Force-Locking Attack on Sync Hotstuff

Atsuki Momose
Nagoya University

Jason Paul Cruz
Osaka University

January 24, 2020

Abstract

Blockchain, which realizes state machine replication (SMR), is a fundamental building block of decentralized systems, such as cryptocurrencies and smart contracts. These systems require a consensus protocol in their global-scale, public, and trustless networks. In such an environment, consensus protocols require high resiliency, which is the ability to tolerate a fraction of faulty replicas, and thus synchronous protocols have been gaining significant research attention recently. Abraham et al. proposed a simple and practical synchronous SMR protocol called Sync Hotstuff (to be presented in IEEE S&P 2020). Sync Hotstuff achieves 2Δ latency, which is near optimal in a synchronous protocol, and its throughput without lock-step execution is comparable to that of partially synchronous protocols. Sync Hotstuff was presented under a standard synchronous model as well as under a weaker, but more realistic, model called mobile sluggish model. Sync Hotstuff also adopts an optimistic responsive mode, in which the latency is independent of Δ. However, Sync Hotstuff has a critical security vulnerability with which an adversary can conduct double spending or denial-of-service attack. In this paper, we present an attack we call *force-locking* attack on Sync Hotstuff. This attack violates the *safety*, i.e., consistency of agreements, of the protocol under the standard synchronous model and the *liveness*, i.e., progress of agreements, of all versions of the protocol, including the mobile sluggish model and responsive mode. The force-locking attack is not only a specific attack on Sync Hotstuff but also on some general blockchain protocols. After describing the attack, we will present some refinements to prevent this attack. Our refinements remove the security vulnerability on Sync Hotstuff without any performance compromises. We will also provide formal proofs of the security for each model.

1 Introduction

1.1 Synchronous State Machine Replication

Blockchain is a scheme of state machine replication (SMR) [17, 18]. For the past three decades, many SMR protocols have been developed and deployed on small- and medium-sized networks, such as server or database replication in a data center. In such applications, *performance*, i.e., high throughput and short latency, is prioritized over *resiliency*, i.e., ability to tolerate a fraction of faulty replicas. In such use cases, synchronous protocols are not considered as a practical solution because of the following problems. First, classical synchronous protocols are slow because they need large number of rounds, which depend on the known upper bound of the network delay Δ, to achieve *finality*, i.e., agreement on a history of a state machine. Moreover, synchronous protocols need lock-step execution where all replicas start and end at the same time, and thus the latency is dependent on the number of synchronous

rounds. Second, the standard synchronous model, which ensures network synchrony for all honest replicas that faithfully follow protocol instructions, is unrealistic in practice because small network partitions may occur frequently. Finally, in practice, the synchronous parameter Δ is overestimated to ensure sufficient security but consequently making the protocol much slower. For these reasons, partially synchronous protocols [4, 8, 12, 11] are much favored because they can tolerate network partitions to ensure the *safety*, i.e., consistency of agreements, of a network. Moreover, partially synchronous protocols do not rely on network synchrony to finalize, and therefore they can achieve *responsiveness*, i.e., the latency is dependent on actual bound δ and not on known bound Δ. Although some synchronous protocols can tolerate the presence of minority faulty replicas (hereafter we assume byzantine fault), partially synchronous protocols can only tolerate up to 1/3 of faulty replicas, which is not enough to be favored.

The creation of blockchain technology in 2008 [13, 16, 9] and its applications to decentralized systems, such as cryptocurrencies and smart contracts [20, 7, 6], has made the SMR protocol to be considered not only for replication in local networks but also in global-scale, public, and trustless networks. Such public networks require higher resiliency than local private networks because of the presence of more malicious activities, as evident in the significant number of cyber attacks that are conducted across the Internet on a daily basis. Therefore, many researchers have tried to address the problems described above and develop practical synchronous protocols. To solve long latency problem, Thunderella [19] introduced the new notion of *optimistic responsiveness*, under which finality is independent of Δ in the optimistic situation that can occur under protocol assumptions. The strong standard synchronous model is not practical in real-world applications, and thus many researchers have tried to rethink and improvise the synchronous model. Guo et al. [10] introduced a new model called *weak synchrony*. In this model, not malicious but offline replicas, i.e., they cannot satisfy the synchrony assumption because of some network failure, are treated as honest replicas and the set of offline replicas can change over time. Such consideration is important in a long-term deployment of a network, such as a public blockchain where a replica can join, leave, or stay dormant at any given time.

Abraham et al. presented a synchronous protocol called Sync Hotstuff. It has a simple design and avoids the problems described above by using several elegant but natural techniques. First, it uses the synchronous parameter Δ only for checking the absence of *equivocation*, i.e., conflicting block proposals, and thus minimizing reliance on the synchrony. With this technique, Sync Hotstuff achieves a latency of 2Δ, which is near optimal in the synchronous model. They argued that 2Δ may be the optimal latency because a round trip may be necessary to check the response of a network. Moreover, Sync Hotstuff does not use lock-step execution, and thus it achieves a throughput that is comparable to that of partially synchronous protocols. To consider a more realistic synchronous model, they also applied the weakly synchronous model but they called it *mobile sluggish model*. This model ensures the safety of a *sluggish* replica, i.e., an offline or slow replica, by collecting enough messages to check the finality of a *prompt* replica, i.e., a replica that respects the synchrony assumption. Furthermore, it achieves optimistic responsiveness by checking the agreement on the optimistic situation with synchronous mode to switch the responsive mode.

1.2 Block-Chaining and Locking

The main features of a blockchain scheme include blocking, i.e., batching of commands, and chaining, i.e., construction of a chain by hash references. These features translate SMR into a simple problem of selecting one path or chain from a growing tree and enable the resolution of a conflict with descendant/ascendant relation. The finality of a block finalizes all of the ascendants, and thus even if some replicas did not finalize a block that others did at a given time, they simply need to finalize a descendant of that block in the future. This simple and versatile feature of a blockchain makes blockchain widely used as a framework for SMR protocols today.

In most voting-based blockchain protocols, a middle state of a block is used to reach a final state. The middle state ensures that a block is voted for by a *quorum* of replicas, e.g., a majority or more than 2/3 of replicas– this is called *certified* in Sync Hotstuff, *justified* in Casper FFG [3], *notarized* in PiLi [5], and *prepared* in Hotstuff [21]; hereafter we call it certified following Sync Hotstuff. In addition, these protocols compare the freshness of a block based on, for example, view number, block height, or epoch number. A certified block is finalized when it is assured that any fresher block conflicting with it will not be certified, i.e., it cannot collect a quorum of votes, by the rule of where to vote defined in the protocol and therefore will not be finalized, providing safety to the network. The rule of where to vote is sometimes called the *fork choice rule*, which provides not only safety but also liveness. In some protocols, the fork choice is always in a subtree which has a root block as long as a root change event does not happen. We say that a replica is *locked* on the root block, and call the root change event *unlocking*. In some protocols, a replica is locked on the freshest certified block. This is natural because the finality rule is constructed such that a finalized block includes all certified blocks that are fresher in its ascendant. The locking scheme is an important building block of the fork choice rule, and thus it should be constructed carefully because it is directly related to the security of the protocol.

