Multi-Threshold Byzantine Fault Tolerance

Atsuki Momose
Intelligent Systems Laboratory, SECOM CO., LTD.
Mitaka, Tokyo, Japan
Nagoya University
Nagoya, Aichi, Japan
atsuki.momose@gmail.com

Ling Ren
University of Illinois at Urbana-Champaign
Urbana, IL, USA
renling@illinois.edu

ABSTRACT

Classic Byzantine fault tolerant (BFT) protocols are designed for a specific timing model, most often one of the following: synchronous, asynchronous or partially synchronous. It is well known that the timing model and fault tolerance threshold present inherent trade-offs. Synchronous protocols tolerate up to $n/2$ Byzantine faults, while asynchronous or partially synchronous protocols tolerate only up to $n/3$ Byzantine faults. In this work, we generalize the fault thresholds of BFT and introduce a new problem called multi-threshold BFT. Multi-threshold BFT has four separate fault thresholds for safety and liveness under synchrony and asynchrony (or partial-synchrony), respectively. Decomposing the fault thresholds in this way allows us to design protocols that provide meaningful fault tolerance under both synchrony and asynchrony (or partial synchrony). We establish tight fault thresholds bounds for multi-threshold BFT and present protocols achieving them. As an example, we show a BFT state machine replication (SMR) protocol that tolerates up to $2n/3$ faults for safety under synchrony while tolerating up to $n/3$ faults for other scenarios (liveness under synchrony as well as safety and liveness under partial synchrony). This is strictly stronger than classic partially synchronous SMR protocols. We also present a general framework to transform known partially synchronous or asynchronous BFT SMR protocols to additionally enjoy the optimal $2n/3$ fault tolerance for safety under synchrony.

CCS CONCEPTS

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

KEYWORDS

Distributed Systems; Byzantine Fault Tolerance; Blockchain

ACM Reference Format:
Atsuki Momose and Ling Ren. 2021. Multi-Threshold Byzantine Fault Tolerance. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3460120.3484554

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea.

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484554

1 INTRODUCTION

Byzantine fault-tolerance (BFT) is a fundamental problem in distributed systems [34, 44, 45]. It also serves as the algorithmic foundation of blockchain [40], which replicates a ledger across mutually distrusting organizations. Designing efficient BFT protocols is of both theoretical and practical interests today.

The classic BFT protocol design first selects a timing assumption, usually from the three models below: synchrony, asynchrony, or partially synchrony. It is well known that there is an inherent trade-off between the timing model and the fault threshold. Synchronous BFT protocols [20, 45] tolerate up to $f < n$ or $f < n/2$ Byzantine faults (depending on the variant of the problem) but break down when the network is not synchronous. On the flip side, asynchronous or partially synchronous protocols tolerate network asynchrony but tolerate only up to $f < n/3$ Byzantine faults [22]. This motivates the following natural question:

Can we design BFT protocols that enjoy more than one-third fault-tolerance under synchrony while at the same time tolerating some (ideally one-third) faults in asynchrony or partial synchrony?

This question was recently studied and partially answered in a series of elegant works by Blum et al. [7–9]. They showed it is possible for a BFT protocol to simultaneously tolerate $f_a < n/3$ faults under asynchrony and $f_a \leq f_s < n/2$ faults under synchrony, if and only if $2f_s + f_a < n$. Clearly, the bound $2f_s + f_a < n$ implies that f_s and f_a are always lower than what is achievable in a single timing model, i.e., $f_s < n/2$ for synchrony and $f_a < n/3$ for asynchrony/partial synchrony. In other words, their protocol provides an interesting trade-off but cannot strictly improve classic single-model protocols. And they showed that this is inherent with the standard BFT definition.

In this work, we further generalize the BFT problem to circumvent the above barrier. We separate the fault tolerance thresholds for each timing model and for *safety* and *liveness*, two well-established properties of distributed algorithms. This generalization gives us a new insight on BFT: it is possible to improve safety under synchrony while preserving the other fault thresholds, i.e., liveness under synchrony and safety/liveness under asynchrony (or partial synchrony). At the technical level, we combine techniques from state-of-the-art synchronous and asynchronous (or partially synchronous) protocols. The outcome is a class of simple and efficient solutions with optimal fault tolerance that are *strictly stronger* than classic single-model protocols. More specifically, we show a protocol tolerating $2n/3$ faults for safety under synchrony while tolerating $n/3$ faults for liveness under synchrony and safety/liveness under asynchrony (or partial synchrony). In comparison, existing asynchronous (or partially synchronous) protocols tolerate $n/3$ faults for both safety

and liveness, under both synchrony and asynchrony (or partial synchrony).

This higher safety under synchrony can be very useful as it helps disincentivize rational players from attacking the system. If they do not have control over the network infrastructure, they would need to persuade a very large fraction of nodes to collude with them in order to succeed in breaking safety.

Multi-threshold BFT. We introduce multi-threshold BFT (or MT-BFT for short), a generalized notion of the BFT where the fault tolerance thresholds are defined separately for safety and liveness under synchrony and asynchrony (or partial synchrony). The MT-BFT protocol is parameterized by four thresholds $\beta_a, \gamma_a, \beta_s, \gamma_s$. The protocol achieves safety in the presence of β_a faults and liveness in the presence of γ_a faults in an asynchronous (or partially synchronous) network; at the same time, the protocol achieves safety in the presence of β_s faults and liveness in the presence of γ_s faults in a synchronous network.

It is worth noting that the problem considered in Blum et al. can be viewed as a special form of the more general problem we introduce here, with the restriction of $\beta_a = \gamma_a = f_a$ and $\beta_s = \gamma_s = f_s$.

Tight fault tolerance bounds of MT-BFT RBC. To motivate the MT-BFT model, we first consider reliable broadcast (RBC), which is a simple and standard variant of consensus problems. We show that MT-BFT RBC can be achieved if and only if $2\gamma_s + \beta_a < n$. Namely, there is an inherent trade-off between the asynchronous (or partially synchronous) safety threshold β_a and the synchronous liveness threshold γ_s. This result is already very interesting. Notice that there is no trade-off between the asynchronous (or partially synchronous) safety (i.e., β_a) and the synchronous safety (i.e., β_s). Thus, it means that we can design a BFT protocol that enjoys an arbitrary high (even $\beta_s = n-1$) synchronous safety while preserving the fault tolerance of classic protocols in asynchrony (or partially synchrony) with $\beta_a = \gamma_a < n/3$.

Tight fault tolerance bounds of MT-BFT SMR. Next, we apply the idea to state machine replication (SMR), which is a practical formulation of the consensus problem (and it is the interface blockchains provide). Notably, the fault tolerance thresholds for BFT SMR differ slightly from the previous ones for RBC because SMR protocols must provide public verifiability, i.e., external clients can verify the correctness of the committed ledger. We show that with this public (external) verifiability requirement, an additional constraint comes in: we need $\beta_s + \gamma_s < n$.

This extra condition means that BFT SMR cannot tolerate arbitrarily high fault for synchronous safety. This is not surprising as it is well known BFT SMR can tolerate at most $f < n/2$ faults [45], unlike Byzantine broadcast which does not require public verifiability and may tolerate an arbitrarily number of faults. As easily seen, this $f < n/2$ bound for SMR is a special form of our new bound with the restriction of $\beta_s = \gamma_s = f$.

Nonetheless, one can still achieve a safety threshold of up to $\beta_s < n - \gamma_s$ while preserving the partial synchronous fault tolerance. We present a simple MT-BFT SMR protocol that can achieve optimal thresholds in the dual timing model of synchrony and partial synchrony. An interesting point in the design space is to achieve $\beta_s < 2n/3$ tolerance for safety under synchrony while preserving the tolerance for partial synchrony and liveness at $\beta_a = \gamma_a = \gamma_s < n/3$. This is strictly stronger than classic asynchronous or partially synchronous protocols.

In addition, our protocol allows tuning parameter on the optimal trade-off curve to best suit the application. For example, one can prioritize safety (e.g., $\beta_a < n/2$, $\beta_s < 3n/4$) at the cost of liveness (e.g., $\gamma_a = \gamma_s < n/4$) for safety critical applications. One can also prioritize the synchronous setting (e.g., $\gamma_s < 9n/20$, $\beta_s < 11n/20$) while tolerating a small number of faults (e.g., $\beta_a = \gamma_a < n/10$) anticipating occasional network failure.

A framework to upgrade to optimal synchronous safety. After giving customized new protocols above, we devise a general framework to transform existing protocols to our MT-BFT paradigm with optimal synchronous safety. To elaborate, the framework converts any MT-BFT SMR protocol parameterized by $(\beta'_a, \gamma'_a, \beta'_s, \gamma'_s)$ into a MT-BFT SMR protocol with optimal synchronous safety $\beta_s = n - \gamma'_s - 1$ tolerating the same thresholds otherwise, i.e., $\beta_a = \beta'_a$, $\gamma_a = \gamma'_a$, and $\gamma_s = \gamma'_s$. Existing partially synchronous, e.g., PBFT [14], HotStuff [47], and asynchronous protocols, e.g., HoneyBadgerBFT [38], BEAT [21], Dumbo [26], can be viewed as MT-BFT SMR protocols with $\beta'_a = \gamma'_a = \beta'_s = \gamma'_s < n/3$. Hence, they can be upgraded to tolerate an optimal $\beta_s < 2n/3$ while preserving other thresholds $\beta_a = \gamma_a = \gamma_s < n/3$. The framework requires only two communication steps and a synchronous waiting step in addition to running the underlying protocol. This helps improve the resilience of these existing protocols in more versatile scenarios with minimum overhead.

Summary of contributions. In summary, this paper provides the following results.

(1) We introduce multi-threshold BFT (MT-BFT), a generalized version of the BFT problem with separate thresholds for safety and liveness under synchrony and asynchrony (or partial synchrony) (Section 2.2).
(2) We establish tight bounds on the fault tolerance thresholds for MT-BFT reliable broadcast (Section 3) and state machine replication (Section 4).
(3) We present a framework to convert existing partially synchronous or asynchronous BFT SMR protocols to additionally enjoy optimal synchronous safety fault tolerance (Section 5).

