Making Synchronous BFT Protocols Secure in the Presence of Mobile Sluggish Faults

Justin Kim
justin314kim@gmail.com
Montgomery High School

Vandan Mehta
vandan.mehta2000@gmail.com
Rutgers University

Kartik Nayak
kartik@cs.duke.edu
Duke University

Nibesh Shrestha
nxs4564@rit.edu
Rochester Institute of Technology

ABSTRACT

BFT protocols in the synchronous setting rely on a strong assumption: every message sent by a party will arrive at its destination within a known bounded time. To allow some degree of asynchrony while still tolerating a minority corruption, recently, in Crypto'19, a weaker synchrony assumption called *mobile sluggish* faults was introduced. In this work, we investigate the support for mobile sluggish faults in existing synchronous protocols such as Dfinity, Streamlet, Sync HotStuff, OptSync and the optimal latency BFT protocol. We identify key principles that can be used to "compile" these synchronous protocols to tolerate mobile sluggish faults.

1 INTRODUCTION

Byzantine fault tolerant (BFT) protocols relying on a network synchrony assumption can tolerate up to one-half Byzantine faults. However, the synchrony assumption may be too strong in practice; it requires every message sent by a replica to arrive at its destination within a known bounded delay Δ. In practice, this assumption may not hold all the time due to irregularities in the sender's or receiver's network. To tolerate such aberrations, one can rely on a partially synchronous or asynchronous network. However, under these network assumptions, consensus is impossible in the presence of more than one-third fraction of Byzantine faults.

In a recent work from Crypto'19, Guo, Pass, and Shi [7] presented a weakly synchronous model where the synchrony assumption holds for most of the network but allows for messages from a few replicas to be arbitrarily delayed. In particular, the model states that, at any time, a fraction of the replicas are honest and prompt, i.e., they respect the synchrony assumption. The remaining replicas can either be sluggish (their messages can be delayed) or Byzantine. Moreover, during the protocol execution, the sluggish replicas can be mobile. Thus, over the course of execution of the protocol, it is possible for every honest party to be sluggish at some point. A subsequent work, thus, referred to this model as the *mobile sluggish synchronous* model [2]. This model is stronger than partial synchrony or asynchrony; however, we have a better fault tolerance. As Guo et al. [7] show, we can have consensus protocols where the fraction of Byzantine and sluggish replicas together is up to one-half. The model is also a strict generalization of synchrony since synchrony requires there to be no sluggish replicas.

Subsequent to the work of Guo et al. [7], there have been multiple works presenting consensus protocols under a synchrony assumption as well as a version that tolerates mobile sluggish faults [2, 3, 5]. In these works, the techniques to make a synchronous protocol tolerant to mobile sluggish faults seem specific to the protocol itself. In this work, we ask,

What support do existing synchronous protocols have for mobile sluggish faults? Can we compile existing synchronous protocols to their mobile sluggish counterparts?

We address these questions by analyzing many of the existing synchronous protocols, namely, Dfinity [8], Streamlet [4], OptSync [9], Sync HotStuff [2] and the optimal latency BFT protocol [3], and presenting their mobile sluggish counterparts. We identify a common theme to make these protocols secure under the weaker synchrony model. Compiling *any* synchronous protocol to one under the weaker model still remains an open question.

2 MODEL AND NOTATIONS

We consider n replicas in a reliable, authenticated all-to-all network, where up to f replicas can be Byzantine or sluggish at any time t. Honest replicas that are not sluggish are prompt. Messages sent by sluggish replicas may suffer arbitrary delays while messages sent by prompt replicas will respect the synchrony bound. Specifically, if a replica r is prompt at time t, then any message sent by r at time $\leq t$ will arrive at a replica r' prompt at time t' if $t' \geq t + \Delta$. The set of sluggish replicas can arbitrarily change at every instant of time. We denote the number of sluggish replicas by d and Byzantine replicas by b such that $f = b + d$. Without loss of generality, we assume $n = 2f + 1$. Thus, at least $f + 1$ replicas are honest and prompt at any time. We assume standard digital signatures and public-key infrastructure (PKI). We use $\langle x \rangle_p$ to denote a signed message x by replica p and $H(x)$ to denote the invocation of the random oracle H on input x.