1.3 Our Contribution

1.3.1 Force-Locking Attack

In this paper, we present the *force-locking attack* on Sync Hotstuff. This attack violates the safety of the protocol under the standard synchronous model (hereafter denoted by Π_{std}) and the liveness of all versions of the protocol, including the mobile sluggish model (hereafter denoted by Π_{slg}) and with responsive mode (hereafter denoted by Π_{rsp}). This attack is not specific on Sync Hotstuff but is a general form of attack scheme that can be applied to any locking-based protocol, wherein a locking point is determined by a replica's past messages. Thus, the messages sent by faulty replicas controlled by an adversary also affect the choice of honest replicas. When a block is lockable, i.e., past messages sent by honest replicas make it ready for messages adoptively created by faulty replicas to change the lock, the adversary can change the lock whenever it wants. Moreover, replicas may sometimes receive different messages due to the network delay. To make matters worse, the network may be controlled fully by the adversary, except for the delay bound in the synchronous model. With a biased network delay and the messages sent by faulty replicas, an adversary can control the observed messages and change and split the locking point of some honest replicas. Consequently, a protocol that relies on the locking rule and without any protection from the force-locking attack can have its safety and liveness violated. In practice, the bouncing attack on Casper

FFG, which is the core consensus protocol for Ethereum 2.0, is a variant of the force-locking attack scheme [2, 14, 15]. This attack simply changes the lock before some block is finalized by releasing the messages of faulty replicas to lock a lockable block that conflicts the block that was expected to be finalized. This attack can be repeated indefinitely to violate the liveness of a network. On the other hand, Sync Hotstuff relies on the locking rule for both its safety and liveness and has no protection against force-locking. In Sync Hotstuff, a block proposer called a *leader* is selected, and the reign of the leader is called a *view*. The process in the reign is called steady state protocol, and the process to change the leader is called view-change protocol. A locking point is a freshest certified block. In Π_{std}, a block is finalized when the maximum round-trip time of 2Δ has passed and no equivocation is observed within that time frame. The authors of Sync Hotstuff ensured that all honest replicas vote for the same block when there is no equivocation. However, this case will not happen when honest replicas are locked on different blocks, and therefore cannot vote for the same block. In this case, a block is finalized in some replicas but not certified, and safety is violated by the extension of a conflicting certified block. In all versions of the protocol (i.e., Π_{std}, Π_{slg}, and Π_{rsp}), a replica is always locked on a freshest certified block. However, in the presence of force-locking, honest replicas and also an honest leader can be locked on different blocks. In this case, an honest leader's proposal cannot be voted for by some honest replicas and cannot be finalized, therefore violating the liveness.

1.3.2 Refinements and Proofs

We also present some refinements to the protocols of Sync Hotstuff to prevent our proposed force-locking attack. For the liveness, to ensure that an honest proposal will be voted for by all honest replicas, we modified the view-change protocol such that if the leader is honest, then all the proposals in its view are consistent and any inconsistency can only come from another view. Intuitively, we added a rule to submit its lock in the view-change and add a waiting time for the next leader to collect all the locks from honest replicas. We also added a refinement to the locking rule as follows: for the first block after view-change, a replica is locked on its submission in the view-change. Since an honest leader can collect all the locks from honest replicas and propose according to them, honest replicas can vote for the proposal of the honest leader. For the safety, the refinement described above is insufficient when a faulty leader makes a proposal that ignores the submissions of honest replicas. The absence of equivocation cannot ensure the existence of a *certificate* (i.e., a quorum of votes). To address this problem, we added the rule to check for the existence of a certificate before the round-trip waiting time to ensure the absence of equivocation. Besides ensuring the absence of equivocation, this refinement additionally guarantees the existence of certificate in all honest replicas, and therefore ensuring safety. Note that because the synchronous waiting time in the steady state protocol is not changed and it does not have lock-step execution, these refinements do not compromise the protocol's performance. Finally, we will give formal proofs of security for each version of the protocol.

1.4 Paper Organization

In Section 2, we briefly review Sync Hotstuff, including some definitions and conventions. In Section 3, we describe the force-locking attack in detail with specific attack scenarios. In Section 4, we propose some refinements to Sync Hotstuff and formally prove the safety and

liveness of the refined protocols. In Section 5, we provide the conclusion and future direction.

2 Sync Hotstuff

In this section, we briefly review Sync Hotstuff, including some basic definitions and conventions which are also used in other blockchain-based SMR protocols.

2.1 State Machine Replication

A state machine replication protocol (SMR) is used for building a fault-tolerant service that processes client requests. The service consists of n replicas, up to f of which can be faulty. The threshold parameter f is also called fault threshold or resiliency. All honest replicas consistently commit client requests into a linearizable log. An SMR protocol is expected to provide at least the following two security properties: (i) safety: if an honest replica commits a value in its log position, any other honest replica commits the same value at the same position, and (ii) liveness: each client request is eventually committed by all honest replicas. Sync Hotstuff assumes $n = 2f + 1$, i.e., minority fault.

2.2 Network model

The network consists of pairwise, authenticated communication channels between replicas. We assume the use of digital signatures and a public-key infrastructure (PKI) and use $\langle x \rangle_p$ to denote a message x signed by replica p, but we omit the signer p when the context is clear.

For the synchrony assumption, the *standard synchronous model* is first introduced. It simply states that a message sent at time t by any replica arrives at another replica by time $t + \Delta$. Later in the paper, we assume that the network is controlled by an adversary that can reorder or delay messages but the adversary can only control the network under synchronous compliance, and thus the protocol does not deviate from the synchrony assumption. A weaker model called *mobile sluggish model* was first introduced by Guo et al. [10] as *weak synchrony*. In this model, a *sluggish* replica, i.e., a replica that does not respect the synchrony assumption, is still treated as honest. On the other hand, a replica that respects the synchrony assumption is called a *prompt* replica. Furthermore, the set of sluggish replicas can change arbitrarily at any time under the constraint that $f = d + b$, where d and b indicate the number of sluggish and faulty replicas, respectively. It should be noted that the standard synchronous model is a special case of the mobile sluggish model with $d = 0$.

2.3 Overview and Definitions

Sync Hotstuff consists of two sub protocols, namely, the steady state protocol and the view-change protocol. In the steady state, a stable leader proposes a new entry for the log and then other replicas vote for the proposal for confirmation. The reign of the leader is called a view and is identified by a monotonically increasing view number. The leader of each view is simply selected in a round robin manner. When a leader is lazy to progress or some byzantine activity is detected, other honest replicas blame the leader and transition into the view-change protocol where they switch to another view and leader.

Sync Hotstuff uses the blockchain scheme. In the steady state protocol, the leader proposes a block $B_k = (b_k, h_{k-1})$, where k is a height in the blockchain, b_k is a batch of client's

requests, and h_{k-1} is a hash of the predecessor in the blockchain that the block extends, i.e., $h_{k-1} = H(B_{k-1})$. The genesis block is defined as $\perp$. A block is *valid* if (i) its predecessor is valid or $\perp$, and (ii) its proposed value meets application-level validity condition and is consistent with its chain of ancestors. If a block B_k is an ancestor of another B_l ($l \geq k$), we say B_l extends B_k, and if a leader proposes two blocks that do not extend each other, we say the blocks equivocate each other.

In Sync Hotstuff, a proposal of a block must contain a set of signatures on the predecessor from a quorum of replicas (i.e., $f + 1$) from the same view and it is called a certificate. A certificate for a block B in view v is denoted by $\mathcal{C}_v(B)$. All certified blocks are ranked with first its view number and then its height in the blockchain. During protocol execution, each replica keeps track of all signatures for all blocks and keeps updating the highest certified block it knows. Each replica uses the highest certified block as a lock, which is the point that we use in our force-locking attack.

2.4 Protocol Specifications

In this subsection, we review the specification of each version of the Sync Hotstuff protocol.

2.4.1 The Protocol under the Standard Synchronous Model

Figure 1 shows the steady state protocol under the standard synchronous model (hereafter denoted by Π_{std}.steady). The steady state protocol runs in iteration. The leader proposes a block extending the highest certified block it knows, and the proposal must contain a certificate. If the leader has been in the steady state, it should extend the previous block it has proposed in the same view, and if it is the first proposal after the view-change, it should extend the highest certified block it knows. Each replica, upon receiving a block B_k, broadcasts a vote for the block if it is the first valid block for the height by the leader and it extends the highest certified block it knows. After voting for a block B_k, a replica starts a timer for the height commit-timer$_k$, and after waiting for 2Δ time and no equivocation has been observed within this time, it commits B_k and all its ancestors. Note that these are all processed without lock-step and commit is non-blocking, i.e., the next iteration is started without waiting for the timer and the blocks at each height are processed concurrently.

