Probabilistic Byzantine Fault Tolerance

Diogo Avelãs*
LASIGE, Faculdade de Ciências,
Universidade de Lisboa, Portugal
dinoroba@proton.me

Hasan Heydari*
LASIGE, Faculdade de Ciências,
Universidade de Lisboa, Portugal
hheydari@ciencias.ulisboa.pt

Eduardo Alchieri
Universidade de Brasilia,
Brasil
alchieri@unb.br

Tobias Distler
Friedrich-Alexander-Universität
Erlangen-Nürnberg, Germany
distler@cs.fau.de

Alysson Bessani
LASIGE, Faculdade de Ciências,
Universidade de Lisboa, Portugal
anbessani@ciencias.ulisboa.pt

ABSTRACT

Consensus is a fundamental building block for constructing reliable and fault-tolerant distributed services. Many Byzantine fault-tolerant consensus protocols designed for partially synchronous systems adopt a pessimistic approach when dealing with adversaries, ensuring safety even under the worst-case scenarios that adversaries can create. Following this approach typically results in either an increase in the message complexity (e.g., PBFT) or an increase in the number of communication steps (e.g., HotStuff). In practice, however, adversaries are not as powerful as the ones assumed by these protocols. Furthermore, it might suffice to ensure safety and liveness properties with high probability. To accommodate more realistic and optimistic adversaries and improve the scalability of BFT consensus, we propose ProBFT (Probabilistic Byzantine Fault Tolerance). ProBFT is a leader-based probabilistic consensus protocol with a message complexity of $O(n\sqrt{n})$ and an optimal number of communication steps that tolerates Byzantine faults in permissioned partially synchronous systems. It is built on top of well-known primitives, such as probabilistic Byzantine quorums and verifiable random functions. ProBFT guarantees safety and liveness with high probability even with faulty leaders, as long as a supermajority of replicas is correct and using only a fraction (e.g., 20%) of messages exchanged in PBFT. We provide a detailed description of ProBFT's protocol and its analysis.

CCS CONCEPTS

• **Computing methodologies → Distributed algorithms**; • **Computer systems organization → Dependable and fault-tolerant systems and networks**.

KEYWORDS

Byzantine fault-tolerance, Consensus, Probabilistic protocols, Byzantine quorum systems

ACM Reference Format:
Diogo Avelãs, Hasan Heydari, Eduardo Alchieri, Tobias Distler, and Alysson Bessani. 2024. Probabilistic Byzantine Fault Tolerance. In *ACM Symposium on Principles of Distributed Computing (PODC '24), June 17–21, 2024, Nantes, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3662158.3662810

1 INTRODUCTION

Context. Consensus is a fundamental building block for constructing reliable and fault-tolerant distributed services, where participants agree on a common value from the initially proposed values. This problem is primarily used to implement state machine replication [16, 50, 51] and atomic broadcast [26, 36, 48], and attracted considerable attention in the last few years, mainly due to its significant role in blockchains [5, 43, 55] and decentralized payment systems (e.g., [35]). Due to its importance and widespread applicability, consensus has been extensively studied in diverse system models, considering various synchrony assumptions and a spectrum of failure models, ranging from fail-stop to Byzantine, across permissioned and permissionless settings [11, 33, 47, 53].

Many Byzantine fault-tolerant (BFT) consensus protocols (e.g., PBFT [13] and HotStuff [56][1]) adopt a pessimistic approach when dealing with Byzantine participants and adversaries, ensuring the safety of protocols even when Byzantine participants behave completely arbitrarily under the worst-case scenarios that corruption and scheduling adversaries can create. That is, they typically consider adversaries that choose a corruption strategy and manipulate the delivery time of messages based on the entire history of the system, including the past and current states of replicas, as well as their exchanged messages.

Protocols following the pessimistic approach are built on the idea of (1) making non-revocable decisions by considering the opinions of a quorum of replicas, as opposed to relying solely on a single replica, and (2) requiring the quorums used for making decisions to intersect in at least one correct replica. Although effective, ensuring deterministic quorum overlaps poses inherent challenges in achieving both resource efficiency and high performance. Some protocols (e.g., PBFT) approach this conflict by opting for low latency and applying message-exchange patterns with quadratic message

*The two first authors contributed equally to this research.

[1] Strictly speaking, PBFT [13] and HotStuff [56] are state machine replication protocols, not consensus protocols. In this paper, when we refer to PBFT and HotStuff, we specifically discuss the single-shot versions of these protocols presented in [7], which address the consensus problem.

PODC '24, June 17–21, 2024, Nantes, France

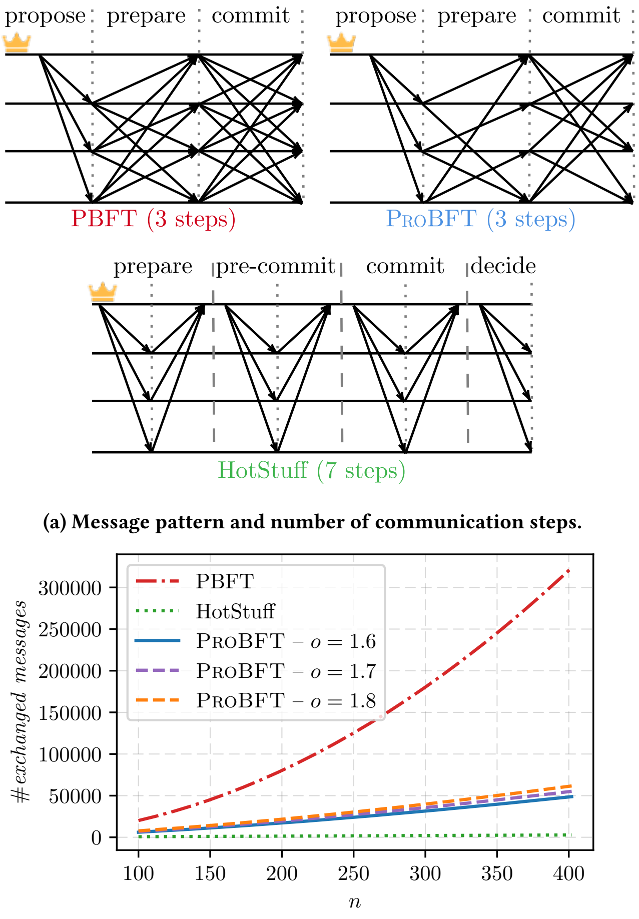
complexity. However, this can be prohibitively expensive, especially for BFT systems with a large number of replicas. Other protocols (e.g., HotStuff) aim at reducing message complexity at the cost of adding extra communication steps. Unfortunately, this approach leads to increased end-to-end response times.