All of the protocols we consider make progress through a series of numbered *views.* A *view* is usually coordinated by a distinct leader where the leader proposes values in the form a block to make progress. Each block references its predecessor to form a block chain. We call a block's position in the chain as its height. A block B_k at height k has the format, $B_k := (b_k, H(B_{k-1}))$ where b_k denotes the proposed payload at height k, B_{k-1} is the block at height $k - 1$ and $H(B_{k-1})$ is the hash digest of B_{k-1}. A block B_k *extends* a block B_l ($k \geq l$) if B_l is an ancestor of B_k.

A block certificate on a block B_k consists of $f + 1$ distinct signatures in a view v and is represented by $C_v(B_k)$. Two blocks B_k and $B'_{k'}$ *equivocate* one another if they are not equal to *and* do not extend one another.

3 TOLERATING MOBILE SLUGGISH FAULTS: KEY IDEA

Synchronous protocols require all messages sent by every honest replica to arrive at its destination within a known bounded delay. At a high level, these messages are used by a replica to (i) learn the state of other replicas, or (ii) make deductions based on absence of messages within a specific time. Partially synchronous and asynchronous protocols rely only on the former; typically, a replica updates its state after receiving messages from a quorum of other replicas. The use of absence-of-messages crucially enables synchronous protocols to circumvent the lower bound by Dwork et al. [6].

In the presence of mobile sluggish faults, a key requirement, thus, is to enable communication between sluggish and prompt honest replicas. We make a simple observation: if a sluggish replica s receives a message m from a quorum of $f+1$ replicas, then within the next Δ time all honest and prompt replicas will receive the message m too (assuming m was sent in an all-to-all communication). This is because at least one of the $f + 1$ senders of m is honest and prompt at a time before s receives this quorum of messages. Assuming all-to-all communication is used by the protocol, all replicas will receive m within Δ time. Moreover, from the perspective of s, only one of the $f + 1$ messages is guaranteed to be from an honest and prompt replica; the remaining messages can potentially be from Byzantine or sluggish replicas.

In the context of protocols that we analyzed, the above generally breaks down to two simple rules. In order to commit a block B_k, a mobile-sluggish protocol needs to have: (i) the certificate for B_k, i.e., $C_v(B_k)$, should be buried deep enough and a replica needs to wait at least Δ time after receiving a sufficiently buried certificate, (ii) a replica needs to commit only after hearing from $f+1$ replicas stating that it has not received equivocations or equivocating certificates.

The first constraint ensures that at least one replica, say replica p, that voted for B_k is honest and prompt when a replica starts waiting (say, at time t). So, all prompt replicas will learn about B_k within next Δ time (i.e., by time $t + \Delta$). This prevents the prompt replicas from deciding on other conflicting values. How deep should a certificate be buried depends on the underlying synchronous consensus protocol. For example, protocols like Dfinity and Streamlet make decisions based on whether or not there are any equivocating certificates. In order for certificates to be propagated among the prompt replicas, $C_v(B_k)$ should be buried at least 2 deep, i.e., a replica needs to receive $C_{v'}(C_v(B_k))$ before it starts waiting. When the replica does not detect any equivocation or equivocating certificates, the replica makes a decision on B_k and we call this step as *pre-commit*.

Despite receiving a sufficiently deep certificate and waiting long enough, a sluggish replica may still not detect equivocation or equivocating certificates and commit on conflicting values. The second constraint prevents conflicting commits on such occasions. Waiting for confirmations from $f+1$ replicas ensures safety because either equivocation or equivocating certificates could not have missed all $f + 1$ replicas.

We note that Sync HotStuff [2] used similar ideas to tolerate mobile-sluggish faults. In this work, we abstract these ideas so they can be applied more generally to other protocols. Existing protocols like Dfinity [1, 8] and Streamlet [4] already meet the first criteria we presented. Thus, adding the second rule of waiting confirmations from $f + 1$ replicas can easily make these protocols tolerate mobile-sluggish faults. We present detailed protocols in the subsequent sections.

In general, the above rules suffice for all protocols that we analyzed. However, the first rule requires a party to wait for at least Δ time after burying $C_v(B_k)$. Protocols like OptSync [9] are designed to commit faster and decide as soon as a unique certificate for a value is formed (and achieve responsiveness). Modifying this protocol to meet above constraints does make the protocol secure in the presence of mobile-sluggish faults at the expense of slower fast commits. In Section 6, we explore an alternative direction to allow the protocol to commit responsively.