The main technique of Sync Hotstuff is the waiting for 2Δ before commit. The authors argue that this ensures the safety informally and also formally in the Theorem 3 in [1] because of the following reasons. Suppose a replica r commits a block B_k at time t, then it must have voted for B_k at time $t - 2\Delta$ and the vote reached all honest replicas before $t - \Delta$. Then, all honest replicas did not vote for any equivocating blocks before $t - \Delta$ because otherwise the vote would have reached r before t and r could not commit B_k. Since all honest replicas did not vote for any equivocating blocks before $t - \Delta$, they must have voted for B_k by $t - \Delta$, and thus B_k was certified. Moreover, since all honest replica voted for B_k by $t - \Delta$, they must have not voted for any equivocating blocks after $t - \Delta$. This means that conflicting blocks are never certified, ensuring safety. However, this argument is incorrect, and we will explain it in Section 3.2.

Figure 2 shows the view-change protocol (hereafter denoted by Π_{std}.change). Similar to other SMR protocols, this protocol is used for preventing a network from being stuck. First, to prevent a faulty leader from not progressing, honest replicas broadcast blame messages if less than p blocks are proposed by the leader within $(2p+1)\Delta$ time or equivocating proposals

Π_{std}.steady

Let v be the current view number, L be the leader of the current view, and r be a replica. Replica r executes the following protocol.

1. **Propose:** If $r = L$, broadcast $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}) \rangle_L$, where $B_k = (b_k, h_{k-1})$, b_k is a batch of new client requests, $v' \leq v$, and B_{k-1} is the highest certified block L knows.

2. **Vote:** Upon receiving $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}) \rangle_L$, if this is the first valid proposal for height k and B_{k-1} is a highest certified block r knows, then broadcast the proposal and a vote in the form of $\langle \mathsf{vote}, B_k, v \rangle$ and set $\mathsf{commit\text{-}timer}_k$ to 2Δ and start counting down.

3. **(Non-blocking) Commit:** When $\mathsf{commit\text{-}timer}_k$ reaches 0 and no equivocation in view v has been observed so far, commit B_k and all its ancestors.

Figure 1: The steady state protocol under the standard synchronous model

are observed. If a quorum of replicas blames the leader, the replicas quit the current view, wait for Δ, and then start the next view by sending the highest certified block they know to the next leader.

Abraham et al. argued that the view-change protocol ensures that *"a view-change will not happen if the current leader is honest"* in Theorem 4 in [1]. This property should hold to ensure the liveness of the network. However, this argument is incorrect, and we will explain the problem in Section 3.3.

2.4.2 The Protocol under the Mobile Sluggish Model

Figure 3 shows the steady state protocol under the mobile sluggish model (hereafter denoted by Π_{slg}.steady). The gray colored parts are the same as in the protocol under standard synchrony. The differences include the following rules: (i) the timer for a block is set after the following two blocks are proposed, and (ii) a block is committed after a quorum of replicas waited for 2Δ. These marginal rules are for sluggish replicas to safely commit blocks. The first rule ensures that a replica has $\mathcal{C}_v(\mathcal{C}_v(B_k))$, which means that a quorum of replicas already has this certificate. The assumption $f = d + b$ ensures that all sets of a quorum of replicas include at least a prompt replica, and thus ensures that at least a prompt replica has certificate. The second rule ensures in the same way that at least one prompt replica waited for 2Δ after at least one prompt replica received the certificate and that it did not observe any equivocation within the waiting time. Therefore, all prompt replicas (thus a quorum of replicas) have certificate and will never vote for equivocating blocks, and thus conflicting blocks will not be certified. The difference in the safety argument between this protocol and the protocol under standard synchrony is that the existence of a certificate of a block is checked before waiting for 2Δ. This point prevents attack on safety, and we use this technique to modify the protocol for standard synchrony (See Section 4.1 for details).

The view-change protocol (hereafter denoted by Π_{slg}.change) is the same as Π_{std}.change. Therefore, the liveness can be violated in the same way.

Π_{std}.change

Let L and L' be the leaders of views v and $v+1$, respectively. Replica r executes the following protocol.

1. **Blame:** If less than p blocks are received from L in $(2p+1)\Delta$ time in view v, broadcast $\langle \mathsf{blame}, v \rangle$. If a leader L proposes equivocating blocks, broadcast $\langle \mathsf{blame}, v \rangle$ and the two equivocating blocks.

2. **Quit old view:** Upon receiving $f+1$ $\langle \mathsf{blame}, v \rangle$ messages, broadcast them and quit view v (abort all commit-timer and stop voting in view v).

3. **Status:** Wait for Δ time and enter view $v+1$. Upon entering view $v+1$, send a highest certified block to L' and transition back to steady state.

Figure 2: The view-change protocol under the standard synchronous model

2.4.3 The Protocol with Optimistic Responsiveness

Figures 4 and 5 show the steady state protocol and view-change protocol with responsive mode, respectively (hereafter denoted by Π_{rsp}.steady and Π_{rsp}.change, respectively). Π_{rsp} achieves optimistic responsiveness, as first introduced in Thunderella [19]. Π_{rsp} uses the same technique introduced in Thunderella, i.e., $3n/4$ votes ensure the absence of equivocating blocks without synchronous waiting. The certificate of a block containing more than $3n/4$ signatures is called a strong certificate. In the voting phase of the steady state protocol, when a proposal contains a strong certificate for the predecessor, a replica switches to the responsive mode. The main techniques include (i) reaching agreement on the switch to the responsive mode, and (ii) taking over the blocks committed responsively to the next view in the view change. In the first technique, at least one block must be committed synchronously in the responsive mode to ensure that a quorum of prompt replicas has switched to the responsive mode and that a commit in the synchronous mode (i.e., without strong certificate) will never happen in the view. In the second technique, some modifications to the view-change protocol ensure that a quorum of replicas gets all certificates of committed blocks before entering the next view. A replica must have $\mathcal{C}_v(\mathcal{C}_v(blame))$ before entering the next view to ensure that at least one prompt replica quit the view. In addition, the waiting for 2Δ after receiving $\mathcal{C}_v(\mathcal{C}_v(blame))$ ensures that a quorum of prompt replicas receives all committed blocks. Suppose a block is committed at time t and a prompt replica enters the next view before $t+\Delta$, then at least one replica quit the view before $t-2\Delta$ and thus preventing the commit.

The modification on the view-change protocol ensures that a quorum of replicas receives all certificates of committed blocks before entering the next view. However, despite this modification, the liveness can still be violated in a similar manner as in the standard protocol (see Section 3.3 for details).

3 Attacks

In this section, we present our proposed force-locking attack on Sync Hotstuff. This attack violates the safety and liveness of Sync Hotstuff. We first introduce force-locking and then

Π_{slg}.steady

Let v be the current view number, replica L be the leader of the current view, and r be a replica. Replica r executes the following protocol.

1. **Propose:** If $r = L$, broadcast $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}) \rangle_L$, where $B_k = (b_k, h_{k-1})$, b_k is a batch of new client requests, $v' \leq v$, and B_{k-1} is the highest certified block L knows.

2. **Vote:** Upon receiving $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}) \rangle_L$, if this is the first valid proposal for height k and B_{k-1} is a highest certified block r knows, then broadcast the proposal and a vote in the form of $\langle \mathsf{vote}, B_k, v \rangle$ and set $\mathsf{pre\text{-}commit\text{-}timer}_{k-2}$ to 2Δ and start counting down.

