Force-Locking Attack on Sync Hotstuff

Atsuki Momose
Nagoya University
momose@sqlab.jp

December 25, 2019

Abstract

Blockchain which realize state machine replication (SMR) is widely studied recently as the fundamental building block of decentralized cryptocurrency and smart contract which need consensus mechanism in the global scale public trustless network. In such situation larger resiliency (e.g., minority fault)of the protocol is favorable, that motivate some research on synchronous protocol which have been studied only on the theoretical level but not for realistic use. Abraham et al. published a synchronous SMR protocol called Sync Hotstuff at ePrint (which will appear in IEEE S&P 2020) which is extremely simple and practical. It achieve 2Δ latency which is near optimal in a synchronous model, and without lock-step execution its throughput is comparable to that of partially synchronous protocols. They present not only for standard synchronous model but for weaker model called mobile sluggish model which is more realistic. And it also adopts optimistic responsive mode where its latency is independent of Δ. However, there is a critical security vulnerability. In this paper, we present force-locking attack on Sync Hotstuff. This attack violate safety of the protocol for standard synchronous model, and liveness of all versions of the protocol including for the mobile sluggish model and with responsive mode. This attack is not only a specific attack on Sync Hotstuff but a general form of attack scheme in the blockchain protocol we call *force-locking*. We then present some refinements to prevent this attack. Our modification remove its security vulnerability without any performance compromises. We also give formal proofs of 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 protocol have been developed which are considered to be deployed in a small or middle sized network such as server or database replication in the data center. And in such applications, performance is prioritized more than resiliency. In such use cases, synchronous protocol is not considered as a practical solution for the following reasons. First, classical synchronous protocols are too slow since they need large number of rounds which depend on Δ (the known upper bound of the network delay) to finalize. Furthermore, they need lock-step execution (i.e., all replicas start and end at the same time) that makes the latency depend on the number of synchronous round. Second, the standard synchronous model which ensure network synchrony for all honest replicas is unrealistic in practice since small network partition may happen frequently. And in the practical use, synchronous parameter Δ is

overestimated to ensure enough security that make it much slower. For these reasons, partial synchronous protocols [4, 8, 12, 11] are much favored. They can tolerate any network failure to ensure safety. And since they do not rely on network synchrony to finalize, they achieve responsiveness (i.e., the latency is independent of known bound Δ but actual bound δ). Although some synchronous protocol can tolerate under the presence of minority byzantine replicas whereas partial synchronous protocol can only tolerate up to 1/3, that is not enough to be favored.

However, since the appearance of the blockchain in 2008 [13, 16, 9] and its applications such as cryptocurrency and smart contract [20, 7, 6], SMR protocol is considered not only for replication in the local scale but for the global scale public trustless network. In such situation, resiliency is much more required because the presence of malicious activity is more realistic than local private network. Indeed a huge number of cyber attacks are conducted across the internet on a daily basis. In order to benefit from the advantage of resiliency of synchronous protocol, many researchers try to address the above problems recently. To rethink the synchronous model which is more fit in the real world than strong classical synchronous model, Guo et al [10] introduced a new model they call "weak synchrony". In this model, not malicious but offline replica (i.e., cannot respect synchrony assumption) because of some network failure is still treated as honest and the set of offline replicas can change over time. This modeling is so important in the long term deployment such as public blockchain where anyone may sleep sometimes. For the long latency problem, Thunderella [19] introduced the new notion of "optimistic responsiveness". It can finalize independent of Δ in the optimistic situation that is not assured by assumption but can appear if faulty replicas are few.

In these circumstances, Abraham et al presented an synchronous protocol called Sync Hotstuff. It has extremely simple design but avoid the above problems with several elegant but natural techniques. First it uses the synchronous parameter Δ only for checking the absence of equivocation, minimizing the reliance on the synchrony. With this technique, it achieves 2Δ latency which is near optimal in the synchronous model. They argued that it may probably be the optimal latency since a round trip may be necessary to check the response of the network. Additionally it does not use lock-step execution. That make it possible to achieve throughput that is comparable to that of partially synchronous protocol. To consider more realistic synchronous model, they also applied the weakly synchronous model but they call it *mobile sluggish model*. It ensure the safety of a *sluggish* (i.e., offline or slow) replica by collecting enough messages to check the finality of a *prompt* (i.e., respecting the synchrony assumption) replica. Futhermore it also achieves optimistic responsiveness by checking the agreement on the optimistic situation with synchronous mode to switch the responsive mode. Even with these additional deal, it is still simple.

1.2 Block-Chaining and Locking

The main features of the blockchain scheme is not just blocking (i.e., batching commands) but chaining, (i.e., constructing a chain by hash references). That translate the SMR into a simple problem, selecting one path or chain from growing tree and it enable defining conflict with descendant/ascendant relation. And as such, finality of a block finalizes all the ascendants. This greatly simplify the discussion, in that even if some replicas finalize a block and others not in sometime, they just need to finalize a descendant of the block in the future. For these simplicity and versatility, blockchain scheme is widely used as a framework for SMR protocol today.