2 MULTI-THRESHOLD BFT

2.1 Preliminaries

Reliable broadcast (RBC). In reliable broadcast (RBC), a designated sender r_s looks to broadcast an input value b_{in} to a set of n replicas, and each replica outputs a value. A RBC protocol needs to achieve the following safety and liveness properties.

(1) **Safety.**
   (a) *Consistency.* If two honest replicas output values b and b', respectively, then $b = b'$.
   (b) *Integrity.* If the designated sender is honest, no honest replica outputs a value $b \neq b_{in}$.
(2) **Liveness.**
   (a) *Validity.* If the designated sender is honest, then all honest replicas output some value.

(b) *Totality.* If an honest replica commits a value, all honest replicas output some value.

We remark that the standard RBC validity property says "if the designated sender is honest, all honest replicas output the sender's value". It has both safety and liveness components, so we separate it into integrity and validity following [17]. While this separation may look verbose at first glance, it follows the convention that safety captures "nothing bad happens" and liveness captures "something happens".

State machine replication (SMR). The problem we are more interested in is the state machine replication (SMR) problem [14, 45]. A SMR protocol uses a number of servers, called replicas, to provide an abstraction of a single non-faulty server . A SMR protocol orders transactions from clients into a totally ordered list that grows in length, called a log. Replicas and clients repeatedly output new transactions at increasing positions of the log. Since a SMR protocol ultimately services clients, it needs to provide *public verifiability*. Namely, there is a predefined Boolean function Verify; a replica or a client outputs a log of transactions $\log = [tx_0, tx_1, ...tx_j]$ if and only if there is a publicly verifiable proof π such that $\text{Verify}(\log, \pi) = 1$. A SMR protocol then provides the following safety and liveness:

(1) **Safety.** If $[tx_0, tx_1, ..., tx_j]$ and $[tx'_0, tx'_1, ..., tx'_{j'}]$ are output by two honest replicas or clients, then $tx_i = tx'_i$ for all $i \leq \min(j, j')$.
(2) **Liveness.** If a transaction tx is input to at least one honest replica, then every honest replica eventually outputs a log containing tx.

Although some prior works [2, 47] do not explicitly mention the public verifiability property in their SMR definitions, they all implicitly achieve it. A typical way to achieve it is to have a client collect signatures on the log from $f + 1$ replicas, which serve as a publicly verifiable proof for the log.

It is easy to see that a SMR protocol solves RBC by outputting a transaction signed by the sender that resides at the smallest log height. We will elaborate on this in Section 4.1. Therefore, an impossibility result for RBC applies to SMR as well.

Timing model. The three most common network timing models in distributed computing are synchrony, asynchrony, and partially synchrony. In a synchronous network, every message sent by an honest replica will be received by the recipient within a known upper bound Δ. If there is no such bound on the communication delay, the network is said to be asynchronous. A partially synchronous network has both synchronous and asynchronous periods. For convenience, it is usually assumed that the network is asynchronous at first, but becomes synchronous after an unknown time called global stabilization time (GST) denoted T_g [22].

Even in the synchronous model, our protocols do not assume any synchronized clocks across replicas, which is commonly assumed in lock-step synchronous protocols such as Blum et al. where all replicas do each operation at the same time. We only assume that each replica's locally measured Δ time is a correct upper bound for the network delay.

Fault model. We assume Byzantine faults that can behave arbitrary. All protocols presented in this paper tolerate adaptive corruption that can happen anytime during the protocol execution. A replica that is not faulty throughout the execution is said to be honest and faithfully execute the protocol.

Other assumptions. We assume the use of digital signatures and public-key infrastructure (PKI) for the set of replicas, i.e., the public-keys of all replicas are known to all replicas and clients. We use the notation $\langle x \rangle_r$ to denote a message x signed by a replica r. We also assume a cryptographic hash function H. As is commonly done in BFT protocols, we abstract away the details of cryptography and assume they are ideal.

2.2 Multi-Threshold BFT

We introduce multi-threshold BFT (MT-BFT), a generalized definition of the BFT problem that separates the fault tolerance for safety and liveness under synchrony and asynchrony (or partially synchrony), and capture the trade-offs between them. To elaborate, a multi-threshold BFT protocol is parameterized by four thresholds $0 < \beta_a, \gamma_a, \beta_s, \gamma_s < n$ and achieves the following two guarantees simultaneously.

(1) Safety in the presence of β_a faults and liveness in the presence of γ_a faults in an asynchronous (or partially synchronous) network.
(2) Safety in the presence of β_s faults and liveness in the presence of γ_s faults in a synchronous network.

Note that the synchronous model is clearly a stronger assumption (hence easier problem) than the asynchronous (or partially synchronous) model. Thus, we always have $\beta_a \leq \beta_s$ and $\gamma_a \leq \gamma_s$. We also note that this paper focuses on the $\gamma \leq \beta$ case. This means safety is considered more important than liveness as it takes more adversarial nodes to break safety than to break liveness. This is consistent with the philosophy of partial synchrony: safety is always maintained whereas liveness is achieved only under good conditions. We remark that the case $\gamma > \beta$ means replicas are allowed to make progress in a unsafe manner (e.g., output conflicting values). It may make sense if the protocol has some notion of "recovering" capability where unsafe decisions are eventually resolved, but we have not seen a clear formalization for such a notion and we leave it as future work.

When we consider asynchrony together with synchrony, we refer to the dual timing model as the async-sync model; likewise, when we consider partial synchrony together with synchrony, we refer to the dual timing model as the psync-sync model. Existing asynchronous [26, 38] (or partially synchronous [14, 47]) BFT protocols are already MT-BFT protocols in the async-sync (or psync-sync) model with $\beta_a = \gamma_a = \beta_s = \gamma_s < n/3$.

3 TIGHT FAULT TOLERANCE BOUND OF MT-BFT RELIABLE BROADCAST

This section establishes a tight bound on the fault thresholds of MT-BFT for reliable broadcast (RBC). As mentioned before, the impossibility result below applies to SMR as SMR implies RBC.

3.1 Fault Tolerance Limit of MT-BFT RBC

We first show that any MT-BFT protocol has an inherent trade-off between the asynchronous safety and the synchronous liveness formulated as $\beta_a + 2\gamma_s < n$. Although the proof is a straightforward

extension of the Blum et al. [7, 8] and a classic proof by Dwork et al. [22], we show the proof in detail to showcase the roles of the different thresholds in the proof. To strengthen the result, we prove the lower bound in the psync-sync model.

Theorem 3.1. *There does not exist a MT-BFT RBC protocol such that for a certain $n > 0$, its threshold parameters satisfy $\beta_a + 2\gamma_s \geq n$.*

Proof. Suppose for the sake of contradiction that there exists a MT-BFT RBC protocol whose fault threshold parameters satisfy $\beta_a + 2\gamma_s = n$ (the proof can be easily extended to $\beta_a + 2\gamma_s > n$). We consider a network with three partitions P, Q, and R, with sizes $|P| = |R| = \gamma_s > 0$ and $|Q| = \beta_a > 0$. The designated sender r_s is in Q. Consider the three executions below.

In the first execution (W1), the network is synchronous and all messages are instantly delivered. All replicas in P crash, i.e., do not send any message to other replicas. The sender r_s has an input value b_1. Since the protocol achieves liveness in the presence of γ_s faults, all replicas in R output b_1 at some time T_1.

The second execution (W2) is symmetric to the first one. The network is synchronous, all messages are instantly delivered, and all replicas in R crash. The sender r_s has an input value $b_2 \neq b_1$. Since the protocol achieves liveness in the presence of γ_s faults, all replicas in P output b_2 at some time T_2.

In the third execution (W3), the network is partially synchronous and $T_g > \max\{T_1, T_2\}$. All replicas in Q are Byzantine. Q behave towards R and P as in W1 and W2, respectively. All messages between P and R are delayed by T_g, but all other messages are instantly delivered. Then, replicas in R cannot distinguish W1 and W3 by $T_1 < T_g$, and they output b_1 as in W1. Similarly, replicas in P cannot distinguish W2 and W3 by $T_2 < T_g$, and they they output $b_2 \neq b_1$ as in W2. This violates the supposition that the protocol achieve consistency against $|Q| = \beta_a$ Byzantine faults. □

As mentioned, the bound above is a straightforward generalization of the bound proven by Blum et al [7, 8], which shows no protocol can tolerate f_a Byzantine faults under asynchrony (i.e., $f_a = \beta_a = \gamma_a$) and f_s Byzantine faults under synchrony (i.e., $f_s = \beta_s = \gamma_s$) if $f_a + 2f_s \geq n$. But once we separate fault tolerance for safety and liveness, an interesting and crucial observation is that β_a and β_s do not constrain each other. This means we may achieve a higher synchronous safety tolerance independent of the asynchronous safety tolerance.

3.2 A MT-BFT RBC Protocol with Optimal Fault Tolerance

We present a MT-BFT RBC protocol with optimal fault tolerance in the async-sync model. The protocol supports any parameter choices within the feasible region. Namely, a protocol designer can first pick $\gamma_s < n/2$. Then, other parameters are determined as

(1) $\beta_a = n - 2\gamma_s - 1$
(2) $\beta_s = n - 1$
(3) $\gamma_a = \min\{\beta_a, \gamma_s\}$

This is optimal given Theorem 3.1 and also establishes the tightness of Theorem 3.1.

Protocol description and intuition. The protocol is given in Figure 1. The protocol follows the common quorum-based design. An available quorum of honest replicas vote for a proposal b from the sender, forming a quorum-certificate $C(b)$. As our protocol requires a quorum availability under both synchrony and asynchrony, we naturally use a quorum of $|C| = n - \gamma_s$ (note that $\gamma_s \geq \gamma_a$). This easily achieves validity property of the protocol. The integrity property is also easily achieved by checking a sender's signature on the value. The core of the protocol combines respective techniques of purely asynchronous and purely synchronous protocols and make them work with the optimal thresholds above to achieve consistency and totality. We elaborate more in detail below.