4 DFINITY UNDER MOBILE SLUGGISH FAULT MODEL

We now present Dfinity [1, 8] secure under mobile sluggish fault model using our guideline. In each view, every replica makes a proposal and waits for 2Δ time. A block B_k proposed by a replica p in view k has the following format: $B_k = \langle b_k, r, C_{k-1}(B_{k-1})\rangle_p$ where r is a unique verifiable rank obtained using Verifiable Random Functions (VRF). In Dfinity, a smaller rank is a better rank and the replica with the smallest rank is the leader of the view.

After the wait, each replica votes for the best ranked block(s) it has received so far. Whenever a replica votes for a block, it also forwards the block to all other replicas. If a replica is forwarded a block with a rank equal to or better than the best ranked block it has voted so far, it votes for the block. This process continues until a certificate is formed. As soon as an honest replica obtains a certificate, it broadcasts the certificate to all other replicas and enters the next view.

A replica pre-commits a block B_{k-2} (proposed in view $k - 2$) in view k and sends commit message for block B_{k-2} if all valid view $k - 1$ blocks it has seen so far extends a common block B_{k-2}. Observe that this condition was sufficient for Dfinity protocol to commit. However to handle mobile-sluggish failures, replicas only pre-commit with this condition. An honest replica commits when it receives $f + 1$ commit messages in the same view. Observe that Dfinity protocol decides based on whether an equivocating certificate exists or not. Thus, as per our guideline, we ensure there is a double certificate on block B_{k-2} and wait for $f + 1$ commit messages.

4.1 Safety and Liveness

Lemma 1 (Unique extensibility). *If an honest replica commits a block B_{k-2} in view k, then B_{k-2} is uniquely extensible.*

Proof. Suppose an honest replica p received $f+1$ $\langle \text{commit}, B_{k-2}, k\rangle$ messages for block B_{k-2} in view k and commits a block B_{k-2}. A set R of at least $d + 1$ honest replicas pre-committed B_{k-2} in view k.

Suppose for the sake of contradiction B_{k-2} is not uniquely extensible. Then at some point B'_{k-2} is extended by some block B'_{k-1}. For B'_{k-1} to be certified, at least one honest and prompt replica, say replica p', has to vote for B'_{k-1} in iteration $k - 1$. Let t be the time when replica p' votes for B'_{k-1} in view $k - 1$. However, if replica

Local state. A replica p keeps track of all valid iteration-k blocks in a set $\mathcal{B}_k$.

1. **Propose and wait.** Arbitrarily select a certified block $B_{k-1} \in \mathcal{B}_{k-1}$. Create $B_k := \langle v, r, C(B_{k-1})\rangle_p$ and broadcast it. Wait for 2Δ time.
2. **Vote.** Let B_k be the best ranked block in $\mathcal{B}_k$. If replica p has not voted for B_k, vote for B_k and forward it to all replicas. If multiple blocks tie for the best rank, vote for and forward all of them. Repeat this step until a certificate $C(B_k)$ for some B_k is received.
3. **Forward certificates.** Upon receiving a certificate $C(B_k)$, broadcast it and enter the next iteration.
4. **Pre-commit iteration** $(k-2)$**.** If all valid B_{k-1} have the same predecessor block B_{k-2}, pre-commit B_{k-2} and broadcast $\langle \text{commit}, B_{k-2}, k\rangle_p$ to all replicas. $\triangleright$ This step can be executed anytime after Step 1.
5. **Commit:** On receiving $\langle \text{commit}, B_k, k+2\rangle$ from $f+1$ replicas, commit B_k and all its ancestors.