In practice, however, implementing punishment mechanisms, like those used in existing Proof-of-Stake blockchains (e.g., [8, 12]), can make it costly for Byzantine participants to deviate from its specification, knowing that their actions may lead to detection and subsequent punishment. Besides, adversaries are not as powerful as the ones assumed by protocols following the pessimistic approach. Accordingly, in many real-world applications, it is sufficient to assume a static corruption adversary, which chooses a corruption strategy at the beginning of the execution of a consensus instance, as well as an adversarial scheduling that manipulates the delivery time of messages independently of the sender's identifier and if it is faulty or not. Given this, ensuring safety and liveness with a high probability might be acceptable in many practical scenarios. This paper describes a new protocol for these less pessimistic practical scenarios that requires less message exchanges and still keeps optimal latency. This protocol is called ProBFT (Probabilistic Byzantine Fault Tolerance).

Overview of ProBFT. ProBFT is a BFT leader-based consensus protocol that operates in permissioned partially synchronous systems and probabilistically ensures liveness and safety properties. It achieves the optimal *good-case latency* of three communication steps [1], just like PBFT, albeit with a message complexity of $O(n\sqrt{n})$. Figure 1 compares ProBFT, PBFT, and HotStuff in terms of the number of communication steps and exchanged messages for different numbers of replicas. ProBFT's resource efficiency and scalability improvements are enabled by a unique combination of building blocks that are usually not employed in traditional BFT protocols, including probabilistic quorum systems [37], a mechanism to configure the degree of communication redundancy, and a verifiable random function (VRF) [38].

Like PBFT and HotStuff, ProBFT operates in a sequence of views, each having a designated leader responsible for proposing a value. The protocol consists of two modes of execution – normal case and view-change. The normal case starts when the leader broadcasts its proposal through a Propose message. Since the leader might be Byzantine and send distinct proposals to different replicas, non-Byzantine replicas need to communicate with each other to check that they received the same proposal. With this aim, upon receiving a Propose message, a correct replica multicasts the proposal to a sample of $o \times q$ distinct replicas taken uniformly at random from the set of replicas, where $q = O(\sqrt{n})$ and $o > 1$ is a real constant. Upon receiving Prepare messages from a probabilistic quorum with size q, a correct replica multicasts a Commit message to another random sample composed of $o \times q$ distinct replicas. Upon receiving Commit messages from a probabilistic quorum with size q, a correct replica decides on the proposed value.

ProBFT's normal case execution relies on probabilistic quorums to solve consensus. That is, in contrast to traditional BFT protocols (e.g., PBFT, HotStuff, and randomized protocols like [10, 41]), we abandon the requirement of quorums strictly having to intersect and instead only aim at quorums overlapping with high probability. As a key benefit, this strategy enables us to keep the number of communication steps at a minimum while significantly reducing quorum sizes, thereby improving resource consumption and scalability. Specifically, for a system with n replicas, ProBFT employs probabilistic quorums of size $q = l\sqrt{n}$, with $l \geq 1$ being a configurable, typically small constant [37]. For example, for $l = 2$ and $n = 100$, a replica can make progress after receiving 20 matching messages from different replicas, which is a significant reduction when compared with the 67 messages necessary in PBFT.



(a) Message pattern and number of communication steps.

(b) Number of exchanged messages.

Figure 1: Comparing the normal case of three consensus protocols – PBFT, ProBFT, and HotStuff – regarding the number of communication steps and message complexity.

Since a comparably small number of messages is sufficient to advance a phase in ProBFT, to offer resilience against Byzantine behavior, it is crucial to prevent faulty replicas from manipulating the decisions in probabilistic quorums (e.g., by flooding the system with their own messages). In ProBFT, this is achieved by delegating the selection of message recipients to a globally known VRF. That is, in the protocol phases relying on probabilistic quorums, replicas do not freely pick the destinations of their messages but instead are required to send the messages to the specific group of recipients determined by the VRF.

Although novel, ProBFT is heavily based on PBFT, being thus somewhat simple to understand and implement. Nonetheless, the probabilistic nature of the protocol makes its analysis far from trivial. More specifically, the main challenges encountered in analyzing ProBFT are:

- In the analysis of probabilistic algorithms, we often deal with *independent* events, enabling the application of powerful and more straightforward bounds like the Chernoff bounds [42]. However, in ProBFT, the probability of forming probabilistic quorums by replicas is *dependent*. That is, as replicas multicast their Prepare (resp., Commit) messages to random samples, knowing that a replica has received Prepare (resp., Commit) messages from a probabilistic quorum decreases the chance of other replicas to receive Prepare (resp., Commit) messages from a probabilistic quorum. This dependency complicates ProBFT's analysis.
- The probability of deciding a value by a replica depends on the number of replicas that multicast matching Commit messages. Additionally, the number of replicas that multicast their Commit messages depends on the number of replicas that multicast their Prepare messages. This dual dependency layer for computing the probability of deciding a value by a replica adds complexity to the analysis of ProBFT.
- A Byzantine leader might send multiple proposals to violate safety. Rather than examining each possible case individually, we find the optimal behavior for a Byzantine leader, considering that it intends to maximize the probability of safety violation.

In summary, besides the design of ProBFT, the main technical contribution of this paper is the analysis of the protocol, represented by the following theorem:

Theorem 1 (Informal main result)**.** ProBFT guarantees liveness with probability 1 and safety with a probability of $1 - exp(-\Theta(\sqrt{n}))$.

Paper organization. The remainder of the paper is organized as follows. Section 2 introduces our system model and describes the key techniques employed in ProBFT. Section 3 describes the ProBFT protocol. Section 4 presents the correctness proofs of ProBFT. Section 5 presents a numerical analysis of the protocol. Finally, Sections 6 and 7 discuss related work and conclude the paper, respectively.