Asynchronous quorum intersection. Existing asynchronous protocols rely on a quorum intersection argument to achieve consistency. In short, two quorums of $2f + 1$ replicas out of $n = 3f + 1$ intersect at at least $f + 1$ replicas. This rules out conflicting quorum certificates. Then, totality is achieved by having replicas forward certificates to make other replicas output the same value.

As can be expected from the proof, the quorum intersection argument works in the optimal thresholds $\beta_a = n - 2\gamma_s - 1$. Two quorums of $n - \gamma_s$ replicas intersect at $2(n - \gamma_s) - n = \beta_a + 1$ replicas, ruling out conflicting certificates. This guarantees both consistency and totality under asynchrony.

Synchronous equivocation-checking. The common approach for synchronous BFT protocols to achieve consistency under minority corruption is equivocation-checking. In short, replicas, before outputting, forward the sender's proposal and wait for Δ to rule out sender equivocation [4]. Our protocol builds on this technique to achieve consistency. But we need a little tweak for liveness. A simple equivocation check achieves consistency because a replica stops outputting if it detects equivocation. A faulty sender can exploit this to break liveness (in particular totality): after some honest repica outputs a value, the sender rushes to send a conflicting proposal to another honest replica to stop it from outputting. To prevent this attack, we perform equivocation check on both proposals and certificates. We first perform equivocation check on proposal to rule out conflicting votes among honest replicas. Then, after receiving a certificate, we perform equivocation check on certificates to rule out conflicting decisions. The latter easily achieves consistency with an arbitrary $\beta_s < n$ faults even if there are conflicting certificates as replicas just stop outputting. When there are fewer than γ_s faults, we can additionally show that conflicting certificates cannot exist. Note that a quorum of $n - \gamma_s$ replicas have at least an honest replica (because $\gamma_s < n/2$). Hence, if an honest replica outputs a value, no honest replica votes for other values. Faulty replicas cannot stop honest replicas from outputting. This ensures totality with γ_s faults.

Correctness of the protocol. We prove safety and liveness under synchrony and asychrony under respective fault thresholds.

Lemma 3.2 (Safety). *If the network is synchronous and there are at most $\beta_s = n - 1$ faults, then safety holds.*

Proof. Since a replica outputs a value b only if it receives $\langle \text{propose}, b \rangle_{r_s}$ signed by the sender, integrity holds. We next prove consistency. Suppose for the sake of contradiction that honest replicas output two different values b and b'. Let r and r' be the first

Each replica r runs the following steps to output a value.

(1) **Propose.** At the beginning of the execution, if $r = r_s$, broadcasts its input value b_{in} in the form of $\langle \text{propose}, b_{in} \rangle_r$.

(2) **Vote.** Upon receiving the first $\langle \text{propose}, b \rangle_{r_s}$, broadcast it and wait for Δ. Then, if it has not received $\langle \text{propose}, b' \rangle_{r_s}$ for $b \neq b$, broadcast $\langle \text{vote}, b \rangle_r$.

(3) **Output.** Upon receiving a quorum of $n - \gamma_s$ $\langle \text{vote}, b \rangle_*$ denoted $C(b)$, broadcast them and wait for Δ. Then, if it has not received $C(b')$ for a different value $b' \neq b$ and it has received $\langle \text{propose}, b \rangle_{r_s}$, output the value b.

Figure 1: An optimal multi-threshold BFT RBC protocol.

honest replicas that have voted for b and b' at time t and t', respectively. Without loss of generality, we assume $t \leq t'$. As r outputs b at t, it must have received and broadcast $C(b)$ at $t - \Delta$. Then, all honest replicas including r' must have received it by $t \leq t'$. Therefore, r' could not have outputted $b' \neq b$ at t', a contradiction. □

Lemma 3.3 (Liveness). *If the network is synchronous and there are at most γ_s faults, then liveness holds.*

Proof. We first prove that, if two certificates $C(b)$ and $C(b')$ are both formed, then $b = b'$. Suppose for the sake of contradiction that $C(b)$ and $C(b')$ for two different values b and b' are both formed. As $n - \gamma_s > \gamma_s$, both values must have been voted by honest replicas. Let r and r' be the first honest replicas that have voted for b and b' at time t and t', respectively. Without loss of generality, we assume $t \leq t'$. As r votes for b at t, it must have received and broadcast $\langle \text{propose}, b \rangle_{r_s}$ at $t - \Delta$. Then, all honest replicas including r' must have received it by $t \leq t'$. Therefore, r' could not have voted for $b' \neq b$ at t', a contradiction.

An honest designated sender broadcasts $\langle \text{propose}, b_{in} \rangle_{r_s}$, and all $n - \gamma_s$ honest replicas vote for b_{in} forming a certificate $C(b_{in})$. As no conflicting certificate exists, all honest replicas output the value b_{in}. Thus, the validity holds.

If an honest replica outputs a value b, then it must have received and broadcast $C(b)$, which is received by all honest replicas. As no conflicting certificate exists, all honest replicas output the value b. Thus, the totality holds. □

Lemma 3.4 (Safety). *If the network is asynchronous and there are at most β_a faults, then safety holds.*

Proof. Integrity proof is identical to Lemma 3.2. We prove consistency. Suppose for the sake of contradiction that honest replicas output two different values b and b', then two certificates $C(b)$ and $C(b')$ are both formed. Let C and C' be the two sets of $n - \gamma_s$ replicas that have voted for b and b'. As C and C' intersect at $2(n - \gamma_s) - n = \beta_a + 1$ replicas, at least an honest replicas must have voted for both values, a contradiction. □

Lemma 3.5 (Liveness). *If the network is under asynchrony and there are at most γ_a faults, then liveness holds.*

Proof. Due to a quorum-intersection argument like the proof of Lemma 3.4 in the presence of $\gamma_a \leq \beta_a$ faults, there cannot be conflicting certificates. The rest of the proof is identical to Lemma 3.3. □

Efficiency. When the leader is honest, the latency of the protocol is two rounds plus 2Δ time; under synchrony, it can be written as $2\Delta + 2\delta$ where δ is the actual network delay and is usually significantly smaller than the conservative delay bound Δ. Our protocol requires $O(n^2)$ messages, matching that of the Bracha's broadcast [10]. The communication complexity (in bits) of our protocol is $O(n^3)$ as replicas send certificates containing $O(n)$ signatures. This is more expensive than Bracha's $O(n^2)$ RBC. It can be reduced to $O(n^2)$ using threshold signatures [13, 25].

4 TIGHT FAULT TOLERANCE BOUND ON MT-BFT SMR IN THE PSYNC-SYNC MODEL

This section establishes a tight bound on the fault thresholds of MT-BFT for state machine replication in the psync-sync model.

4.1 Fault Tolerance Limit of MT-BFT SMR

As mentioned before, a SMR protocol can solve RBC under the same condition. Suppose we have a SMR protocol. The n replicas participating in the RBC execution run the SMR protocol. The designated sender r_s generates a transaction containing the signed input value $\langle b_{in} \rangle_{r_s}$. A replica outputs the first transaction in the SMR log that is signed by the sender r_s. That is, when a replica outputs in SMR a log $[tx_1, tx_2, ..tx_l]$ where tx_i is $\langle b \rangle_{r_s}$ and no tx_j $(j < i)$ is signed by r_s, then the replica outputs b in RBC. The safety and liveness of RBC easily follow from the safety and liveness of SMR. Therefore, the constraint $\beta_a + 2\gamma_s < n$ in Theorem 3.1 applies to SMR as well.

However, this is not the full picture for SMR. We show that for MT-BFT SMR, due to the need for public verifiability, another constraint exists between safety and liveness under synchrony formulated as $\beta_s + f_s < n$.

Public verifiability. To prove this bound, we introduce a variant of RBC called publicly verifiable reliable broadcast (PVRBC). The only difference from the regular RBC in Section 2.1 is that we add the public verifiability property to RBC. As in the case of SMR, this means there is a predefined Boolean function Verify and a replica outputs a value b if and only if it obtains a publicly verifiable proof π such that $\text{Verify}(b, \pi) = 1$. We can still use the aforementioned reduction to solve PVRBC using SMR, so a negative result for PVRBC also applies to SMR.

Now we show the bound $\beta_s + f_s < n$ for MT-BFT PVRBC below.

Theorem 4.1. *There does not exist a MT-BFT PVRBC protocol such that for a certain $n > 0$, its threshold parameters satisfy $\beta_s + \gamma_s \geq n$.*

Proof. Suppose for the sake of contradiction there exists a MT-BFT PVRBC protocol whose threshold parameters satisfy $\beta_s + \gamma_s = n$ for a certain $n > 0$ (trivially extended for $\beta_s + \gamma_s > n$). Suppose the

network consists of two partitions P and Q with size $\gamma_s > 0$ and $\beta_s > 0$, respectively. A designated sender r_s is in Q. In this setting, there are three possible executions below.

In the first execution (W1), all replicas are honest. The sender r_s has an input value b. By the liveness, every honest replica outputs b, i.e., obtains a proof π such that $\mathsf{Verify}(b, \pi) = 1$.

In the second execution (W2), all γ_s replicas in P crash. The sender r_s has an input value $b' \neq b$. By the liveness in the presence of γ_s faults, every honest replica outputs b', i.e., obtains a proof π' such that $\mathsf{Verify}(b', \pi') = 1$.

In the third execution (W3), all β_s replicas in Q are Byzantine. Q behave towards P as in W1. Replicas in P cannot distinguish W1 and W3, so they behave as in W1. Thus, Byzantine replicas in Q can obtain π such that $\mathsf{Verify}(b, \pi) = 1$. On the other hand, Q can simulate the execution of W2 in which P crash. Thus, Byzantine replicas in Q can also generate π' such that $\mathsf{Verify}(b', \pi) = 1$. This means honest replicas can receive proofs π and π' for two different values $b \neq b'$ such that $\mathsf{Verify}(b, \pi) = 1$ and $\mathsf{Verify}(b', \pi') = 1$. This violates consistency. However, as the number of faults is $|Q| = \beta_s$, consistency should hold. This is a contradiction. □

The extra constraint shows that BFT SMR can achieve higher (though not perfect) synchronous safety at the cost of synchronous liveness. This observation helps us find new improvements to classic asynchronous and partially synchronous protocols to potentially tolerate $\beta_s < 2n/3$ for synchronous safety.