Figure 1: Dfinity protocol under mobile-sluggish fault model.

p' votes for B'_{k-1} at time t in view $k-1$, at least $f+1$ honest and prompt replicas will receive B'_{k-1} at time $t+\Delta$ time before they pre-commit in view k. At least one of these prompt replicas belongs to set R and would not pre-commit. A contradiction. □

Theorem 2 (Safety). *Honest replicas always commit the same block B_k for each view k.*

Proof. Suppose an honest replica p commits a block B_k in view $j \geq k+2$ by committing block B_{j-2} and another honest replica p' commits B'_k in view $j' \geq k+2$ by committing $B_{j'-2}$. Due to the commit rule, B_{j-2} and $B_{j'-2}$ are both uniquely extensible (Lemma 1). If $j = j'$, B_{j-2} and $B_{j'-2}$ must be the same block (and extend the same B_k) in order for both to be uniquely extensible. Else, without loss of generality, assume $j' > j$. Since B_{j-2} is uniquely extensible, $B_{j'-2}$ extends B_{j-2}, and the two blocks extend the same B_k. □

Theorem 3 (Liveness). *If the leader of view k is honest and prompt, view-k block B_k is committed at the end of view $k+2$.*

Proof. Since an honest leader does not equivocate, the block proposed by the honest leader is uniquely extensible. If the leader of view k is honest, block B_k is uniquely extensible. Moreover, if the leader is prompt, $f+1$ honest and prompt replicas will vote for B_k and hence will be committed at the end of view $k+2$. □

5 STREAMLET UNDER MOBILE SLUGGISH FAULT MODEL

We now explain how to make Streamlet [4] secure under mobile sluggish fault model using our guideline (refer Figure 2). In each view e, a designated leader L_e proposes a block B_k by extending on a longest certified chain. Replicas vote on block B_k only if B_k extends the longest certified chain it has seen so far. An honest replica p pre-commits B_k and sends commit message for B_k if it has observed six consecutive blocks B_k though B_{k+5} are certified and does not detect certification of an equivocating block at these same 6 lengths. Observe that this condition was sufficient for Streamlet to commit block B_k. Since, Streamlet makes decision based on certification of equivocating block and it already makes decision on blocks that are six-block deep, as per our guideline, the protocol only needs to wait $f+1$ commit message before committing.

5.1 Safety and Liveness

Lemma 4. *If block B_k is certified in view e, then $f+1$ honest replicas must have received $C_v(B_{k-1})$ by the beginning of view $e+2$ where B_k extends B_{k-1}.*

Proof. Since B_k is certified in view e, at least one honest and prompt replica (say replica p) has voted for B_k in the vote phase in view e. Replica p's vote arrives a set of $f+1$ honest and prompt replicas by the beginning of view $e+1$ and receive $C_v(B_{k-1})$. Trivially, $f+1$ honest replicas receive $C_v(B_{k-1})$ by the the beginning of view $e+2$.

□

Lemma 5. *Suppose there is a certified chain in honest view containing 2 adjacent blocks B_k and B_{k+1} with consecutive view numbers e and $e+1$, respectively, then no block of height k on an view greater than $e+2$ can be certified in an honest view.*

Proof. By Lemma 4 , $f+1$ honest replicas have observed the predecessor block B_k of block B_{k+1} by the beginning of view $e+3$. Hence $f+1$ replicas will not vote for an equivocating block at height k or lower in an view $e+3$ or greater. Hence no block of length k or less and view number greater than $e+2$ can be certified in an honest view. □

Theorem 6 (Safety). *Suppose two certified chains containing $B_{k\ldots k+5}$ and $B'_{k'\ldots k'+5}$ both trigger commit rule, then it must be that one of these blocks must extends the other.*

Proof. Suppose for the sake of contradiction that the above is not true and two conflicting chains $B_{k\ldots k+5}$ and $B'_{k'\ldots k'+5}$ trigger commit rule i.e., $B_k \neq B'_k$ and $B_{k+1} \neq B'_{k+1}$. Suppose B_k through B_{k+5} were proposed in views e through $e+5$. Given than B_k trigger commit rule, a set R of at least $d+1$ honest replicas pre-commit B_k and all replicas in R observed certification of blocks B_k through B_{k+5} and no certification of equivocating blocks at height k through $k+5$.