2 PRELIMINARIES

2.1 System Model

We consider a distributed system composed of a finite set Π of n processes, which we call replicas, among which up to $f < n/3$ might be subject to Byzantine failures [29] and not behave according to the protocol specification. A non-faulty replica is said to be *correct*. During execution, we denote by Π_C and Π_F the sets of correct and faulty replicas, respectively. The system is partially synchronous [7, 20] in which the network and replicas may operate asynchronously until some *unknown global stabilization time* GST, after which the system becomes synchronous, with *unknown time bounds for communication and computation.*

We assume that each replica has a unique ID, and it is infeasible for a faulty replica to obtain additional IDs to launch a *Sybil attack* [17]. We consider a *static corruption adversary*, i.e., Π_F is fixed at the beginning of execution by the adversary. Byzantine replicas may collude and coordinate their actions. It is important to note that while Byzantine replicas may be aware of Π_F, the correct replicas are unaware of Π_F and only know the value of f. Furthermore, we assume an adversarial scheduler that manipulates the delivery time of messages independent of the sender's identifier, its past and current states, and whether it is Byzantine or not.

Each replica signs outgoing messages with its private key and only processes an incoming message if the message's signature can be verified using the sender's public key. We denote $\langle \mathrm{T}, m \rangle_i$ as a message of type T with content m signed by replica i. We assume that the distribution of keys is performed before the system starts. At run-time, the private key of a correct replica never leaves the replica and, therefore, remains unknown to faulty replicas. In contrast, faulty replicas might learn the private keys of other faulty replicas. In practical settings, it is a standard assumption that the adversary does not have unlimited computational resources; therefore, they cannot break cryptographic primitives.

2.2 Consensus

We assume there is an application-specific `valid` predicate to indicate whether a value is acceptable [7, 9, 51]. Assuming that each correct replica proposes a valid value, any protocol that solves probabilistic consensus satisfies the following properties:

- **Validity.** The value decided by a correct replica satisfies the application-defined `valid` predicate.
- **Probabilistic Agreement.** Any two correct replicas decide on different values with probability ρ depending on the number of existing Byzantine replicas and quorum/sample sizes.
- **Probabilistic Termination.** Every correct replica decides with probability 1.

The first two properties are safety properties, while the last one is a liveness property.

2.3 Single-shot PBFT

Since ProBFT follows a structure very similar to PBFT [13], we briefly review the single-shot version of the latter considering the use of a *synchronizer* [7]. This version is a leader-based consensus protocol that operates in a succession of views produced by the synchronizer, each having a designated leader defined in a round-robin way. Each protocol view consists of three steps and works as follows (see Figure 2):

- **Propose (or pre-prepare) phase.** The leader of view v is responsible for proposing a value to the other replicas. With this aim, the leader broadcasts a value through a Propose message. A correct leader must carefully choose the value to ensure that if a correct replica has decided on a value in a previous view, it will propose the same value.
- **Prepare phase.** Upon receiving a Propose message from a replica i in view v, a correct replica broadcasts a Prepare message if i is the leader of v, and the proposed value is a valid proposal.
- **Commit phase.** Upon receiving Prepare messages from a quorum of replicas, a correct replica broadcasts a Commit message.

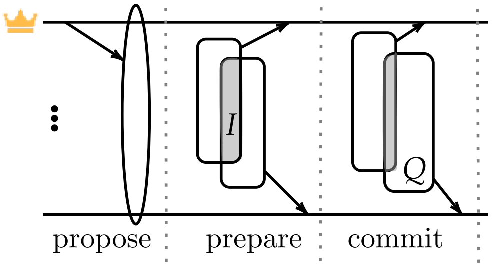


Figure 2: Overview of PBFT. Each correct replica broadcasts its Prepare and Commit messages. The size of any quorum is $|Q| = \lceil (n+f+1)/2 \rceil$. The set of replicas I in the intersection of two quorums contains at least one correct replica.

A replica decides the value proposed by the leader upon receiving Commit messages from a quorum of replicas. There are multiple scenarios where replicas cannot decide a value in a view v, like when the leader is Byzantine and remains silent. In those scenarios, the synchronizer transitions the view from v to $v+1$, changing the designated leader.

2.4 Verifiable Random Function

A *verifiable random function* (VRF) [23, 24] enables the random selection of a subset from a given set, ensuring that the selection process is verifiable and secure. We assume a globally known VRF that provides the following two operations:

- **VRF_prove($K_{p,i}$, z, s) ⇒ S_i, P_i.** Given the private key $K_{p,i}$ for a replica i, a seed z, and a positive integer s, VRF_prove selects a sample S_i containing the IDs of s distinct replicas uniformly at random. Along with S_i, this operation returns a proof P_i, enabling other replicas to verify whether the sample was obtained using this operation.
- **VRF_verify($K_{u,i}$, z, s, S_i, P_i) ⇒ bool.** Given the public key $K_{u,i}$ of replica i, a seed z, a positive integer s, a sample S_i, and its associated proof P_i, VRF_verify determines whether S_i is a valid sample generated using VRF_prove with the given parameters. It returns true if the sample and proof are valid and false otherwise.

The VRF should provide the following guarantees [24]:

- **Uniqueness.** A computationally limited adversary must not be able to produce two different proofs P_i and P'_i for the same input parameters $K_{u,i}$, z, and s.
- **Collision resistance.** Even when a private key is compromised, it should be infeasible for an adversary to find two distinct seeds z and z' for which the VRF_prove returns the same sample.
- **Pseudorandomness.** For an adversarial verifier without knowledge of the proof, the corresponding sample should be indistinguishable from a randomly selected set of replica IDs.

3 PROBFT

3.1 Overview

ProBFT is a leader-based probabilistic consensus protocol that operates in a succession of views produced by the synchronizer. In ProBFT, illustrated in Figure 3, one of the replicas is assigned the role of *leader* in each view, meaning that this replica is in charge of proposing a value. ProBFT's consensus process comprises three phases of message exchange between replicas – *propose*, *prepare*, and *commit*, just like in PBFT. In the *propose* phase, the leader proposes a value by broadcasting it to other replicas, which is then agreed on in the subsequent *prepare* and *commit* phases.