3. **(Non-blocking) Pre-commit:** When $\mathsf{pre\text{-}commit\text{-}timer}_k$ reaches 0 and no equivocation in view v has been observed so far, pre-commit B_k and broadcast $\langle \mathsf{commit}, B_k, v \rangle$.

4. **(Non-blocking) Commit:** Upon receiving $\langle \mathsf{commit}, B_k, v \rangle$ from $f + 1$ distinct replicas with the same v, commit B_k and all its ancestors.

Figure 3: The steady state protocol under the mobile sluggish model

describe specific attack scenarios.

3.1 Force-Locking

In a blockchain protocol, each replica or block-proposer selects a path from the genesis block to a leaf block in the block tree to vote for or extend following a rule, which is often called the fork choice rule. In some blockchain protocols, a leaf block is selected from a subtree in the block tree that changes over time. We say that a replica is locked on a block when it is the root of the subtree. In Sync Hotstuff, a replica is always locked on the highest certified block it knows, and a newly certified block with a higher rank unlocks the replica to this block. In every protocol, the lock is determined by the messages received from all replicas, including faulty replicas. Therefore, a coalition of faulty replicas can intentionally change a lock or prevent locking by sending or withholding its messages if the protocol does not prevent such action. In addition, even in the synchronous model, biased message delay within the assumption can split the lock of each honest replica. Consequently, a protocol with naively constructed locking rule can lose its safety or liveness. In Sync Hotstuff, an adversary can split the lock or highest certified block of each honest replica for at most Δ time by sending the votes of faulty replicas on a certifiable block (i.e., the block was already voted for by honest replicas and is ready to be certified) to some honest replicas with delay 0 and to others with delay Δ. This scenario is not a problem in the case where a leader steadily extends its proposal because the highest certified block is its proposal and it would be blamed for equivocation if it does not extend this block. However, when a view-change happens and the view of honest replicas on the highest certified block in the previous view is split, the proposal of the next leader cannot collect votes from some honest replicas even though the

Π_{rsp}.steady

Let v be the current view number, replica L be the leader of the current view, and r be a replica. Replica r executes the following protocol.

1. **Propose:** If $r = L$, broadcast $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}) \rangle_L$, where $B_k = (b_k, h_{k-1})$, b_k is a batch of new client requests, $v' \leq v$, and B_{k-1} is the highest certified block L knows.

2. **Vote:** Upon receiving $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}) \rangle_L$, if this is the first valid proposal for height k and B_{k-1} is a highest certified block r knows, then broadcast the proposal and a vote in the form of $\langle \mathsf{vote}, B_k, v \rangle$. (i) If $\mathcal{C}_{v'}(B_{k-1})$ is a strong certificate, switch to the responsive mode and only vote for blocks with strong certificates for the rest of this view. (ii) Else, set $\mathsf{pre\text{-}commit\text{-}timer}_{k-2}$ to 2Δ and start counting down.

3. **(Non-blocking) Pre-commit:** (i) If at least one block has been committed in the responsive mode of this view, pre-commit B_{k-2} and broadcast $\langle \mathsf{commit}, B_{k-2}, v \rangle$.(ii) When $\mathsf{pre\text{-}commit\text{-}timer}_{k-2}$ reaches 0, and no equivocation in view v has been observed so far, pre-commit B_{k-2} and broadcast $\langle \mathsf{commit}, B_{k-2}, v \rangle$.

4. **(Non-blocking) Commit:** Upon receiving $\langle \mathsf{commit}, B_k, v \rangle$ from $f + 1$ distinct replicas with the same v, commit B_k and all its ancestors.

Figure 4: The steady state protocol with responsive mode

leader cannot be blamed, consequently violating the safety and liveness of the protocol.

3.2 Attack on Safety

We first describe the attack on the safety of Π_{std}. As described in the previous section, a replica commits a block after waiting for 2Δ time and if it did not observe any equivocation within that time. The authors of Sync Hotstuff argued that this ensures the existence of certificate on the block since all honest replicas will vote for the block if no equivocation occurred. However, this is not the case when view-change happens and the highest certified block in the previous view for honest replicas is split and some honest replicas cannot vote for the proposal that extends a certified block with lower rank. In this case, the block is committed but not certified and then the blockchain extends the other certified block, violating the safety of the protocol.

3.2.1 Attack Scenario

We assume that at time t all replicas start the view v and recognize B_0 as the highest certified block. In practice, when the protocol starts, i.e., $t = 0$, all replicas start the view $v = 1$ and recognize the genesis block $\perp$ as the highest certified block. We define some notations to describe the sets of honest and faulty replicas. L_1 and L_2 denote faulty leaders at views v and $v + 1$, respectively. R_f denotes a set of faulty replicas of size f and it includes L_1 and L_2. R_1 and R_2 each denote a set of honest replicas of size $(f + 1)/2$, and every replica in the

Π_{rsp}.change

Let L and L' be the leaders of views v and $v+1$, respectively. Replica r executes the following protocol.

1. **Blame:** If less than p blocks are received from L in $(2p+1)\Delta$ time in view v, broadcast $\langle \mathsf{blame}, v \rangle$. If a leader L proposes equivocating blocks, broadcast $\langle \mathsf{blame}, v \rangle$ and the two equivocating blocks.

2. **Quit old view:** Upon receiving $f+1$ $\langle \mathsf{blame}, v \rangle$ messages, broadcast them along with $\langle \mathsf{blame2}, v \rangle$, and then quit view v (abort all commit-timer and stop voting in view v).

3. **Status:** Upon receiving $f+1$ $\langle \mathsf{blame2}, v \rangle$ messages, wait for 2Δ time and enter view $v+1$. Upon entering view $v+1$, send a highest certified block to L' and transition back to steady state.

Figure 5: The view-change protocol with responsive mode

same set can communicate with delay 0 or communicate through the network with standard synchrony if it is in a different set. All faulty replicas as well as the network are controlled by the adversary. The total number of replicas is $n = 2f + 1$, where f is the number of faulty replicas controlled by the adversary, and this situation is within the fault assumption in Sync Hotstuff. We present a specific scenario chronologically where the safety is violated as follows:

- $t + 2.5\Delta$
  1. L_1 sends $\langle \mathsf{propose}, B_1, v, \mathcal{C}_{v-1}(B_0) \rangle$ to R_1 with delay 0.
  2. R_1 receives propose from L_1 and broadcasts $vote_1 = \langle \mathsf{vote}, B_1, v \rangle$ with delay Δ.
- $t + 3\Delta$
  1. R_2 broadcasts $blame_1 = \langle \mathsf{blame}, v \rangle$ with delay Δ since it did not receive $p = 1$ proposal within $3\Delta = (2p + 1)\Delta$.
  2. R_f sends $blame_2 = \langle \mathsf{blame}, v \rangle$ to R_2 with delay 0, as if it also blames the leader by timeout.
  3. R_2 receives more than $f + 1$ blame messages ($blame_3 = blame_1 \cup blame_2$), broadcasts all the blame messages with delay Δ, and starts waiting for Δ.
- $t + 3.5\Delta$
  1. R_2 receives $vote_1$ from R_1, but it cannot certify B_1 since its size is less than $f + 1$.
  2. R_f sends $vote_2 = \langle \mathsf{vote}, B_1, v \rangle$ to R_1 with delay 0.
  3. R_1 receives more than $f + 1$ votes ($vote_3 = vote_1 \cup vote2$) and then certifies B_1.
- $t + 4\Delta$
  1. R_1 receives more than $f+1$ blame messages $blame_3$, broadcasts all blame messages, and starts waiting for Δ.

