The lower bound and the proof closely follow Dwork et al. [5]. For clarity, we present the bound in the Byzantine broadcast formulation. Recall that in Byzantine broadcast, a designated *sender* tries to broadcast a value to n parties. A solution needs to satisfy three requirements:

(termination) all honest parties eventually commit,

(agreement) all honest parties commit on the same value, and

(validity) if the sender is honest, then all honest parties commit on the value it broadcasts.

Theorem 8. *There exists no Byzantine broadcast protocol that simultaneously satisfy the following:*

- *termination, agreement and validity as defined above;*

- *tolerates $f \geq n/3$ Byzantine faults;*
- *terminates in less than Δ time if the designated sender is honest.*

Proof. Suppose such a protocol exists. Divide parties into three groups P, Q and R, each of size f. Parties in P and Q are honest and parties in R are Byzantine. Consider the following three scenarios. In Scenario A, parties in R remain silent and an honest designated sender sends 0. In this scenario, parties in P and Q commit 0 in less than Δ time. In Scenario B, parties in R remain silent and an honest designated sender sends 1. In this scenario, parties in P and Q commit 1 in less than Δ time. In Scenario C, the designated sender is Byzantine and sends 0 to P and 1 to Q; parties in R behave like Q in Scenario A to P, and behave like P in scenario B to Q. Messages between P and Q take Δ to reach. This scenario is indistinguishable from Scenario A to P and indistinguishable from Scenario B to Q. Thus, P commits 0 and Q commits 1 in less than Δ time, violating agreement. □

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 *Financial Cryptography and Data Security (FC)*, 2019.

[2] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, and Ling Ren. Dfinity consensus, explored. Cryptology ePrint Archive, Report 2018/1153, 2018.

[3] Danny Dolev, Joseph Halpern, Barbara Simons, and Ray Strong. Dynamic fault-tolerant clock synchronization. *Journal of the ACM*, 42(1):143–185, 1995.

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

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

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

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

[8] Shengyun Liu, Christian Cachin, Vivien Quéma, and Marko Vukolic. XFT: practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 485–500. USENIX Association, 2016.

[9] 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.

[10] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT Consensus in the Lens of Blockchain. *arXiv preprint arXiv:1803.05069*, 2018.