A remark on Schneider [45]. It has been stated, without a proof, that BFT SMR can tolerate only $f < n/2$ faults due to the need for public verifiability [45]. While intuition is clear for this claim, we are not aware of a rigorous proof for it. We observe that it is a special form of our Theorem 4.1 with a restriction of $t = \beta_s = \gamma_s$. So we have provided a rigorous proof for this well-known result.

4.2 A MT-BFT SMR Protocol with Optimal Fault Tolerance

We present a MT-BFT SMR protocol with optimal fault tolerance in the psync-sync model. Our protocol allows any parameter choice within the optimal trade-offs. Namely, a protocol designer can first pick $\gamma_s < n/2$. Then, other parameters are determined as follows

(1) $\beta_a = n - 2\gamma_s - 1$
(2) $\beta_s = n - \gamma_s - 1$
(3) $\gamma_a = \min\{\beta_a, \gamma_s\}$

This is optimal given Theorem 3.1 and 4.1 and also establish the tightness of the bounds for MT-BFT SMR in the psync-sync model.

Commit and public verifiability. When proving a negative result, we had to use the most general definition of SMR. But in designing a protocol, we can use a set of safety and liveness conditions convenient for us as long as they are sufficient for SMR. Towards this end, we introduce the *batching* technique and the *commit* operation that are widely used in the SMR literature [2, 11, 14, 31, 35, 45, 47].

Note that batching transactions into blocks is compatible with the SMR definition in Section 2.1 where each position contains a single transaction. Transactions within a block are totally ordered, so a log of blocks can be flattened into a log of transactions. For example, suppose we have a log of blocks $[B_0, B_1, \ldots]$ with $B_0 = [tx_{0,1}, tx_{0,2}, \ldots, tx_{0,\ell_0}]$ and $B_1 = [tx_{1,1}, tx_{1,2}, \ldots, tx_{1,\ell_1}]$. This can be interpreted as a log of transactions $[tx_{0,1}, tx_{0,2}, \ldots, tx_{0,\ell_0}, tx_{1,1}, tx_{1,2}, \ldots, tx_{1,\ell_1}, \ldots]$. The publicly verifiable proof for a block serves as the publicly verifiable proof for each transaction in that block.

When a replica *commits* a new block, it broadcasts a signature on the block. With some foresight, we will adopt the recent paradigm of chained SMR in which the last block of a log uniquely identifies the entire log, and hence signing a block is equivalent to signing the the entire log up to the block. Then, a set of signatures on a block from $n - \gamma_s = \beta_s + 1$ replicas forms a publicly verifiable proof for the log up to that block. Hereafter, when designing new protocols, we aim at achieving the following safety and liveness properties.

(1) **Safety.** If two honest replicas commit two logs $[B_0, B_1, ..., B_j]$ and $[B'_0, B'_1, ..., B'_{j'}]$, then $B_i = B'_i$ for all $i \leq \min(j, j')$.
(2) **Liveness.** Every transaction is eventually committed by all honest replicas.

The above safety and liveness allow us to focus on replicas and deal with clients and public verifiability easily in a single extra step. This is sufficient for the safety and liveness for SMR defined in Section 2.1. Simply observe that at least one honest replica must commit a log in order for the log to have a publicly verifiable proof; If all honest replicas commit a log, all honest replicas will obtain a publicly verifiable proof for it.

Intuitive overview of the protocol. At a high level, we combine a partially synchronous SMR protocol (PBFT [14]) and a synchronous SMR protocol (Sync HotStuff [2]). These two protocols share a similar view-by-view construction. In the steady state of each view, a leader L proposes a value to the next log position and replicas vote and commit the value. When no progress is being made (possibly because of a faulty leader), replicas replace the leader and enter the next view using a view change procedure.

We further observe that these two protocols also share two fundamental steps in achieving safety despite their different timing models and fault thresholds. In particular, they both need to guarantee safety within a view and across views. Let $|C|$ be the quorum size; $|C| = 2f + 1$ out of $n = 3f + 1$ in PBFT and $|C| = f + 1$ out of $n = 2f + 1$ in Sync HotStuff, where f is the fault tolerance threshold.

(P1) consistency within a view: If an honest replica commits a value b in view v, no conflicting value b' has a certificate in view v.

(P2) consistency across views: If an honest replica commits a value b in view v, more than $n - |C|$ honest replicas receive a certificate of b from view v before entering the next view $v + 1$.

PBFT achieves P1 using the standard quorum intersection technique. With up to f Byzantine faults out of $n = 3f + 1$ replicas, two quorums of $|C| = 2f + 1$ replicas intersect at at least one honest replica (because $2|C| - n = f + 1$). Having two conflicting certificates from the same view would imply that an honest replica voted for two equivocating proposals, which cannot happen. For P2, PBFT uses two rounds of voting, so an honest replica commits a value only after $|C| = 2f + 1$ replicas (at least $f + 1$ honest) voted for the certificate of that value.

Sync HotStuff, on the other hand, cannot rely on quorum intersection because it tolerates f Byzantine faults out of $n = 2f + 1$ replicas. Instead, it uses synchronous waiting periods to achieve these properties. An honest replica r commits a value only after

waiting for 2Δ time and detecting no leader equivocation or view-change. The equivocation check makes sure no honest replica votes for conflicting values. This rules out conflicting certificates in this view, achieving P1. Checking view-change of other replicas makes sure no honest replica enters the next view early. This leaves enough time for all honest replicas to receive (from r) a certificate for the committed value before entering the next view, achieving P2.

Combining PBFT and Sync HotStuff. Our key observation is that the orthogonal techniques of PBFT and Sync HotStuff are compatible under the three constraints at the beginning of this section and a quorum size $|C| = n - \gamma_s$, as we elaborate below.

PBFT's quorum intersection ensures that two quorums of size $n - \gamma_s$ intersect at $2(n - \gamma_s) - n = n - 2\gamma_s = \beta_a + 1$ replicas. Under partial synchrony, this ensures P1 with up to β_a faults, which is our target fault threshold for partially synchronous safety. Moreoever, a quorum of $|C| = n - \gamma_s = \gamma_s + \beta_a + 1$ replicas contains at least $\gamma_s + 1 > n - |C|$ honest replicas, so PBFT's two-phase voting ensures P2 under partial synchrony.

Under synchrony, we can show the 2Δ waiting periods ensure that P1 and P2 still hold. When an honest replica commits a value, the 2Δ waiting period before commit still ensures that no honest replica votes for a conflicting value, hence ruling out a conflicting certificate (P1 holds) under β_s faults because $|C| = n - \gamma_s = \beta_s + 1$. Similarly, the 2Δ waiting period in view change ensures that all $n - \beta_s = \gamma_s + 1 > n - |C|$ honest replicas receive a certificate before entering the next view (P2 holds).

Therefore, combining the techniques of PBFT and Sync HotStuff seamlessly with a quorum size of $|C| = n - \gamma_s$ gives a MT-BFT SMR protocol with optimal fault thresholds.

4.3 Protocol Description

Block chaining. Following recent BFT protocols [2, 47], we utilize the "block chaining" paradigm to simplify the protocol. In the steady state protocol, the leader proposes a block containing a list of transactions and a hash digest of the previous block. Thus, a block determinies a unique hash chain for all previous blocks in the log. Any chains starts at a hard-coded *genesis block*, and the distance from the genesis block to a block B in the chain is called the *height* of block B. A block of height k, denoted B_k, is formatted as $(d_k, H(B_{k-1}))$ where d_k is a set of transactions and $H(B_{k-1})$ is the hash of the predecessor block B_{k-1}. The genesis block B_0 can be written as $B_0 = (\perp, \perp)$. We say a block $B_k = (d_k, h_{k-1})$ is *valid* if (i) it is the genesis block or (ii) there is a valid block B_{k-1} and $h_{k-1} = H(B_{k-1})$. We say a block B_k *extends* B_l if $B_k = B_l$ or B_k is a descendant of B_l. If two blocks do not extend one another, we say they *conflict* with each other. Conflicting blocks can occur due to network asynchrony or faulty leaders. If two conflicting blocks are signed in the same view by the same leader, they form a proof of leader *equivocation*.

Each view is identified by a monotonically increasing integer denoted $v \geq 1$, and has a leader L selected in a round robin manner (e.g., $v \mod n$). In each view v, the steady state protocol (Figure 2) runs the following steps in iterations. Note that each step is triggered by an "upon" event and is hence non-blocking. Thus, subsequent blocks B_{k+1}, B_{k+2} can be proposed before the previous block B_k is committed.

Propose. The leader L of view v proposes a block $B_k = (d_k, H(B_{k-1}))$ in the form of $\langle \text{propose}, B_k, C_v(B_{k-1}), v \rangle_L$. The certificate in the proposal must be the certificate for the predecessor B_{k-1} in the current view v. In the steady state, the leader L proposes B_k upon receiving $C_v(B_{k-1})$ for its previous proposal B_{k-1} in the current view. For the first proposal after entering view v, the certificate $C_v(B_{k-1})$ is formed during the view change protocol and will be described later.

Vote. Upon receiving a valid proposal $\langle \text{propose}, B_k, C_v(B_{k-1}), v \rangle_L$ by the leader L of the current view v, a replica r votes for the block B_k in the form of $\langle \text{vote}, B_k, v \rangle_r$, if the replica has not received an equivocating proposal signed by L in the current view v.

