Hybrid-BFT: Optimistically Responsive Synchronous Consensus with Optimal Latency or Resilience

Atsuki Momose[1], Jason Paul Cruz[2], and Yuichi Kaji[1]

[1]Nagoya University, momose@sqlab.jp, kaji@icts.nagoya-u.ac.jp
[2]Osaka University, jpmcruz@ymail.com

April 19, 2020

Abstract

The breakthrough of blockchain in many decentralized cryptocurrencies has reactivated studies on consensus under network synchrony, which has better security than a consensus under network asynchrony but has been considered to be impractical and only theoretical so far. One of the biggest concerns is the speed of transaction processing. To solve this concern, transactions can be processed *responsively*, i.e., without reliance on synchrony. Another approach is to minimize reliance on synchrony to achieve optimal synchronous latency. In this paper, we consider answering the question *"Can we achieve both responsiveness and optimal synchronous latency?"*. To do this, we first show some theoretical possibilities and impossibilities in achieving both responsiveness and optimal synchronous latency, and then we present a practical blockchain or state machine replication protocol we call Hybrid-BFT. Hybrid-BFT can process transactions responsively under normal situation, i.e., small number of faults, while it can achieve optimal synchronous latency even under worst situation. Furthermore, Hybrid-BFT achieves responsive leader change, making it completely free from synchronous delay under normal situation.

1 Introduction

1.1 Consensus under Network Synchrony

A consensus protocol, specifically *state machine replication* [40, 37, 38], is a crucial component in a fault-tolerant computer system. For the past three decades, it has been studied in the areas of distributed algorithms and cryptographic protocols. The main application considered in these studies is a server/database replication service managed by one entity. In such a private environment, performance is prioritized at the cost of *resilience*, which is the maximum number of faults allowed denoted by f. In such situation, consensus under network synchrony has been considered to be only theoretical and an impractical solution due to the following two concerns: (i) To be naturally considered, consensus relying on network synchrony makes latency dependent on the estimated network delay bound Δ, which creates a trade-off between security and latency; (ii) The *standard synchrony* model in which partitions are not allowed is unrealistic in a real-world network environment. To make matters worse, to simplify its model, it has been discussed under lockstep execution where each round is fully synchronized [20, 1], consequently making its latency longer and introducing an additional trade-off.

Since the invention of blockchain following the *Nakamoto Consensus* [34, 36, 22] and its practical use in many decentralized cryptocurrency projects, high resilience has become an important requirement in a public network environment where malicious behaviors occur frequently. Such a consideration has reactivated studies on consensus under network synchrony to use its inherent resilience. In a network with n replicas, consensus under network synchrony can tolerate $< n/2$ byzantine faults while consensus under asynchrony or *partial synchrony* can only tolerate $< n/3$ byzantine faults [21].

Several prior works have tackled the two concerns mentioned above. The first concern, i.e., the latency to process transactions, has been the biggest hindrance towards the practical use of consensus under network synchrony. This latency problem can be tackled using two approaches. The first approach is to process transactions responsively, i.e., with latency of $O(\delta)$, where δ is the actual network delay which is generally $\delta \ll \Delta$. It has been shown that an assumption of $f \geq n/3$ does not always achieve responsiveness [2, 37]. Pass et al. showed that responsiveness can be achieved when the number of actual faults f_{opt} is substantially smaller (e.g., $f_{opt} < n/4$) than what is allowed in its worst case (e.g., $f < n/2$)– they call this property *optimistic responsiveness* [39]. The second approach is to minimize reliance on network synchrony to achieve optimal latency in terms of Δ. Abraham et al. first presented a near optimal $2\Delta + O(\delta)$ protocol [2] and achieved optimal $\Delta + O(\delta)$ in their subsequent work [4]. To tackle the second concern, Guo et al. introduced *weak synchrony* [24] which allows some honest replicas to be offline. In the real world, replicas being offline is common and can be caused by a number of events, e.g., network failure or power failure. Abraham et al. constructed state machine replication protocols under the weak synchrony model.

1.2 Our Contributions

In this paper, we answer the following natural question:

"Can we construct a state machine replication protocol that achieves both optimistic responsiveness and optimal synchronous latency?"

To do this, we first show some theoretical possibilities and impossibilities in achieving both optimistic responsiveness and optimal synchronous latency. Then, we present a practical blockchain or state machine replication protocol we call Hybrid-BFT that can achieve both optimistic responsiveness and optimal/near optimal synchronous latency.

1.2.1 Theoretical Bounds on Latency/Resilience

To simplify the discussion on the theoretical possibilities and impossibilities mentioned above, we first consider *reliable broadcast*, a single-shot consensus primitive, and formally define its latency and optimistic responsiveness. On the negative side, we show that there is an inherent trade-off between its *synchronous latency*, i.e., the latency under f faults, and its *optimistic resilience* f_{opt}. Assuming optimal optimistic resilience to achieve optimistic responsiveness, both optimal $\Delta + O(\delta)$ synchronous latency and optimistic responsiveness cannot be achieved. On the positive side, we show that both optimal $\Delta + O(\delta)$ synchronous latency and optimistic responsiveness can be achieved when the optimistic resilience is compromised. On the other hand, assuming optimal optimistic resilience, we can achieve both near-optimal $2\Delta + O(\delta)$ synchronous latency and optimistic responsiveness. We can consider these protocols as latency favoring or resilience favoring in the trade-off described

above.

Prior works that tried to achieve optimistic responsiveness have only one "mode" of commitment. Put simply, only one of responsive or synchronous commitment is allowed at a time, and if one fails, then switch to the other one. This "single mode" construction makes the synchronous latency of previous protocols longer. In this paper, we introduce "hybrid commitment", which allows the execution of the two commitment modes simultaneously. By interactively committing between two modes, our protocol achieves both optimistic responsiveness and optimal or near-optimal synchronous latency without introducing any inconsistency.

Furthermore, all of these latency favoring or resilience favoring protocols are special cases of one unified reliable broadcast protocol where differences are all absorbed into the protocol parameters. Therefore, our protocol can be considered to be natural and reasonable with respect to theoretical features.

1.2.2 Practical State Machine Replication

After the theoretical discussions, we extend the reliable broadcast protocols to introduce Hybrid-BFT. Hybrid-BFT has a simple leader-full construction, where each predetermined leader is assigned to each time frame of constant length called *view*. We also use the so-called *locking* mechanism [33], which is used in many state machine replication protocols, to maintain consistency across different views. Prior works only needed to care about one mode of commitment, while our proposed protocol needs to care about two modes of commitment. Each commitment in two modes needs to be properly handed over to the next view, which also has two modes. Therefore, we need to care about four relations of locking. This *"hybrid locking"* is a critical component in extending the single-shot reliable broadcast protocol with hybrid commitment into a state machine replication protocol without introducing any inconsistency.

Furthermore, by combining the *three-phase commitment* technique first introduced by Hotstuff [46, 47] with our hybrid locking mechanism, our protocol achieves responsive leader/view change under the presence of a small number of faults. This is a strong and desired feature in a practical implementation because the processing of transactions is completely free from synchronous delay Δ at all times during protocol execution, while leaders continuously change in a democratic manner.

Finally, we extend Hybrid-BFT to be secure under weak synchrony, hereafter called *mobile sluggish synchrony*. The technique we use for synchronous commitment to be robust against offline replicas is similar to the technique in prior works, but constructing a hybrid commitment/locking mechanism and proving its security under mobile sluggish synchrony is not trivial.

1.3 Related Works

We introduce some recent works closely related to ours to compare and clarify our contributions.

1.3.1 Optimistic Responsiveness

Optimistic responsiveness was first introduced by Thunderella [39]. In Thunderella, transactions are first processed responsively through a fast pass, and if this fast pass fails, transactions are processed through a fallback pass with an underlying synchronous blockchain, e.g., Nakamoto Consensus. In the fast pass, transactions are proposed by a predetermined leader and processed by a committee when the number of faults is very small. When transactions fail to be processed responsively,

they are instead processed slowly under synchronous blockchain. This fallback mechanism can be regarded as a mode switching mechanism, which makes the synchronous latency longer.

Sync Hotstuff [2] applies optimistic responsiveness to a classical state machine replication with a bimodal construction, i.e., *steady-state* and *view-change*. Sync Hotstuff has synchronous and responsive modes, and it starts with the synchronous mode. To switch to the responsive mode, at least one block needs to be processed synchronously. If this block collects votes from $> 3n/4$ replicas, then it is processed responsively. Therefore, Sync Hotstuff cannot process transactions responsively immediately after the start of each view. Furthermore, if faulty replicas pretend to be honest at first and then stop voting after the switch to the responsive mode, the protocol cannot progress. Therefore, Sync Hotstuff cannot ensure liveness under the worst situation where f replicas are faulty.

1.3.2 Optimal Synchronous Latency

As mentioned above, prior works have achieved optimal synchronous latency without optimistic responsiveness. Sync Hotstuff first achieved near-optimal $2\Delta + O(\delta)$ latency. It achieves safety by interactively preventing conflicting transactions from being processed. In other words, Sync Hotstuff is "symmetric", i.e., when two replicas try to process conflicting transactions, both transactions are rejected. It is also conjectured that $2\Delta + O(\delta)$ is the optimal synchronous latency because interactions between honest replicas are required for symmetric commitment. However, Abraham et al. achieved optimal $\Delta + O(\delta)$ latency in their subsequent work [4] by non-interactively preventing conflicting transactions from being processed. Their work, in contrast to Sync Hotstuff, is "asymmetric", i.e., when two replicas try to process conflicting transactions, one of the transactions is processed while the other one is rejected.

These works have contributed in making synchronous protocols practical. However, considering today's computer systems that are required to process requests extremely fast, a latency that depends on the securely overestimated delay bound Δ is not enough.

1.4 Paper Organization

In Section 2, we introduce the model under which our protocols are executed. In Section 3, we show some theoretical possibilities and impossibilities to achieve optimistic responsiveness and optimal synchronous latency. In Section 4, we present Hybrid-BFT. In Section 5, we extend Hybrid-BFT for mobile sluggish synchrony. In Section 6, we review additional related works. Finally, in Section 7, we conclude with a summary of our work.

2 Model

We define a protocol Π as an algorithm for a set of *nodes* or *replicas*. Each protocol execution is directed by an *environment* $\mathcal{Z}$, which captures all aspects of external actions, such as clients requests, and proceeds in an atomic time step. At the start of protocol execution, $\mathcal{Z}$ spawns a set of replicas denoted by $\mathcal{N}$ of size n. All replicas in a subset $\mathcal{H} \subset \mathcal{N}$ are honest and faithfully follow Π. The remaining replicas $\mathcal{N} \setminus \mathcal{H}$ of size at most f are byzantine faulty, i.e., fully controlled by an *adversary* $\mathcal{A}$, and behave arbitrarily. We assume *static adversary* for simplicity. At the start of protocol execution, after spawning all replicas, $\mathcal{Z}$ determines which replicas are honest or faulty and cannot be changed throughout the protocol execution. $\mathcal{A}$ is in charge of all communications

between replicas. Here, we assume *standard synchrony*. If an honest replica r sends a message x to another honest replica r' at time t, then r' receives x before $t + \delta$. The delay parameter δ is determined by $\mathcal{Z}$ at the start of protocol execution within the constraint that $\delta \leq \Delta$. All replicas are informed of Δ but not δ, and thus δ can be regarded as an actual network delay that cannot be observed in the real world. $\mathcal{A}$ can delay and reorder, but cannot rewrite, messages within the constraint above. We will extend the standard synchrony to weaker synchrony model called *mobile sluggish synchrony* later in this paper. We assume the use of authenticator and use $\langle x \rangle_r$ to denote a message x signed by a replica r. We omit the signer r when it is clear from the context. We assume the use of collision-resistant hash function H.

3 Reliable Broadcast

In this section we show some theoretical lower/upper bounds on latency and resilience of optimistically responsive consensus protocols. Although our final goal is to construct state machine replication protocols, for simplicity we first deal with reliable broadcast [30, 8], a well-known single-shot consensus primitive.

3.1 Definitions

In a reliable broadcast protocol, a designated sender in a set of replicas broadcasts a value in a consistent manner. At the start of protocol execution, on spawning of replicas, $\mathcal{Z}$ selects a sender $r_{send} \in \mathcal{N}$, informs all replicas, and sends a value v as input to r_{send}. Each honest replica commits v following the protocol. The security properties of reliable broadcast are defined as in Definition 1.

Definition 1 (Security of Reliable Broadcast)**.** *A reliable broadcast protocol must provide the following properties with some polynomials T_{tot}, T_{val} in protocol parameters.*

1. Consistency. *If any two honest replicas commit v and v', respectively, then $v = v'$.*
2. T_{tot}-Totality. *If an honest replica commits a value at time t, then every honest replica commits this value before $t + T_{tot}$.*
3. T_{val}-Validity. *If the sender is honest, then every honest replica commits a value before T_{val}.*

To define the optimistic responsiveness of a reliable broadcast protocol, we first define a property called *optimistic validity* as in Definition 2, which is an extension of the Validity property described above. Here after we call *synchronous latency* as T_{val} and *optimistic latency* as T_{opt}.

Definition 2 (Optimistic Validity)**.** *A reliable broadcast protocol has* (T_{opt}, f_{opt})-Optimistic Validity *if it satisfies the following: If the sender is honest and up to $0 < f_{opt} < f$ (we call it* optimistic resilience*) replicas are faulty, then every honest replica commits a value before T_{opt}.*

We now define the *optimistic responsiveness* of a reliable broadcast protocol as in Definition 3. In the definition of optimistic responsiveness in [39], the optimistic latency is described by a polynomial for versatility. However, most responsive protocols have latency of some constant factor of δ, and thus we follow this definition for simplicity and practicality.