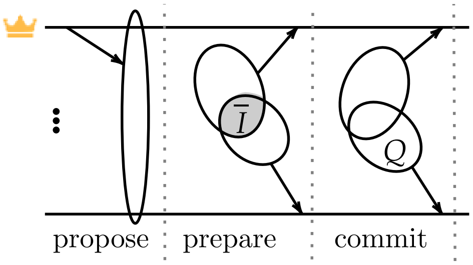


Figure 3: Overview of ProBFT. The size of any probabilistic quorum is $q = |Q| = O(\sqrt{n})$. Each correct replica multicasts its Prepare and Commit messages to random samples of sizes $o \times q$, where o is a constant. The set of replicas $\bar{I}$ in the intersection of two probabilistic quorums contains at least one correct replica with high probability.

Given the linear message complexity of the *propose* phase, we restrict the utilization of probabilistic quorums to the two remaining phases. That is, in each of the *prepare* and *commit* phases, any replica relies on the VRF to determine a subset of replicas with size $o \times q$ to whom its messages should be sent, where $o > 1$ is a constant. Any replica progresses by receiving messages from a probabilistic quorum with size $q = l\sqrt{n}$, being l also a small constant (e.g., 2). The constant o defines how large the random subset of replicas contacted on each phase by each replica is when compared with the probabilistic quorum size. Bigger values of o increase the probability of forming a probabilistic quorum (with q replicas), increasing the chance of the protocol to terminate (see Section 4), albeit generating more messages (see Figure 1b). As a result, in contrast to the $O(n^2)$ communication complexity associated with traditional protocols such as PBFT, ProBFT's message complexity for the *prepare* and *commit* phases is $O(n\sqrt{n})$.

As seed for computing its recipient sample with the VRF, a replica is required to use $v \,||\, \mathrm{T}$, which is a concatenation of the current view v, and an identifier T representing the phase/message type ("prepare" for Prepare and "commit" for Commit). Each replica starts a *prepare* phase upon receiving the leader's proposal and a *commit* phase upon forming a probabilistic quorum, i.e., receiving q matching messages. Forcing replicas to involve the VRF this way and thereby apply deterministic seeds has the following key benefits:

(1) With the inputs of VRF_prove being dictated by the protocol, faulty replicas cannot control their recipient sample for a particular view and phase. Consequently, a faulty replica cannot deliberately favor a certain subset of replicas, for example, in support of a faulty leader trying to trick these replicas into forming a probabilistic quorum for a specific value.
(2) Since a replica's recipient samples are computed from its private key (see Section 2.4), faulty replicas cannot predict the

individual samples of correct replicas in advance. That is, at the start of a view, faulty replicas do not know the upcoming *prepare* and *commit* phase message-exchange patterns between correct replicas, thereby making it inherently difficult for them to identify promising attack targets (e.g., correct replicas that are included in none or only in a few recipient samples of other correct replicas).

(3) With the *prepare* and *commit* phase recipient samples usually differing (due to the use of the phase parameter in the seed), correct replicas are more likely to observe the misbehavior of a faulty leader. For example, if a faulty leader performs equivocation by proposing different values to different subsets of replicas, then the phase-specific recipient samples increase the probability of a correct replica learning about the existence of contradictory proposals.

In summary, the use of the VRF for selecting recipient samples in the *prepare* and *commit* phases significantly strengthens ProBFT's resilience against malicious behaviors.

3.2 Protocol Specification

Algorithm 1 presents ProBFT specification. This description assumes a synchronizer exactly like the one presented in [7]. The protocol proceeds in a series of views, with each new view having a fixed leader responsible for proposing a value to be decided. Every replica in the system can determine the leader for a view v with the `leader` predicate.

$$\texttt{leader}(v) = (v - 1 \ mod \ n) + 1$$

Upon receiving a notification from the synchronizer to transition to view v, a replica stores v in a variable *curView* and sets a flag *voted* to `false` to record that it has not yet received any proposal from the leader in the current view. If $v = 1$, the leader is free to broadcast its proposal (line 7). However, for other views, a correct leader must be careful in choosing its proposal because if a correct replica has already decided on a value in a prior view, the leader is obligated to propose the same value. To facilitate this, upon entering a view $v > 1$, a correct replica sends a NewLeader message to the leader of v, providing information about the latest value it accepted in a prior view (line 9). Any message exchanged in the protocol is tagged with the sender's view. A receiver will only accept a message if its own view stored in the *curView* variable matches the view of the sender.

For any view $v > 1$, the leader of v waits until it receives NewLeader messages from a *deterministic quorum* of replicas. After computing its proposal, the leader broadcasts the proposal, along with some supporting information, in a Propose message (lines 11-16). After presenting the rest of the protocol, we will describe the process for computing the proposal. Since a Byzantine leader may send different proposals to different replicas, correct replicas need to communicate with others to ensure they have received the same proposal. With this aim, correct replicas process the leader's proposal in two phases – *prepare* and *commit*.

Upon receiving a Propose message m containing a proposal x from a replica j in view v, a correct replica i starts the *prepare* phase if it is currently in view v, it has not processed a Propose message in this view, and message m satisfies the `safeProposal` predicate (also explained later), which ensures that a Byzantine leader cannot reverse decisions reached in a prior view (with high probability). The replica then stores x in *curVal* and sets voted to `true` (line 18). Afterward, replica i uses the VRF to select a random sample S_P to which it sends a Prepare message.