Certificates and ranking. A quorum of $|C| = n - \gamma_s$ votes form a *quorum certificate* (or certificate for short). This is the key ingredient of many SMR protocols including ours. To elaborate, each replica r votes for a block B_k (at height k proposed by the leader of the current view v in the form of $\langle \text{vote}, B_k, v \rangle_r$. A quorum of $\langle \text{vote}, B_k, v \rangle_*$ messages from distinct replicas form a certificate for block B_k in view v, denoted $C_v(B_k)$. We say a block B_k is certified in view v if the certificate $C_v(B_k)$ is formed. Certificates are ranked first by view number and then by height. For example, $C_v(B_k)$ is ranked higher than $C_{v-1}(B_{k+1})$ but lower than $C_v(B_{k+1})$.

Pre-commit. Upon receiving a certificate $C_v(B_k)$, a replica r forwards it to all other replicas. Then, after waiting for 2Δ, it broadcasts $\langle \text{commit}, B_k, v \rangle_r$. The 2Δ waiting step helps maintain consistency both within a view and across views (i.e., P1 and P2) under synchrony as in Sync HotStuff. The commit message corresponds to the second vote in PBFT, which helps maintain consistency across views (i.e., P2) under partial synchrony.

Commit. Finally, upon collecting a quorum of $\langle \text{commit}, B_k, v \rangle_*$, a replica forwards them to all other replicas, and commits the block B_k. As we mentioned before, a replica commits a log by broadcasting a signature on the log instead of each block or transaction. However, as all blocks are chained by hash references, the block B_k works as a snapshot of the log that consists of B_k and all its ancestors. Therefore, we hereafter say a replica commits a block/log interchangeably.

The view change protocol (Figure 3) monitors the progress in the steady state and changes the view when the leader exhibits a faulty behavior or replicas fail to make progress.

Blame. A replica blame the view by broadcasting $\langle \text{blame}, v \rangle_r$ if it detects leader equivocation or suspects the leader is misbehaving such as stalling progress or censoring transactions. A transaction tx is considered to be censored if it has not been committed by $\max\{T_{tx}, T_v\} + \Lambda$ where T_{tx} is the time it receives tx, and T_v is the time it starts view v, and Λ is a given time (the specific value is discussed later). In the case of the leader's equivocation, the replica forwards the two equivocating proposals to all other replicas as proof of misbehavior of the current leader, and stops all processes in the steady state of view v.

Status. A quorum of distinct $\langle \text{blame}, v \rangle_*$ is called a blame-certificate for view v, denoted $\mathcal{B}_v$. Upon receiving $\mathcal{B}_v$, a replica r forwards it

to all other replicas, and stops all processes in the steady state of view v. Then, the replica r sets a variable lock_r to a highest ranked certificate to lock on and sends it to the leader L' of the next view $v+1$ in a message $\langle \text{status}, \text{lock}_r, v\rangle_r$. At this point, replica r starts the new view $v+1$.

New-View. A quorum of $\langle \text{status}, \text{lock}, v\rangle_*$ messages form a status-certificate, denoted $\mathcal{S}_v$. Upon receiving $\mathcal{S}_v$, the new leader L' of view $v+1$ picks a highest certificate lock_{v+1} among $\mathcal{S}_v$, and broadcasts $\langle \text{new-view}, \text{lock}_{v+1}, \mathcal{S}_v, v+1\rangle_{L'}$. One status message in $\mathcal{S}_v$ must come from an honest replica. Since more than $n-|C|$ honest replicas lock on all blocks committed till the previous view v (by P2), the selected highest certificate lock_{v+1} always extends all committed blocks.

First-Vote. Upon receiving a first $\langle \text{new-view}, C_{v''}(B_{k'}), \mathcal{S}_v, v+1\rangle_{L'}$, a replica r first checks that the selected certificate $C_{v''}(B_{k'})$ has a highest rank in $\mathcal{S}_v$. If the check passes, a replica r forwards it to all other replicas and sends $\langle \text{vote}, B_{k'}, v+1\rangle_r$. This step forms the first certificate in the new view $v+1$ that all subsequent proposals in view $v+1$ should extend.

4.4 Correctness of the Protocol

We prove the safety and liveness of the protocol. We say a replica *directly* commits a block B_k in a view v if it is due to the replica receiving a quorum of $n-\gamma_s$ $\langle \text{commit}, B_k, v\rangle$. If a replica commits a block B_k as a result of directly committing one of its descendant, then we say B_k is *indirectly* committed.

Safety under partial synchrony. We prove safety under partial synchrony in the presence of at most β_a faulty replicas.

Lemma 4.2 (Consistency within a View). *If two certificates $C_v(B_k)$ and $C_v(B_l)$ from the same view v exist, then B_k and B_l do not conflict with each other.*

Proof. Suppose for the sake of contradiction two conflicting certificates $C_v(B_k)$ and $C_v(B_l)$ both exist, then at least an honest replica must have voted for both, because two quorums of $n-\gamma_s$ intersect at an honest replica, i.e., $2(n-\gamma_s)-n > \beta_a$. This cannot happen because an honest replica stops voting in view v as soon as it detects equivocating proposals. □

Lemma 4.3. *If an honest replica directly commits a block B_k in a view v, then at least γ_s+1 honest replicas receive $C_v(B_k)$ before entering view $v+1$.*

Proof. Suppose an honest replica directly commits a block B_k in a view v, then it must have received $n-\gamma_s$ $\langle \text{commit}, B_k, v\rangle$, out of which at least $n-\gamma_s-\beta_a > \gamma_s$ must be from honest replicas. These γ_s+1 honest replicas must have received the certificate $C_v(B_k)$ before sending $\langle \text{commit}, B_k, v\rangle$ before entering view $v+1$. □

Lemma 4.4 (Consistency across Views). *If an honest replica directly commits a block B_k in a view v, then for all view $v' \geq v$, if a certificate $C_{v'}(B_l)$ exists, then B_l does not conflict with B_k.*

Proof. We prove by induction on the view number. The base case (i.e., $v'=v$) follows from Lemma 4.2. We prove for the inductive step (i.e., view $v'+1$). Suppose a certificate $C_{v'+1}(B_l)$ exists. Let B_m be the block of lowest height in the chain that is certified in view $v'+1$. Then, at least an honest replica must have voted for B_m in the First-Vote phase of the view change protocol. Otherwise, blocks extending B_l (including B_l) could not have been certified in view $v'+1$, as every valid proposal should contain a certificate for the previous block in the same view. In the First-Vote phase, the honest replica must have voted for B_m in response to $\langle \text{new-view}, C_{v''}(B_m), \mathcal{S}, v'+1\rangle_{L'}$ message from the leader L' of view $v'+1$, and $C_{v''}(B_m)$ ($v'' \leq v'$) should be the highest certificate in $\mathcal{S}$. However, recall that at least γ_s+1 honest replicas (denoted R) receive $C_v(B_k)$ before entering view $v+1 \leq v'+1$ (Lemma 4.3). As R and $\mathcal{S}$ ($|\mathcal{S}| \geq n-\gamma_s$) intersect, $C_{v''}(B_m)$ should be as highly ranked as $C_v(B_k)$. By the inductive hypothesis, B_m should extend B_k. Therefore, B_l (which extends B_m) does not conflict with B_k. □

Lemma 4.5 (Safety for Replica). *Honest replicas do not commit conflicting blocks.*

Proof. Suppose two blocks B_k and $B_{k'}$ are committed due to directly committed descendants B_l and $B_{l'}$, respectively. As all directly committed blocks are certified, by Lemma 4.4, B_l and $B_{l'}$ do not conflict with each other. Therefore, B_k and $B_{k'}$ are both ancestors of B_l (and $B_{l'}$), and do not conflict with each other. □

Safety under synchrony. We next prove the safety under synchrony, assuming in the presence of at most β_s faulty replicas.

Lemma 4.6 (Consistency within a View). *If an honest replica directly commits a block B_k in a view v, then a certificate $C_v(B_l)$ for a conflicting block B_l cannot exist.*

Proof. Suppose an honest replica directly commits a block B_k in a view v, then it must have received $n-\gamma_s$ $\langle \text{commit}, B_k, v\rangle$. At least one of them is from an honest replica since $n-\gamma_s > \beta_s$. Let t be the first time when an honest replica (say r) sends $\langle \text{commit}, B_k, v\rangle$, then r must have received and broadcasted $C_v(B_k)$ by $t-2\Delta$. Thus, all honest replicas must have received $C_v(B_k)$ by $t-\Delta$, and could not have voted a conflicting block B_l after that. But if any honest replica voted for B_l (and forwarded the proposal of B_l) before $t-\Delta$, then r must have received the conflicting proposal of B_l by t. Then, r would have then stopped all processes in the steady state of view v and could not have sent commit for B_k at t, a contradiction. Thus, no honest replica could not voted for B_l. As $n-\gamma_s > \beta_s$, there are not enough votes for B_l to form $C_v(B_l)$ □

Lemma 4.7. *If an honest replica directly commits a block B_k in a view v, then at least γ_s+1 honest replicas receive $C_v(B_k)$ before entering the next view $v+1$.*

Proof. Suppose an honest replica directly commits a block B_k in a view v, then it must have received $n-\gamma_s$ $\langle \text{commit}, B_k, v\rangle$. At least one of them is from an honest replica since $n-\gamma_s > \beta_s$. Let t be the first time when an honest replica (say r) sends $\langle \text{commit}, B_k, v\rangle$, then r must have received and broadcasted $C_v(B_k)$ by $t-2\Delta$. Thus, all honest replicas must have received $C_v(B_k)$ by $t-\Delta$. We just need to prove that no honest replica has entered the next view before $t-\Delta$. Suppose for the sake of contradiction an honest replica enters the next view $v+1$ before $t-\Delta$, then it must have received and broadcasted $\mathcal{B}_v$ before $t-\Delta$. Then, r must have received $\mathcal{B}_v$ before t, and could not have sent commit for B_k at t, a contradiction. Thus,

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

(1) **Propose.** Upon receiving $C_v(B_{k-1})$, if $r = L$, broadcast $\langle \text{propose}, B_k, C_v(B_{k-1}), v\rangle_L$ where B_k extends B_{k-1}.

(2) **Vote.** Upon receiving a valid proposal $\langle \text{propose}, B_k, C_v(B_{k-1}), v\rangle_L$, if it has not detected equivocation in the current view v, broadcast the proposal and $\langle \text{vote}, B_k, v\rangle_r$.