Definition 3 (Optimistic Responsiveness)**.** *A reliable broadcast protocol has* f_{opt}-Optimistic Responsiveness *if it has* $(O(\delta), f_{opt})$-Optimistic Validity.

3.2 Trade-off between Latency and Resilience

We first show the negative aspect, i.e., impossibility results, of the theoretical properties of optimistically responsive reliable broadcast protocols. We show that there is an inherent trade-off between latency and resilience. First, we provide the tight upper bound on the optimistic resilience to achieve optimistic responsiveness in Theorem 1. This upper bound is introduced in [39] for state machine replication, and thus we easily extend the proof for reliable broadcast to be reviewed.

Theorem 1 (Resilience for Optimistic Responsiveness)**.** *Assuming $f_{opt} \geq \frac{n-f}{2}$, there are no reliable broadcast protocols that achieve f_{opt}-Optimistic Responsiveness.*

Proof. Suppose for the sake of contradiction that there exists a reliable broadcast protocol Π that achieves $(C_0 \cdot \delta, f_{opt})$-Optimistic Validity for $f_{opt} \geq \frac{n-f}{2}$ and a constant C_0. We lead to a contradiction by showing a scenario where Π violates its consistency. In the scenario, we use a network delay parameters δ_0 that satisfies $C_0 \cdot \delta_0 < \Delta$. We consider three worlds where each honest replica runs Π to commit a value. In each world, there are three sets of replicas A, B, C, whose size is each $|A| = |\mathcal{C}| = f_{opt}, |B| = n - 2f_{opt} \leq f$, and a sender $r_{send} \in B$.

In world 1, only A is crashed and $\delta = \delta_0$. r_{send} receives a value v_0 from $\mathcal{Z}$ and broadcasts it. At time $t = \delta_0$, each replica receives v_0. From the assumption, C commits v_0 at time $t = C_0 \cdot \delta_0$ communicating with B and not communicating with A.

In world 2, only C is crashed and $\delta = \delta_0$. r_{send} receives a value v_1 from $\mathcal{Z}$ and broadcasts it. At time $t = \delta_0$, each replica receives v_1. From the assumption, A commits v_1 at time $t = C_0 \cdot \delta_0$ communicating with B and not communicating with C.

In world 3, only B is faulty and $\delta = \Delta$. Additionally $\mathcal{A}$ transports messages with delay Δ between A and C and with δ_0 for other links. r_{send} receives a value from $\mathcal{Z}$ but ignores it and sends v_0 to C and v_1 to A. Furthermore, B behaves like in world 1 for C and world 2 for A. From the view of C, worlds 1 and 3 are indistinguishable before $C_0 \cdot \delta_0$ because the messages from A cannot be reached before $C_0 \cdot \delta_0 < \Delta$, thus C commits v_0 at $C_0 \cdot \delta_0$. In the same way, A commits v_1 at $C_0 \cdot \delta_0$. As a result, A and C commit different values, violating the consistency, which is a contradiction. □

We now provide the upper bound on the optimistic resilience to achieve the optimal synchronous latency, i.e., $\Delta + O(\delta)$-Validity, in Theorem 2. In a latter section, we will show that this bound is tight.

Theorem 2 (Resilience for Optimal Synchronous Latency)**.** *Assuming $f_{opt} \geq n - 2f$, there are no reliable broadcast protocols that achieve $\Delta + O(\delta)$-Validity and f_{opt}-Optimistic Responsiveness.*

Proof. Suppose for the sake of contradiction that there exists a reliable broadcast protocol Π that achieves $\Delta + C_0 \cdot \delta$-Validity and $(C_1 \cdot \delta, f_{opt})$-Optimistic Validity for some constant C_0, C_1 and $f_{opt} \geq n - 2f$. We lead to a contradiction by showing a scenario where Π violates its consistency. We illustrate the scenario chronologically in Figure 1. In the scenario, we use network delay parameters δ_0, δ_1 that satisfy $C_0 \cdot \delta_0 < \delta_1 < C_1 \cdot \delta_1 < \Delta < \delta_0 + \Delta < C_0 \cdot \delta_0 + \Delta < \delta_1 + \Delta$. We consider three worlds where each honest replica runs Π to commit a value. In each world, there are three sets of replicas A, B, C, whose size is each $|A| = f, |B| = n - f - f_{opt} \leq f, |\mathcal{C}| = f_{opt}$, and a sender $r_{send} \in B$.

In world 1, only A is faulty and $\delta = \delta_0$. r_{send} receives a value v_0 from $\mathcal{Z}$ and broadcasts it. At time $t = \delta_0$, each replica receives v_0. However, A behaves as if it did not receive v_0 by time δ_1, and

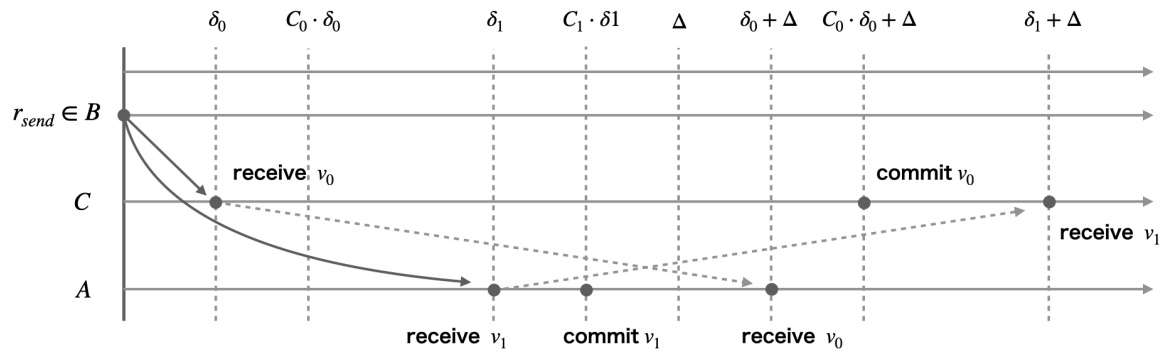

Figure 1: The scenario where consistency is violated in world 3 described in the proof of Theorem 2. The faulty sender r_{send} sends different values v_0 to C and v_1 to A, and $\mathcal{A}$ transport them with different delays. Finally, honest replicas in C and A commit v_0 and v_1, respectively, without interacting with each other.

crashes after δ_1. From the assumption, C commits v_0 at time $t = C_0 \cdot \delta_0 + \Delta$ communicating with B and ignoring A.

In world 2, only C is crashed and $\delta = \delta_1$. r_{send} receives a value v_1 from $\mathcal{Z}$ and broadcasts it. At time $t = \delta_1$, each replica receives v_1. From the assumption, A commits v_1 at time $t = C_1 \cdot \delta_1$ communicating with B and not communicating with A.

In world 3, only B is faulty and $\delta = \Delta$. Additionally $\mathcal{A}$ transports messages with delay δ_0 between B and C, with δ_1 between B and A, and with Δ between A and C. r_{send} receives a value from $\mathcal{Z}$ but ignores it and sends v_0 to C and v_1 to A. Furthermore, B behaves like in world 1 for C and world 2 for A. From the view of C, worlds 1 and 3 are indistinguishable before $C_0 \cdot \delta_0 + \Delta$ because the messages from A after the time when A receives v_1 (i.e., $t = \delta_1$) cannot be reached before $C_0 \cdot \delta_0 + \Delta < \delta_1 + \Delta$, thus C commits v_0 at $\Delta + C_0 \cdot \delta_0$. On the other hand, from the view of A, worlds 2 and 3 are indistinguishable before $C_1 \cdot \delta_1$ because all messages from C cannot be reached before $C_1 \cdot \delta_1 < \Delta$, thus A commits v_1 at $C_1 \cdot \delta_1$. Therefore, A and C commit different value, violating the consistency, which is a contradiction. □

Summarizing the theorems above, we can see, as in Figure 2, the inherent gap between the optimal resilience to achieve optimistic responsiveness (the green line) and the optimal resilience to achieve optimistic responsiveness and optimal synchronous latency (the yellow line). We can also see as a corollary that we can achieve optimal synchronous latency and optimistic resilience only when $f = \frac{n}{3}$. Therefore, when we assume $f > \frac{n}{3}$ and optimal optimistic resilience, the optimal synchronous latency is $k \cdot \Delta + O(\delta)$ for $k > 1$. Furthermore, it can be seen that it is impossible to achieve optimistic responsiveness and optimal synchronous latency when $f \geq \frac{n-1}{2}$.

3.3 Protocols with Optimal Latency/Resilience

We now show the positive aspect, i.e., the possibility results, of the theoretical properties of optimistically responsive reliable broadcast protocols. In the previous section, we provided the upper bound on the resilience to achieve optimal synchronous latency. We show that this bound is tight as formally argued in Theorem 3.

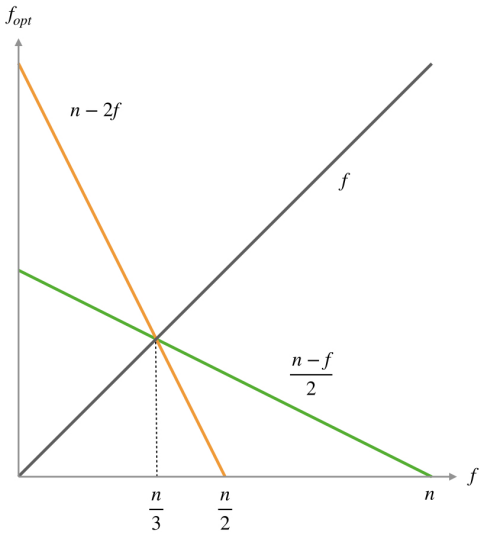

Figure 2: The upper bound on each optimistic resilience f_{opt}. The yellow line is the optimal resilience to achieve optimistic responsiveness, the green one is the optimal resilience to achieve optimistic responsiveness and optimal synchronous latency.

Theorem 3 (Reliable Broadcast with Optimal Synchronous Latency)**.** *Assuming* $\frac{n}{3} \leq f < \frac{n-1}{2}$ *and* $0 < f_{opt} < n-2f$*, a reliable broadcast protocol can achieve both* $\Delta+O(\delta)$*-Validity and* f_{opt}*-Optimistic Responsiveness.*

As shown in the previous section, when we assume $f > \frac{n}{3}$ and optimal optimistic resilience, the optimal synchronous latency is $k \cdot \Delta + O(\delta)$ for $k > 1$. We show that the near optimal synchronous latency $2\Delta + O(\delta)$ is possible, as formally argued in Theorem 4.

Theorem 4 (Reliable Broadcast with Optimal Resilience)**.** *Assuming* $\frac{n}{3} \leq f < \frac{n}{2}$ *and* $0 < f_{opt} < \frac{n-f}{2}$*, a reliable broadcast protocol can achieve both* $2\Delta + O(\delta)$*-Validity and* f_{opt}*-Optimistic Responsiveness.*

To prove these theorems, we construct two reliable broadcast protocols that favor latency and resilience, respectively. Surprisingly, these two protocols are special cases of a unified protocol in which the differences are all absorbed into the protocol parameters.

3.3.1 The Unified Protocol

We introduce the unified reliable broadcast protocol Π_{RBC}, parameterized by a constant k, optimistic resilience f_{opt}, and resilience f. The constant k directly determines the synchronous latency of Π_{RBC}, i.e., Π_{RBC} has $k \cdot \Delta + O(\delta)$ synchronous latency. In Figure 3 we describe Π_{RBC} in detail. Π_{RBC} consists of three sub protocols, Π_{sync}, Π_{rsp}, and Π_{blame}.

Π_{sync} is the *synchronous commitment* protocol that commits a value even under the worst situation where f replicas are faulty. Upon receiving a value b from a designated sender, a replica

$\Pi_{RBC}(k, f_{opt}, f)$

Let r_{send} be the designated sender, and r be a replica. The sender r_{send} broadcast the value b input from $\mathcal{Z}$ in the form of $\langle \mathsf{propose}, b \rangle_{r_{send}}$. Replica r concurrently executes the following sub protocols Π_{sync}, Π_{rsp} and Π_{blame}.

Π_{sync}**:**

1. **Sync:** Upon receiving $\langle \mathsf{propose}, b \rangle_{r_{send}}$, set timer to $k \cdot \Delta$ and start counting down.
2. **Vote:** When timer reaches 0, broadcast a vote in the form of $\langle \mathsf{vote}, b \rangle$.
3. **Commit:** Upon receiving $\mathcal{C}(b)$, commit b.

Π_{rsp}**:**

1. **Vote:** Upon receiving $\langle \mathsf{propose}, b \rangle_{r_{send}}$, broadcast a vote in the form of $\langle \mathsf{vote}, b \rangle$.
2. **Commit:** Upon receiving $\mathcal{S}(b)$, commit b.

Π_{blame}**:**

1. **Blame:** Upon receiving two different values sent from r_{send}, stop voting in Π_{sync} and Π_{rsp}.

Figure 3: Optimistically Responsive RBC with $k \cdot \Delta$ Synchronous Latency

waits for $k \cdot \Delta$ to synchronously check the absence of *equivocation*, i.e., conflicting values sent by a faulty sender. After the synchronous waiting, the replica votes for b. Finally, upon receiving a *normal certificate* for b denoted by $\mathcal{C}(b)$, which is a set of votes for b from at least $n - f$ distinct replicas, the replica commits b.