2. R_2 has waited for Δ since it quit the old view, so it starts the next view $v+1$ and sends its highest certified block B_0 to the next leader L_2. Here, R_2 does not recognize B_1 as the highest certified block.
3. L_2 sends $\langle \mathsf{propose}, B'_1, v, \mathcal{C}_{v-1}(B_0)\rangle$ to R_2 with delay 0.
4. R_2 broadcasts $vote_4 = \langle \mathsf{vote}, B'_1, v+1\rangle$ with delay Δ, then it sets $\mathsf{commit\text{-}timer}_1$ to 2Δ and starts counting down.

- $t + 4.5\Delta$

1. R_2 now received more than $f+1$ votes $vote_3$, and then certifies B_1. However, this certification is too late.

- $t + 5\Delta$

1. R_1 has waited for Δ since it quit the old view, so it starts the next view $v+1$ and sends its highest certified block B_1 to the next leader L_2.
2. R_1 receives $vote_4$ but does not vote since B'_1 does not extend from its highest certified block B_1.

- $t + 6\Delta$

1. R_2 commits B'_1 since the $\mathsf{commit\text{-}time}_1$ reached 0 and no equivocation in view $v+1$ was observed.

At $t+6\Delta$, R_2 commits B'_1 but its highest certified block is B_1. Therefore, the blockchain will extend from B_1 conflicting with B'_1 after the next view, and safety will be lost.

3.3 Attack on Liveness

We now describe the attack on liveness. This attack can be applied to all three protocols, but we first describe the attack on Π_{std} and Π_{slg}, and later show a small modification for the attack on the protocol with responsive mode. The attack scheme is almost the same as the attack on safety, except that this attack prevents honest leaders from continuously proposing a valid block. Consider the following scenario. The view of honest replicas on the highest certified block is split in the previous view, but only the next honest leader sees the old lower certified block while other honest replicas see the newly certified block. The honest leader proposes extending the old block but it cannot receive a certificate for its proposal and cannot continue proposing a valid block and be blamed. Although this attack does not bring a permanent failure on finality, an honest leader should not be blamed to ensure the liveness of the network, and thus the liveness is somewhat violated. It should also be noted that this is a violation of validity in the byzantine broadcast formulation, which requires that an honest leader's proposal should be committed.

3.3.1 Attack Scenario

We assume that at time t all parties start the view v and recognize B_0 as the highest certified block. L_1 and L_2 denote the leaders at views v and $v+1$, respectively, but in this case $L1$ is faulty and $L2$ is honest. R_f denotes a set of faulty replicas of size f, including L_1. R_h denotes a set of honest replicas of size f excluding L_2. This situation is also within the fault

assumption in Sync Hotstuff. Then, we describe a specific scenario chronologically where the liveness is violated as follows:

- $t + 2.5\Delta$
  1. L_1 sends $\langle \mathsf{propose}, B_1, v, \mathcal{C}_{v-1}(B_0) \rangle$ to R_h with delay 0.
  2. R_h receives $\mathsf{propose}$ from L_1 and broadcasts $vote_1 = \langle \mathsf{vote}, B_1, v \rangle$ with delay Δ.
- $t + 3\Delta$
  1. L_2 broadcasts $blame_1 = \langle \mathsf{blame}, v \rangle$ with delay Δ since it did not receive $p = 1$ proposal within $3\Delta = (2p + 1)\Delta$.
  2. R_f sends $blame_2 = \langle \mathsf{blame}, v \rangle$ to L_2 with delay 0, as if it also blames the leader by timeout.
  3. L_2 receives more than $f+1$ blame messages ($blame_3 = blame_1 \cup blame_2$), broadcasts all the blame messages with delay Δ, and starts waiting for Δ.
- $t + 3.5\Delta$
  1. R_h receives $vote_1$ from L_2, but it cannot certify B_1 since its size is less than $f + 1$.
  2. R_f sends $vote_2 = \langle \mathsf{vote}, B_1, v \rangle$ to R_h with delay 0.
  3. R_h receives more than $f + 1$ votes including its vote ($vote_3 = vote_1 \cup vote2$), then it certifies B_1.
- $t + 4\Delta$
  1. R_h receives more than $f + 1$ blame messages $blame_3$, broadcasts all the blame messages with delay Δ, and starts waiting for Δ.
  2. L_2 has waited for Δ since it quit the old view, so it starts the next view $v + 1$. Here, L_2 does not recognize B_1 as the highest certified block.
  3. L_2 sends $\langle \mathsf{propose}, B'_1, v, \mathcal{C}_{v-1}(B_0) \rangle$ to R_2 with delay Δ.
- $t + 4.5\Delta$
  1. L_2 now receives more than $f + 1$ votes $vote_3$, and then certifies B_1. However, this certification is too late.
- $t + 5\Delta$
  1. R_h has waited for Δ since it quit the old view, so it starts the next view $v + 1$ and sends its highest certified block B_1 to the next leader L_2.
  2. R_h receives the proposal $vote_4$ $\langle \mathsf{propose}, B'_1, v, \mathcal{C}_{v-1}(B_0) \rangle$ but it does not vote since B'_1 does not extend from their highest certified block B_1.

The block B'_1 proposed by L_2 does not extend B_1, which is the highest certified block in R_h, and thus R_h does not vote. Therefore, the honest proposal B'_1 cannot collect $f + 1$ votes, and L_2 cannot propose a valid block anymore.

We can apply this attack to the protocol with responsive mode as follows: R_f sends its votes for B_1 to R_h just before L_2 starts the next view after waiting for 2Δ.

4 Refinements

In this section, we present some refinements to prevent the force-locking attack and guarantee the safety and liveness of the protocols. We describe the modified protocols for all models and formally prove their safety and liveness.

4.1 The Protocol under the Standard Synchronous Model

Figures 6 and 7 show the refined protocols under the standard synchronous model (hereafter denoted by Π^*_{std}). The gray colored parts are the same as the original protocol under the standard synchronous model, i.e., Π_{std}. The refinements are mainly divided into two parts corresponding to safety and liveness.

For the safety, a timer for a block is set after a following block is proposed to ensure that a replica already has a certificate for the block when it sets the timer for that block. Although this is not enough to ensure that all honest replicas vote for the block (but a quorum of replicas already voted for the block), the absence of equivocation within 2Δ ensures that certificates for conflicting blocks are not created in the view. This technique is inspired by the original protocol for the mobile sluggish model, where the timer for a block is set after the following two blocks are proposed.

For the liveness, we modify both the steady state protocol and the view-change protocol. First, the problem in the original protocol is that the next leader cannot collect all locks of honest replicas and the leader's proposal cannot be voted for. To address this problem, a replica submits its current lock (hereafter denoted by $lock(r, v)$ as the submission of replica r at view v) to the next leader after quitting the current view, and then waits for 2Δ before entering the next view. This ensures that the next leader can collect all locks of honest replicas before entering the next view since all honest replicas quit a view within Δ and waiting for another Δ is enough for the next leader to collect all of the submissions. In addition, for the first proposal after view-change, replicas check if the block extends a certified block with higher rank than the lock it submitted in the view-change. Since the leader received all the submissions and an honest leader extends the highest certified block it knows, an honest leader's proposal can be voted for by all honest replicas.

4.1.1 Safety and Liveness

We formally prove the safety and liveness of the modified protocols. First, we define additional notations. Let $pred(B)$ denote the predecessor of a block B, $view(B)$ denote the view where a block B is proposed, $>_{rank}$ denote the order of blocks in rank, $A >_{rank} B$ denote that A is higher in rank than B, and $\mathcal{H}$ denote a set of all honest replicas, including prompt and sluggish replicas. As defined in [1], we say that a block is committed directly if an honest replica commits it if no equivocation is observed within 2Δ after the replica votes, and we say that a block is committed indirectly if it is a result of directly committing a block extending the block.