Since $B'_{k'}$ is committed, both B'_k and B'_{k+1} must have been certified. By Lemma 5, no block of height k can be certified in view $e+3$ or greater. Thus, if block B'_{k+1} was certified, it must be in view $e' \leq e+2$. However, if B'_{k+1} was certified, by Lemma 4, $f+1$ honest replicas would have received a certificate for B'_k by view $e'+2$. At least one of them, say replica p belongs to set R. Thus, replica p would not pre-commit and B_k would not be committed. A contradiction. □

Theorem 7 (Liveness). *If there are 8 honest and prompt leaders in consecutive views $e, e+1, \ldots, e+7$, then a new block that was not committed before view e will be committed.*

Proof. Since honest leaders do not equivocate and there are $f+1$ honest and prompt replicas at any time, there will be a series

For every view e = 1, 2, . . . :

1. **Propose.** At the beginning of view e, view e's leader L does the following: The leader L broadcasts $\langle \text{propose}, B_k, e, C_v(B_{k-1})\rangle_L$ where B_k extends B_{k-1} and B_{k-1} is one of the longest certified chains L has seen so far.
2. **Vote.** During view e, every replica p does the following. Upon receiving the first valid proposal $\langle \text{propose}, B_k, e, C_v(B_{k-1})\rangle_L$ from view e's leader L, vote for the proposed block iff it extends from one of the longest certified chains it has seen at the time. To vote for the proposed block B_k, replica i simply broadcasts $\langle \text{vote}, e, B_k\rangle_p$.
3. **Pre-Commit.** On observing last 6 blocks with consecutive view numbers and moreover no conflicting block has been certified at these same 6 lengths, then, pre-commit the prefix of this chain removing the last 5 blocks. Let B_k be the prefix, then broadcast $\langle \text{commit}, e', B_k\rangle_p$.
4. **Commit.** On receiving $\langle \text{commit}, e', B_k\rangle$ from $f + 1$ distinct replicas, commit B_k and all its ancestors.

Figure 2: Streamlet under mobile-sluggish model.

of blocks proposed by leaders of view $e + 2$ through $e + 7$. Thus, block proposed by honest leader of view $e + 2$ will committed. □

6 OPTSYNC UNDER MOBILE SLUGGISH FAULT MODEL

OptSync [9] is an optimistically responsive synchronous protocol that commits at network speed when some *optimistic* conditions are met i.e, $\lfloor 3n/4 \rfloor + 1$ replicas behave honestly. It contains two distinct commit rules that exist simultaneously–(i) the optimistic commit rule that commits immediately when $\lfloor 3n/4 \rfloor + 1$ replicas vote for a block (ii) synchronous commit rule that commit within 2Δ from voting and detecting no equivocation. A set of $\lfloor 3n/4 \rfloor + 1$ votes for a block B_k forms a unique certificate , we call *responsive* certificate (denoted by $C_v^{3/4}(B_k)$). We use the notion of chain certificate and ranking introduced in OptSync for ranking chains. Due to space constraints, we refer readers to OptSync [9] for more details on chain certificates and chain ranking rules.

We can follow the guidelines presented in Section 3 to make OptSync mobile-sluggish secure at the expense of slower commits. In order to facilitate responsive commits, we first present an alternative direction, we call *two-blames technique*, to support responsive commits that helps in propagating the unique certificate among prompt replicas to ensure a conflicting certificate cannot be formed at some later point in time.

Two blames technique. First, the protocol is modified to commit when a set R of $f + 1$ replicas say they have the unique certificate. Second, the fallback protocol (e.g., *view-change* protocol) is modified to include two types of blame messages. When a replica receives $f + 1$ blame messages (blame certificate) , it forwards the blame certificate along with the unique certificate and sends second blame message (e.g., quit-view message in Figure 4). If a replica receives $f + 1$ quit-view (quit-view certificate) messages at time t, at least one of them, say replica p, should be prompt at time t. Since, replica p forwarded a blame certificate at time $\leq t$, prompt replicas at time $t + \Delta$ will receive the blame certificate. At least one of the prompt replicas at time $t + \Delta$ belongs to set R which broadcasts the unique certificate. Within next Δ time, $f + 1$ prompt replicas receive the unique certificate by time $t + 2\Delta$. The safety of the protocol relies on the fact that these $f + 1$ honest replicas that received the unique certificate do not vote for conflicting values.

Using the above guideline, we present mobile-sluggish secure version of OptSync is presented in Figures 3 and 4. In order to make the synchronous commit rule secure in the presence of mobile-sluggish faults, a replica waits for a double certificate for block B_k and waits for 2Δ time and no equivocation before pre-committing. Waiting for double certificate for block B_k ensures $f + 1$ honest replicas receive $C_v(B_k)$ before view-change is executed. Note that the optimistic commit rule commits immediately. As per our guideline, we modify the view-change protocol of OptSync with the two blames technique (refer Figure 4) to propagate the unique responsive certificate. In addition, the protocol also requires a replica to wait for $f + 1$ pre-commits before committing to prevent sluggish replicas from committing before hearing from majority replicas. Observe that the protocol has two different commit messages i.e., sync-commit for synchronous pre-commits and resp-commit for responsive pre-commits.