Π_{rsp} is the *responsive commitment* protocol that commits a value responsively under the optimistic situation where up to f_{opt} replicas are faulty. Upon receiving a value b from a designated sender, the replica immediately votes for b. And then, upon receiving a *strong certificate* for b denoted by $\mathcal{S}(b)$, which is a set of votes for b from at least $n - f_{opt}$ distinct replicas, the replica commits the b. Hereafter, we refer to *certificate* without distinction between normal and strong. The focal point of this construction is that Π_{rsp} concurrently executes the synchronous commitment and the responsive commitment. This removes any mode-switching mechanism that leads to another synchronous waiting. However, if these two commitment protocols or also protocol instances executed by different replicas arbitrarily vote for values, equivocating certificates, i.e., certificates for equivocating values, are created, consequently breaking the consistency.

Π_{blame} is the *blame* protocol that stops the creation of equivocating certificates. Upon receiving equivocation, a replica stops voting for any values from then on, consequently preventing equivocating certificates from being created. This blame mechanism becomes effective if equivocations can be detected properly. Therefore, to complete its security, we need to ensure equivocation detection. Furthermore, we need to consider two concurrently executed protocol instances, which are not considered in the single-mode protocols. This mechanism can be compressed into the parameter constraints. Later in this section, by introducing two types of parameter settings, we explain how parameter constraints can be used to detect equivocation and how they affect the latency and resilience of a protocol. It can easily be calculated that Π_{blame} has synchronous latency of $k \cdot \Delta + 2\delta$ and optimistic latency of 2δ. The additional 2δ is for a value to be received by replicas and for a vote to be received by replicas.

3.3.2 The Latency-Favoring Protocol

We instantiate a protocol from the unified protocol Π_{RBC}. We first introduce the protocol with optimal synchronous latency $\Pi_{RBC}(1, n-2f-1, f)$. The novel point of this parameter setting is that we can ensure that at least one honest replica is in any intersection of two types of quorums. Each type of certificate has two quorums with sizes of $|\mathcal{C}| = n - f$ and $|\mathcal{S}| > 2f$. Therefore, the minimum size of intersection is $|\mathcal{C}| + |\mathcal{S}| - n > f$. This technique prevents synchronous and responsive commitment protocols from creating equivocating certificates without relying on the network synchrony.

3.3.3 The Resilience-Favoring Protocol

We introduce a protocol that achieves optimal optimistic resilience $\Pi_{RBC}(2, \lceil \frac{n-f}{2} - 1 \rceil, f)$. In this parameter setting, we cannot rely on the quorum intersection to prevent synchronous and responsive commitment protocols from creating equivocating certificates. Instead, the additional synchronous waiting prevents the creation of equivocating certificates. We explain how the synchronous waiting for 2Δ prevents the creation of equivocating certificates by considering two cases. Suppose an honest replica r receives a value v at time t and starts waiting for 2Δ in a synchronous commitment protocol. If another honest replica r' in a responsive commitment protocol votes for a different value v', we can consider two cases, i.e., r' votes before or after $t + \Delta$. Considering the former case, v' is received by r before $t + 2\Delta$, preventing r from voting for v. The latter case cannot occur because v should be received by r' before $t + \Delta$, preventing r' from voting for v'.

3.4 Proofs of Security

We provide formal proofs of security for each protocol.

3.4.1 The Latency-Favoring Protocol – $\Pi_{RBC}(1, n-2f-1, f)$

Assuming $\frac{n}{3} \leq f < \frac{n-1}{2}$, $\forall 0 < f_{opt} < n-2f$, the protocol $\Pi_{RBC}(1, n-2f-1, f)$ satisfies consistency, δ-Totality, $\Delta + 2\delta$-Validity, and f_{opt}-Optimistic Responsiveness. The last three properties can be proven in straightforward manner, and thus we only show the proof of consistency in detail. Consider that $|\mathcal{S}| > 2f \geq \frac{n+f}{2}$ under $\frac{n}{3} \leq f$.

Lemma 1 (Certified without Equivocation)**.** *If a value v is certified, then any value $v' \neq v$ is not certified.*

Proof. If a value v is certified, then we can consider the two cases, i.e., normally or strongly certified. In the normally certified case, at least one honest replica r voted for v. Let t be the time when r voted for v. Suppose a v' is certified, then at least one of $\mathcal{C}(v')$ or $\mathcal{S}(v')$ is created. For $\mathcal{C}(v')$ to be created, at least an honest replica r' must have voted for v' after waiting for Δ. If r' voted before t, r' must have received v' before $t - \Delta$ and r must have received v' before t, preventing r from voting for v. For the same reason r' could not have voted after t. Therefore, $\mathcal{C}(v')$ cannot be created. For both $\mathcal{S}(v')$ and $\mathcal{C}(v)$ to be created, at least one honest replica needs to vote for v and v' because the minimum size of intersection of these two sets of replicas each voted for v and v' is $|\mathcal{C}| + |\mathcal{S}| - n > (n - f) + 2f - n > f$. This event cannot occur.

In the strongly certified case, at least $n - f_{opt}$ replicas voted for v. Suppose a v' is certified, then at least one of $\mathcal{C}(v')$ or $\mathcal{S}(v')$ is created. We can exclude both of two cases by considering their

quorum intersection. We can adopt the previous argument regarding $\mathcal{C}(v)$. For both $\mathcal{S}(v')$ and $\mathcal{S}(v)$ to be created, at least one honest replica needs to vote for v and v' because the minimum size of intersection of these two sets of replicas each voted for v and v' is $2|\mathcal{S}| - n > f$. This event cannot occur.

Considering both cases above, $v' \neq v$ is not certified. □

Theorem 5 (Consistency)**.** *If any two honest replicas commit v and v', respectively, then $v = v'$.*

Proof. If an honest replica commits v, then v must have been certified. By Lemma 1, if a value v' is certified, then $v = v'$ must hold. Therefore, if v' is committed, then $v = v'$. □

3.4.2 The Resilience-Favoring Protocol – $\Pi_{RBC}(2, \lceil \frac{n-f}{2} - 1 \rceil, f)$

Assuming $\frac{n}{3} \leq f < \frac{n}{2}$, $\forall 0 < f_{opt} < \frac{n-f}{2}$, the protocol $\Pi_{RBC}(1, \lceil \frac{n-f}{2} - 1 \rceil, f)$ satisfies consistency, δ-Totality, $2\Delta + 2\delta$-Validity, and f_{opt}-Optimistic Responsiveness. The last three properties can be proven in a straightforward manner, and thus we only show the proof of consistency in detail.

Lemma 2 (Certified without Equivocation)**.** *If a value v is certified, then any value $v' \neq v$ is not certified.*

Proof. If a value v is certified, then we can consider two cases, i.e., normally or strongly certified. In the normally certified case, at least one honest replica r voted for v. Let t be the time when r voted for v. Suppose a v' is certified, then at least one of $\mathcal{C}(v')$ or $\mathcal{S}(v')$ is created. We can prove that the former case cannot occur based on the proof of Lemma 1. For $\mathcal{S}(v')$ to be created, at least one honest replica r' needs to vote for v'. If r' voted for v' before $t - \Delta$, then r should have received v' before t, preventing r from voting for v. r' could not vote for v' after $t - \Delta$ because r received v before $t - \Delta$. Therefore, $\mathcal{S}(v')$ cannot be created.

In the strongly certified case, at least $n - f_{opt}$ replicas voted for v. Suppose a v' is certified, then at least one of $\mathcal{C}(v')$ or $\mathcal{S}(v')$ is created. Using the same discussion as in the normally certified case, the strongly certified case cannot occur based on the proof of Lemma 1.

Considering both cases above, $v' \neq v$ is not certified. □

Theorem 6 (Consistency)**.** *If any two honest replicas commit v and v', respectively, then $v = v'$.*

Proof. By Lemma 2, we can prove this in the same way as the proof of Theorem 5. □

4 State Machine Replication under Standard Synchrony

In the previous section, we described a reliable broadcast protocol to show some theoretical possibility and impossibility results for optimistically responsive consensus. In this section, we extend it into a practical blockchain or state machine replication protocol called Hybrid-BFT.

4.1 Overview and Definitions

We first review state machine replication. In a state machine replication protocol, replicas jointly commit an ever-growing history of requests from clients to build a fault-tolerant computer system. State machine replication can be considered as an extension of a single-shot consensus, such as reliable broadcast, to a sequential consensus. At each time, $\mathcal{Z}$ inputs a value or client request to

honest replicas. Each honest replica commits this input to a position in its log, which is a linearized history of inputs. The security properties of state machine replication are defined as in Definition 4.

Definition 4 (Security of State Machine Replication)**.** *A state machine replication protocol must provide the following properties:*

1. Safety. *If any two honest replicas commit v and v', respectively, at the same log position, then $v = v'$.*

2. Liveness. *All inputs received by honest replicas are eventually committed by all honest replicas.*

The protocol proceeds in a time frame with fixed length called *view* and is identified by a monotonically increasing number starting from 0. In each view, a replica designated as *leader* proposes blocks, and replicas try to commit the proposals. In each view, replicas try to commit blocks synchronously as well as responsively. The operation to commit one block is similar to that in Π_{RBC}, but additionally it uses a mechanism that ensures consistency across different views.

The focal point of the protocol is that it achieves responsive view change, i.e., a block B in view v and a block B' in view $v+1$ proposed by two leaders, respectively, can be committed with delay of $O(\delta)$. This is a strong feature of the protocol because a client request can be processed without synchronous delay at any time during protocol execution while changing leaders periodically. We will explain how this can be achieved later in this section. We will first introduce some terms and techniques to simplify the description of the protocol.

4.1.1 Blocking and Chaining

Blocking and chaining are the main features of a blockchain scheme. Blocking means batching of client requests with predetermined order to efficiently commit them together in a subsequence of the log. Blocking is adopted by many state machine replication protocols and is essential in a practical implementation. For simplicity, we consider one client request for one block. Chaining means constructing a hash chain with a block containing a hash reference of its predecessor. Chaining is a strong feature that can be used to translate state machine replication into a simple problem of selecting one path from an ever-growing tree of blocks. Chaining can also be used to handle conflicts in the ordering of descendant/ascendant relations. Therefore, a commitment of a block can be regarded as a commitment of all its ancestors, further simplifying the problem.

4.1.2 Block and its Rank

Each block includes (i) an input b_k from $\mathcal{Z}$ and (ii) a hash h_{k-1} of its predecessor. A *height* of a block is defined as the number of ascendants in the blockchain. Each block of height k is denoted by B_k and formatted as $B_k = (b_k, h_{k-1})$. All blocks have a common ascendant called *genesis block* denoted by B_0 and formatted as $B_0 = (b_0, 0)$. All replicas receive a common genesis block up front along with other protocol parameters. We say a block $B_k = (b_k, h_{k-1})$ is *valid* if (i) there is a valid block B_{k-1} and $h_{k-1} = H(B_{k-1})$ or (ii) B_k is the genesis block.

We define a *rank* between blocks. All blocks in a tree are ranked first by view number, and then by height. For example, a block B_k in view v is higher in rank than a block B_{k+1} in view $v-1$, and a block B_k in view v is higher in rank than a block B_{k-1} in view v.

4.1.3 Conflict and Equivocation

We say a block B *extends* B' if $B = B'$ or B is a descendant of B'. We say blocks B and B' are *conflicting* with each other if they are not equal and do not extend one another. Additionally, if blocks B and B' are conflicting with each other and they are created by the same replica, we say that B and B' are *equivocating* each other. Note that even if a proposer of one of the two blocks is honest, a conflict can occur but not an equivocation, and thus an equivocation is considered to be a byzantine behavior.

4.1.4 Certificate

In the reliable broadcast protocol, we define a certificate for a value. This certificate also serves as a proof that a block was voted for by a quorum of replicas. We use $\mathcal{C}_v(B)$ to denote a *normal certificate* for a block B proposed in view v, and it contains a set of signed votes from at least $n-f$ distinct replicas for block B. This certificate is mainly used for synchronous commitment of blocks just like in Π_{RBC}. On the other hand, for responsive commitment, we use $\mathcal{S}_v(B)$ to denote a *strong certificate* for a block B proposed in view v, and it contains a set of signed votes from $n - f_{opt}$ distinct replicas for block B. Additionally, we define a *higher order certificate* to be used as a proof of repeated vote. We use $\mathcal{S}_v^i(B)$ to denote an *i-th order strong certificate* for a block B proposed in view v, which is a set of signed votes from at least $n - f_{opt}$ distinct replicas for block $\mathcal{S}_v^{i-1}(B)$. Note that $\mathcal{S}_v(B)$ is equal to $\mathcal{S}_v^1(B)$. We only use the higher order certificate to make our protocol easier to understand. In a practical implementation, this higher order certificate can be realized as a type of vote to compress communication complexity.

4.2 Protocol Description

Similar to the reliable broadcast protocol Π_{RBC}, we first construct a unified protocol $\Pi_{SMR}(k, f_{opt}, f)$ and then instantiate some parameter settings. The protocol $\Pi_{SMR}(k, f_{opt}, f)$, which consists of four subprotocols Π_{sync}, Π_{rsp}, Π_{blame}, and Π_{leader}, is described in Figure 7. The first three subprotocols are similar to those in Π_{RBC}, while the last additional subprotocol has a role similar to the designated sender r_{send}. The protocol proceeds in view with length of 7Δ. A prior known leader L for the view v runs Π_{leader} to propose blocks to extend the blockchain, and all replicas run Π_{sync} and Π_{rsp} to jointly commit the proposed blocks synchronously and responsively. Π_{blame} is used to stop voting when L tries to propose equivocating blocks. These subprotocols perform the same process as in Π_{RBC} to commit a block in one view, but an additional mechanism is used to maintain consistency across different views.