Lemma 1 (Locked Honest)**.** *$\forall r \in \mathcal{H}, \forall v$ if $lock(r, v-1) >_{rank} pred(B)$, then r will not vote for B in view v.*

Proof. If $lock(r, v-1) >_{rank} pred(B)$, then $view(pred(B)) \leq v-1$ must hold. Thus, by the second rule of Vote in Figure 6, r will not vote for B. □

Π^*_{std}.steady

Let v be the current view number, L be the leader of the current view, and r be a replica. Replica r executes the following protocol.

1. **Propose:** If $r = L$, broadcast $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}) \rangle_L$ where $B_k = (b_k, h_{k-1})$, b_k is a batch of new client requests, $v' \leq v$, and B_{k-1} is the highest certified block L knows.

2. **Vote:** Upon receiving $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}) \rangle_L$

   (a) If $v' = v$, B_{k-1} is a highest certified block r knows and no equivocation in view v has been observed so far, then broadcast a vote in the form of $\langle \mathsf{vote}, B_k, v \rangle$, set $\mathsf{commit\text{-}timer}_{k-1}$ to 2Δ, and start counting down.

   (b) If $v' < v$, B_{k-1} is as high as $lock(r, v-1)$ and no equivocation in view v has been observed so far, then broadcast a vote in the form of $\langle \mathsf{vote}, B_k, v \rangle$.

3. **(Non-blocking) Commit:** When $\mathsf{commit\text{-}timer}_k$ reaches 0 and no equivocation in view v has been observed so far, commit B_k and all its ancestors.

Figure 6: The refined steady state protocol under the standard synchronous model

Lemma 2 (Certified without Equivocation)**.** *If an honest replica directly commits B_k in view v, then (i) every honest replica receives $\mathcal{C}_v(B_k)$ before it quit view v, and (ii) every honest replica does not vote for any B equivocating B_k in view v.*

Proof. Suppose a replica r directly commits B_k in view v at time t, then r must have voted for B_{k+1} at $t - 2\Delta$, and thus all honest replicas have $\mathcal{C}_v(B_k)$ at time $t - \Delta$. This ensures that all honest replicas neither quit view v nor detected any equivocation in view v before $t - \Delta$, since otherwise, before t, r must have received more than $f + 1$ blame messages or detected an equivocation, either of which would prevent the commit. □

Lemma 3. *If an honest replica directly commits B_k in view v, then $\forall v' \geq v, \forall r' \in \mathcal{H}$ (i) $lock(r', v')$ extends B_k and (ii) $\forall B$ $(view(B) = v')$ if B is certified and $B >_{rank} B_k$, then B extends B_k.*

Proof. We will prove Lemma 3 through induction on the view number. We first prove it for the base case ($v' = v$). By the second part of Lemma 2, any certified block in view v does not conflict with B_k, and thus all certified blocks in view v higher than B_k should extend B_k, which proves the second part. By the first part of Lemma 2, every honest replica will submit its lock whose rank is higher than or equal to that of B_k, and since all certified blocks in view v do not conflict with B_k, every honest replica will submit its lock which extend B_k, which proves the second part. We next prove it for the inductive step. We prove it by contradiction. For the second part, suppose a certified block B does not extend B_k. Let B_{min} be the ascendant in view $v' + 1$ with minimum rank, then $view(pred(B_{min})) \leq v'$ must hold. By the inductive hypothesis, $view(pred(B_{min})) < v$ must hold and it implies $\forall r' \in \mathcal{H}.\ pred(B_{min}) <_{rank} lock(r', v')$. By Lemma 1, B_{min} could not collect a vote from an

Π^*_{std}.change

Let L and L' be the leaders of views v and $v+1$, respectively, and r be a replica. Replica r executes the following protocol.

1. **Blame:** If less than p blocks are received from L in $(2p+1)\Delta$ time in view v, broadcast $\langle \mathsf{blame}, v \rangle$. If a leader L proposes equivocating blocks, r broadcasts $\langle \mathsf{blame}, v \rangle$ and the two equivocating blocks.

2. **Quit and Lock:** Upon receiving $f+1$ $\langle \mathsf{blame}, v \rangle$ messages, broadcast them, quit view v (abort all commit-timer and stop voting in view v), and send a highest certified block denoted by $lock(r, v)$ to L'.

3. **Warmup:** After waiting for 2Δ, transition back to steady state.

Figure 7: The refined view-change protocol under the standard synchronous model

honest replica, which contradicts that the block is certified. For the first part, suppose in an honest replica r'' that $B = lock(r'', v'+1)$ does not extend B_k, then by the second part which was already proven, $view(B) < v$ must hold. Since an honest replica submits the highest certified block it knows, B would not be selected since at least B_k is higher than B, which is a contradiction. □

Lemma 4 (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 ranks extend B_k.*

Proof. By the second part of Lemma 2, B'_k cannot collect a vote from an honest replica, and thus it will not be certified, which proves the first part. The second part was already proven in the second part of Lemma 3. □

Theorem 1 (Safety)**.** *Honest replicas always commit the same block B_k for each height k.*

Proof. In [1], this is proven only by the Unique Extensibility Lemma (Lemma 2 in [1]), which we proved in Lemma 4. □

Theorem 2 (Liveness)**.** *(i) A view-change will not happen if the current leader is honest, (ii) a byzantine leader must propose p blocks in $(2p+1)\Delta$ time to avoid a view-change, and (iii) if k is the highest height at which some honest replica has committed a block in view v, then leaders in subsequent views must propose blocks at heights higher than k.* (These arguments are the same as in Theorem 4 in [1]).

Proof. Suppose at time t an honest replica quit the view for the first time in all honest replicas, then every honest replica quit the view before $t+\Delta$ and every lock submitted by an honest replica is received by the next leader before $t+2\Delta$. If the next leader is honest, it proposes a block extending the highest certified block it knows and does not commit any equivocation. Then, all honest replicas can vote for the proposal and create a certificate for the proposal. The maximum lag time of entering the view is Δ, and the maximum time to propose a block and receive its certificate from honest replicas is the maximum round-trip delay 2Δ, and thus

Π^*_{slg}.steady

Let v be the current view number, replica L be the leader of the current view, and r be a replica. Replica r executes the following protocol.

1. **Propose:** If $r = L$, broadcast $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}) \rangle_L$, where $B_k = (b_k, h_{k-1})$, b_k is a batch of new client requests, $v' \leq v$, and B_{k-1} is the highest certified block L knows.

2. **Vote:** Upon receiving $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}) \rangle_L$

   (a) If $v' = v$, B_{k-1} is a highest certified block r knows, and no equivocation in view v has been observed so far, then broadcast a vote in the form of $\langle \mathsf{vote}, B_k, v \rangle$ and if $view(B_{k-2}) = v$, set $\mathsf{pre\text{-}commit\text{-}timer}_{k-2}$ to 2Δ and start counting down.

   (b) If $v' < v$ and B_{k-1} is as high as $lock(r, v-1)$ and no equivocation in view v has been observed so far, then broadcast a vote in the form of $\langle \mathsf{vote}, B_k, v \rangle$.

3. **(Non-blocking) Pre-commit:** When $\mathsf{pre\text{-}commit\text{-}timer}_k$ reaches 0 and no equivocation in view v has been observed so far, pre-commit B_k and broadcast $\langle \mathsf{commit}, B_k, v \rangle$.

4. **(Non-blocking) Commit:** Upon receiving $\langle \mathsf{commit}, B_k, v \rangle$ from $f + 1$ distinct replicas with the same v, commit B_k and all its ancestors.