In most voting based blockchain protocol, there is a middle state of a block to reach the final state. The state is to ensure that the block is voted by enough replicas, e.g., *certified* in Sync Hotstuff, *justified* in Casper FFG [3], *notarized* in PiLi [5], *prepared* in Hotstuff [21], hereafter we call it certified following Sync Hotstuff. Additionally they have some kind of freshness of a block to compare e,g,. view number, block hight, epoch number. And a certified block is finalized when it is assured that any fresher block conflicting with it will not be certified (i.e., they cannot collect enough vote) by the rule of where to vote defined in the protocol, and thus will not be finalized, this provide safety. The rule of where to vote is sometimes called *fork choice rule*. The fork choice rule provide not just safety but also liveness. And in some protocols, the fork choice is always in a subtree which has a root block as long as the root change event not happens. We say a replica is *locked* on the root block, and call the root change event *unlocking*. In some protocol, replica is locked on the freshest certified block. This is quite natural because finality rule is constructed so that a finalized block include all certified blocks that is fresher in its ascendant. The locking scheme is so important building block of fork choice that it should be constructed carefully since it is directly related with security of the protocol.

1.3 Our Contribution

1.3.1 Force-Locking Attack

In this paper, we present *force-locking attack* on Sync Hotstuff. This attack violate safety of the protocol for the standard synchronous model, and liveness of all versions of the protocol including for the mobile sluggish model and with responsive mode. This is not a specific attack on Sync Hotstuff but a general form of attack scheme we call force-locking which can be applied to any locking based protocol. In every protocol, locking point is determined by replica's past messages. Thus, the messages sent by faulty replicas which is controlled by the adversary also affect the choice. When a block is lockable i.e., honest replica's past messages make it ready for faulty replica's messages to change the lock to the point, the adversary can change the lock whenever he want. And by the network delay, the messages received by each replica is sometimes different. Additionally to make the matters worse, network may fully be controlled by the adversary except for delay bound in the synchronous model. By the biased network delay and messages sent by faulty replicas, the adversary can control the view on past messages and change and split the locking point in any honest replicas. If a protocol relying on the locking rule for its safety/liveness without any prevention from the force-locking, they can be violated. In practice, bouncing attack on Casper FFG the core consensus protocol for Ethereum2.0 is a variant of this attack scheme [2, 14, 15]. This attack is simply change the lock before some block is finalized by releasing faulty replica's messages to lock a lockable block conflicting the block expected to be finalized, and repeat it definitely to violate liveness. On the other hands, Sync Hotstuff relies on the locking rule for both its safety and liveness and has no prevention from force-locking. As other classical protocols, in Sync Hotstuff, a block proposer called leader is selected, and the reign of the leader is called view. And the process in the reign is called steady state protocol and the process to change the leader is called view-change protocol. And the locking point is freshest certified block. In the protocol for standard synchronous model, block is finalized when maximum round-trip time 2Δ passed and no equivocation (i.e., conflicting proposal by the same leader) is observed so far. The authors argued that it ensure that all honest replicas vote for the block since there

is no equivocation. However, this is not the case when honest replicas are locked on different blocks and cannot vote for the block. In this case, the block is finalized in some replicas but not certified, and safety is violated by extending conflicting certified block. In each version of the protocol (for standard synchrony, for mobile sluggish, with optimistic responsive), at every time replica is locked on a freshest certified block. However in the presence of force-locking, honest replicas and also honest leader can be locked on some different blocks. In this case, honest leader's proposal cannot be voted by some honest replicas and cannot be finalized, this violate the liveness.

1.3.2 Refinements and Proofs

We also present some simple refinements to prevent force-locking. First for liveness, in order for honest proposal to be voted by all honest replicas, we modify the view-change protocol since if the leader is honest all the proposals in its view are consistent and any inconsistency is only from the different view. Intuitively, we add 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. Additionally we add a little bit modification to the locking rule as follows: for the first block after view-change, replica is locked on its submission in the view-change. Since honest leader can collect all the locks from honest replicas and propose according to them, honest replicas can vote for it. For the safety, this modification is not enough since faulty leader may propose ignoring the submissions. The main problem is that the absence of equivocation cannot ensure the existence of certificate. To address this problem, we add the rule to check the existence of certificate before the round-trip waiting to ensure the absence of equivocation. Despite the modification, honest replica's vote is not be guaranteed though, existence of certificate in all honest replicas is guaranteed. This ensure that honest replicas will not vote for conflicting block, and thus ensure safety. Note that because the synchronous waiting in the steady state is not changed and it is still without lock-step execution, these modifications do not bring any performance compromise. Finally, we also give formal proofs of security for each version of the protocol.

1.4 Paper Organization

In section 2, we briefly review the Sync Hotstuff and some definitions and conventions. In section 3, we present the force-locking attack in detail with specific attack scenario. In section 4, we propose some refinements to the Sync Hotstuff and formally prove the safety and liveness of the modified protocols. We conclude with the summary of this paper in section 6.