(3) **Pre-commit.** Upon receiving $C_v(B_k)$, broadcast it, wait for 2Δ, and broadcast $\langle \text{commit}, B_k, v\rangle_r$.

(4) **Commit.** Upon receiving a quorum of $\langle \text{commit}, B_k, v\rangle_*$, forward them, and commit B_k and all its ancestors.

Figure 2: An optimal multi-threshold BFT SMR – steady state protocol.

Let v be the view number and replica L and L' be the leader of the view v and v', respectively. A replica r run the following in view v.

(1) **Blame.** When either two conditions below holds, broadcast $\langle \text{blame}, v\rangle_r$.
   (a) The replica r cannot commit a transaction tx by $\max\{T_{tx}, T_v\} + \Lambda$ where T_{tx} is the time it receives tx, and T_v is the time it starts view v, and Λ is a given time-out.
   (b) The replica r receives two equivocating proposals.
   In the third case, broadcasts the two proposals, and stop all processes in the steady state of view v immediately.

(2) **Status.** Upon receiving $\mathcal{B}_v$, broadcast it, stop all processes in the steady state of view v, and send $\langle \text{status}, \text{lock}_r, v\rangle_r$ to L' where lock_r is the highest certificate known to r. Enter the next view $v + 1$.

(3) **New-View.** Upon receiving a quorum of $\langle \text{status}, \text{lock}, v\rangle_*$ denoted $\mathcal{S}_v$, if $r = L'$, broadcast $\langle \text{new-view}, \text{lock}_{v+1}, \mathcal{S}_v, v + 1\rangle_{L'}$, where lock_{v+1} is a highest certificate in $\mathcal{S}_v$.

(4) **First-Vote** Upon receiving the first $\langle \text{new-view}, C_{v'}(B_{k'}), \mathcal{S}_v, v + 1\rangle_{L'}$, broadcast it and $\langle \text{vote}, B_{k'}, v + 1\rangle_r$, if $C_{v'}(B_{k'})$ is the highest certificate in $\mathcal{S}_v$.

Figure 3: An optimal multi-threshold BFT SMR – view change protocol.

all $n - \beta_s \geq \gamma_s + 1$ honest replicas receive $C_v(B_k)$ before entering the next view $v + 1$, and the lemma holds. □

The rest of the safety proof under synchrony is the same as the safety proof under partial synchrony.

Liveness. We prove liveness under partial synchrony when at most $\gamma_a \leq \gamma_s$ replicas are faulty. Liveness under synchrony can be proven in the same way, and we omit the details of the proof. We prove that, with a reasonable timeout of $\Lambda = O(\Delta)$, every transaction is eventually committed by all honest replicas.

Lemma 4.8. *There exists a timeout $\Lambda = O(\Delta)$ such that every transaction is eventually committed by all honest replicas.*

Proof. Let v_g be the highest view among honest replicas at the global standardization time T_g. Then, all honest replicas receive $\mathcal{B}_{v_g-1}$ by $T_g + \Delta$ and enter view v_g. We first prove that if an honest replica (say r) permanently stays in a view $v \geq v_g$, then every transaction will eventually be committed by all honest replicas in view v.

If r permanently stays in a view $v \geq v_g$, then no honest replica enters a higher view $v' > v$ because that will make r eventually enter view v' as well. If an honest replica commits a transaction by committing a block B_k in view v, then the replica must have received and broadcast a quorum of $n - \gamma_s$ $\langle \text{commit}, B_k, v\rangle_*$, and all honest replicas will also commit B_k and the transaction. Thus, if any transaction tx is not eventually committed by r, then it is not committed in view v by any honest replica. Then, all honest replicas send $\langle \text{blame}, v\rangle$, receive $\mathcal{B}_v$, and enter the next view $v+1$. Therefore, if r permanently stays in a view $v \geq v_g$, then every transaction is eventually committed by all honest replicas in view v.

To complete the proof, it remains to show such a view indeed exists. In fact, after the global standardization time, a view v with an honest leader L is a view that all honest replicas permanently stay in. Let t be the time when the first honest replica enters this view v. Then, the honest leader L collects a quorum of $\langle \text{status}, \text{lock}, v - 1\rangle_*$, broadcasts a valid $\langle \text{new-view}, C_{v'}(B_{k'}), \mathcal{S}_v, v + 1\rangle_L$, and receives a certificate $C_v(B_{k'})$, and thus proposes a first block in the view v containing all transactions that have not committed yet by $t + O(\Delta)$. After that L proposes a new block every 2Δ time (or less) upon collecting a certificate for the previous proposal. Each proposed block will collect a quorum of $n - \gamma_s$ vote and commit within $O(\Delta)$ time. Therefore, every honest replica commits every transaction tx by $\max\{T_v, T_{tx}\} + O(\Delta)$, and thus stays in view v. □

Remark on censorship resistance. Our protocol uses the standard and widely used stable leader approach of PBFT [14] where a leader is replaced only if it is believed to be faulty. Thus, the protocol sets a timeout for each transaction to prevent a faulty leader from censoring specific transactions. But this approach assumes an honest leader is able to propose all transactions in time. In practice, an adversary may create a large number of dummy transactions to saturate the system and that would create an issue for the stable leader approach. An alternative is to revolve the leader after every block proposal such as in HotStuff [47]. The revolving leader approach offers simpler censorship resistance but is less efficient as a leader cannot proposal multiple blocks in a non-blocking fashion.

4.5 Efficiency

The latency of our protocol is $2\Delta + 3\delta$. It almost matches the latency of Sync HotStuff ($2\Delta + 2\delta$). The number of messages sent to commit a block is $n+3n^2$, which also close to that of PBFT ($n+2n^2$) and Sync HotStuff ($n+n^2$). The amount of bits sent to commit a block is $O(n^3)$ as some message contains a linear number of signatures, which matches that of Sync HotStuff but is more expensive than PBFT's $O(n^2)$ bit complexity. It can be reduced to $O(n^2)$ using threshold signatures. Our protocol commits blocks every 2δ as a leader can propose a subsequent block upon collecting a quorum of votes for the previous proposal, which also matches that of PBFT and Sync HotStuff. Therefore, we expect our protocol to have similar performance as PBFT and Sync HotStuff.

4.6 Parameter Choices

Our protocol allows flexible parameter choices within the trade-offs between safety and liveness as well as synchronous and partially synchronous security. There are several characteristic parameter choices with suited applications.

Strengthen the classic partially synchronous setting. A prominent parameter choice is $\beta_a = \gamma_a = \gamma_s < n/3$ and $\beta_s < 2n/3$, where the protocol has the same thresholds as classic partially synchronous protocols in terms of safety/liveness under partial synchrony and liveness under synchrony while tolerating a supermajority of faults for safety under synchrony. Permissioned blockchains [5, 6, 16] and some proof-of-stake cryptocurrencies [12, 24] using partially synchronous BFT protocols can strengthen their security with our protocol.

Strong safety for safety-critical systems. One can opt for stronger safety at the cost of liveness. For example, with a liveness threshold of $\gamma_s = \gamma_a < n/4$, our protocol tolerates $\beta_a < n/2$ and $\beta_s < 3n/4$. Such a parameter choice can be useful in safety-critical applications such as payment systems with high-value transactions [32].

Highly available BFT tolerating partial synchrony. Another interesting parameter choice is to favor high availability under synchrony while tolerating a small fraction of faults under partial-synchrony. For example, with $\gamma_s < 9n/20$ and $\beta_s < 11n/20$ for synchrony, our protocol tolerates $\beta_a = \gamma_a < n/10$ under partial-synchrony. Such a parameter choice can be used in an environment where network failures occur occasionally [11, 19, 29].

5 A FRAMEWORK TO UPGRADE TO OPTIMAL SYNCHRONOUS SAFETY

This section presents a framework to convert any MT-BFT SMR protocol parameterized by $(\beta'_a, \gamma'_a, \beta'_s, \gamma'_s)$ into a MT-BFT SMR protocol with a synchronous safety of $\beta_s = n - \gamma'_s - 1$ while preserving the other thresholds $\beta_a = \beta'_a$, $\gamma_a = \gamma'_a$, and $\gamma_s = \gamma'_s$. Namely, the framework upgrades a base protocol to enjoy optimal synchronous safety (due to Theorem 4.1) without compromising other fault thresholds.

Concretely, existing partially synchronous protocols, e.g., PBFT [14], HotStuff [47], and asychronous protocols, e.g., HoneyBadgerBFT [38], BEAT [21], Dumbo [26], can be viewed as MT-BFT SMR protocols with $\beta_a = \gamma_a = \beta_s = \gamma_s < n/3$. The framework in this section upgrades these protocols to enjoy synchronous safety of $\beta_s < 2n/3$ while preserving the other thresholds of $\beta_a = \gamma_a = \gamma_s < n/3$.

5.1 Protocol Description

Intuitive overview. Our key idea is to compose the given base SMR protocol with a multi-threshold broadcast protocol, such as the multi-threshold RBC protocol in Section 3.2. The output of the SMR protocol is like a virtual sender of the RBC: if a party commits a block in SMR, it treats this block as the proposal it receives from the sender in the subsequent RBC. A block is committed in the converted protocol only if it is committed both in the base protocol and in the RBC protocol.

It is then easy to see that the converted protocol is live if both the base protocol and the RBC protocol are live, and they commit the same block. More interestingly, the converted protocol is safe if *either* the base protocol *or* the RBC protocol is safe. From another angle, in order to break safety in the converted protocol, the adversary needs to break the safety of both the base protocol and the RBC protocol. This is the key observation that leads to the upgraded synchronous safety tolerance. Even when the base protocol violates safety under synchrony, in the presence of $n - \gamma'_s - 1 > \beta'_s$ faults, the RBC protocol, with its higher synchronous safety tolerance, guards the safety of the converted protocol.