6.1 Safety and Liveness

Claim 8. *If a block B_k is pre-committed directly in view v using the responsive pre-commit rule, then a responsive certificate for an equivocating block $B'_{k'}$ does not exist in view v.*

Proof. If replica r pre-commits B_k due to the responsive pre-commit rule in view v then, r must have received $\lfloor 3n/4 \rfloor + 1$ votes, i.e., $C_v^{3/4}(B_k)$, forming a quorum for B_k in view v. A simple quorum intersection argument shows that a responsive certificate for equivocating block $B'_{k'}$ cannot exist. □

Claim 9. *If $d + 1$ honest replicas pre-commit B_k in view v using synchronous pre-commit rule, then (i) no equivocating block is certified in view v (ii) $f + 1$ honest replicas lock on a chain certificate CC such that $\text{tip}(CC)$ extends B_k before entering view $v + 1$.*

Proof. Suppose a set R of $d + 1$ honest replicas pre-commit a block B_k using synchronous pre-commit rule. Let replica $r \in R$ be the earliest replica that pre-committed at time t. Replica r must have received $f + 1$ proposals for B_{k+1} (which contains $C_v(B_k)$) by time $t - 2\Delta$. At least one of them, say replica r', is honest and prompt at time $t - 2\Delta$. Let R' be the set of $f + 1$ honest and prompt replicas at time $t - \Delta$. Every replica in R' receives the proposal for B_{k+1} by time $t - \Delta$.

Observe that no replica in R' will vote for equivocating blocks after time $t - \Delta$. If any replica in R' voted for an equivocating block before $t - \Delta$, its broadcast of the equivocating propose message will reach all honest replicas that are prompt at time t by time t. At least one replica in R would be prompt at time t. This replica would have detected an equivocation and would not pre-commit; a

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

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

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

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

(a) *Responsive.* If $\lfloor 3n/4 \rfloor + 1$ votes for B_k, i.e., $C_v^{3/4}(B_k)$ have been received, pre-commit B_k and broadcast $\langle \text{resp-commit}, B_k, v\rangle_r$.

(b) *Synchronous.* If commit-timer$_{v,k}$ reaches 0, pre-commit B_k and broadcast $\langle \text{sync-commit}, B_k, v\rangle$.

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

(a) *Responsive.* On receiving $f + 1$ resp-commit messages for B_k in view v, commit B_k and all its ancestors. Stop commit-timer$_{v,k}$.

(b) *Synchronous.* On receiving $f + 1$ sync-commit messages for B_k, commit B_k and all its ancestors.

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

- *Blame.* For $p > 0$, if fewer than p proposals trigger r's votes in $(2p+4)\Delta$ time in view v broadcast $\langle \text{blame}, v\rangle_r$. If leader equivocation is detected, broadcast $\langle \text{blame}, v\rangle_r$ along with the equivocating proposals.
- *Quit view on $f + 1$ blame messages.* Upon gathering $f + 1$ distinct blame messages, broadcast $\langle \text{quit-view}, v, CC\rangle$ along with $f + 1$ blame messages where CC is the highest ranked chain certificate known to r. Abort all view v timers, and stop voting in view v.

Figure 3: OptSync Steady state protocol under mobile sluggish model.

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

(1) **Status.** On receiving $f + 1$ quit-view messages, quit view v, set view-timer$_{v+1}$ to 2Δ and start counting down. When view-timer$_{v+1}$ expires, update its chain certificate CC to the highest possible rank. Set lock$_{v+1}$ to CC and send $\langle \text{status}, \text{lock}_{v+1}\rangle_r$ to L'. Enter view $v + 1$.

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

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

Figure 4: The view-change protocol with mobile sluggish faults

contradiction. Thus, an equivocating block will not get any vote from R' in view v and will not be certified in view v. This proves part (i) of the claim.