Figure 8: The refined steady state protocol under the mobile sluggish model

an honest leader proposes at least p blocks within $(2p + 1)\Delta$ time. Therefore, all honest replicas cannot blame the leader and view-change will not happen, which proves the first part. If a faulty replica does not propose p blocks within $(2p + 1)\Delta$, then all honest replicas blame the leader and view-change happens, which proves the second part. For the last part, by the second part of Lemma 2, if the leader does not propose a block with a height higher than k within 3Δ, all honest replicas will not vote for the proposal and blame the leader. □

4.2 The Protocol under the Mobile Sluggish Model

Figure 8 shows the refined steady state protocol under the mobile sluggish model (hereafter denoted by Π^*_{slg}). The prevention of the attack on the liveness is similar to that in the standard synchronous model. As a minor refinement we additionally describe explicitly that the pre-commit timer is set only if the block is proposed in the same view. This may be implicitly included in the original protocol, although it was not mentioned explicitly in [1]. The refined view-change protocol, i.e., Π^*_{slg}.change, is the same as that under the standard synchronous model, i.e., Π^*_{std}.change.

4.2.1 Safety and Liveness

The safety of Π_{slg} is not violated. However, the proof in [1] relies on the following argument: *"If an honest replica directly commits a block B_l in view v, then $f+1$ honest replicas vote for B_l in that view"* (the first part of Lemma 6 in [1]). This argument is incorrect as shown in our proposed attack on the liveness. Moreover, it is not clear that our refinement to prevent the attack on liveness cannot bring any security holes that can violate the safety of Π^*_{slg}. For these reasons, we formally prove the safety of Π^*_{slg}. On the other hand, liveness of Π^*_{slg} is proved only under the standard synchronous model (recall that standard synchrony is a special case of the mobile sluggish model). Note however that the original protocol Π_{slg} does not guarantee the liveness even under the standard synchronous model by the attack on liveness.

Lemma 5 (Certified without Equivocation)**.** *If an honest replica directly commits a block B_k in view v, then there is a quorum of honest replicas R and $\forall r \in R$ (i) r receives $\mathcal{C}_v(B_k)$ before it quit view v and (ii) r does not vote for any block B equivocating B_k in view v.*

Proof. The proof scheme is almost the same as that of Lemma 6 in [1]. Suppose an honest replica directly commits a block B_k in view v, then a quorum of replicas (say R_0) pre-commits B_k. Let the earliest pre-commit among all honest replicas in R_0 be performed by r at time t. Then, r must have voted for B_{k+2} at $t-2\Delta$. This ensures that a quorum of replicas votes for B_{k+1} before $t-2\Delta$, and at least one replica in the quorum set is prompt. Therefore, at $t-\Delta$, a quorum of prompt replicas (say R) at the time must have received $\mathcal{C}_v(B_k)$. If a replica in R quits the view before $t-\Delta$ or observes any equivocation, a quorum of blame messages or equivocating blocks is received by at least a prompt replica in R_0 before it commits the block. This would prevent the commit and proves the first part. Since all replicas in R do not observe any equivocation before $t-\Delta$, they do not vote for the equivocating blocks in the same view, which proves the second part. □

Lemma 6. *If an honest replica directly commits a block B_k in view v, then $\forall v' \geq v$ there exists a quorum of honest replicas R and (i) $\forall r \in R$, $lock(r, v')$ extends B_k and (ii) for each certified block B in view v', if $B >_{rank} B_k$, then B extends B_k.*

Proof. We will prove it through induction on the view number. We first prove it for the base case ($v' = v$). By the second part of Lemma 5, any certified block in view v does not conflict with B_k, and thus all certified blocks in view v that have height higher than B_k should extend B_k, which proves the second part. By the first part of Lemma 5, a quorum of honest replicas receives $\mathcal{C}_v(B_k)$ before quitting view v and will submit its lock whose rank is higher than or equal to that of B_k. Since all certified blocks in view v do not conflict with B_k, they will submit their locks that extend B_k, which proves the second part. We next prove it for the inductive step. We prove it by contradiction. For the second part, suppose a certified block B in view $v'+1$ does not extend B_k. Let B_{min} be the ascendant in view $v'+1$ with minimum rank, then $view(pred(B_{min})) \leq v'$ must hold. By the inductive hypothesis, $view(pred(B_{min})) < v$ must hold and it implies $\forall r' \in \mathcal{R}.\ pred(B_{min}) <_{rank} lock(r', v')$ (R is a quorum of honest replicas that exists by the inductive hypothesis). By Lemma 1 which is also easily proven for this model, B_{min} could not collect votes from replicas in R, which contradicts that the block is certified. For the first part, suppose in an honest replica $r'' \in R$ that $B = lock(r'', v'+1)$ does not extend B_k, then by the second part which was already proven, $view(B) < v$ must hold. Since an honest replica submits the highest certified block

it knows, B would not be selected because at least B_k is higher than B. By the inductive hypothesis, $\forall r'' \in R$, $lock(r'', v')$ is higher than B_k, which is a contradiction. □

Lemma 7 (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 ranks extend B_k.*

Proof. By the second part of Lemma 5, B'_k cannot collect votes from honest replicas, and thus it will not be certified, which proves the first part. The second part was already proven in the second part of Lemma 6. □

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

Proof. This is proven in the same way as in Theorem 1 by the Unique Extensibility Lemma, which we already proved in Lemma 7. □

Theorem 4 (Liveness)**.** *Under standard synchrony (i.e., $d = 0$), (i) a view-change will not happen if the current leader is honest, (ii) a byzantine leader must propose p blocks in $(2p+1)\Delta$ time to avoid a view-change, and (iii) if k is the highest height at which an honest replica has committed a block in view v, then leaders in subsequent views must propose blocks at heights higher than k.*

Proof. We can easily prove in the same way as for the modified protocol under the standard synchronous model. □

4.3 The Protocol with Optimistic Responsiveness

Figures 9 and 10 show the refined protocol with responsive mode (hereafter denoted by Π^*_{rsp}). In the steady state protocol, we added some minor refinements as in Π^*_{slg} as follows: (i) prevention of the attack on liveness: for the first proposal after the view-change, a replica votes if the predecessor is higher than the lock it submitted at the end of the previous view, and (ii) explicit description: in setting the timer for the block or pre-commit the block, check that the block is from the same view. Waiting after submitting its lock in the Warmup phase of the view-change protocol is also a part of the prevention of the attack on liveness. The main solution is to deal with a responsive commit. In the Lock phase, we added another waiting for 2Δ after receiving a quorum of *blame*2 messages. This ensures that when some honest replicas committed a block directly, its certificate will be received by a quorum of honest replicas before the replicas submit their locks.

4.3.1 Safety and Liveness

We formally prove the safety. However, the liveness is not guaranteed under minority fault assumption because an adversary can violate the liveness immediately after switching to responsive mode by stopping the votes from faulty replicas. This can also be applied to the original protocol, i.e., Π_{rsp}, and the liveness proof is not complete in [1].

Lemma 8. *If an honest replica directly commits a block B_k in view v, then there is a quorum of honest replicas R and $\forall r \in R$, r receives $\mathcal{C}_v(B_k)$ before submitting $lock(r, v)$.*

Π^*_{rsp}.steady

Let v be the current view number, replica L be the leader of the current view, and r be a replica. Replica r executes the following protocol.

1. **Propose:** If $r = L$, broadcast $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1})\rangle_L$, where $B_k = (b_k, h_{k-1})$, b_k is a batch of new client requests, $v' \leq v$, and B_{k-1} is the highest certified block L knows.

2. **Vote:** Upon receiving $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1})\rangle_L$,

   (a) If $v' = v$ and B_{k-1} is a highest certified block r knows and no equivocation in view v has been observed so far, then broadcast a vote in the form of $\langle \mathsf{vote}, B_k, v\rangle$ and (i) If $\mathcal{C}_{v'}(B_{k-1})$ is a strong certificate, switch to the responsive mode and only vote for blocks with strong certificates for the rest of this view; (ii) Else, if $view(B_{k-2}) = v$, then set $\mathsf{pre\text{-}commit\text{-}timer}_{k-2}$ to 2Δ and start counting down.