4.2.1 Hybrid Locking

To maintain consistency of commitments across different views, we introduce a mechanism called *locking*. Each honest replica has its own lock which points to a block in the blockchain, and the replica can only vote for a block if that block is as high in rank as its lock. Each honest replica sets its lock at the end of a view so that the lock is higher in rank than all blocks committed by honest replicas in the view. This ensures that honest replicas are locked on a descendant of all committed blocks if consistency in the same view is maintained. This mechanism prevents creation of a certificate for a block conflicting with committed blocks after the subsequent views, therefore ensuring the safety of the protocol. The locking mechanism should be carefully constructed such

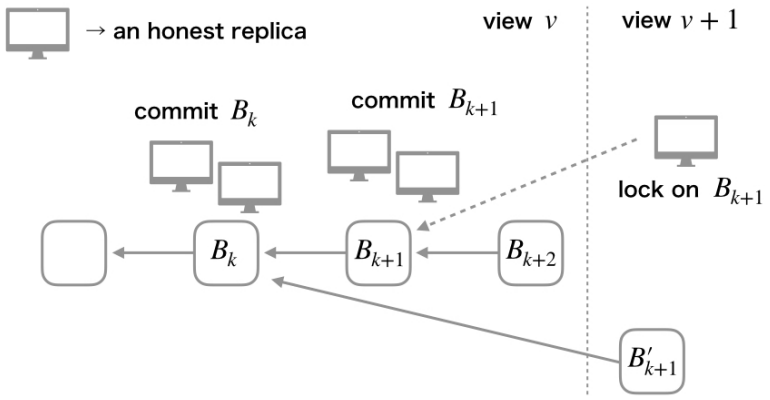

Figure 4: Locking for safety. An honest replica must be locked on B_{k+1} or its descendant block at the end of view v to prevent it from voting for B'_{k+1} conflicting with a committed block B_{k+1} after view $v+1$.

that it does not impair the liveness of the protocol. This mechanism also need to ensure that a subsequent leader knows which blocks honest replicas are locked on so that its proposal can be voted for and committed. We illustrate examples in Figures 4 and 5 to understand locking more intuitively.

The locking mechanism is being used in many state machine replication protocols that only need to consider a single instance for commitment. In our proposed protocol, we need to consider two instances to interactively set their locks. An overview of our proposed "hybrid-locking" is shown in Figure 6. We have two types of lock, sync-lock and rsp-lock, which are used for synchronous and responsive commitments, respectively. Each lock is handed over to each protocol instance of the next view v (gray arrows). To ensure safety, all committed blocks in each protocol instance of the previous view $v-1$ should be set to each lock (red arrows labeled with S-(1–4)). To ensure liveness, a next leader needs to be notified of each lock so that it can propose appropriate blocks (red arrows labeled with N-(1,2)).

4.2.2 The leader protocol – Π_{leader}

In this protocol, the selected leader L of a view proposes blocks to extend the blockchain. Upon receiving an input b_k from $\mathcal{Z}$, L broadcasts a signed block B_k, which includes b_k and a hash h_{k-1} of the predecessor block it extends. To ensure liveness, the proposal needs to be voted for by honest replicas to be committed. To do so, the leader proposes a block that extends a descendant of the highest certified block it knows after (i) receiving status messages from at least $|\mathcal{S}| = n - f_{opt}$ distinct replicas or (ii) waiting for Δ after the start of the view. This ensures that the block is as high in rank as the rsp-lock of all honest replicas, which works for (N-2). We will explain why it works after explaining the responsive commitment protocol in detail.

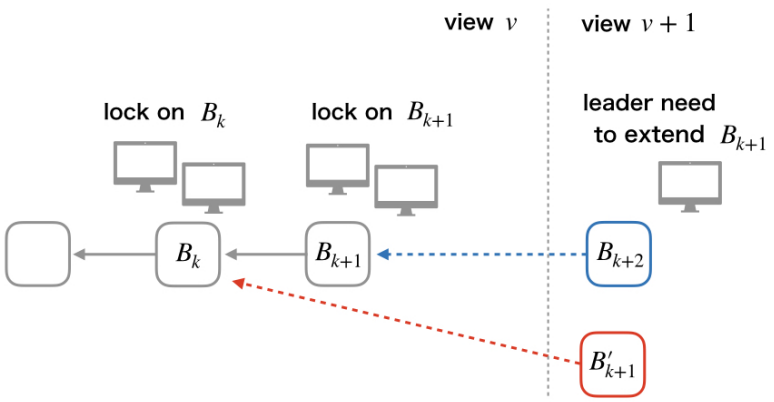

Figure 5: Locking for liveness. The leader of view $v+1$ can propose B_{k+2} extending B_{k+1}, but not B'_{k+1} to be voted for by honest replicas.

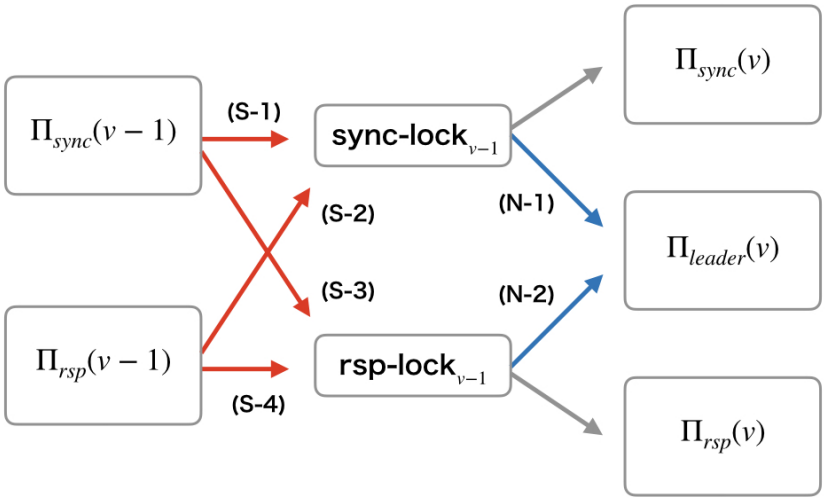

Figure 6: Overview of the hybrid-locking mechanism

4.2.3 The synchronous commitment protocol – Π_{sync}

This protocol ensures that a block is committed synchronously even under the worst situation where f replicas are faulty. The basic process of committing a block is almost the same as in Π_{RBC}. Upon receiving a signed proposal from the leader, a replica waits for $k \cdot \Delta$ and votes for the block if the block extends a descendant of a certified block as high as its sync-lock in the previous view. Finally, upon receiving a normal certificate for the block, the replica commits the block and all its ancestors.

In addition to the basic process described above, three processes related to locking are used. (i) At 5Δ, i.e., 2Δ before the end of the view, the replica stops all voting and commitment, waits for Δ, and then sets the highest certified block to its sync-lock. This process ensures that the replica receives all blocks synchronously committed by honest replicas and reflected to its sync-lock, as shown in (S-1) of Figure 6. (ii) At 6Δ, after setting its sync-lock, the replica sends its sync-lock to the next leader, as shown in (N-1). (iii) Until Δ, the replica collects third-order strong certificates $\mathcal{S}^3_{v-1}(B)$ from the previous view $v-1$. It helps to reflect all responsively committed blocks to sync-lock, which works for (S-2). We will explain why it works after explaining the responsive commitment protocol in detail.

4.2.4 The responsive commitment protocol – Π_{rsp}

This protocol ensures that a block is committed responsively under the optimistic situation where up to f_{opt} replicas are faulty. Due to the locking mechanism, the commitment operation in a view has a slightly different construction from that in Π_{RBC}. Upon receiving a signed proposal from the leader, the replica votes for the block if the block extends a descendant of a certified block as high as its rsp-lock in the previous view. Upon receiving a strong certificate $\mathcal{S}_v(B)$, the replica vote for the block. The voting for higher order certificate is repeated until the replica receives a third-order strong certificate $\mathcal{S}^3_v(B)$ for the block, and then the replica commits the block. At the end of the view ($\mathsf{clock}_v = 7\Delta$), the replica sends the highest certified block it knows to the next leader as a status message and then sets a second-order strong certificate $\mathcal{S}^2_v(B)$ or sync-lock, whichever is higher, to its rsp-lock. All responsively committed blocks are third-order and strongly certified, which ensures that at least $|\mathcal{S}| - f$ honest replicas are locked on the block. This prevents the responsive commitment protocol in a subsequent view from creating a conflicting strong certificate, which works for (S-4). Additionally, since sync-lock is also included a candidate for rsp-lock, it ensures that synchronously committed blocks are also reflected on rsp-lock, as shown in (S-3).

4.2.5 Achieving responsive view change.

The responsive view change considering a single instance has already been achieved in Hotstuff. However, our proposed protocol needs to achieve responsive view change considering two instances, one of which is synchronously operated. (N-2), (S-2), and (N-1) are closely related with it.

An honest replica locked on a second-order strongly certified block indicates that at least $|\mathcal{S}| - f$ honest replicas have strong certificate for the block. Therefore, collecting status messages from at least $|\mathcal{S}|$ distinct replicas ensures that the leader knows their rsp-lock. Additionally, even under the worst situation where f replicas are faulty (even if they pretended to be honest initially), waiting for at most Δ to collect status messages before proposal ensures that rsp-lock is properly handed over to the leader.

All responsively committed blocks are received by all honest replicas before Δ at the beginning of the next view, so it is easy to understand that collecting third order strong certificate for Δ

$\Pi_{SMR}(k, f_{opt}, f)$

At $7v \cdot \Delta$ for all $0 \leq v$, replica r ends view $v-1$ (except for initial view $v=0$) and starts the view v. Replica r concurrently executes subprotocols Π_{sync}, Π_{rsp}, and Π_{blame}, and it initializes clock_v to 0 and starts counting up. A leader L of the view v executes Π_{leader}.

Π_{leader}**:**

1. **First-Propose:** Upon receiving an input b_k from $\mathcal{Z}$, if a block has not been proposed yet in the current view, broadcast $\langle \mathsf{propose}, B_k, v \rangle_L$, where $B_k = (b_k, h_{k-1})$ and h_{k-1} is a hash of a descendant of the highest certified block L knows, if at least one of the following conditions holds.
   (a) It has received status messages from at least $|\mathcal{S}|$ distinct replicas.
   (b) When clock_v reaches Δ.
2. **Propose** Upon receiving an input b_k from $\mathcal{Z}$, if a block has already been proposed in the current view, broadcast $\langle \mathsf{propose}, B_k, v \rangle_L$, where $B_k = (b_k, h_{k-1})$ and h_{k-1} is the hash of the previous proposal.

Π_{sync}**:**

Collect $\mathcal{S}^3_{v-1}(B)$ until Δ and update $\mathsf{sync\text{-}lock}_{v-1,r}$ with B if it is higher in rank than currently set $\mathsf{sync\text{-}lock}_{v-1,r}$, then start the following processes.

1. **Sync:** Upon receiving $\langle \mathsf{propose}, B_k, v \rangle_L$, set timer to $k \cdot \Delta$ and start counting down.
2. **Vote:** When timer reaches 0, if B_k is a descendant of a certified block $B \geq_{rank} \mathsf{sync\text{-}lock}_{v-1,r}$, broadcast a vote in the form of $\langle \mathsf{sync\text{-}vote}, B_k, v \rangle$.
3. **Commit:** Upon receiving $\mathcal{C}_v(B_k)$, commit B_k.
4. **Status:** When clock_v reaches 5Δ, stop voting and commitment.
5. **Lock:** When clock_v reaches 6Δ, set $\mathsf{sync\text{-}lock}_{v,r}$ to the highest certified block and send it to the next leader.

Π_{rsp}**:**

1. **Vote:** Upon receiving $\langle \mathsf{propose}, B_k, v \rangle_L$, if B_k is a descendant of a certified block $B \geq_{rank} \mathsf{rsp\text{-}lock}_{v-1,r}$, broadcast a vote in the form of $\langle \mathsf{rsp\text{-}vote}, B_k, v \rangle$.
2. **Lock-Vote:** Upon receiving $\mathcal{S}_v(B_k)$, broadcast a vote in the form of $\langle \mathsf{rsp\text{-}vote}, \mathcal{S}_v(B_k), v \rangle$.
3. **Commit-Vote:** Upon receiving $\mathcal{S}^2_v(B_k)$, broadcast a vote in the form of $\langle \mathsf{rsp\text{-}vote}, \mathcal{S}^2_v(B_k), v \rangle$.
4. **Commit:** Upon receiving $\mathcal{S}^3_v(B_k)$, commit B_k.
5. **Status & Lock:** When clock_v reaches 7Δ, send the highest certified block $\mathcal{C}_v(B)$ or $\mathcal{S}_v(B)$ to the next leader in the form of $\langle \mathsf{status}, \mathcal{C}_v(B)/\mathcal{S}_v(B), v \rangle$. Set a highest $\mathcal{S}^2_v(B)$ or $\mathsf{sync\text{-}lock}_{v,r}$, whichever is higher, to $\mathsf{rsp\text{-}lock}_{v,r}$.

Π_{blame}**:**

1. **Blame:** Upon receiving two equivocating blocks sent from L, stop voting in Π_{sync} and Π_{rsp}.