2 Sync Hotstuff

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

2.1 State Machine Replication

A state machine replication protocol (SMR) is used for building a fault tolerant service to process 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 non-faulty replicas

consistently commit client requests into linearizable log. A SMR protocol is expected to provide at least the following two security properties: (i) *safety*: if a non-faulty replica commit a value in its log position, the value committed at the same position by any non-faulty replica is also the same. (ii) *liveness*: each client requests is eventually committed by all non-faulty replicas. In the asynchrony or partial synchrony, it is known that resiliency is up to 1/3. Whereas, in synchrony, it is known that minority fault is possible (i.e., $n = 2f + 1$), and Sync Hotstuff also assume the same.

2.2 Network model

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

For synchrony assumption, the *standard synchrony model* is first introduced. It is simply state 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 network is controlled by the adversary (i.e., adversary can reorder or delay messages), but the adversary can only control the network under this synchronous compliance, thus do not deviate from the assumption. The weaker model called *mobile sluggish model* is introduced. This model was first introduced by Guo et al [10] as "weak synchrony". In this model, a *sluggish* replica, i.e., a replica which cannot respect the synchronous assumption, is still treated as honest. On the other hand, a replica which respect the synchronous assumption is called *prompt*. Futhermore, the set of sluggish replicas can change arbitrarily at every time, under the constraint that $f = d + b$ (d and b are each the number of sluggish and byzantine replicas). It should be noted that the standard synchronous model is a special case of mobile sluggish model that $d = 0$.

2.3 Overview and Definitions

Sync Hotstuff consists of two sub protocols. The first is called steady state protocol and the main procedure to process client requests. In the steady state, a stable leader propose an new entry for log and then other replicas vote for it to confirm the proposal. The reign of the leader is called view and is identified by a monotonically increasing view number. The leader of each view is simply selected through round robin. When a leader is lazy to progress or some byzantine activity is detected, other replica blame the leader and transition into the second protocol called view-change and switch the view and leader.

Sync Hotstuff use the blockchain scheme. In the steady state protocol, the leader propose a block $B_k = (b_k, h_{k-1})$ (k is a height in the blockchain), which consists of b_k: a batch of client's requests and h_{k-1}: a hash of predecessor in the blockchain on which the block extend, i.e., $h_{k-1} = H(B_{k-1})$. The genesis block is defined as $\perp$. The validity of a block is defined as follows: 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 ancestor. If a block B_k is an ancestor of another B_l ($l \geq k$), we say B_l extend B_k, and if a leader propose two blocks which are not extend one another, we say the blocks equivocate one another.

In Sync Hotstuff, a proposal of a block must contain a set of signatures on the predecessor from a quorum (i.e., $f + 1$) of replicas from the same view and it is called certificate. A certificate for a block B in view v is denoted as $\mathcal{C}_v(B)$. All certified blocks are ranked by first views, and heights, i.e., higher view higher rank, and higher hight higher rank if the views

are the same. During the protocol execution, each replica keeps track of all signatures for all blocks and keeps updating the highest certified block to its knowledge. As we mentioned before, the highest certified block is used as a lock in each replica, and this is the point we use in the attack in the later section.

2.4 Protocol Specifications

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

2.4.1 The Protocol for the Standard Synchronous Model

The protocol under the standard synchrony is so simple. The Figure 1 show the steady state protocol under the standard synchrony. The steady state protocol runs in iteration. The leader proposes a block extending the highest certified block known to the leader, and it must contains the 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 known to the leader. Each replica, upon receiving a block B_k, broadcast a vote for it if it is the first valid block for the height by the leader and it extend the highest certified block known to the replica. Once replica vote for a block B_k, it start a timer for the height $\mathsf{commit\text{-}timer}_k$, and after waiting for 2Δ time, if no equivocation observed so far, it commit B_k and all its ancestors. Note that these are all processed without lock-step and commit is non-blocking, i.e, start next iteration without waiting the timer, blocks for each height are processed concurrently.

The main technique is the waiting for 2Δ before commit. The authors argue that this ensure safety informally and also formally in the Theorem 3 in [1], because of the following reasons in brief. Suppose a replica r commit a block at time t, then it voted for the block at time $t - 2\Delta$ and the vote reached before $t - \Delta$ at all honest replicas. If an honest replica r' voted some equivocating block before $t-\Delta$ it would reached r before t and it would prevented the commit, thus all honest replica voted for the block, and the block must be certified. And all honest replicas never vote for equivocating block, no conflicting block of the same height can be certified. However, this argument is incorrect, and we will explain it in the later section.

The Figure 2 show the view-change protocol. As other SMR protocols, this protocol is for a prevention of stuck. First to prevent a byzantine leader from not progressing, if less than p blocks are proposed by the leader within $(2p+1)\Delta$ time or equivocating proposals are observed, honest replicas broadcast the blame messages. And if a quorum of replicas blame the leader, the view are quitted, and after waiting for Δ, the replica start the next view and send a highest certified block to the next leader.