   (b) If $v' < v$, B_{k-1} is as high as $lock(r, v-1)$ and no equivocation in view v has been observed so far, then broadcast a vote in the form of $\langle \mathsf{vote}, B_k, v\rangle$.

3. **(Non-blocking) Pre-commit:** (i) If at least one block has been committed in the responsive mode of this view, if $view(B_{k-2}) = v$, pre-commit B_{k-2} and broadcast $\langle \mathsf{commit}, B_{k-2}, v\rangle$.(ii) When $\mathsf{pre\text{-}commit\text{-}timer}_{k-2}$ reaches 0 and no equivocation in view v has been observed so far, pre-commit B_{k-2} and broadcast $\langle \mathsf{commit}, B_{k-2}, v\rangle$.

4. **(Non-blocking) Commit:** Upon receiving $\langle \mathsf{commit}, B_k, v\rangle$ from $f+1$ distinct replicas with the same v, commit B_k and all its ancestors.

Figure 9: The refined steady state protocol with responsive mode

Proof. Suppose an honest replica directly commits a block B_k in view v, then a quorum of replicas (say R_0) pre-commits B_k. Let the earliest pre-commit among all honest replicas in R_0 be performed by r at time t. Then, r must have voted for B_{k+2} before t. This ensures that a quorum of replicas vote for B_{k+1} before t, and at least one replica in the quorum set is prompt. Therefore, at $t + \Delta$, a quorum of prompt replicas (say R) at the time must have received $\mathcal{C}_v(B_k)$. Suppose for a contradiction that a replica in R submits its lock before $t+\Delta$, then it receives a quorum of *blame*2 messages before $t - \Delta$. This ensures that a quorum of replicas quit the view before $t - \Delta$, and prompt replicas at t receive $f + 1$ blame messages since at least one replica in the quorum set is prompt. Since at least one replica in R_0 (say r') is prompt at t, the quorum of blame messages would prevent the pre-commit in r', which is a contradiction. □

Lemma 9. *If an honest replica directly commits a block B_k in view v, then $\forall v' \geq v$ there exists a quorum of honest replicas R and (i) $\forall r \in R$, $lock(r', v')$ extends B_k and (ii) for all certified blocks B in view v, if $B >_{rank} B_k$, then B extends B_k.*

Proof. These are the same arguments in Lemma 6, therefore we only prove it for the responsive

Π^*_{rsp}.change

Let L and L' be the leaders of views v and $v+1$, respectively. Replica r executes the following protocol.

1. **Blame:** If less than p blocks are received from L in $(2p+1)\Delta$ time in view v, broadcast $\langle \mathsf{blame}, v \rangle$. If a leader L proposes equivocating blocks, broadcast $\langle \mathsf{blame}, v \rangle$ and the two equivocating blocks.

2. **Quit old view:** Upon receiving $f+1$ $\langle \mathsf{blame}, v \rangle$ messages, broadcast them along with $\langle \mathsf{blame2}, v \rangle$ and quit view v (abort all commit-timer and stop voting and commit in view v).

3. **Lock:** Upon receiving $f+1$ $\langle \mathsf{blame2}, v \rangle$, wait for 2Δ and send a highest certified block to L'.

4. **Warmup:** After waiting for 2Δ, enter view $v+1$ and transition back to steady state.

Figure 10: The refined view-change protocol with responsive mode

mode. We will prove it through induction on the view number. We prove it for the base case ($v' = v$). First, if B_k is directly committed in the responsive mode, there does not exist a certificate for any block equivocating B_k, which proves the second part. We prove it in the same way as in Lemma 8 in [1]. If B_k is directly committed in the responsive mode, there exists a strong certificate for B_k. Thus, there cannot exist a strong certificate for a block equivocating B_k, otherwise $f+1 = (3n/4+1) + (3n/4+1) - n$ replicas would have voted for it. At least one ancestor of B_k should be committed synchronously in the responsive mode and at most d honest replicas (i.e., sluggish replicas) are left behind in the synchronous mode, which is not enough to create a certificate. The first part is clear by Lemma 8 and the second part was already proven. The inductive step is almost as same as in the proof of Lemma 6, thus we skip it here. □

Lemma 10 (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 ranks extend B_k.*

Proof. As shown in Lemma 9, any block equivocating B_k cannot collect a quorum of certificates, this proves the first part. The second part was already proven in the second part of Lemma 9. □

Theorem 5 (Safety)**.** *Honest replicas always commit the same block B_k for each height k.*

Proof. This is proven in the same as manner as in Theorem 1 by the Unique Extensibility Lemma, which we proved in Lemma 10. □

5 Conclusion

In this paper, we presented the force-locking attack on Sync Hotstuff. This attack can violate the safety and liveness of the protocol, which are necessary security properties for a state machine replication protocol. We also presented the protocols for each model with reasonable refinements and formally proved their security. Our refinements remove the security vulnerability in Sync Hotstuff without any performance compromises.

As mentioned in [1], the mobile sluggish model is still far from being applied to an actual network because an offline replica may not receive all the messages sent before Δ at the time it wakes up. Moreover, in the responsive mode, liveness failure can still happen even with our refinements. These will be investigated in future works.

References

[1] Abraham, I., Malkhi, D., Nayak, K., Ling, R., Maofan, Y.: Sync hotstuff: Simple and practical synchronous state machine replication. IACR Cryptology ePrint Archive, Report 2019/270 (2019), `https://eprint.iacr.org/2019/270`

[2] Buterin, V.: Beacon chain casper mini-spec. Ethereum Research (2018), `https://ethresear.ch/t/beacon-chain-casper-mini-spec/2760`

[3] Buterin, V., Griffith, V.: Casper the friendly finality gadget. arXiv preprint arXiv:1710.09437 (2017)

[4] Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: 3rd Symposium on Operating Systems Design and Implementation. pp. 173–186. USENIX (1999)

[5] Chan, T.H.H., Pass, R., Shi, E.: Pili: An extremely simple synchronous blockchain. IACR Cryptology ePrint Archive, Report 2018/980 (2018), `https://eprint.iacr.org/2018/980`

[6] Cruz, J.P., Kaji, Y.: The bitcoin network as platform for trans-organizational attribute authentication. In: Third International Conference on Building and Exploring Web Based Environments. pp. 29–36. IARIA (2015)

[7] Cruz, J.P., Kaji, Y., Yanai, N.: Rbac-sc: Role-based access control using smart contract. IEEE Access **6**, 12240–12251 (2018)

[8] Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM **35**(2), 288–323 (1988)

[9] Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Eurocrypt 2015. pp. 281–310. Springer (2015)

[10] Guo, Y., Pass, R., Shi, E.: Synchronous, with a chance of partition tolerance. IACR Cryptology ePrint Archive, Report 2019/179 (2019), `https://eprint.iacr.org/2019/179`

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

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

[13] Nakamoto, S., et al.: Bitcoin: A peer-to-peer electronic cash system (2008)

[14] Nakamura, R.: Analysis of bouncing attack on ffg. Ethereum Research (2019), `https://ethresear.ch/t/analysis-of-bouncing-attack-on-ffg/6113`

[15] Nakamura, R.: Prevention of bouncing attack on ffg. Ethereum Research (2019), `https://ethresear.ch/t/prevention-of-bouncing-attack-on-ffg/6114`

[16] Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Eurocrypt 2017. pp. 643–673. Springer (2017)

[17] Pass, R., Shi, E.: Hybrid consensus: Efficient consensus in the permissionless model. In: DISC. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)

[18] Pass, R., Shi, E.: The sleepy model of consensus. In: Asiacrypt 2017. pp. 380–409. Springer (2017)

[19] Pass, R., Shi, E.: Thunderella: Blockchains with optimistic instant confirmation. In: Eurocrypt 2018. pp. 3–33. Springer (2018)

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

[21] Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: Bft consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. pp. 347–356. ACM (2019)