Figure 7: Hybrid-BFT- State Machine Replication under Standard Synchrony

and reflecting them on sync-lock at the start of synchronous commitment works for (S-2). However the most importantly, we need to ensure that the sync-lock set after the beginning of the view also properly handed over to the leader even though the leader can propose a block responsively without waiting for Δ. We can also rely on the third-order certificate in this aspect. If a block is third-order strongly certified, it ensure that at least $|\mathcal{S}| - f$ honest replicas are locked on it with rsp-lock, so if rsp-lock is guaranteed to be handed over to the leader, sync-lock set after synchronous waiting also properly handed over to the leader.

Summarizing the techniques above, hybrid-locking, which can be constructed by using higher order certificates and synchrony under the worst situation, can be used to achieve responsive view change without any inconsistencies.

4.3 Evaluation

4.3.1 Latency and Throughput

The latency to process client requests can fluctuate because of the periodical leader change. Therefore, we show both the maximum latency and minimum latency. Note that we evaluate the *good-case latency* [4], which is the latency when leaders are honest. Under the worst situation where f replicas are faulty, client request are committed through Π_{sync}, which typically has 2 rounds of communication including proposal and voting. Including synchronous timeout, the minimum latency is $k \cdot \Delta + 2\delta$. A client request waiting for a leader change needs 3Δ, including timeout in the Status phase, Lock phase, and at the beginning of the next view. Therefore, the maximum latency is $(k+3)\Delta + 2\delta$. Under normal situation where up to f_{opt} replicas are faulty, client requests are committed through Π_{rsp}, which typically has 4 rounds of communication including proposal and 3 rounds of voting. Thus, the minimum latency is 4δ. A client request waiting for a leader change needs a round of communication for the next leader to collect status messages. Therefore, the maximum latency is 5δ. As mentioned before, latency is still responsive even with leader change and therefore it is practical.

Upon receiving client requests, the leader can propose blocks at any time without waiting for commitments or certificate of its ascendants. Therefore, theoretically, there is no limit on throughput even when the size of a block is fixed (the block size is often limited for estimated Δ to be secure [19, 42]).

4.3.2 Resilience

Resilience should be set as protocol parameters in a practical implementation. For a resilience-favoring protocol, $f < n/2$ (i.e., 50%) and $f_{opt} < n/4$ (i.e., 25%) can be set under the optimal setting. On the other hand, for a latency-favoring protocol, $f < 2n/5$ (i.e., 40%) and $f_{opt} < n/5$ (i.e., 20%), for example, can be set. A resilience-favoring protocol has higher resilience than a latency-favoring protocol because of the trade-off discussed in Section 3.2. Nevertheless, it is still reasonable and secure than inherent bound $f < n/3$ (i.e., about 33%) under asynchrony or partial synchrony.

4.3.3 Leader Election

A practical implementation of either a public or private network needs to carefully consider a leader election mechanism. Example mechanisms include a round-robin election and a random election using an unbiasable randomeness source [44, 25]. Leader election mechanisms depend on the target application of a network, and thus we do not discuss them in detail here.

4.4 Proofs of Security

First, we define notations. Let $view(B)$ denote the view where a block B is proposed, $>_{rank}$ denote the order of block in rank, and $A >_{rank} B$ denote that A is higher in rank than B. Global time is sometimes described as (v, t), where v is the view number and t is a local time that clock_v indicates. We use this notation to denote a state of each variable change over time, e.g, $\mathsf{rsp\text{-}lock}_{v,r}(v, \Delta)$ denotes a value of $\mathsf{rsp\text{-}lock}_{v,r}$ at (v, Δ). Let $\rightarrow$ denote a descendant/ascendant relation, i.e., $A \rightarrow B$ denotes that A extends B. This notation is also used for describing conflict/equivocation. $A \not\Leftrightarrow B$

denotes that A and B are conflicting with each other, and if A and B have the same view number, it denotes that A and B equivocate each other.

4.4.1 Latency-Favoring Protocol – $\Pi_{SMR}(1, n-2f-1, f)$

In this subsection, we assume $\frac{n}{3} \leq f < \frac{n-1}{2}$.

Lemma 3 (Certified without Equivocation)**.** *If a block B in view v is certified, then any block B' equivocating B is not certified.*

Proof. We can prove it in the same way as in the proof of Lemma 1. □

Lemma 4. *If an honest replica directly and responsively commits a block B_k in view v, $\forall v' \geq v$, (i) $\exists R \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{S}| - f), \forall r \in R$, $\mathsf{rsp\text{-}lock}_{v',r} \rightarrow B_k$, (ii) $\forall r \in \mathcal{H}$, $\mathsf{sync\text{-}lock}_{v',r} \rightarrow B_k$, and (iii) $\forall B$ $(view(B) = v')$, if B is certified and $B \geq_{rank} B_k$, then $B \rightarrow B_k$.*

Proof. We prove it by induction on the view number. We first prove for the base case $v' = v$. (iii) is easily shown from Lemma 3. Suppose an honest replica directly and responsively commits a block B_k in view v, then it observes $\mathcal{S}_v^3(B_k)$ in view v. This ensures that $\exists R \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{S}| - f)$, $\forall r \in R$, r observes $\mathcal{S}_v^2(B_k)$, and thus r sets its $\mathsf{rsp\text{-}lock}_{r,v}$ to B_k or a block B which is certified and higher in rank than B_k. By (iii) which was already proven, block B extends B_k, and therefore $\mathsf{rsp\text{-}lock}_{r,v}$ extends B_k, which proves (i). Furthermore, all honest replicas observe $\mathcal{S}_v^3(B_k)$ before Δ in the next view, thus all honest replicas set their $\mathsf{sync\text{-}lock}_{r,v}$ to B_k or a block B which is certified and higher in rank than B_k. By (iii) which was already proven, block B extends B_k, and therefore $\mathsf{sync\text{-}lock}_{r,v}$ extends B_k, which proves (ii).

Next, we prove for the inductive step. We first prove (iii). Suppose for the sake of contradiction that $\exists B$ $(view(B) = v' + 1)$, where B is certified and $B \geq_{rank} B_k$ and $B \not\rightarrow B_k$. Then, there exists a certified block B_{min} with lowest rank in all certified blocks which satisfy $B \rightarrow B_{min}$ and $view(B_{min}) = v' + 1$. We can consider the two cases where B_{min} is normally and strongly certified. If B_{min} is normally certified, then $\exists r \in \mathcal{H}$, r sync-vote for B_{min}. This ensures that $\exists B_{min}^{-1}, \forall r \in \mathcal{H}$, $\mathsf{sync\text{-}lock}_{r,v'} \leq_{rank} B_{min}^{-1}$ and $B_{min} \rightarrow B_{min}^{-1}$ and $view(B_{min}^{-1}) \leq v'$ and B_{min}^{-1} is certified. If $B_{min}^{-1} \geq_{rank} B_k$, then by the inductive hypothesis of (iii), $B_{min}^{-1} \rightarrow B_k$, which contradicts that $B \not\rightarrow B_k$. If $B_{min}^{-1} <_{rank} B_k$, then by the inductive hypothesis of (ii), $B_{min}^{-1} <_{rank} \mathsf{sync\text{-}lock}_{r,v}$, which is a contradiction. If B_{min} is strongly certified, then $\exists R \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{S}| - f), \forall r \in R$, r rsp-vote for B_{min}. This ensures that $\exists B_{min}^{-1}, \forall r \in R$, $\mathsf{rsp\text{-}lock}_{r,v'} \leq_{rank} B_{min}^{-1}$ and $B_{min} \rightarrow B_{min}^{-1}$ and $view(B_{min}^{-1}) \leq v'$ and B_{min}^{-1} is certified. If $B_{min}^{-1} \geq_{rank} B_k$, then by the inductive hypothesis of (iii), $B_{min}^{-1} \rightarrow B_k$, which contradicts that $B \not\rightarrow B_k$. If $B_{min}^{-1} <_{rank} B_k$, then by the inductive hypothesis of (i), $\exists R' \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{S}| - f), \forall r' \in R', \mathsf{rsp\text{-}lock}_{r',v'} >_{rank} B_{min}^{-1}$. Since $R \cap R' \neq \emptyset$, then $\exists r'' \in R, \text{rsp-lock}_{r'',v'} >_{rank} B_{min}^{-1}$, which is a contradiction.

We now prove (i). Suppose for the sake of contradiction, for R which is guaranteed to exist for view v' by the inductive hypothesis of (i), that $\exists r \in R, \text{rsp-lock}_{v'+1,r} \not\rightarrow B_k$. Let B_0 be this $\text{rsp-lock}_{v'+1,r}$. By (iii) which was already proven, $B_0 <_{rank} B_k$. Since $\text{rsp-lock}_{v',r} \rightarrow B_k$, then $\text{rsp-lock}_{v',r} >_{rank} B_0$. Since rsp-lock cannot be updated with a block with lower rank, B_0 cannot be $\text{rsp-lock}_{v'+1,r}$, which is a contradiction.

Finally, we prove (ii). Suppose for the sake of contradiction that $\exists r \in \mathcal{H}, \text{sync-lock}_{v'+1,r} \not\rightarrow B_k$. Let B_0 be this $\text{sync-lock}_{v'+1,r}$. By (iii) which was already proven, $B_0 <_{rank} B_k$. Since $\text{sync-lock}_{v',r} \rightarrow B_k$, then $\text{sync-lock}_{v',r} >_{rank} B_0$. Since rsp-lock cannot be updated with a block with lower rank, B_0 cannot be $\text{sync-lock}_{v'+1,r}$, which is a contradiction. □

Lemma 5. *If an honest replica directly and synchronously commits a block B_k in view v, $\forall v' \geq v$, (i) $\forall r \in \mathcal{H}$, $\mathsf{sync\text{-}lock}_{v',r} \rightarrow B_k$ and $\mathsf{rsp\text{-}lock}_{v',r} \rightarrow B_k$, and (ii) $\forall B$ $(view(B) = v')$, if B is certified and $B \geq_{rank} B_k$, then $B \rightarrow B_k$.*

Proof. We prove it by induction on the view number. We first prove for the base case $v' = v$. (ii) is easily shown from Lemma 3. Suppose an honest replica directly and synchronously commits a block B_k in view v, then it observes $\mathcal{C}_v(B_k)$ before 5Δ in view v. This ensures that $\forall r \in \mathcal{H}$, r observes $\mathcal{C}_v(B_k)$ before 6Δ in view v, and thus r sets its $\mathsf{rsp\text{-}lock}_{r,v}$ and $\mathsf{sync\text{-}lock}_{r,v}$ to B_k or a block B which is certified and higher in rank than B_k. By (ii) which was already proven, the block B extends B_k, and therefore $\mathsf{rsp\text{-}lock}_{r,v}$ and $\mathsf{sync\text{-}lock}_{r,v}$ extend B_k, which proves (i).

Next, we prove for the inductive step. We first prove (ii). Suppose for the sake of contradiction that $\exists B$ $(view(B) = v'+1)$, B is certified and $B \geq_{rank} B_k$ and $B \not\rightarrow B_k$. Then, there exists a certified block B_{min} with lowest rank in all certified blocks which satisfies $B \rightarrow B_{min}$ and $view(B_{min}) = v'+1$. We can consider the two cases when B_{min} is normally and strongly certified. If B_{min} is normally certified, then $\exists r \in \mathcal{H}$, r sync-vote for B_{min}. This ensures that $\exists B^{-1}_{min}, \forall r \in \mathcal{H}$, $\mathsf{sync\text{-}lock}_{r,v'} \leq_{rank} B^{-1}_{min}$ and $B_{min} \rightarrow B^{-1}_{min}$ and $view(B^{-1}_{min}) \leq v'$ and B^{-1}_{min} is certified. If $B^{-1}_{min} \geq_{rank} B_k$, then by the inductive hypothesis of (ii), $B^{-1}_{min} \rightarrow B_k$, which contradicts that $B \not\rightarrow B_k$. If $B^{-1}_{min} <_{rank} B_k$, then by the inductive hypothesis of (i), $B^{-1}_{min} <_{rank} \mathsf{sync\text{-}lock}_{v,r}$, which is a contradiction. If B_{min} is strongly certified, then $\exists R \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{S}| - f), \forall r \in R$, r rsp-vote for B_{min}. This ensure that $\exists B^{-1}_{min}, \forall r \in R$, $\mathsf{rsp\text{-}lock}_{r,v'} \leq_{rank} B^{-1}_{min}$ and $B_{min} \rightarrow B^{-1}_{min}$ and $view(B^{-1}_{min}) \leq v'$ and B^{-1}_{min} is certified. If $B^{-1}_{min} \geq_{rank} B_k$, then by the inductive hypothesis of (ii), $B^{-1}_{min} \rightarrow B_k$, which contradicts that $B \not\rightarrow B_k$. If $B^{-1}_{min} <_{rank} B_k$, then by the inductive hypothesis of (i), $\exists R' \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{S}| - f), \forall r' \in R', \mathsf{rsp\text{-}lock}_{r',v'} >_{rank} B^{-1}_{min}$. Since $R \cap R' \neq \emptyset$, then $\exists r'' \in R, \text{rsp-lock}_{r'',v'} >_{rank} B^{-1}_{min}$, which is a contradiction.