The other cases are more straightforward. With $\gamma s'$ faults under synchrony or $\gamma a'$ faults under asynchrony, both the base protocol and the RBC protocol are safe and live. This also means the base protocol functions as an honest virtual sender for the RBC. In this case, by the integrity and validity properties, the RBC protocol commits the same block as the base protocol. Thus, the converted protocol achieves both safety and liveness. With $\beta s'$ faults under synchrony or $\beta a'$ faults under asynchrony, both the base protocol and the RBC protocol are safe (but not necessarily live). The converted protocol is also safe. We remind the readers that we only consider the case with $\gamma s' \leq \beta s'$ and $\gamma a' \leq \beta a'$ in this paper. That means a multi-threshold protocol may enjoy safety without liveness; but we do not consider the case in which a protocol enjoys liveness without safety.

Actual framework with consistent broadcast. An acute reader may have noticed that we did not use RBC's totality property in the above argument. Indeed, totality is not needed and RBC is a slight overkill. We used RBC in this intuitive overview for convenience since we have already introduced RBC. In the actual framework, we use a multi-threshold *consistent broadcast*, which is a weaker primitive than RBC as it does not require totality.

The framework is given in Figure 4. Let Π_{base} be the underlying base protocol parameterized by $(\beta'_a, \gamma'_a, \beta'_s, \gamma'_s)$. Similar to the RBC protocol in Section 3.2, we use a quorum-based design with a quorum size of $n - \gamma'_s > \gamma'_s + \beta'_a$ for consistent broadcast. Such a quorum contains at least one honest replica in the presence of γ'_s faults or β'_a faults. Thus, when the sender is honest, no certificate can be formed for any value other than the sender's input. Therefore,

validity and integrity hold. Consistency under synchrony is easily achieved by a similar synchronous equivocation check mechanism. As in Section 4, a quorum of $\langle \text{commit}, B_k \rangle_*$ messages serves as proof for the log up to block B_k.

5.2 Correctness of the Framework

We prove the correctness of the our framework. Let Π_{base} be the given base protocol with threshold parameters of $(\beta'_a, \gamma'_a, \beta'_s, \gamma'_s)$. We prove that the converted protocol achieves the optimal $\beta_s = n - \gamma'_s - 1$ while tolerating the same $\beta_a = \beta'_a$, $\gamma_a = \gamma'_a$, and $\gamma_s = \gamma'_s$. Note that the base protocol assumes either an asynchronous or a partially synchronous network. We will use the asynchronous case for convenience. The proof below directly applies to the partially synchronous case.

Lemma 5.1 (Safety). *If the network is asynchronous and there are at most β'_a faults, then safety holds.*

Proof. In order for the converted protocol to commit two conflicting blocks, they first need to be committed by the base protocol Π_{base}, which will not happen under the said condition. □

Lemma 5.2 (Safety). *If the network is synchronous and there are at most $n - \gamma'_s - 1$ faults, then safety holds.*

Proof. Suppose for the sake of contradiction that these two distinct blocks B_k and B'_k are committed at the same height by the converted protocol. There must be a quorum of $n-\gamma'_s$ $\langle \text{commit}, B_k \rangle_*$ messages and one of them must be from an honest replica. For the same reason, there must be a $\langle \text{commit}, B'_k \rangle_*$ message from an honest replica. Let t be the time when the first honest replica r sent $\langle \text{commit}, B_k \rangle_r$, and t' be the time when the first honest replica r' sent $\langle \text{commit}, B'_k \rangle_{r'}$. Without loss of generality, we assume $t \le t'$. Then, r must have received and broadcast a certificate $C(B_k)$ at $t-\Delta$. It will be received by r' by time $t \le t'$. It would have prevented r' from sending $\langle \text{commit}, B'_k \rangle_{r'}$ at t', a contradiction. □

Lemma 5.3 (Liveness). *If the network is asynchronous and there are at most γ'_a, then liveness holds.*

Proof. Due to the liveness of Π_{base} in the presence of γ'_a faults, all honest replicas keep committing new blocks in Π_{base}. Let B_k be a new block committed in Π_{base}. Then, all honest replicas broadcast $\langle \text{vote}, B_k \rangle$, and a certificate $C(B_k)$ is created. Due to the safety of Π_{base} in the presence of $\gamma'_a \le \beta'_a$ faults, no honest replica votes for a conflicting block B'_k, and thus a certificate $C(B'_k)$ cannot be formed. Therefore, all honest replicas broadcast $\langle \text{commit}, B_k \rangle$ and receive a quorum of $\langle \text{commit}, B_k \rangle_*$. Thus, all honest replicas commit B_k. □

The proof of liveness of the protocol under synchrony is similar.

5.3 Discussions

Overheads added by the framework. In terms of latency, the framework adds two extra rounds plus a Δ waiting time. In terms of communication complexity, the framework introduces $3n^2$ additional messages; some messages are linear-sized certificates, which can again be reduced to a constant size using threshold signatures.

Responsiveness. As seen above, our framework (and the protocol in Sections 3 and 4) is non-responsive, i.e., the latency depends on a pre-defined estimated bound Δ. We next show that the lack of responsiveness is inherent if we want higher synchronous safety.

It is well known that any BFT protocol that is safe with t faults cannot commit *responsively*, i.e., faster than the delay bound Δ, in the presence of $\frac{n-t}{2}$ faults, even if designated sender or the leader is honest [42, 43]. This bound can be easily generalized to MT-BFT: no MT-BFT protocol can commit responsively in the presence of $\frac{n-\beta_s}{2}$ faults. We prove the bound using RBC as an example.

Theorem 5.4. *There does not exists a MT-BFT RBC protocol that is safe under β_s Byzantine faults under synchrony, and responsive in the presence of $\frac{n-\beta_s}{2}$ Byzantine faults.*

Proof. Suppose for the sake of contradiction that there exists an MT-BFT RBC protocol that is responsive in the presence of $\frac{n-\beta_s}{2}$ Byzantine faults. We consider a network with three partitions P, Q and R, with sizes $|P| = |R| = \frac{n-\beta_s}{2}$ and $|Q| = \beta_s$. The designated sender r_s is in Q. Consider the three executions below.

In the first execution (W1), all messages are instantly delivered and all replicas in P crash. The sender r_s has an input value b_1. Since the protocol is responsive in the presence of $\frac{n-\beta_s}{2}$ Byzantine faults, all replicas in R commit b_1 within Δ time.

The second execution (W2) is symmetric to the first one. All messages are instantly delivered and all replicas in R crash. The sender r_s has an input value $b_2 \neq b_1$. Since the protocol is responsive in the presence of $\frac{n-\beta_s}{2}$ Byzantine faults, all replicas in P commit b_2 within Δ time.

In the third execution (W3), all replicas in Q are Byzantine. Q behave towards R and P as in W1 and W2, respectively. All messages between P and R are delivered with delay Δ, but all other messages are instantly delivered. Then, replicas in R receive no messages from P by time Δ and cannot distinguish W1 and W3 by time Δ. Thus, they commit b_1 before Δ as in W1. Similarly, replicas in P cannot distinguish W2 and W3 before Δ, and they commit $b_2 \neq b_1$ before Δ as in W2. This violates the supposition that the protocol achieves consistency in the presence of $|Q| = \beta_s$ Byzantine faults. □

Existing partially synchronous protocols such as PBFT, when viewed as MT-BFT, tolerate $\beta_a = \beta_s < n/3$ faults; thus, they always satisfy the above condition and, if the leader is honest, can commit responsively. However, as we achieve $\beta_s \ge n/3 > \beta_a$, it is inevitable that the protocol (while live) cannot be responsive in the presence of $n/3$ faults. Nonetheless, our protocols can still be made *optimistically responsive* [43], i.e., responsive when the number of actual faults is less than $\frac{n-\beta_s}{2}$, using techniques in the literature [39, 46].

Higher safety after GST. However, our upgrading framework actually provides a stronger guarantee than the MT-BFT psync-sync model defined in Section 2.2. In our definition in Section 2.2, if the network is partially synchronous, a protocol does not need to provide higher safety tolerance even after GST. In other words, a MT-BFT protocol only needs to provide higher safety of β_s only if the network is synchronous at all time. In comparison, the upgrading framework in this section provides safety in the presence of β_s faults after GST in a partially synchronous network. Let us briefly prove this. Let t_1 be the first time after GST T_g that an honest replicas commits a block B_k of height k. Then all honest replicas receive

Let Π_{base} be the given base protocol parameterized by $(\beta'_a, \gamma'_a, \beta'_s, \gamma'_s)$. A replica r runs the following steps in addition to running the base protocol Π_{base}. A quorum is $|C| = n - \gamma'_s$.

(1) **Vote.** Upon committing a block B_k at height k in Π_{base}, broadcast $\langle \text{vote}, B_k \rangle_r$.

(2) **Pre-commit.** Upon receiving a quorum of $\langle \text{vote}, B_k \rangle_r$ denoted $C(B_k)$, broadcast it and wait for Δ. Then, broadcast $\langle \text{commit}, B_k \rangle_r$.

(3) **Commit.** Upon receiving a quorum of $\langle \text{commit}, B_k \rangle_*$ and Π_{base} has committed B_k, then commit B_k.

Upon receiving $C(B_k)$ and $C(B'_k)$ for conflicting blocks B_k and B'_k, stop all operations above for height k immediately.

Figure 4: A framework to convert a MT-BFT SMR protocol with γ'_s synchronous liveness into a MT-BFT SMR protoocl with optimal synchronous safety $\beta_s = n - \gamma'_s - 1$.

$C(B_k)$ by t_1. Therefore, no honest replicas commit any other block B'_k of height k after that.