Algorithm 1 ProBFT – replica i.

```
upon newView($v$)
 1: $curView \leftarrow v$
 2: $curVal \leftarrow \bot$
 3: $voted \leftarrow$ false
 4: $blockView \leftarrow$ false
 5: $proposal \leftarrow \bot$
 6: if $curView = 1 \land i = \texttt{leader}(curView)$
 7:     broadcast $\langle \text{Propose}, \langle curView, \texttt{myValue}() \rangle_i, \bot \rangle_i$
 8: else if $curView > 1$
 9:     send $\langle \text{NewLeader}, curView, preparedView, preparedVal, cert \rangle_i$
                                                        to leader($curView$)
upon receiving $\{\langle \text{NewLeader}, v, view_j, val_j, cert_j \rangle_j : j \in Q\} =$
    $M$ from a deterministic quorum $Q$
10: pre: $curView = v \land i = \texttt{leader}(v) \land$
    $(\forall m \in M : \texttt{validNewLeader}(m))$
11: $v_{max} \leftarrow \max\{view_j : \langle \text{NewLeader}, v, view_j, \_, \_ \rangle_j \in M\}$
12: $val_{max} = \texttt{mode}\{val_j : \langle \text{NewLeader}, v, v_{max}, val_j, \_ \rangle_j \in M\}$
13: if $val_{max} \neq \bot$
14:     broadcast $\langle \text{Propose}, \langle v, val_{max} \rangle_i, M \rangle_i$
15: else
16:     broadcast $\langle \text{Propose}, \langle v, \texttt{myValue}() \rangle_i, M \rangle_i$
upon receiving $\langle \text{Propose}, \langle v, x \rangle_j, \_ \rangle_j = m$
17: pre: $blockView =$ false $\land$ $curView = v \land voted =$ false $\land$
    $\texttt{safeProposal}(m)$
18: $curVal, voted, proposal \leftarrow x,$ true$, m$
19: $S_p, P_p \leftarrow \texttt{VRF\_prove}(K_{p,i}, v \mid\mid \text{``prepare''}, o \times q)$
20: send $\langle \text{Prepare}, \langle v, x \rangle_j, S_p, P_p \rangle_i$ to $S_p$
upon receiving $\{\langle \text{Prepare}, \langle v, x \rangle_*, S, P \rangle_j : j \in Q\} = C$ from a
    probabilistic quorum $Q$
21: pre: $blockView =$ false $\land$ $curView = v \land curVal = x \land$
    $voted =$ true $\land$ $(\forall \langle \_, \_, S, P \rangle_j \in C : i \in S \land$
    $\texttt{VRF\_verify}(K_{u,j}, v \mid\mid \text{``prepare''}, o \times q, S, P))$
22: $preparedVal, preparedView, cert \leftarrow curVal, curView, C$
23: $S_c, P_c \leftarrow \texttt{VRF\_prove}(K_{p,i}, v \mid\mid \text{``commit''}, o \times q)$
24: send $\langle \text{Commit}, \langle v, x \rangle_*, S_c, P_c \rangle_i$ to $S_c$
upon receiving $\{\langle \text{Commit}, \langle v, x \rangle_*, S, P \rangle_j : j \in Q\} = M$ from a
    probabilistic quorum $Q$
25: pre: $blockView =$ false $\land$ $preparedVal = x \land$
    $curView = preparedView = v \land (\forall \langle \_, \_, S, P \rangle_j \in M : i \in S \land$
    $\texttt{VRF\_verify}(K_{u,j}, v \mid\mid \text{``commit''}, o \times q, S, P))$
26: decide($curVal$)
upon receiving $\langle \_, \langle v, x \rangle_j, \ldots \rangle_* = m$
27: pre: $blockView =$ false $\land$ $curView = v \land j = \texttt{leader}(v) \land$
    $voted =$ true $\land$ $curVal \neq x$
28: $blockView \leftarrow$ true
29: broadcast $m$, broadcast $proposal$
```

A correct replica waits until receiving a set C of Prepare messages from a probabilistic quorum. We call this set of messages a *prepared certificate* for a proposed value x in a view v if it satisfies the following predicate:

$$\begin{aligned}&\texttt{prepared}(C, v, x, j) \iff\\ &\quad \exists Q : |Q| = q \land C = \{\langle \textsc{Prepare}, \langle v, x\rangle_i, S_k, P_k\rangle_k : k \in Q\} \land\\ &\quad i = \texttt{leader}(v) \land (\forall \langle\_, \_, S_k, P_k\rangle_k \in C : j \in S_k \land\\ &\qquad \texttt{VRF\_verify}(K_{u,k}, v \mid\mid \text{“prepare”}, o \times q, S_k, P_k))\end{aligned}$$

Once a replica j creates a prepared certificate for a value x in a view v (i.e., j prepares x), it stores x, v, and this certificate in *preparedVal*, *preparedView*, and *cert*, respectively. Afterward, the replica generates a new random sample of replicas S_c to whom it will multicast a Commit message (lines 22-24). Every correct replica that multicasts a Commit message enters the *commit* phase. It then waits until receiving Commit messages from a probabilistic quorum. It is worth noting that a correct replica neither sends a Commit message nor processes a received Commit message (line 25) if it has not yet prepared a value. After observing a quorum of Commit messages, a correct replica with a prepared certificate decides on the proposed value (line 26).

Recall that when the synchronizer triggers a `newView` notification in a replica for a view greater than one, the replica sends a NewLeader message to the new leader. If a replica has created a prepared certificate in a prior view, it sends that certificate in the NewLeader message. This allows the leader to generate its proposal based on a quorum of well-formed NewLeader messages that can be checked using the following predicate:

$$\begin{aligned}&\texttt{validNewLeader}(\langle \textsc{NewLeader}, v, \mathit{view}, \mathit{val}, \mathit{cert}\rangle_j) \iff\\ &\qquad \mathit{view} < v \land \mathit{view} \neq 0 \Rightarrow \texttt{prepared}(\mathit{cert}, \mathit{view}, \mathit{val}, j)\end{aligned}$$

The leader chooses its proposal by selecting the value prepared in the most recent view by more replicas (lines 11-12). If no such prepared values exist, it uses its own proposal provided by the function `myValue()`. Since a faulty leader may not follow this rule, it is essential for the correct replicas to validate that the leader adheres to this selection rule for proposals. With this aim, the leader's Propose message contains the NewLeader messages received by the leader, in addition to the proposal. A correct replica checks the validity of the proposed value by redoing the leader's computation using the following predicate:

$$\begin{aligned}&\texttt{safeProposal}(\langle \textsc{Propose}, \langle v, x\rangle_j, M\rangle_j) \iff\\ &\quad v \geq 1 \land j = \texttt{leader}(v) \land \texttt{valid}(x) \ \land (v = 1 \lor\\ &\qquad (|M| \geq \lceil (n + f + 1)/2 \rceil \land (\forall m \in M : \ \texttt{validNewLeader}(m)) \land\\ &\qquad\quad (\exists v_{max} = \texttt{max}\{\mathit{view}_k : \langle \textsc{NewLeader}, v, \mathit{view}_k, \_, \_\rangle_k \in M\} \land\\ &\qquad\qquad x = \texttt{mode}\{\mathit{val}_k : \langle \textsc{NewLeader}, v, v_{max}, \mathit{val}_k, \_\rangle_k \in M\})))\end{aligned}$$

When a correct replica detects that the leader is faulty, i.e., receiving messages from any replica with different proposals signed by the leader, it instantly blocks the current view and waits for the synchronizer to trigger a new view (lines 27-29). Besides, it informs other replicas about this misbehavior. It is important to emphasize that informing other replicas is necessary only when the leader is Byzantine and sends distinct proposals. Therefore, it does not impact the message complexity of the protocol when the leader is correct.