The author argued that the view-change protocol ensure that *"a view-change will not happen if the current leader is honest"* in Theorem 4 in [1]. This property should be held since this ensure the liveness. However, this argument is incorrect, and we will explain it in the later section.

2.4.2 The Protocol for the Mobile Sluggish Model

Figure 3 shows the steady state protocol for the mobile sluggish model. The gray colored parts are the same as the protocol under standard synchrony. The differences are that (i) the timer for a block is set after the following two blocks are proposed, and (ii) a block is committed

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

1. **Propose:** The leader L broadcasts $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}))\rangle_L$ 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 known to L.

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 known to the replica, 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 is observed so far, commit B_k and all its ancestors.

Figure 1: The steady state protocol under standard synchrony

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

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 $\mathsf{commit\text{-}timer}$ 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 standard synchrony

after a quorum of replicas waited for 2Δ. These marginal rules are for sluggish replicas to safely commit blocks. The first one ensure that the replica have $\mathcal{C}_v(\mathcal{C}_v(B_k))$ and this means that a quorum of replicas already have certificate. The assumption that $f = d + b$ ensure that all sets of a quorum of replicas include at least a prompt replica, thus it ensure that at least a prompt replica have certificate. Additionally, the second rule ensure in the same way that at least one prompt replica waited for 2Δ after at least one prompt replica received the certificate, and it did not observe any equivocation within the waiting time. Therefore, all prompt replicas thus a quorum of replicas have certificate and never vote for equivocating blocks, thus conflicting blocks will not be certified. The difference of safety argument between the protocol under the standard synchrony is that the existence of certificate of a block is checked before waiting 2Δ. This point prevent attack on safety, and we use this technique to modify the protocol for standard synchrony in the later section.

The view-change protocol is the same as for the standard synchrony. Therefore as we mentioned before we can violate liveness in the same way.

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

1. **Propose:** The leader L broadcasts $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}))\rangle_L$ 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 known to L.

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 known to the replica, 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 is observed so far, pre-commit B_k and broadcast $\langle \mathsf{commit}, B_k, v\rangle$.

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

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

2.4.3 The Protocol with Optimistic Responsiveness

Figure 4 and 5 shows the steady state protocol and view-change protocol each with responsive mode. This protocol achieve optimistic responsiveness, which is first introduced in Thunderella [19]. The technique is the same as what Thunderella introduced: $3n/4$ votes ensure the absence of equivocating block without synchronous waiting. The certificate of a block containing more than $3n/4$ signatures is called strong certificate. In the voting phase of steady state, when a proposal contains the strong certificate for the predecessor, the replica switch to the responsive mode. The main techniques are (i) to reach agreement on the switch to the responsive mode, and (ii) to take over the blocks committed responsively to the next view in the view change. For the first part, at least one block must be committed synchronously in the responsive mode. It ensure that quorum of prompt replica have switched to the responsive mode and commit in the synchronous mode (i.e., without strong certificate) will never happen in the view. For the second part, some modification to the view-change protocol ensure that quorum of replica get all certificates of committed blocks before entering the next view. First, replica must have $\mathcal{C}_v(\mathcal{C}_v(blame))$ before entering the next view, this ensure that at least one prompt replica quitted the view. Additionally the waiting for 2Δ after receiving $\mathcal{C}_v(\mathcal{C}_v(blame))$ ensure that quorum of prompt replicas receive all committed blocks, because suppose a block is committed at time t and a prompt replica enter the next view before $t+\Delta$, at least one replica quitted the view before $t-2\Delta$ and that would prevented the commit.

The modification to the view-change protocol ensure that quorum of replicas receive all certificates of committed blocks before entering the next view. However, despite this modification, liveness can be still violated in the same as the standard protocol. We will show the detail in the later section.

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

1. **Propose:** The leader L broadcasts $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}))\rangle_L$ 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 known to L.

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 known to the replica, 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 is observed so far, pre-commit B_{k-2} and broadcast $\langle \mathsf{commit}, B_{k-2}, v\rangle$.

4. **(Non-blocking) Commit:** On When $\langle \mathsf{commit}, B_k, v\rangle$ 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

3 Attacks

In this section, we present force-locking attack on Sync Hotstuff. This attack violate safety and liveness of Sync Hotstuff. We first introduce the force-locking and later describe specific attack scenarios.

3.1 Force-Locking

In the blockchain protocol, each replica or block-proposer select a path from genesis block to a leaf block in the block tree to vote or extend following the rule. This rule is often called the fork choice rule. In some blockchain protocol the leaf block is selected from a subtree in the block tree that change over time. We call the root block of the subtree as a lock and the root change event as unlocking or simply locking. In Sync Hotstuff each replica always locked on the highest certified block known to the replica, and newly certified block with higher rank unlock the replica to the block. In every protocol, the lock is determined by the messages received from all replicas including faulty replica. Therefore, a coalition of faulty replicas can intentionally change the lock or prevent locking by sending or withholding its messages if the protocol does not appropriately take care about it. Additionally, even in the synchronous model, biased message delay within the assumption can split the lock of each honest replica. The protocol with naively constructed locking rule can lose its safety or liveness. In Sync Hotstuff, the 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., honest replicas already voted on it and ready to be certified) to some honest replicas with delay 0

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

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 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

and to others with delay Δ. In the case where a leader steadily extend its proposal, it is not a problem since highest certified block is its proposal and without extending it, the leader is blamed for equivocation. However, when a view-change happens and honest replica's view on highest certified block in the previous view split, the proposal of the next leader cannot collect votes from some honest replica even though the leader cannot be blamed, and it lead the safety/liveness violation.