Finally, we prove (i). Suppose for the sake of contradiction that $\exists r \in \mathcal{H}, \text{sync-lock}_{v'+1,r} \not\rightarrow B_k$ or $\text{rsp-lock}_{v'+1,r} \not\rightarrow B_k$. Let B_0 be this $\text{sync-lock}_{v'+1,r}$ or $\text{rsp-lock}_{v'+1,r}$. By (ii) which was already proven, $B_0 <_{rank} B_k$. Since $\text{sync-lock}_{v',r} \rightarrow B_k$ and $\text{sync-lock}_{v'+1,r} \rightarrow B_k$, then $\text{sync-lock}_{v',r} >_{rank} B_0$ and $\text{rsp-lock}_{v',r} >_{rank} B_0$. Since sync-lock and rsp-lock cannot be updated with a block with lower rank, B_0 cannot be $\text{sync-lock}_{v'+1,r}$ or $\text{rsp-lock}_{v'+1,r}$, which is a contradiction. □

Lemma 6 (Unique Extensibility)**.** *If an honest replica commit directly commits a block B_k in view v, then (i) there does not exist a certificate for block $B'_k \neq B_k$ in the same view, and (ii) all certified blocks with higher rank extend B_k.*

Proof. These are proven in (iii) of Lemmas 4 and 5 where B_k is responsively and synchronously committed, respectively. □

Theorem 7 (Safety)**.** *If an honest replica commits a block B_k, then all honest replicas do not commit a different block B'_k for the height k.*

Proof. Suppose two blocks B_k and B'_k are committed at height k. Suppose each commitment is a result of a direct commitment of B_l and $B_{l'}$, and every directly committed block is certified. Therefore, by Lemma 6, $B_l \leftrightarrow B_{l'}$, thus $B_k = B_{k'}$. □

Lemma 7 (Leader observes $\mathsf{rsp\text{-}lock}$)**.** *If an honest leader L of view v proposes a block B, then there exits a block B^{-1} which is certified and $\forall r \in \mathcal{H}, B^{-1} \geq_{rank} \mathsf{rsp\text{-}lock}_{v-1,r}$.*

Proof. An honest replica r sets a highest second-order strong certificate $\mathcal{S}^2_{v-1}(B')$ for a block B' or $\mathsf{sync\text{-}lock}_{v-1,r}(v-1, 6\Delta)$, whichever is higher, to $\mathsf{rsp\text{-}lock}_{v-1,r}$. If the latter one is selected, then L already observed sync-lock$_{v-1,r}(v-1, 6\Delta)$ before proposal since L receives it from all honest replicas by $(v-1, 7\Delta)$. If the former one is selected, then $\exists R \subset \mathcal{H}$ $(|R| \geq |\mathcal{S}| - f), \forall r' \in R$, r' observes $\mathcal{S}_{v-1}(B')$. If L proposes B before (v, Δ), L receives status messages from at least $|\mathcal{S}| - f$ honest replicas (say R'). Since $R \cap R' \neq \emptyset$, L observes $\mathcal{S}_{v-1}(B')$. If L proposes B after (v, Δ), L receives status messages from all honest replicas, thus L observes $\mathcal{S}_{v-1}(B')$ through the status messages. Therefore, since L observes all certified blocks which are set to $\mathsf{rsp\text{-}lock}$ in some honest replicas before proposal and L proposes a block extending a highest certified block B^{-1} it knows, then B^{-1} is as high as all $\mathsf{rsp\text{-}lock}$ in honest replicas. □

Lemma 8 (Leader observes $\mathsf{sync\text{-}lock}$)**.** *If an honest leader L of view v proposes a block B, then there exits a block B^{-1} which is certified and $\forall r \in \mathcal{H}, B^{-1} \geq_{rank} \mathsf{sync\text{-}lock}_{v-1,r}(v, \Delta)$.*

Proof. An honest replica r sets a highest $\mathcal{S}^3_{v-1}(B')$ at (v, Δ) or a highest certified block at $(v-1, 6\Delta)$, whichever is higher, to $\mathsf{sync\text{-}lock}_{v-1,r}$. If the latter one is selected, then L already observed sync-lock$_{v-1,r}(v-1, 6\Delta)$ before proposal since L receives it from all honest replicas by $(v-1, 7\Delta)$. If the former one is selected, then $\exists R \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{S}| - f), \forall r' \in R$, r observes $\mathcal{S}^2_{v-1}(B')$. If L propose B before (v, Δ), L receives status messages from at least $|\mathcal{S}| - f$ honest replicas (say R'). Since $R \cap R' \neq \emptyset$, L observes $\mathcal{S}_{v-1}(B')$. If L proposes B after (v, Δ), L receives status messages from all honest replicas, thus L observes $\mathcal{S}_{v-1}(B')$ through the status messages. Therefore, since L observes all certified blocks which are set to $\mathsf{sync\text{-}lock}$ in some honest replicas before proposal and L proposes a block extending a highest certified block B^{-1} it knows, then B^{-1} is as high as all $\mathsf{sync\text{-}lock}$ in honest replicas. □

Theorem 8 (Liveness)**.** *If the leader L of view v is honest, then all honest replicas (i) commit responsively at least one block if up to f_{opt} replicas are faulty, and (ii) always commit synchronously at least one block.*

Proof. If L is honest, then L proposes a block B by (v, Δ) and all honest replicas receive the block by $(v, 2\Delta)$. By Lemmas 7 and 8, all honest replicas rsp-vote for B by $(v, 2\Delta)$ and sync-vote for B by $(v, 3\Delta)$. Therefore, if up to f_{opt} replicas are faulty, then all honest replicas receive $\mathcal{S}^3_v(B)$ by $(v, 5\Delta)$ and responsively commit B. On the other hand, all honest replicas always receive $\mathcal{C}_v(B)$ by $(v, 4\Delta)$ and synchronously commit B. □

4.4.2 Resilience-Favoring Protocol – $\Pi_{SMR}(2, \lceil \frac{n-f}{2} - 1 \rceil, f)$

We assume $\frac{n}{3} \leq f < \frac{n}{2}$. We can prove its safety and liveness in the same way as for $\Pi_{SMR}(1, n - 2f - 1, f)$, so we omit the details of the proof.

5 State Machine Replication under Mobile Sluggish Synchrony

In this section, we extend Hybrid-BFT to be secure under the weaker synchrony model called *mobile sluggish synchrony*.

5.1 Definitions

5.1.1 Mobile Sluggish Synchrony

We first extend the standard synchrony into mobile sluggish synchrony. In the standard synchrony, up to f replicas are faulty and fully controlled by $\mathcal{A}$. On the other hand, in the mobile sluggish synchrony, up to $b \leq f$ replicas are faulty. Instead, $\mathcal{A}$ can control communications of up to $f - b$ honest replicas without synchrony, i.e., up to $f - b$ replicas are honest but under asynchrony. The remaining $n-f$ replicas are honest and under synchrony. Furthermore, the honest but asynchronous replicas are mobile, i.e., they change over time among all honest replicas. Formally, at each time t, there is a set of replicas $\mathcal{P}_t \subseteq \mathcal{H}$ ($|\mathcal{P}_t| \geq n - f$), and if $r \in \mathcal{P}_t$ sends a message x to a replica $r' \in \mathcal{P}_{t'}$, r' receives the message x by t' if $t' \geq t + \Delta$. We say a replica r is *prompt* if $r \in \mathcal{P}_t$ at t, and a r is *sluggish* if $r \in \mathcal{H} \setminus \mathcal{P}_t$ at t.

5.1.2 Qualification and Certificate

We define a qualification for a data. Qualification is a proof for a data to be recognized by at least one prompt replica. A replica can create a qualify message for a data to acknowledge its recognition of the data. The *qualification* for a data is a set of signed qualify messages for the data created by at least $n - f$ distinct replicas. A qualification for a data x is denoted by $\mathcal{Q}(x)$. Assuming $f < n/2$, a qualification ensures that at least one of the replicas qualifying a data is prompt at that time since $n - f > f$. Therefore, the data is assured to be recognized by all prompt replicas by Δ after that. The normal certificate has almost the same role because its size is the same, i.e., $|\mathcal{Q}| = |\mathcal{C}|$. However, we distinguish them from each other to clarify the differences between Π^*_{SMR} and Π_{SMR}.

5.2 Protocol Description

The protocol Π^*_{SMR} is described in detail in Figure 8. Its construction is similar to that of Π_{SMR} but with small differences. For clarity, the modified parts are black colored and the gray colored parts are exactly the same as in Π_{SMR}. The two main modifications include (i) the qualification for a proposal to be voted for Π_{sync}, and (ii) the qualification for a certificate to be committed for both Π_{sync} and Π_{rsp}. Both modifications ensure consistency, with the former within the same view and the latter across different views and related to the locking.

5.2.1 Qualification for Proposal

Upon receiving a proposal, a replica broadcasts a qualify messages for the block and set its timer after receiving a qualification to the block. This modification prevents equivocating certificates from being created.

We first explain how two equivocating normal certificates are prevented from being created. As explained in Section 5.1, a qualification for a block indicates that at least one prompt replica recognizes the block. This ensures that all prompt replicas recognize the block Δ after that. When two equivocating blocks are qualified, all prompt replicas receive both blocks but do not vote for the latter one. If all prompt replicas do not vote for a block, then the block cannot be certified. Therefore, the certification for the latter qualified block cannot be created.

We now explain how a normal certificate and a strong certificate equivocating each other are prevented from being created. In this case, we consider $\Pi^*_{SMR}(2, \lceil \frac{n-f}{2} - 1 \rceil, f)$ because we can rely on the quorum intersection for $\Pi^*_{SMR}(1, n-2f-1, f)$, as explained in the previous section. Suppose

a block B is qualified at t, then all prompt replicas recognize B at $t + \Delta$ and do not vote for an equivocating block B' after that. On the other hand, when a prompt replica recognizes B' before $t + \Delta$, B' is received by all prompt replicas before $t + 2\Delta$. Both normal and strong certificates need at least one vote from a prompt replica, and thus both of them cannot be created.

5.2.2 Qualification for Normal Certificate

Upon receiving a normal certificate for a block, a replica broadcasts a qualify message for the certificate and commits the block after receiving a qualification for the certificate. This ensures that a synchronously committed block can be properly handed over to the next view through the lock, which works for (S-1) and (S-3) in Figure 6.

A qualified certificate for a block indicates that at least one prompt replica recognizes the certificate. This ensures that all prompt replicas will receive the certificate in the Status process and will be locked on it at the end of the view. As mentioned above, both normal and strong certificates need at least one vote from a prompt replica, and thus the lock prevents any conflicting blocks with the committed block from being certified after the next view.

5.2.3 Qualification for Third-Order Strong Certificate

As in the synchronous commitment, we also need to ensure that all prompt replicas are locked on all responsively committed blocks by some honest replicas. For rsp-lock, i.e., (S-4) in Figure 6, we can guard in the same way as in Π_{SMR}. On the other hand, we need to ensure that all committed blocks are reflected on sync-lock, which works for (S-2) in Figure 6. To do this, similar to the synchronous commitment, upon receiving a third-order strong certificate, a replica broadcasts a qualify message for the certificate and commits the block after receiving a qualification.

5.3 Evaluation

The latency to process client requests in Π^*_{SMR} is longer than that in Π_{SMR} because of the additional communication rounds for two qualifications. The latency under the normal and worst situations can be calculated in the same way as in Π_{SMR}. Under the worst situation where f replicas are faulty, the minimum latency is $k \cdot \Delta + 4\delta$ and the maximum latency is $(k + 3)\Delta + 4\delta$. Under the normal situation where up to f_{opt} replicas are faulty, the minimum latency is 5δ and the maximum latency is 6δ. Under $\delta \ll \Delta$, the latency is not so different from that in Π_{SMR}.

5.4 Proofs of Security

5.4.1 Latency-Favoring Protocol – $\Pi^*_{SMR}(1, n - 2f - 1, f)$

In this subsection, we assume $\frac{n}{3} \leq f < \frac{n-1}{2}$.

Lemma 9 (Certified without Equivocation)**.** *If a block B in view v is certified, then any block B' equivocating B is not certified.*

Proof. Suppose for the sake of contradiction that two blocks B and B' equivocating each other are both certified in view v. We can consider three cases: (i) B and B' are both normally certified, (ii) B and B' are both strongly certified, and (iii) B is normally certified and B' is strongly certified.

$\Pi^{*}_{SMR}(k, f^{<}_{opt}, f^{<})$

At $9v \cdot \Delta$ for all $0 \leq v$, replica r ends view $v-1$ (except for initial view $v=0$) and starts the view v. Replica r concurrently executes subprotocols Π_{sync}, Π_{rsp} and Π_{blame}, and initializes clock_v to 0 and starts counting up. A leader L of the view v executes Π_{leader}.

Π_{leader}**:**

1. **First-Propose:** Upon receiving an input b_k from $\mathcal{Z}$, if a block has not been proposed yet in the current view, broadcast $\langle \mathsf{propose}, B_k, v \rangle_L$, where $B_k = (b_k, h_{k-1})$, and h_{k-1} is a hash of a descendant of the highest certified block L knows, if at least one of the following conditions holds.
   (a) It has received status messages from at least $|\mathcal{S}|$ distinct replicas.
   (b) When clock_v reaches Δ.
2. **Propose** Upon receiving an input b_k from $\mathcal{Z}$, if a block has already been proposed in the current view, broadcast $\langle \mathsf{propose}, B_k, v \rangle_L$, where $B_k = (b_k, h_{k-1})$, and h_{k-1} is the hash of its previous proposal.

Π_{sync}**:**

Collect $\mathcal{S}^3_{v-1}(B)$ until Δ and update $\mathsf{sync\text{-}lock}_{v-1,r}$ with B if it is higher in rank than currently set $\mathsf{sync\text{-}lock}_{v-1,r}$, then start the following processes.