3.3 Message and Communication Complexities

ProBFT's message complexity is $O(n\sqrt{n})$, as computed based on four terms: $O(n)$ for NewLeader messages, $O(n)$ for Propose messages, $O(n\sqrt{n})$ for Prepare messages, and $O(n\sqrt{n})$ for Commit messages. Further, in ProBFT, for any view greater than one, a new leader sends a Propose message with a certificate containing a full (not probabilistic) quorum of NewLeader messages to all replicas. Each NewLeader message might contain a prepared certificate with a probabilistic quorum of Prepare messages. Hence, ProBFT's communication complexity is $O(n^2\sqrt{n})$. Note that ProBFT has this communication complexity only when a view-change occurs. In the first view, there is no need to send NewLeader messages to the leader, as avoided in practical instantiations of PBFT (e.g., [6, 14]). Therefore, ProBFT's best-case communication complexity is $\Omega(n\sqrt{n})$, contrary to PBFT, which still has $\Omega(n^2)$.

4 PROBFT PROOF OUTLINE

This section outlines the correctness proofs of ProBFT. More specifically, we show that ProBFT satisfies three properties of probabilistic consensus, i.e., Validity, Probabilistic Termination, and Probabilistic Agreement. Here we discuss the main arguments of the proofs and refer to the extended version [3] for the full proofs.

4.1 Validity

Recall that the Validity property states that the value decided by a correct replica satisfies the application-defined `valid` predicate. In ProBFT, before deciding a value x, a correct replica i must receive x as a proposal from the leader and verify its validity using the `safeProposal` predicate (line 17). In this predicate, several conditions must be satisfied, one of which is that $\texttt{valid}(x)$ must be *true*. Accordingly, a value decided by a correct replica is valid, satisfying the Validity property.

4.2 Probabilistic Termination

To demonstrate that ProBFT satisfies Probabilistic Termination, suppose there is a non-empty subset of correct replicas that have not decided a value by GST. Recall that ProBFT employs a round-robin mechanism for changing the leaders. Accordingly, among the replicas that have not made a decision by GST, there will be some correct replica i that performs the leader's role by proposing a value x in a view v. Note that all correct replicas receive such a proposal during v as the system is synchronous after GST and multicast their Prepare messages. We show that any correct replica decides x with a high probability during view v if it has not already decided a value. To do so, we first demonstrate that any correct replica receives Prepare messages from a probabilistic quorum of replicas with a high probability. The following theorem demonstrates that with a proper value of o, such an event occurs for a correct replica with a probability of at least $1 - exp(-\sqrt{n})$, i.e., forming such a quorum with high probability, even if all Byzantine replicas remain silent.

Theorem 2. Suppose each correct replica takes a sample composed of $o \times l\sqrt{n}$ distinct replicas, uniformly at random from Π, and multicasts a message to them, where $o \in [1, 3.732(n/(n-f))]$, and $l \geq 1$. Provided that a replica forms a probabilistic quorum upon receiving $l\sqrt{n}$ messages, the probability of forming such a quorum is at least $1 - exp(-\sqrt{n})$.

Recall that a correct replica prepares the value proposed by the leader upon receiving Prepapre messages from a probabilistic quorum. According to Theorem 2, a correct replica prepares the value proposed by a correct leader with high probability, so almost all correct replicas send Commit messages to their randomly selected samples. It is clear that the probability of receiving Commit messages from a probabilistic quorum is less than the probability of receiving Prepare messages from a probabilistic quorum. However, the following theorem demonstrates that every correct replica receives Commit messages from a probabilistic quorum with a high probability, resulting in deciding a value with a high probability.

Theorem 3. After GST, if the leader of view v is correct, then each correct replica decides a value in v with a probability of at least $1 - 2(n - f) \times exp\big(- \Theta(\sqrt{n})\big)$.

Proving the above theorem constitutes the most complex part of demonstrating that ProBFT satisfies the Probabilistic Termination property. This complexity arises from the following sources:

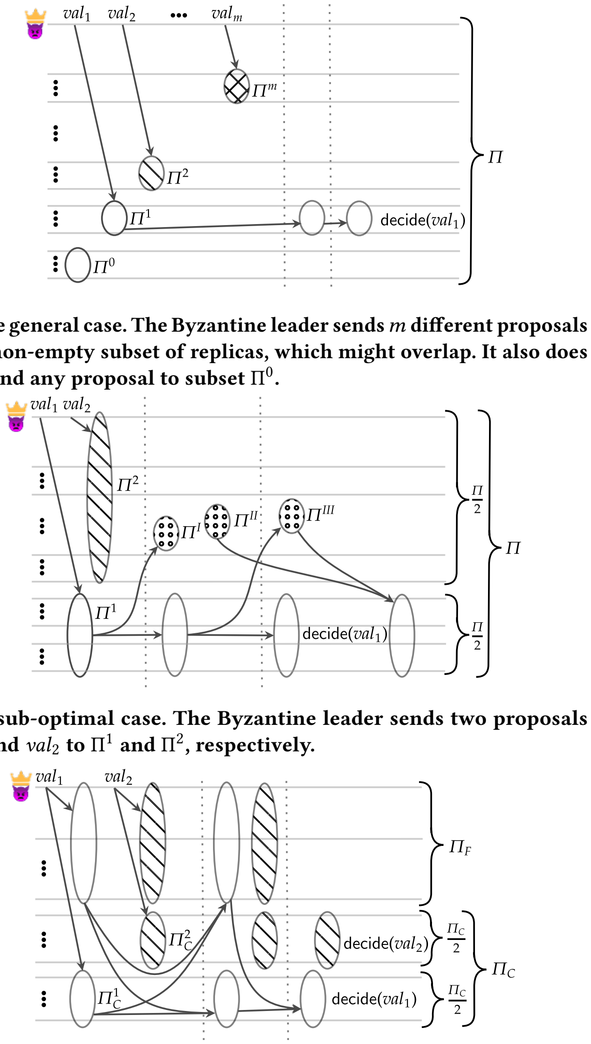
- The probability of forming probabilistic quorums by replicas is *dependent*. That is, as replicas multicast their Prepare (resp., Commit) messages to random samples, knowing that a replica has received Prepare (resp., Commit) messages from a probabilistic quorum decreases the chance of other replicas to receive Prepare (resp., Commit) messages from a probabilistic quorum. This dependency prevents us from directly using well-known and sharp bounds like the Chernoff bounds [42]. To circumvent this dependency, we use the notion of negative association [19], enabling us to leverage the Chernoff bounds.
- The number of replicas that receive Commit messages from probabilistic quorums *depends on* the number of replicas that receive Prepare messages from probabilistic quorums. We address this dual dependency inherent in computing the probability of deciding a value by a replica by conditioning the probability of forming quorums by Commit messages on the probability of forming quorums by Prepare messages.