3.2 Safety Attack on the Standard Protocol

We first describe the attack on safety of the protocol under standard synchronous model. As described in the previous section, a block is committed if the replica waited for 2Δ time and no equivocation observed so far. The author argued that this ensure the existence of certificate on the block since without equivocation all honest replicas vote for it. However, this is not the case when view-change happens and honest replica's view on the highest certified block in the previous view split and some honest replica cannot vote for the proposal extend a certified block with lower rank. In this case, the block is committed but not certified, and after that, the blockchain extend the other certified block and safety is violated.

3.2.1 Attack Sceneario

We assume that at time t all parties start the view v and they all recognize B_0 as the highest certified block. This is a natural case, in practice when the protocol start i.e., $t = 0$, all parties start the view $v = 1$ and all recognize the genesis block $\perp$ as the highest certified block. We define some notations to describe the set of honest/faulty replicas. L_1 and L_2 denote the leaders at view v and $v+1$ each and both of them are faulty. R_f denote a set of faulty replicas of size f and it include L_1 and L_2. R_1 and R_2 each denote a set of honest replicas of size $(f+1)/2$, and every parties in the same set can communicate with delay 0 but if in the different set, they communicate through the network with standard synchrony. All faulty replicas are controlled by the adversary, and as we mentioned before the network is also controlled by the adversary. The total number of replica is $n = 2f + 1$ and faulty

replicas controlled by the adversary is f, this situation is within the fault assumption in Sync Hotstuff. Then, we describe a specific scenario chronologically where the safety is violated.

- $t + 2.5\Delta$
  1. L_1 send $\langle \mathsf{propose}, B_1, v, \mathcal{C}_{v-1}(B_0))\rangle$ to R_1 with delay 0.
  2. R_1 receive propose from L_1 and broadcast $vote_1 = \langle \mathsf{vote}, B_1, v\rangle$ with delay Δ.
- $t + 3\Delta$
  1. R_2 broadcast $blame_1 = \langle \mathsf{blame}, v\rangle$ with delay Δ since they did not receive $p = 1$ proposal within $3\Delta = 2p + 1$.
  2. R_f send $blame_2 = \langle \mathsf{blame}, v\rangle$ to R_2 with delay 0, as if they also blame the leader by timeout.
  3. R_2 receive more than $f + 1$ blame message ($blame_3 = blame_1 \cup blame_2$), then broadcast all the blame messages with delay Δ, and start waiting for Δ.
- $t + 3.5\Delta$
  1. R_2 receive $vote_1$ from R_1, but they cannot certify B_1 since its size is less than f+1.
  2. R_f send $vote_2 = \langle \mathsf{vote}, B_1, v\rangle$ to R_1 with delay 0.
  3. R_1 receive more than $f + 1$ vote ($vote_3 = vote_1 \cup vote2$), then they certify B_1.
- $t + 4\Delta$
  1. R_1 receive more than $f + 1$ blame messages $blame_3$, then broadcast all the blame messages, and start waiting for Δ.
  2. R_2 have waited Δ since they quitted old view, then they start the next view $v+1$, and send their highest certified block B_0 to the next leader L_2. Here, R_2 do not recognize B_1 as the highest certified block.
  3. L_2 send $\langle \mathsf{propose}, B'_1, v, \mathcal{C}_{v-1}(B_0)\rangle$ to R_2 with delay 0.
  4. R_2 broadcast $vote_4 = \langle \mathsf{vote}, B'_1, v + 1\rangle$ with delay Δ, then they set $\mathsf{commit\text{-}timer}_1$ to 2Δ and start counting down.
- $t + 4.5\Delta$
  1. R_2 now receive more than $f + 1$ vote $vote_3$, then certify B_1. But it is too late.
- $t + 5\Delta$
  1. R_1 have waited for Δ since they quitted old view, then start the next view $v + 1$, and send their highest certified block B_1 to the next leader L_2.
  2. R_1 receive $vote_4$ but they do not vote since B'_1 does not extend from their highest certified block B_1.
- $t + 6\Delta$
  1. R_2 commit B'_1 since the $\mathsf{commit\text{-}time}_1$ reaches 0 and no equivocation in view $v+1$ observed so far.