1. **Qualify:** Upon receiving $\langle \mathsf{propose}, B_k, v \rangle_L$, broadcast a qualify message in the form of $\langle \mathsf{qualify}, B_k, v \rangle$.
2. **Sync:** Upon receiving $\mathcal{Q}(B_k)$, set timer to $k \cdot \Delta$ and start counting down.
3. **Vote:** When timer reaches 0, if B_k is a descendant of a certified block $B \geq_{rank} \mathsf{sync\text{-}lock}_{v-1,r}$, broadcast a vote in the form of $\langle \mathsf{sync\text{-}vote}, B_k, v \rangle$.
4. **Qualify:** Upon receiving $\mathcal{C}_v(B_k)$, broadcast a qualify message in the form of $\langle \mathsf{qualify}, \mathcal{C}_v(B_k), v \rangle$.
5. **Commit:** Upon receiving $\mathcal{Q}(\mathcal{C}_v(B_k))$, commit B_k.
6. **Status:** When clock_v reaches 7Δ, stop voting and commitment.
7. **Lock:** When clock_v reaches 8Δ, set $\mathsf{sync\text{-}lock}_{v,r}$ to the highest certified block, and send it to the next leader.

Π_{rsp}**:**

1. **Vote:** Upon receiving $\langle \mathsf{propose}, B_k, v \rangle_L$, if B_k is a descendant of a certified block $B \geq_{rank} \mathsf{rsp\text{-}lock}_{v-1,r}$, broadcast a vote in the form of $\langle \mathsf{rsp\text{-}vote}, B_k, v \rangle$.
2. **Lock-Vote:** Upon receiving $\mathcal{S}_v(B_k)$, broadcast a vote in the form of $\langle \mathsf{rsp\text{-}vote}, \mathcal{S}_v(B_k), v \rangle$.
3. **Commit-Vote:** Upon receiving $\mathcal{S}^2_v(B_k)$, broadcast a vote in the form of $\langle \mathsf{rsp\text{-}vote}, \mathcal{S}^2_v(B_k), v \rangle$.
4. **Qualify:** Upon receiving $\mathcal{S}^3_v(B_k)$, broadcast a qualify message in the form of $\langle \mathsf{qualify}, \mathcal{S}^3_v(B_k), v \rangle$.
5. **Commit:** Upon receiving $\mathcal{Q}(\mathcal{S}^3_v(B_k))$, commit B_k.
6. **Status & Lock:** When clock_v reaches 9Δ , send the highest certified block $\mathcal{C}_v(B)$ or $\mathcal{S}_v(B)$ to the next leader in the form of $\langle \mathsf{status}, \mathcal{C}_v(B)/\mathcal{S}_v(B), v \rangle$. Set a highest $\mathcal{S}^2_v(B)$ or $\mathsf{sync\text{-}lock}_{v,r}$, whichever is higher, to $\mathsf{rsp\text{-}lock}_{v,r}$.

Π_{blame}**:**

1. **Blame:** Upon receiving two equivocating blocks sent from L, stop voting in Π_{sync} and Π_{rsp}.

Figure 8: Hybrid-BFT– State Machine Replication under Mobile Sluggish Synchrony

The latter two cases can be proven in the same way as in Π_{SMR} by the quorum intersection. Thus, we only prove (i) by contradiction.

Let t be the time when B is certified, then $\exists R \subseteq \mathcal{H}(|R| \geq |\mathcal{C}| - b), \forall r \in R, \exists t_0, t_1 (t_0 \leq t_1 \leq t)$, (a) r sync-vote for B at t_1, and (b) B is qualified before $t_0 - \Delta$. Since $R \cap \mathcal{P}_{t_0} \neq \emptyset$ is held, (c) $\forall B_{-1}$, if B_{-1} equivocates B, then B_{-1} is not qualified before $t_0 - \Delta$. Since otherwise, one of the prompt replicas $p \in R \cap \mathcal{P}_{t_0}$ would not sync-vote for B after t_0. In the same way, let t' be the time when B' is certified, then $\exists R' \subseteq \mathcal{H}(|R'| \geq |\mathcal{C}| - b), \forall r' \in R', \exists t'_0, t'_1 (t'_0 \leq t'_1 \leq t')$, (a') r' sync-vote for B'

at t'_1, and (b') B' is qualified before $t'_0 - \Delta$. Here, without loss of generality, let $t_0 \geq t'_0$, then (c) and (b') contradict each other. □

Lemma 10. *If an honest replica directly and responsively commits a block B_k in view v, $\forall v' \geq v$, (i) $\exists R \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{C}|), \forall r \in R$, $\mathsf{sync\text{-}lock}_{v',r} \rightarrow B_k$, (ii) $\exists R \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{S}| - b), \forall r \in R$, $\mathsf{rsp\text{-}lock}_{v',r} \rightarrow B_k$, and (iii) $\forall B$ $(view(B) = v')$, if B is certified and $B \geq_{rank} B_k$, then $B \rightarrow B_k$.*

Proof. We prove it by induction on the view number. We first prove it for the base case ($v' = v$). (iii) is clear from Lemma 9. We prove the (ii). Suppose the honest replica commits B_k at time t, then it observes $\mathcal{Q}(\mathcal{S}_v^3(B_k))$ at $t \leq 9\Delta$. Then, $\exists R \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{S}| - b), \forall r \in R$, r observes $\mathcal{S}_v^2(B_k)$ before t, thus $\mathsf{rsp\text{-}lock}_{v,r} \geq_{rank} B_k$. Combining it together with Lemma 9, $\mathsf{rsp\text{-}lock}_{v,r} \rightarrow B_k$, which proves (ii). Here, the existence of $\mathcal{Q}(\mathcal{S}_v^3(B_k))$ means that at least one prompt replica observes $\mathcal{S}_v^3(B_k)$. Therefore, $\forall p \in \mathcal{P}_{(v+1,\Delta)}$, p observes $\mathcal{S}_v^3(B_k)$ before $(v+1, \Delta)$. This ensures that $\mathsf{sync\text{-}lock}_{v,p}(v+1, \Delta) \geq_{rank} B_k$. Combining it together with Lemma 9, $\mathsf{sync\text{-}lock}_{v,p}(v+1, \Delta) \rightarrow B_k$, and thus $|\mathcal{P}| \geq |\mathcal{C}|$, which proves (i).

Then, we prove for the inductive step. We first prove (iii) by contradiction. Suppose for the sake of contradiction that $\exists B$, $view(B) = v' + 1$, B is certified, $B \geq_{rank} B_k$, and $B \not\rightarrow B_k$. Let B_{min} be a block with lowest rank in a set of blocks which satisfies $B \rightarrow B_{min}$, and $view(B_{min}) = v' + 1$, and B_{min} is certified. Here, we can consider the two cases when B_{min} is normally and strongly certified.

If B_{min} is normally certified, then $\exists R' \subseteq \mathcal{H}$ $(|R'| \geq |\mathcal{C}| - b), \forall r' \in R'$, r' sync-vote for B_{min}. Then $\exists B_{min}^{-1}, \forall r' \in R'$, $\mathsf{sync\text{-}lock}_{r',v'} \leq_{rank} B_{min}^{-1}$, and B_{min}^{-1} is certified, and $B_{min} \rightarrow B_{min}^{-1}$, and $view(B_{min}^{-1}) \leq v'$. If $B_{min}^{-1} \geq_{rank} B_k$, then by the inductive hypothesis of (iii), $B_{min}^{-1} \rightarrow B_k$, which contradicts with $B \rightarrow B_k$. If $B_{min}^{-1} <_{rank} B_k$, then by the inductive hypothesis of (i), $\exists R \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{C}|), \forall r'' \in R$, $\mathsf{sync\text{-}lock}_{v',r''} >_{rank} B_{min}^{-1}$. Since $|R| + |R'| - |\mathcal{H}| \geq (n - f) + (n - f - b) - (n - b) > 0$, thus $R \cap R' \neq \emptyset$. Therefore $\exists r_0 \in R'$, $\mathsf{sync\text{-}lock}_{v',r_0} >_{rank} B_{min}^{-1}$, which is a contradiction.

If B_{min} is strongly certified, then $\exists R' \subseteq \mathcal{H}$ $(|R'| \geq |\mathcal{S}| - b), \forall r' \in R'$, r' rsp-vote for B_{min}. Then $\exists B_{min}^{-1}, \forall r' \in R'$, $\mathsf{rsp\text{-}lock}_{r',v'} \leq_{rank} B_{min}^{-1}$, and B_{min}^{-1} is certified, and $B_{min} \rightarrow B_{min}^{-1}$, and $view(B_{min}^{-1}) \leq v'$. If $B_{min}^{-1} \geq_{rank} B_k$, then by the inductive hypothesis of (iii), $B_{min}^{-1} \rightarrow B_k$, which contradicts with $B \rightarrow B_k$. If $B_{min}^{-1} <_{rank} B_k$, then by the inductive hypothesis of (ii), $\exists R \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{S}| - b), \forall r'' \in R$, $\mathsf{rsp\text{-}lock}_{v',r''} >_{rank} B_{min}^{-1}$. Since $|R| + |R'| - |\mathcal{H}| \geq 2 \cdot (|\mathcal{S}| - b) - (n - b) > 0$, thus $R \cap R' \neq \emptyset$. Therefore $\exists r_0 \in R'$, $\mathsf{rsp\text{-}lock}_{v',r_0} >_{rank} B_{min}^{-1}$, which is a contradiction.

We prove the (i) by contradiction. Suppose for the sake of contradiction, for R which is guaranteed to exist for view v' by the inductive hypothesis of (i), that $\exists r \in R$, $\mathsf{sync\text{-}lock}_{v'+1,r} \not\rightarrow B_k$. Let B_0 be this $\mathsf{sync\text{-}lock}_{v'+1,r}$. By (iii) which was already proven, $B_0 <_{rank} B_k$. Combining it together with the inductive hypothesis of (i), $\mathsf{sync\text{-}lock}_{v',r} >_{rank} B_0$. Since lock cannot be updated with any lower ranked blocks, it contradicts that B_0 is $\mathsf{sync\text{-}lock}_{v'+1,r}$. (ii) is also proven in the same way. □

Lemma 11. *If an honest replica directly and synchronously commits a block B_k in view v, $\forall v' \geq v$, (i) $\exists R \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{C}|), \forall r \in R$, $\mathsf{sync\text{-}lock}_{v',r} \rightarrow B_k$ and $\mathsf{rsp\text{-}lock}_{v',r} \rightarrow B_k$, and (ii) $\forall B$ $(view(B) = v')$, if B is certified and $B \geq_{rank} B_k$, then $B \rightarrow B_k$.*

Proof. We prove it by induction on the view number. We first prove it for the base case ($v' = v$). (ii) is clear from Lemma 9. We prove (i). Suppose the honest replica commits B_k at time t, then it observes $\mathcal{Q}(\mathcal{C}_v(B_k))$ at $t \leq 7\Delta$. Then, $\forall p \in \mathcal{P}_{(v,8\Delta)}$, p observes $\mathcal{C}_v(B_k)$ before $(v, 8\Delta)$. Since $\mathsf{sync\text{-}lock}$ is set at least after 8Δ and $\mathsf{rsp\text{-}lock}$ is set at least after 9Δ in each view, $\forall p \in \mathcal{P}_{(v,8\Delta)}$,

$\mathsf{sync\text{-}lock}_{v,p} \geq_{rank} B_k$ and $\mathsf{rsp\text{-}lock}_{v,p} \geq_{rank} B_k$. Combining it together with Lemma 9, $\forall p \in \mathcal{P}_{(v,8\Delta)}$, $\mathsf{sync\text{-}lock}_{v,p} \rightarrow B_k$ and $\mathsf{rsp\text{-}lock}_{v,p} \rightarrow B_k$. Since $|\mathcal{P}| \geq |\mathcal{C}|$, it proves (i).

Then, we prove for the inductive step. We first prove (ii) by contradiction. Suppose for the sake of contradiction that $\exists B$, $view(B) = v' + 1$, B is certified, $B \geq_{rank} B_k$, and $B \not\rightarrow B_k$. Let B_{min} be a block with lowest rank in a set of blocks which satisfies $B \rightarrow B_{min}$, and $view(B_{min}) = v' + 1$, and B_{min} is certified. Here, we can consider the two cases when B_{min} is normally or strongly certified.

If B_{min} is normally certified, then $\exists R' \subseteq \mathcal{H}$ $(|R'| \geq |\mathcal{C}| - b), \forall r' \in R'$, r' sync-vote for B_{min}. Then $\exists B_{min}^{-1}, \forall r' \in R'$, $\mathsf{sync\text{-}lock}_{r',v'} \leq_{rank} B_{min}^{-1}$, and B_{min}^{-1} is certified, and $B_{min} \rightarrow B_{min}^{-1}$, and $view(B_{min}^{-1}) \leq v'$. If $B_{min}^{-1} \geq_{rank} B_k$, then by the inductive hypothesis of (ii), $B_{min}^{-1} \rightarrow B_k$, this contradicts with $B \rightarrow B_k$. If $B_{min}^{-1} <_{rank} B_k$, then by the inductive hypothesis of (iii), $\exists R \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{C}|), \forall r'' \in R$, $\mathsf{sync\text{-}lock}_{v',r''} >_{rank} B_{min}^{-1}$. Since $|R| + |R'| - |\mathcal{H}| \geq (n-f) + (n-f-b) - (n-b) > 0$, thus $R \cap R' \neq \emptyset$. Therefore $\exists r_0 \in R'$, $\mathsf{sync\text{-}lock}_{v',r_0} >_{rank} B_{min}^{-1}$, which is a contradiction.