6 RELATED WORKS

Byzantine fault-tolerance is a forty-year-old research field in distributed computing and cryptography. Starting from the celebrated work of Lemport et al. [34, 44], it has been studied mostly in a single timing model with one fault threshold. The synchronous model has been mainly the target of theoretical research, assuming perfectly synchronized rounds across all parties [1, 20, 23, 30]. Only recently, following the introduction of Bitcoin, a.k.a., Nakamoto Consensus [40], which is perhaps the first practical synchronous protocol, a couple of works have presented synchronous BFT protocols under the non-lockstep model towards practical use [2, 4, 46]. Yet, it seems there remains strong reluctance in the community to rely on synchrony due to concerns for more severe network failures. Classic studies of practical BFT have mainly focused on the partially synchronous protocols favoring their ease of design and asynchronous safety. Numerous works studied improvements over PBFT [25, 31, 47]. Other works assume the fully asynchronous model [3, 21, 26, 36, 38]. However, these protocols tolerate only $f < n/3$ faults even when the network is under synchrony. Our primary motivation is to resolve this long-standing dilemma between the timing assumption and fault tolerance.

Weakly synchronous model. Some recent works have considered an intermediate model between the synchronous and asynchronous models. Guo et al. [27] introduced the *weakly synchronous* model, where a majority of the participants are honest and synchronous, but the remaining minority may be Byzantine or suffer from a network failure. They presented a Byzantine agreement protocol in this setting and some other works applying it to BFT SMR protocols [2, 4, 15]. However, these works still break down if a majority of the participants experience asynchrony. In contrast, our protocols are safe against one-third faults even under complete asynchrony.

Multiple fault thresholds. There have been previous works that consider different thresholds for different correctness properties in a *single* timing model. UpRight [18] is a BFT SMR protocol allowing different thresholds for safety and liveness in the partially synchronous model. Another recent work [28] studies reliable broadcast and Byzantine agreement with different fault thresholds for different correctness properties and captures optimal trade-offs between them in the asynchronous model. In comparison, our MT-BFT framework captures trade-offs not only between different correctness properties but also between different timing models, notably as we improve synchronous safety without compromising properties in asynchrony or partial synchrony.

Dual timing model. The beautiful recent work of Blum et al. [7–9] is the closest to our work and is the inspiration to our work. They considered the async-sync dual timing model. They provided a first and partial answer the above question. However, it does not completely resolve the above dilemma as it cannot match the fault tolerance of classic single-model protocols. For example, with $\beta_a = \gamma_a < n/3$, their protocol tolerates only $\beta_s = \gamma_s < n/3$, which is the same as classic asynchronous and partially synchronous protocols, while our protocol tolerates $\beta_s < 2n/3$. Moreover, from a practical perspective, the construction of their protocol seems more complicated and works in lockstep rounds. In contrast, our protocol is simpler and closer to deployed practical protocols.

The XFT protocol [35] also considers the psync-sync dual timing model. It extends Paxos [33], a popular crash fault-tolerant (CFT) replication protocol, to tolerate $f < n/2$ Byzantine faults under synchrony. However, under partial-synchrony, the protocol does not tolerate Byzantine faults.

Flexible BFT. Flexible BFT [37] (FBFT) is a recent work that supports two different timing models and also separates thresholds for safety and liveness. FBFT also combines some techniques from the partially synchronous PBFT and the synchronous Sync HotStuff, like our protocol in Section 4. But the key difference is that FBFT does not fully combine the two protocols. Instead, it leaves the two different commit rules untouched, and leaves the responsibility to each client to choose between the two commit rules according to its own belief about the network. This is a fundamental difference in the design goal as FBFT's goal is to support clients with diverse beliefs about the network. If a client in FBFT makes an incorrect assumption about the network, e.g., choose the synchronous commit rule when the network is asynchronous, the protocol does not provide any guarantees to that client.

In contrast, our protocol in Section 4 combines the two protocols including their commit rules from different timing models into a single protocol with a single commit rule. And we maintain the standard model where clients do not choose their own models or commit rules. Our protocol provides all clients with safety and liveness guarantees under both timing models (though with different fault thresholds).

Ebb-and-flow. Another recent work called Ebb-and-flow [41] also takes FBFT's approach of having two commit rules for two different models and leaving the responsibility to the clients to choose between them (though its primary motivation is to support dynamic availability). Interestingly, if we constrain all clients in Ebb-and-flow to choose the partially synchronous commit rule, then their protocol can be regarded as an MT-BFT protocol with $\beta_s < n/2$ and $\beta_a = \gamma_a = \gamma_s < n/3$, which has a higher but not optimal synchronous safety threshold.

7 CONCLUSION AND FUTURE WORK

We introduce multi-threshold BFT, a generalized version of the BFT problem, which defines fault thresholds separately for safety and liveness under synchrony and asynchrony (or partial-synchrony), respectively. Our optimal protocols have strictly stronger fault tolerance than classic BFT protocols. We also present a general framework to upgrade existing protocols to achieve optimal synchronous safety with minimum overhead. Our customized protocol in Section 4 that allows generic (i.e., other than $\beta_a = \gamma_a = \lfloor \frac{n-1}{3} \rfloor$) parameter choices works only in the psync-sync model. Therefore, the tightness of the fault bounds for MT-BFT SMR in the async-sync model is still open and interesting future work.

ACKNOWLEDGMENTS

We thank our shepherd Julian Loss and the anonymous reviewers at ACM CCS 2021 for their helpful feedback. This work was supported in part by gifts from Novi and VMware.

REFERENCES

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. 2019. Synchronous Byzantine Agreement with Expected $O(1)$ Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience. In *Financial Cryptography and Data Security (FC)*. Springer, 320–334.

[2] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. 2020. Sync HotStuff: Simple and Practical Synchronous State Machine Replication. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 106–118.

[3] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically optimal validated asynchronous byzantine agreement. In *ACM Symposium on Principles of Distributed Computing (PODC)*. 337–346.

[4] 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).

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

[6] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. [n.d.]. State machine replication in the Libra Blockchain.

[7] Erica Blum, Jonathan Katz, and Julian Loss. 2019. Synchronous consensus with optimal asynchronous fallback guarantees. In *Theory of Cryptography Conference (TCC)*. Springer, 131–150.

[8] Erica Blum, Jonathan Katz, and Julian Loss. 2020. Network-Agnostic State Machine Replication. *arXiv preprint arXiv:2002.03437* (2020).

[9] Erica Blum, Chen-Da Liu-Zhang, and Julian Loss. 2020. Always have a backup plan: fully secure synchronous MPC with asynchronous fallback. In *Annual International Cryptology Conference (CRYPTO)*. Springer, 707–731.

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

[11] Mike Burrows. 2006. The Chubby lock service for loosely-coupled distributed systems. In *7th Symposium on Operating Systems Design and Implementation (OSDI)*. 335–350.

[12] Vitalik Buterin and Virgil Griffith. 2017. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437* (2017).

[13] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference (CRYPTO)*. Springer, 524–541.

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

[15] T-H Hubert Chan, Rafael Pass, and Elaine Shi. 2018. PiLi: An Extremely Simple Synchronous Blockchain. *IACR Cryptology ePrint Archive, Report 2018/980* (2018).

[16] J.P.Morgan Chase. 2018. Quorum Whitepaper. (2018). https://github.com/jpmorganchase/quorum/blob/master/docs/QuorumWhitepaperv0.2.pdf.

[17] Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. 2012. On the (limited) power of non-equivocation. In *ACM Symposium on Principles of Distributed Computing (PODC)*. 301–308.

[18] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. 2009. Upright cluster services. In *22nd ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*. 277–290.

[19] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. 2013. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)* 31, 3 (2013), 1–22.

[20] Danny Dolev and H. Raymond Strong. 1983. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput.* 12, 4 (1983), 656–666.

[21] Sisi Duan, Michael K Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT made practical. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2028–2041.

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

[23] Paul Feldman and Silvio Micali. 1988. Optimal algorithms for Byzantine agreement. In *20th Annual ACM Symposium on Theory of Computing (STOC)*. 148–161.

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

[25] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. 2019. SBFT: a scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 568–580.

[26] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster asynchronous bft protocols. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 803–818.

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

[28] Martin Hirt, Ard Kastrati, and Chen-Da Liu-Zhang. 2020. Multi-Threshold Asynchronous Reliable Broadcast and Consensus. In *24th International Conference on Principles of Distributed Systems (OPODIS)*.

[29] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. 2010. ZooKeeper: Wait-free Coordination for Internet-scale Systems.. In *USENIX annual technical conference*, Vol. 8.

[30] Jonathan Katz and Chiu-Yuen Koo. 2009. On expected constant-round protocols for byzantine agreement. *J. Comput. System Sci.* 75, 2 (2009), 91–112.

[31] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2007. Zyzzyva: speculative byzantine fault tolerance. In *21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*. 45–58.

[32] Michael Kumhof and Clare Noone. 2018. Central bank digital currencies-design principles and balance sheet implications. (2018).

[33] Leslie Lamport. 2019. The part-time parliament. In *Concurrency: the Works of Leslie Lamport*. 277–317.

[34] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* 4, 3 (1982), 382–401.

[35] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolić. 2016. XFT: Practical fault tolerance beyond crashes. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 485–500.

[36] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *ACM Symposium on Principles of Distributed Computing (PODC)*. 129–138.

[37] Dahlia Malkhi, Kartik Nayak, and Ling Ren. 2019. Flexible byzantine fault tolerance. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1041–1053.

[38] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 31–42.

[39] Atsuki Momose, Jason Paul Cruz, and Yuichi Kaji. 2020. Hybrid-BFT: Optimistically Responsive Synchronous Consensus with Optimal Latency or Resilience. *IACR Cryptology ePrint Archive, Report 2020/406* (2020).

[40] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[41] Joachim Neu, Ertem Nusret Tas, and David Tse. 2020. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. *arXiv preprint arXiv:2009.04987* (2020).
[42] Rafael Pass and Elaine Shi. 2017. Hybrid consensus: Efficient consensus in the permissionless model. In *International Symposium on Distributed Computing (DISC)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
[43] Rafael Pass and Elaine Shi. 2018. Thunderella: Blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Springer, 3–33.
[44] Marshall Pease, Robert Shostak, and Leslie Lamport. 1980. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)* 27, 2 (1980), 228–234.
[45] Fred B Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)* 22, 4 (1990), 299–319.
[46] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. 2020. On the Optimality of Optimistic Responsiveness. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 839–857.
[47] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. Hotstuff: Bft consensus with linearity and responsiveness. In *ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 347–356.