Again, observe that no replica in R' has quit view v by time $t - \Delta$. Otherwise, at least one replica in R would have seen $f + 1$ blame message and wouldn't pre-commit. Thus, every replica in R' receives $C_v(B_k)$ before quitting view v. By part (i) an equivocating certificate does not exist in view v. This implies, every replica in R locks on a chain certificate CC such that tip(CC) extends B_k before entering view $v + 1$. □

Claim 10. *If an honest replica commits a block B_k in view v using synchronous commit rule, then (i) no equivocating block is certified in view v (ii) $f + 1$ honest replicas lock on a chain certificate CC such that* tip(CC) *extends B_k before entering view $v + 1$.*

Proof. If an honest replica commits a block B_k in view v using synchronous commit rule, then at least $d + 1$ honest replicas pre-commit B_k in view v using synchronous pre-commit rule. The rest of the proof follows trivially from Claim 9. □

Claim 11. *If an honest replica commits a block B_k in view v using responsive commit rule, then there does not exist a chain certificate CC in view v, such that $CC > (C_v^{3/4}(B_k), \perp)$ where a block in CC equivocates B_k.*

Proof. Suppose replica r receives a set R of $f + 1$ resp-commit messages for block B_k in view v and commits using responsive commit rule. At least one of them is from an honest replica. By Claim 8, no equivocating block can have a responsive block certificate. So all responsive block certificates must extend B_k. Since we assume that $CC > (C_v^{3/4}(B_k), \perp)$ then it must be that either CC is of the form $(C_v^{3/4}(B_k), C_v^{1/2}(B_\ell))$ and by definition B_ℓ extends B_k, or CC is of the form $(C_v^{3/4}(B_{k'}), C_v^{1/2}(B_{\ell'}))$ where $B_{k'}$ extends B_k and again by transitivity $B_{\ell'}$ must extend B_k. □

Claim 12. *If an honest replica commits a block B_k using responsive commit rule in view v, then $f+1$ honest replicas lock a chain certificate CC such that* tip(CC) *extends B_k before entering view $v + 1$.*

Proof. Suppose replica r commits a block B_k using responsive commit rule in view v. Then, a set R of $d + 1$ honest replicas pre-commit B_k using responsive pre-commit rule in view v. Since, quit-view happens after a pre-commit, whenever replicas in R quit view v, they will send a chain certificate CC such that tip(CC) extends B_k.

Let replica r_1 be the earliest honest replica that enters view $v + 1$ at time t. Replica r_1 must have received quit-view message from $f + 1$ distinct replicas at time $t - 2\Delta$ (due to 2Δ wait during view-change). At least one of them, say replica r_2, must be honest and prompt at time $t - 2\Delta$. Replica r_2 forwards quit-view messages at time $t - 2\Delta$.

Denote the set of $f+1$ honest and prompt replicas at time $t-\Delta$ by R_1. Replicas in R_1 receive r_2's quit-view message by time t and broadcasts quit-view messages. At least one of the replicas in R must be prompt at time $t-\Delta$. This replica sends a quit-view message containing CC. Again, denote the set of $f+1$ honest and prompt replicas at time t by R_2. The quit-view message containing CC arrives replicas in R_2 by time t. We now prove that set R_2 is the required set that satisfies the claim. Since, no honest replicas entered a higher view using synchronous quit view rule, replicas in R_2 will receive and lock CC before entering view $v+1$. □

Lemma 13. *If a block B_k is committed in view v, $f+1$ honest replicas lock on a chain certificate CC such that $\mathsf{tip}(CC)$ extends B_k before entering view $v+1$.*

Proof. Straight forward from Claim 10 part (ii) and Claim 12. □

Lemma 14 (Unique Extensibility). *If an honest replica directly commits a block B_k in view v, and $C_{v'}(B_{k'})$ is a view $v' > v$ block certificate, then $B_{k'}$ extends B_k. Moreover, all honest replicas have $\mathsf{lock}_{v'}$ such that $\mathsf{tip}(\mathsf{lock}_{v+1})$ extends B_k.*

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

For the base case, where $v' = v+1$, the proof that $C_{v'}(\mathsf{tip}(\mathsf{lock}'))$ extends B_k follows from Lemma 13 because the only way such a block can be certified is some honest votes for it. However, all honest replicas are locked on a block that extends B_k and a chain certificate with a higher rank for an equivocating block does not exist. Thus, no honest replica will first vote for a block that does not extend B_k. The second part follows directly from Lemma 13.