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

3.3 Liveness Attack

Next, we describe the attack on liveness. This attack can be applied to all of the three protocols, but first describe the attack on the protocol for standard synchronous model and mobile sluggish model and later modify a little bit for the protocol with responsive mode. The attack scheme is almost as same as that of the attack on safety. The difference is that this attack prevents the honest leaders from continuously propose valid block. The scenario is simply as follows. As same as safety attack, split the honest replica's view on the highest certified block in the previous view, but in this attack only the next honest leader see the old lower certified block and other honest replicas see the newly certified block. The honest leader propose extending the old block but it cannot receive certificate for the proposal and cannot propose valid block continuously and be blamed. Although this attack does not bring the permanent failure of finality, to ensure liveness honest leader should not be blamed, thus liveness is somewhat violated. Also, note that this is the violation of validity in the byzantine broadcast formulation which require that 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 they all recognize B_0 as the highest certified block. L_1 and L_2 denote the leaders at view v and $v+1$ each, and in this case $L1$ is faulty but $L2$ is honest. R_f denote a set of faulty replicas of size f including L_1. R_h denote a set of honest replica 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.

- $t + 2.5\Delta$
  1. L_1 send $\langle \mathsf{propose}, B_1, v, \mathcal{C}_{v-1}(B_0))\rangle$ to R_h with delay 0.
  2. R_h receive propose from L_1 and broadcast $vote_1 = \langle \mathsf{vote}, B_1, v\rangle$ with delay Δ.
- $t + 3\Delta$
  1. L_2 broadcast $blame_1 = \langle \mathsf{blame}, v\rangle$ with delay Δ since they did not receive $p = 1$ proposal within $3\Delta = 2p + 1$.
  2. R_f send $blame_2 = \langle \mathsf{blame}, v\rangle$ to L_2 with delay 0, as if they also blame the leader by timeout.
  3. L_2 receive more than $f + 1$ blame message ($blame_3 = blame_1 \cup blame_2$), then broadcast all the blame messages with delay Δ, and start waiting for Δ.
- $t + 3.5\Delta$
  1. R_h receive $vote_1$ from L_2, but they cannot certify B_1 since its size is less than $f+1$.
  2. R_f send $vote_2 = \langle \mathsf{vote}, B_1, v\rangle$ to R_h with delay 0.
  3. R_h receive more than $f + 1$ vote including himself ($vote_3 = vote_1 \cup vote2$), then they certify B_1.
- $t + 4\Delta$

1. R_h receive more than $f+1$ blame messages $blame_3$, then broadcast all the blame messages with delay Δ, and start waiting for Δ.
2. L_2 have waited Δ since they quitted old view, then they start the next view $v+1$. Here, L_2 does not recognize B_1 as the highest certified block.
3. L_2 send $\langle \mathsf{propose}, B'_1, v, \mathcal{C}_{v-1}(B_0)\rangle$ to R_2 with delay Δ.

- $t + 4.5\Delta$
  1. L_2 now receive more than $f+1$ vote $vote_3$, then certify B_1. But it is too late.
- $t + 5\Delta$
  1. R_h have waited for Δ since they quitted old view, then start the next view $v+1$, and send their highest certified block B_1 to the next leader L_2.
  2. R_h receive the proposal $vote_4$ $\langle \mathsf{propose}, B'_1, v, \mathcal{C}_{v-1}(B_0)\rangle$ but they do not vote since B'_1 does not extend from their highest certified block B_1.

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

We can apply this attack to the protocol with responsive mode easily as follows. R_f send their vote for B_1 to R_h just before the L_2 start the next view after waiting for 2Δ.

4 Modified Protocols

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

4.1 The Protocol for the Standard Synchronous Model

Figure 6 and 7 show the modified protocol for the standard synchronous model. The gray colored parts are the same as the original protocol for the standard synchronous model. The refinements are mainly two parts each for safety and liveness.

First for the safety, timer for a block is set after the following block is proposed. This ensure that when a replica set the timer for a block, it already have the certificate for the block. Although this is not enough to ensure that all honest replicas vote for the block, since a quorum of replicas already voted for the block, to check the absence of equivocation within 2Δ ensure that certificates for any 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 of all, the problem in the original protocol is that the next leader cannot collect all honest replica's lock, and it bring the situation where the leader's proposal cannot be voted. To address this problem, replica submit its current lock (hereafter denote $lock(r, v)$ as the submission of replica r at view v) to the next leader after quitting the current view, and wait for 2Δ before entering the next view. This ensure that the next leader can collect all

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

1. **Propose:** The leader L broadcasts $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}) \rangle_L$ 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 known to L.

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 highest certified block known to the replica and no equivocation in view v is observed so far, then broadcast a vote in the form of $\langle \mathsf{vote}, B_k, v \rangle$ and set $\mathsf{commit\text{-}timer}_{k-1}$ 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 is 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 is observed so far, commit B_k and all its ancestors.

Figure 6: The modified steady state protocol under standard synchrony

honest replica's locks before entering the next view, since all honest replicas quit a view within Δ, waiting for another Δ is enough for the next leader to collect all of the submissions. Additionally, for the first proposal after view-change, replicas check if the block extend 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 extend the highest certified block known to it, honest leader's proposal can be voted by all honest replicas.