In ProBFT, when the leader of view v is correct, a correct replica might not receive enough messages to form quorums, leading to not deciding a value in view v. According to Theorem 3, such an event happens with a low probability. However, as there are infinite views whose leaders are correct, each correct replica decides with probability 1.

Theorem 4 (Main liveness result)**.** In ProBFT, every correct replica eventually decides a value with probability 1.

4.3 Probabilistic Agreement

In ProBFT, different replicas may decide different values since quorum intersections are not guaranteed, but the protocol has to ensure that the probability of agreement violation is low. We begin by computing the probability of ensuring agreement within a view.



(a) The general case. The Byzantine leader sends m different proposals to m non-empty subset of replicas, which might overlap. It also does not send any proposal to subset Π^0.

(b) A sub-optimal case. The Byzantine leader sends two proposals val_1 and val_2 to Π^1 and Π^2, respectively.

(c) The optimal case. Given two sets $\Pi_C^1, \Pi_C^2 \subseteq \Pi_C$ with equal sizes, the Byzantine leader sends only two proposals val_1 and val_2 to $\Pi_C^1 \cup \Pi_F$ and $\Pi_C^2 \cup \Pi_F$, respectively.

Figure 4: Different scenarios in which a Byzantine leader can cause disagreements in a view.

Probabilistic Agreement in a view. In ProBFT, to cause disagreement in a view, it is required that multiple values are decided in the same view by multiple correct replicas. Such a situation only happens when the leader is Byzantine since correct leaders send a single proposal in their views.

There are many cases in which a Byzantine leader can compromise the agreement in a view. Rather than examining every possible case individually, we find the optimal behavior for a Byzantine leader, considering that it intends to maximize the probability of agreement violation. For this purpose, we consider the three cases illustrated in Figure 4. The first case is the most general one, which can be used to derive any possible situation. The second and third cases demonstrate specific situations. In the third case, the probability of agreement violation is greater than or equal to the probability of agreement violation in the second case and any other situation obtained from the first case.

- *The general case.* The Byzantine leader sends multiple proposals $val_1, \dots, val_m$, where $m \geq 2$. As a result, some replica(s) might receive one or even multiple proposals, and some others might not receive any proposals. This case is depicted in Figure 4a.
- *A sub-optimal case.* The Byzantine leader divides the set of replicas into two equally sized groups Π^1 and Π^2. It sends a proposal val_1 to Π^1 and another proposal val_2 to Π^2. This case is depicted in Figure 4b.
- *The optimal case.* The Byzantine leader makes a distinction between correct and Byzantine replicas. It divides the correct replicas into two equally sized groups – Π^1_C and Π^2_C. It sends a proposal val_1 to $\Pi^1_C \cup \Pi_F$ and another proposal val_2 to $\Pi^2_C \cup \Pi_F$. This case is depicted in Figure 4c.

In order to describe why the first case represents the most general situation, we need to present some notations. For each proposal val_i, where $1 \leq i \leq m$, we associate a set $\Pi^i \subset \Pi$ containing each replica p that receives val_i as a proposal from the leader. As the leader might send multiple proposals to a replica, any set Π^i might intersect with another set Π^j, where $1 \leq i < j \leq m$. We denote by Π^0 the set of replicas to which the leader does not send any proposal, but these replicas can receive messages from other replicas. Indeed, $\Pi \setminus \Pi^0$ contains every replica that can multicast Prepare and Commit messages. Furthermore, we denote the correct (resp. Byzantine) replicas within a set Π^i by Π^i_C (resp. Π^i_F).

It is essential to remark that a correct replica p to decide a value val_i requires to form a probabilistic prepare quorum P and a probabilistic commit quorum Q such that $P, Q \subseteq \Pi^i$; otherwise, p does not decide a value due to receiving two distinct values. Consequently, if replica $p \in \Pi^i$ decides a value, the value is val_i.

Using the first case, we can model any situation in the system when the Byzantine leader sends multiple proposals, as we do not impose any restrictions on the leader's behavior. For instance, replica p decides val_i when either (1) it forms a probabilistic prepare quorum and a probabilistic commit quorum, both composed of Byzantine replicas within Π^i, i.e., $P, Q \subseteq \Pi^i_F$, (2) it forms a probabilistic prepare quorum composed of Byzantine replicas and a probabilistic commit quorum composed of correct replicas, i.e., $P \subseteq \Pi^i_F$ and $Q \subseteq \Pi^i_C$, or (3) it forms a probabilistic prepare quorum composed of correct and Byzantine replicas and a probabilistic commit quorum composed of correct replicas, i.e., $P \subseteq \Pi^i$, $P \cap \Pi^i_C \cap \Pi^i_F \neq \emptyset$, and $Q \subseteq \Pi^i_C$.

Beyond the situations where a replica decides a value, the first case demonstrates situations where a replica does not decide a value because of receiving multiple proposals. For example, replica p does not decide a value when either (1) it receives multiple proposals from the leader, i.e., $p \in \Pi^i$ and $p \in \Pi^j$, where $1 \leq i < j \leq m$, (2) it receives at least two different proposals val_i and val_j such that val_i is received from the leader and val_j is received from one of the replicas sending its Prepare message, or (3) it receives at least two different proposals val_i and val_j such that val_i is received from a replica sending its Prepare message, while val_j is received from another replica sending its Commit message.