If B_{min} is strongly certified, then $\exists R'$ $(|R'| \geq |\mathcal{S}| - b), \forall r' \in R'$, r' rsp-vote for B_{min}. Then $\exists B_{min}^{-1}, \forall r' \in R'$, $\mathsf{rsp\text{-}lock}_{r',v'} \leq_{rank} B_{min}^{-1}$, and B_{min}^{-1} is certified, and $B_{min} \rightarrow B_{min}^{-1}$, and $view(B_{min}^{-1}) \leq v'$. If $B_{min}^{-1} \geq_{rank} B_k$, then by the inductive hypothesis of (ii), $B_{min}^{-1} \rightarrow B_k$, this contradicts with $B \rightarrow B_k$. If $B_{min}^{-1} <_{rank} B_k$, then by the inductive hypothesis (i), $\exists R \subseteq \mathcal{H}$ $(|R| \geq |\mathcal{C}|), \forall r'' \in R$, $\mathsf{rsp\text{-}lock}_{v',r''} >_{rank} B_{min}^{-1}$. Since $|R| + |R'| - |\mathcal{H}| \geq (n-f) + (|\mathcal{S}| - b) - (n-b) > 0$, thus $R \cap R' \neq \emptyset$. Therefore $\exists r_0 \in R'$, $\mathsf{rsp\text{-}lock}_{v',r_0} >_{rank} B_{min}^{-1}$, which is a contradiction.

We prove (i) by contradiction. Suppose for the sake of contradiction, for R which is guaranteed to exist for view v' by the inductive hypothesis of (i), that $\exists r \in R$, $\mathsf{sync\text{-}lock}_{v'+1,r} \not\rightarrow B_k$ or $\mathsf{rsp\text{-}lock}_{v'+1,r} \not\rightarrow B_k$. Let B_0 be this $\mathsf{sync\text{-}lock}_{v'+1,r}$. By (ii) which is already proved, $B_0 <_{rank} B_k$. Combining it together with the inductive hypothesis of (i), $\mathsf{sync\text{-}lock}_{v',r} >_{rank} B_0$. Since lock cannot be updated with any lower ranked blocks, it contradicts that B_0 is $\mathsf{sync\text{-}lock}_{v'+1,r}$. For $\mathsf{rsp\text{-}lock}_{v'+1,r}$, we can lead to a contradiction in the same way. □

Lemma 12 (Unique Extensibility)**.** *If an honest replica directly commits a block B_k in view v, then (i) there does not exist a certificate for block $B'_k \neq B_k$ in the same view, and (ii) all certified blocks with higher rank extend B_k.*

Proof. These are proven in (iii) of Lemmas 10 and 11 where B_k is responsively and synchronously committed, respectively. □

Theorem 9 (Safety)**.** *If an honest replica commits a block B_k, then all honest replicas do not commit a different block B'_k for the height k.*

Proof. By using Lemma 12, we can prove this in the same way as in the proof of Theorem 7. □

Liveness is proven in the case where all honest replicas are prompt. It can be proven in a straightforward manner as in Π_{SMR} (Theorem 8), and thus we omit the details of the proof.

5.4.2 Resilience-Favoring Protocol – $\Pi^*_{SMR}(2, \lceil \frac{n-f}{2} - 1 \rceil, f)$

We assume $\frac{n}{3} \leq f < \frac{n}{2}$. We can prove its safety and liveness in the same way as for $\Pi^*_{SMR}(1, n - 2f - 1, f)$, and thus we do not show all details of the proof. However, the proof of Certified without Equivocation Lemma is slightly different, and thus we show the proof in detail.

Lemma 13 (Certified without Equivocation)**.** *If a block B in view v is certified, then any block B' equivocating B is not certified.*

Proof. Suppose for the sake of contradiction that two blocks B and B' equivocating each other are both certified in view v. We can consider three cases: (i) B and B' are both normally certified, (ii) B and B' are both strongly certified, (iii) B is normally certified and B' is strongly certified. (i) can be proven in the same way as in the proof of Lemma 13 and (ii) can be proven using the quorum intersection. We lead to a contradiction for (iii).

Let t be the time when B is certified, then $\exists R \subseteq \mathcal{H}(|R| \geq |\mathcal{C}| - b), \forall r \in R, \exists t_0, t_1(t_0 \leq t_1 \leq t)$, (a) r sync-vote for B at t_1, and (b) B is qualified before $t_0 - \Delta$. Thus, $\forall p \in \mathcal{P}_{t_0-\Delta}, \forall B_{-1}$, if B_{-1} equivocates B, then p does not vote for B_{-1} after $t_0 - \Delta$ since p observes B after $t_0 - \Delta$. Since $R \cap \mathcal{P}_{t_0} \neq \emptyset$ is held, $\forall B_{-1}$, if B_{-1} equivocate B, then B_{-1} is not qualified before $t_0 - \Delta$. Therefore, (c) $\forall p \in \mathcal{P}_{t_0-\Delta}, \forall B_{-1}$, if B_{-1} equivocates B, then p does not vote for B_{-1}. Let t' be the time when B' is certified, then $\exists R' \subseteq \mathcal{H}(|R'| \geq |\mathcal{S}| - b), \forall r' \in R', \exists t'_0 \leq t'$, r' sync-vote for B' at t'_0. Since $R' \cap \mathcal{P}_{t_0-\Delta} \neq \emptyset$ is held, it contradicts with (c). □

6 Additional Related Works

6.1 Blockchain and Permissioned Consensus

In this paper, we discussed consensus protocols for private networks where membership in the network is known beforehand and fixed. We call it *permissioned consensus*. The consensus protocols studied before the invention of well-known Nakamoto Consensus can be classified as permissioned consensus [12, 29]. The Nakamoto Consensus was created to manage a blockchain in a public network where membership changes and even unknown during protocol execution. This protocol created a jointly served publicly accessible decentralized applications [45, 16, 15]. The Nakamoto Consensus uses a *proof-of-work (PoW)* mechanism which is a difficult computational puzzle to sporadically create a "member", i.e., a virtual node with a unit of power to contribute. However, the PoW inherently introduces a couple of problems, including (i) a wasteful and significant power consumption, and (ii) a trade-off between security and scalability.

To solve the first problem, *proof-of-stake (PoS)*, is considered to be one of the most practical solution. Many blockchain-based networks have implemented a PoS-based consensus [7, 26, 17]. In such networks, the membership is defined by a stake distribution that changes over time, making permissioned consensus practical in public networks by using it together with a dynamic membership reconfiguration mechanism. Indeed, many PoS-based protocols [10, 9, 23] have a quorum-based commitment mechanism similar to the permissioned consensus.

The second problem has been the focus of many studies and research in the past decade. To solve this problem, an approach is to extend a chain into a more general tree or directed acyclic graph [43, 41, 31, 42, 6], to scale its throughput. Another approach is using the Nakamoto Consensus to determine a committee in which a permissioned consensus is executed to process transactions fast [37, 18, 27]. However, these approaches are not considered to be practical because they cannot scale in/out. For this reason, *sharding* is much favored lately [28, 48, 32]. In sharding protocols, the whole network is randomly separated into committees called *shards* to process disjointed sets of transactions. Here, permissioned consensus is required for intra-shard consensus.

As summary, PoS and sharding are currently considered to be the most practical alternatives to the Nakamoto Consensus. In fact, a combination of both is being considered by Ethereum [11, 35, 50, 49], which is one of the biggest cryptocurrency projects. Futhermore, blockchain for consortium deployment where membership is predetermined is also attracting significant attention lately [5, 14]. In each case, permissioned consensus is regarded as an important component and

research topic.

6.2 Blockchain under Network Synchrony

Other prior works have proposed a practical blockchain under network synchrony. Dfinity [25, 3] achieved latency of small constant factor of Δ, but it is not optimal and is still connected to synchronous delay. PiLi [13] also achieved optimistic responsiveness and applied mobile sluggish synchrony, but it is not optimal and need long fallback period when it fails to progress responsively on the way. Compared to these works and other works introduced before, Hybrid-BFT fully removes dependency on synchronous delay under normal situation but still progresses with optimal synchronous latency even under the worst situation or when it fails to progress responsively on the way. In this sense, Hybrid-BFT can be considered to be most practical.

7 Conclusion

In this paper, we were able to show that consensus under network synchrony can achieve both optimistic responsiveness and optimal synchronous latency. Through discussions on reliable broadcast, we showed that there is an inherent trade-off between the optimistic resilience and optimal synchronous latency by presenting some lower and upper bounds. Extending the theoretical discussion, we presented a state machine replication protocol called Hybrid-BFT. Hybrid-BFT can be flexibly set its parameters to get required latency and resilience within the trade-off. When the optimistic resilience is compromised, Hybrid-BFT achieves optimal $\Delta + O(\delta)$ synchronous latency, while even under optimal optimistic resilience, it achieves near-optimal $2\Delta + O(\delta)$ synchronous latency. Furthermore, Hybrid-BFT achieves responsive leader change under normal situation where the number of faulty replicas is small. This makes Hybrid-BFTś latency to be completely free from synchronous delay, which is a greatly desired feature in a practical implementation. Finally, we extended Hybrid-BFT to be secure under weaker synchrony model called mobile sluggish synchrony, which allows network partitions and is more realistic than standard synchrony. Therefore, applying this model is also a desired direction today.

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)*, pages 320–334. Springer, 2019.

[2] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ren Ling, and Yin Maofan. Sync hotstuff: Simple and practical synchronous state machine replication. *IACR Cryptology ePrint Archive, Report 2019/270*, 2019. `https://eprint.iacr.org/2019/270`.

[3] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, and Ling Ren. Dfinity consensus, explored. *IACR Cryptology ePrint Archive, Report 2018/1153*, 2018. `https://eprint.iacr.org/2018/1153`.

[4] 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.

[5] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Thirteenth EuroSys Conference*, page 30. ACM, 2018.

[6] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 585–602. ACM, 2019.

[7] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *Financial Cryptography and Data Security (FC)*, pages 142–157. Springer, 2016.

[8] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[9] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, 2016.

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

[11] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[12] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *3rd Symposium on Operating Systems Design and Implementation (OSDI)*, pages 173–186. USENIX, 1999.

[13] T-H Hubert Chan, Rafael Pass, and Elaine Shi. Pili: An extremely simple synchronous blockchain. *IACR Cryptology ePrint Archive, Report 2018/980*, 2018. `https://eprint.iacr.org/2018/980`.

[14] J.P.Morgan Chase. Quorum whitepaper. 2018. `https://github.com/jpmorganchase/quorum/blob/master/docs/QuorumWhitepaperv0.2.pdf`.

[15] Jason Paul Cruz and Yuichi Kaji. The bitcoin network as platform for trans-organizational attribute authentication. In *Third International Conference on Building and Exploring Web Based Environments*, pages 29–36. IARIA, 2015.

[16] Jason Paul Cruz, Yuichi Kaji, and Naoto Yanai. Rbac-sc: Role-based access control using smart contract. *IEEE Access*, 6:12240–12251, 2018.

[17] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography and Data Security (FC)*, pages 23–41. Springer, 2019.

[18] Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin meets strong consistency. In *17th International Conference on Distributed Computing and Networking (ICDCN)*, page 13. ACM, 2016.

[19] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *The IEEE International Conference on Peer-to-Peer Computing*, pages 1–10. IEEE, 2013.

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

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

[22] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 281–310. Springer, 2015.

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

[24] Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In *Annual International Cryptology Conference (CRYPTO)*, pages 499–529. Springer, 2019.

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

[26] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference (CRYPTO)*, pages 357–388. Springer, 2017.

[27] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium*, pages 279–296. USENIX, 2016.

[28] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *IEEE Symposium on Security and Privacy (S&P)*, pages 583–598. IEEE, 2018.

[29] Leslie Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, 2006.

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

[31] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *Financial Cryptography and Data Security (FC)*, pages 528–547. Springer, 2015.

[32] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 17–30. ACM, 2016.

[33] Atsuki Momose and Jason Paul Cruz. Force-locking attack on sync hotstuff. *IACR Cryptology ePrint Archive, Report 2019/1484*, 2019. `https://eprint.iacr.org/2019/1484`.

[34] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

[35] Ryuya Nakamura, Takayuki Jimba, and Dominik Harz. Refinement and verification of cbc casper. In *2019 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 26–38. IEEE, 2019.

[36] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 643–673. Springer, 2017.

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

[38] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *Annual International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 380–409. Springer, 2017.

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

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

[41] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive, Report 2016/1159*, 2016. `https://eprint.iacr.org/2016/1159`.

[42] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security (FC)*, pages 507–527. Springer, 2015.

[43] Yonatan Sompolinsky and Aviv Zohar. Phantom: A scalable blockdag protocol. *IACR Cryptology ePrint Archive, Report 2018/104*, 2018. `https://eprint.iacr.org/2018/104`.

[44] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *IEEE Symposium on Security and Privacy (S&P)*, pages 444–460. IEEE, 2017.

[45] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[46] 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.

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

[48] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 931–948. ACM, 2018.

[49] Vlad Zamfir. Casper the friendly ghost: A correct by construction blockchain consensus protocol. 2017. `https://github.com/ethereum/research/blob/master/papers/CasperTFG/CasperTFG.pdf`.

[50] Vlad Zamfir, Nate Rush, Aditya Asgaonkar, and Georgios Piliouras. Introducing the "minimal cbc casper" family of consensus protocols. 2018. https://github.com/cbc-casper/cbc-casper-paper/blob/master/cbc-casper-paper-draft.pdf.