Let L and L' be the leader if view v and $v+1$, respectively. And r be the replica.

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

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

Figure 7: The modified view-change protocol under standard synchrony

4.1.1 Safety and Liveness

We formally prove the safety and liveness of the modified protocol. To discuss clearly, we define some additional notations. $pred(B)$ denote the predecessor of a block B. $view(B)$ denote the view in which a block B is proposed. $>_{rank}$ describes the order of blocks in rank, and $A >_{rank} B$ means A is higher in rank than B. $\mathcal{H}$ denote a set of all honest replicas including prompt and sluggish. And as defined in [1] we say that a block is committed directly if an honest replica commit it by observing no equivocation within 2Δ after it 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 be held, and thus by the second rule of Vote in Figure 6, r will not vote for B. □

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

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

Lemma 3. *If an honest replica directly commits a block* B_k *in view* v*, then* $\forall v' \geq v, \forall r' \in \mathcal{H}$ *(i)* $lock(r', v')$ *extend* B_k *and (ii) for all certified block* B *in view* v'*, if* $B >_{rank} B_k$ *then* B *extend* 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 2 any certified blocks in view v do not conflict with B_k, thus all certified blocks in view v higher than B_k should extend B_k, which prove 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 block in view v do not conflict with B_k, every honest replica will submit its lock which extend B_k, which prove the second part. We next prove it for the inductive step. We prove it by contradiction. For the second part, suppose some certified block B does not extend B_k. Let B_{min} as the ascendant in view $v'+1$ with minimum rank, then $view(pred(B_{min})) \leq v'$ must be held. By the inductive hypothesis, $view(pred(B_{min})) < v$ must be held and it implies $\forall r' \in \mathcal{H}.\ pred(B_{min}) <_{rank} lock(r', v')$. By Lemma 1, B_{min} could not collect vote by honest replica, it contradict that the block is certified. For the first part, suppose in some honest replica r'', $B = lock(r'', v'+1)$ does not extend B_k, then by the second part which is already proven, $view(B) < v$ must be held. Since honest replica submit the highest certified block known to it, B would not be selected since at least B_k is higher than B, 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 vote from honest replica, thus it will not be certified, which prove the first part. The second part is 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, and (ii) A byzantine leader must propose p blocks in $(2p+1)$ time to avoid a view-change, and (iii) If k is the highest height at which some honest replica has commited a block in view v, then leaders in subsequent views must propose blocks at heights higher than k.* (These arguments are the same as 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 locks submitted by an honest replica is received by the next leader before $t + 2\Delta$. If the next leader is honest, he propose a block extending the highest certified block known to it and not appear any equivocation, then all honest replicas can vote for the proposal, and create a certificate for it. The maximum time lag 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, thus within $(2p + 1)\Delta$ time an honest leader propose at least p blocks. Therefore, all honest replicas cannot blame the leader and view-change will not happen, this prove the first part. If a byzantine replica do not propose p blocks within $(2p + 1)\Delta$, all honest replicas blame the leader and view-change happens, this prove 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 it and will be blamed. □

4.2 The Protocol for the Mobile Sluggish Model

Figure 8 shows the steady state protocol for the mobile sluggish model. The prevention of liveness attack is in the same way as for the standard synchronous model. As a minor modification 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, though it is not mentioned explicitly in [1]. The view-change protocol is the same as that of for the standard synchronous model.

4.2.1 Safety and Liveness

The safety of the original protocol for the mobile sluggish model 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 by the liveness attack. Also, it is not clear that our modification to prevent liveness cannot bring any security holes with which the safety

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

1. **Propose:** The leader L broadcasts $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}) \rangle_L$ 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 known to L.

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 highest certified block known to the replica and no equivocation in view v is 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 is 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 is observed so far, pre-commit B_k and broadcast $\langle \mathsf{commit}, B_k, v \rangle$.

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

Figure 8: The steady state protocol for the mobile sluggish model

is violated. For these reasons, we formally prove the safety. On the other hand, liveness is proved only under standard synchrony (recall that standard synchrony is a special case of the mobile sluggish model). Note however the original protocol is not guarantee the liveness even under standard synchrony by the liveness attack.

Lemma 5 (Certified without Equivocation)**.** *If an honest replica directly commits a 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 (ii) r do not vote any block B equivocating B_k in view v.*