Given that the statement is true for all views below v', the proof that $C_{v'}(\mathsf{tip}(\mathsf{lock}'))$ extends B_k follows from the induction hypothesis because the only way such a block can be certified is if some honest votes for it. An honest party with a lock lock will vote only if $\mathsf{tip}(\mathsf{lock}_{v'})$ has a valid block certificate and lock $\geq \mathsf{lock}_{v'}$. Due to Lemma 13 and the induction hypothesis on all block certificates of view $v < v'' < v'$ is must be that $C_{v'}(\mathsf{tip}(\mathsf{lock}))$ extends B_k. □

Theorem 15 (Safety). *Honest replicas do not commit conflicting blocks for any height ℓ.*

Proof. Suppose for contradiction that two distinct blocks B_ℓ and B'_ℓ are committed at height ℓ. Suppose B_ℓ is committed as a result of B_k being directly committed in view v and B'_ℓ is committed as a result of $B'_{k'}$ being directly committed in view v'. This implies B_k extends B_ℓ and $B'_{k'}$ extends B'_ℓ. Without loss of generality, assume $v \leq v'$; if $v = v'$, further assume $k \leq k'$. If $v = v'$ and $k \leq k'$, by Claim 11 and Claim 10, $B'_{k'}$ extends B_k. Similarly, if $v < v'$, by Lemma 14, $B'_{k'}$ extends B_k. Thus, $B'_\ell = B_\ell$. □

7 EXISTING MOBILE SLUGGISH PROTOCOLS

7.1 Sync HotStuff

Sync HotStuff [2] presents a simple synchronous consensus protocol. In Sync HotStuff, upon receiving a block B_k, a replica votes for the block and waits for 2Δ time. If no equivocation has been detected during the wait, the replica commits B_k immediately. In the same work, a mobile sluggish version of the protocol was presented. In mobile-sluggish version, a replica commits a block B_{k-2} upon receiving a proposal for block B_k, waiting for 2Δ time and receiving $f+1$ commit messages. Essentially, this implies a replica waits for two consecutive certificates on a block B_{k-2} and waits for 2Δ time and $f+1$ commit messages. Hence, mobile-sluggish Sync HotStuff follows our guideline. In addition, this strengthens our guideline.

7.2 Optimal Latency BFT under Mobile Sluggish Fault Model

Optimal latency BFT protocol [3] presents an optimal latency synchrous BFT protocol that commits in $\Delta + O(\delta)$ time. At a high level, the protocol is as follows: On receiving a block proposal B_k, a replica sends an ack message and waits for Δ time. After Δ wait, if the replica does not hear equivocation, it votes for B_k. Voting for B_k after sending an ack message and Δ wait ensure uniqueness of certificate for block B_k. On receiving a certificate for B_k, the replica commits immediately.

The same paper presents mobile-sluggish version of the optimal latency BFT. The synchronous protocol is modified as follows. First the replica waits for $f+1$ ack messages for block B_k and waits for Δ time. After Δ wait, if the replica does not hear an equivocation, it votes for B_k. On receiving a certificate for B_k, the replica pre-commits and broadcasts commit messages. A replica commits immediately if it receives $f+1$ commit messages.

Observe that a replica commits immediately on receiving $f+1$ commit messages and does not wait for propagation of the unique certificate for B_k. As a result, the view-change phase of this protocol requires two blames technique to ensure the unique certificate for B_k is received by $f+1$ honest replicas. This protocol follows our guideline.

REFERENCES

[1] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, and Ling Ren. 2018. Dfinity Consensus, Explored. *IACR Cryptol. ePrint Arch.* 2018 (2018), 1153.

[2] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. 2020. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 106–118.

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

[4] Benjamin Y Chan and Elaine Shi. 2020. Streamlet: Textbook streamlined blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 1–11.

[5] TH Hubert Chan, Rafael Pass, and Elaine Shi. [n.d.]. *Pili: A simple, fast, and robust family of blockchain protocols.* Technical Report. Cryptology ePrint Archive, Report 2018/980, 2018. https://eprint. iacr. org ....

[6] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* 35, 2 (April 1988), 288–323. https://doi.org/10.1145/42282.42283

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

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

[9] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. 2020. On the Optimality of Optimistic Responsiveness. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 839–857.