Brief Announcement: 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.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**.

KEYWORDS

Distributed computing; Byzantine Fault Tolerance; Weak Synchrony

ACM Reference Format:
Justin Kim, Vandan Mehta, Kartik Nayak, and Nibesh Shrestha. 2021. Brief Announcement: Making Synchronous BFT Protocols Secure in the Presence of Mobile Sluggish Faults. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC '21), July 26–30, 2021, Virtual Event, Italy.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3465084.3467954

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.

PODC '21, July 26–30, 2021, Virtual Event, Italy

ACM ISBN 978-1-4503-8548-0/21/07.
https://doi.org/10.1145/3465084.3467954

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 [10], 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. Due to space constraints, we present detailed protocols in the full version [9].

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 [10] 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 4, we explore an alternative direction to allow the protocol to commit responsively.

4 OPTSYNC UNDER MOBILE SLUGGISH FAULT MODEL

OptSync [10] 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 [10] 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.

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 $\langle \text{quit-view}, v, CC\rangle_r$ 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 1: 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 2: The view-change protocol with mobile sluggish faults

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 2). 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 1 and 2. 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 2) 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. Due to space constraints, we present detailed proofs in the full version [9].

Acknowledgment. We would like to thank anonymous reviewers for their feedback. Vandan has been supported by NSF award CCF 1910565.

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] T-H Hubert Chan, Rafael Pass, and Elaine Shi. 2018. PiLi: An Extremely Simple Synchronous Blockchain. *IACR Cryptol. ePrint Arch.* 2018 (2018), 980.

[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] Justin Kim, Vandan Mehta, Kartik Nayak, and Nibesh Shrestha. 2021. Making Synchronous BFT Protocols Secure in the Presence of Mobile Sluggish Faults. *IACR Cryptology ePrint Archive* 2021 (2021), 603.

[10] 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.