Proof. The proof scheme is almost as same as that of the proof of Lemma 6 in [1]. Suppose an honest replica directly commit a block B_k in view v, then a quorum of replicas say R_0 pre-commit 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 ensure that a quorum of replicas vote 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 received $\mathcal{C}_v(B_k)$. If some replica in R quit the view before $t-\Delta$ or observe any equivocations, a quorum of blame messages or equivocating blocks are received by at least a prompt replica in R_0 before it commit the block, and it would prevent the commit, this prove the first part. Since no equivocation is observed before $t-\Delta$ by all replicas in R, they do not vote for the equivocating blocks in the same view, this prove 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) $\forall r \in R$, $lock(r, v')$ extend B_k and (ii) for all*

certified block B in view v', if $B >_{rank} B_k$ then B extend 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 blocks in view v do not conflict with B_k, thus all certified blocks in view v which is higher than B_k should extend B_k, which prove the second part. By the first part of Lemma 5, a quorum of honest replicas receive $\mathcal{C}_v(B_k)$ before quit view v and 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, they will submit its lock which extend B_k, which prove the second part. We next prove it for the inductive step. We prove it by contradiction. For the second part, suppose some block certified block B in view $v'+1$ does not extend B_k. Let B_{min} as the ascendant in view $v'+1$ with minimum rank, then $view(pred(B_{min})) \leq v'$ must be held. By the inductive hypothesis, $view(pred(B_{min})) < v$ must be held 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 proved for this model, B_{min} could not collect vote by replicas in R, it contradict that the block is certified. For the first part, suppose in some honest replica $r'' \in R$, $B = lock(r'', v'+1)$ does not extend B_k, then by the second part which is already proven, $view(B) < v$ must be held. Since an honest replica submit the highest certified block known to it, B would not be selected since at least B_k is higher than B and by the inductive hypothesis, $\forall r'' \in R$, $lock(r'', v')$ is higher than B_k , 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 vote from honest replica, thus it will not be certified, which prove the first part. The second part is 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 proved in the same as in the Theorem 1 by the Unique Extensibility Lemma which we 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, and (ii) A byzantine leader must propose a p blocks in $(2p+1)$ time to avoid a view-change, and (iii) If k is the highest height at which some honest replica has commited 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 for the standard synchronous model. □

4.3 The Protocol with Optimistic Responsiveness

Figure 9 and 10 show the modified protocol with responsive mode. In the steady state protocol, we add some minor modifications as in the mobile sluggish model, including (i) prevention of liveness attack: for the first proposal after the view-change, replica vote if the predecessor is higher than its lock submitted at the end of the previous view, and (ii) explicit

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

1. **Propose:** The leader L broadcasts $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1}) \rangle_L$ 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 known to L.

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 highest certified block known to the replica and no equivocation in view v is 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 $view(B_{k-2}) = v$ 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 is 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 observed so far, pre-commit B_{k-2} and broadcast $\langle \mathsf{commit}, B_{k-2}, v \rangle$.

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

Figure 9: The modified steady state protocol with responsive mode

description: in setting the timer or 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 view-change protocol is also a part of the prevention of liveness attack. The main one is to deal with a responsive commit. In the Lock phase, we add another waiting for 2Δ after receiving a quorum of blame2 messages. This ensure that when some honest replicas committed a block directly, it will be received by a quorum of honest replicas before submitting its lock.

4.3.1 Safety and Liveness

We formally prove the safety. However, liveness is not guaranteed under minority fault assumption since once after switching the responsive mode, adversary can violate the liveness by stopping the vote from faulty replicas. This is also applied to the original protocol and the liveness proof is also not done in [1].

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

Proof. Suppose an honest replica directly commit a block B_k in view v, then a quorum of replicas say R_0 pre-commit B_k. Let the earliest pre-commit among all honest replicas in R_0

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

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 the highest certified block known to the replica to L'

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

Figure 10: The view-change protocol with responsive mode

be performed by r at time t. Then, r must have voted for B_{k+2} before t. This ensure 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 received $\mathcal{C}_v(B_k)$. If some replica in R submit its lock before $t+\Delta$, the replica receives a quorum of blame2 messages before $t-\Delta$. This ensure that a quorum of replicas quit view before $t-\Delta$, and since at least one replica in the quorum set is prompt, prompt replicas at t receives $f+1$ blame messages. Since at least one replica in R_0 is prompt at t, it would prevent the pre-commit in the replica. □

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') extend B_k and (ii) for all certified block B in view v, if $B >_{rank} B_k$ then B extend B_k.*

Proof. These are the same arguments in the Lemma 6, therefore we only prove it for the responsive mode. We will prove it through the 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, and it prove the second part. We prove it in the same way as in the Lemma 8 in [1]. If B_k is directly committed in the responsive mode, there exits a strong certificate for B_k. Thus, there cannot exist a strong certificate for some block equivocating B_k, otherwise $f+1 = (3n/4+1) + (3n/4+1) - n$ replicas would voted for it. At least one ancestor of B_k should be committed synchronously in the responsive mode, at most d honest is left behind in the synchronous mode, this is not enough to create a certificate. The first part is clear by Lemma 8 and the second part which is already proven. The inductive step is almost as same as in the proof of Lemma 6, thus we abbreviate 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 certificate, this prove the first part. The second part is 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 proved in the same as in the 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 is necessary security properties for the state machine replication protocol. We also presented the modified protocols for each models with reasonable refinements and formally prove its security. Our modification remove its security vulnerability without any performance compromises.

As mentioned in [1] the mobile sluggish model is still far from the actual network, since an offline replica may not receive all the messages sent before Δ at the time it wake up. And also, with responsive mode, liveness failure can still happen even with our refinements. These are interesting 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)