In order to grasp why the third case is optimal, consider the following observations:

(1) If the Byzantine leader sends different proposals to a correct replica p, p will detect the misbehavior of the leader and notify all replicas about it (line 29 in Algorithm 1). Hence, sending multiple proposals to p increases the probability that correct replicas avoid deciding some value(s). Note that there is no agreement violation when correct replicas avoid deciding a value. Hence, the Byzantine leader should send only *one* proposal to each correct replica, i.e., $\Pi^i_C \cap \Pi^j_C = \emptyset$, for any $i, j \in \{1, \dots, m\}$, as it intends to increase the probability of agreement violation. Consequently, in the optimal case, the number of proposals the leader sends is bounded by the number of correct replicas, i.e., $m \leq n - f$.
(2) The Byzantine leader should send *two* proposals to increase the probability of agreement violation. To show this result, we prove in Theorem 5 that sending m proposals instead of $m + 1$ proposals, where $m \geq 2$, increases the probability of agreement violation. Given the previous observation that states the leader sends at most $n - f$ proposals, we can now say that the leader prefers to send $n - f - 1$ proposals instead of sending $n - f$ proposals in the optimal case. Likewise, it prefers to send $n-f-2$ proposals instead of sending $n - f - 1$ proposals. Following this line of reasoning, we conclude that the Byzantine leader should send two proposals to increase the probability of agreement violation.
(3) The probability of forming a probabilistic quorum by a replica p for a proposal val increases by expanding the set of replicas that send val. This result is formally presented in Theorem 6. From the previous observation, we know that the leader should send two proposals. Now, we can say that the Byzantine leader should maximize the size of these two sets, resulting in the optimal case depicted in Figure 4c.

Theorem 5. Given a Byzantine leader who intends to send multiple proposals, consider the following two scenarios: (1) given non-empty sets $\Pi^1, \dots, \Pi^{m+1}$ of replicas, where $m \geq 2$, and $|\Pi^1| \leq |\Pi^2| \leq \dots \leq |\Pi^{m+1}|$, the leader sends a distinct proposal to each set, and (2) the leader merges two sets Π^1 and Π^2 to create a set $\Pi^{1,2}$ and sends m proposals to $\Pi^{1,2}, \Pi^3, \dots, \Pi^{m+1}$. The probability of agreement violation in the second scenario is greater than in the first scenario.

Theorem 6. Suppose any replica forms a quorum upon receiving q messages. Consider a set of r replicas, each of which takes a sample composed of $o \times q$ distinct replicas uniformly at random from Π, with the condition that $n < o \times r$, and sends a message to all sample members. The value of r and the probability of a replica forming a quorum are directly proportional.

Since the third case discussed above is optimal, i.e., the probability of compromising the agreement is maximized by a Byzantine leader when it divides the correct replicas into two equally sized groups, Π^1_C and Π^2_C, and sends a proposal val_1 to $\Pi^1_C \cup \Pi_F$ and another proposal val_2 to $\Pi^2_C \cup \Pi_F$, we only analyze this case. We prove that given a Byzantine leader who may send several proposals, the probability of agreement violation in a view under the worst-case scenario is bounded by $exp\big(-\Theta(\sqrt{n})\big)^4$ in the following theorem.

Theorem 7. Given a Byzantine leader who may send several proposals, the probability of agreement violation in a view under the worst-case scenario is at most $exp\big(-\Theta(\sqrt{n})\big)^4$.

Probabilistic Agreement with view change. We now consider the case of a view change. When referring to agreement within different views, we need to guarantee that if at least one correct replica decides on a proposal *val* in a view v, the probability that some correct replica decides on a different proposal *val′* in a view $v' > v$ is negligible. To guarantee this condition, we need to demonstrate that the leader of any view $v'' > v$ proposes *val* with high probability.

Recall that, in Algorithm 1, when the synchronizer notifies a replica to enter a new view v'', the replica informs the leader of v'' about its latest prepared value through a NewLeader message. The leader of v'' waits until it observes a deterministic quorum of NewLeader messages. If at least $\lceil (n + f + 1)/2 \rceil$ correct replicas have prepared *val*, then the leader must propose *val*, regardless of its type, whether Byzantine or correct. The problem occurs when $w < \lceil (n + f + 1)/2 \rceil$ correct replicas have prepared *val*. Note that $w \geq 1$ as we assumed at least one correct replica has decided *val*. One of the following scenarios can happen:

- $1 \leq w \leq f$. If the leader is Byzantine, it can propose any value different than *val*. Besides, if the leader is correct, it proposes a value $val' \neq val$ when the number of NewLeader messages received from replicas that prepared *val′* is greater than the number of NewLeader messages received from replicas that prepared *val*.
- $f + 1 \leq w < \lceil (n + f + 1)/2 \rceil$. If the leader is Byzantine, it can propose a value $val' \neq val$ if the number of replicas that prepared *val′* is greater than the number of replicas that prepared *val*. Besides, if the leader is correct, it proposes a value $val' \neq val$ when the number of NewLeader messages received from replicas that prepared *val′* is greater than the number of NewLeader messages received from replicas that prepared *val*.

We need to ensure that there is a high probability that the system will not be in these scenarios. The following theorem shows this.

Theorem 8. The probability of proposing a value *val′* when another value *val* has been decided by a correct replica in a prior view is at most $exp\big(-(q \times \delta^2)/((\delta + 1) \times (\delta + 2))\big)$, where $\delta = 2n/(o \times (n + f)) - 1$.

Corollary 1 (Main safety result). ProBFT guarantees safety with a probability of at least $1 - exp(-\Theta(\sqrt{n}))$.

5 NUMERICAL EVALUATION

In this section, we illustrate the usefulness of ProBFT by presenting a numerical analysis of the agreement and termination probabilities, considering $q = 2\sqrt{n}$ and $o \in \{1.6, 1.7, 1.8\}$.

Analysis with fixed fault threshold and varying system sizes. In Figure 5, the top sub-figures depict analyses with $f/n = 0.2$ and varying system sizes. The top-left sub-figure depicts the probability of ensuring agreement, considering the worst-case scenario in which there are faulty leaders in each view. It shows that as the system size increases, the probability of ensuring agreement also increases. Besides, the top-right sub-figure depicts the probability of terminating in a view after GST when the leader is correct. It shows the probability of deciding increases as the number of replicas increases.

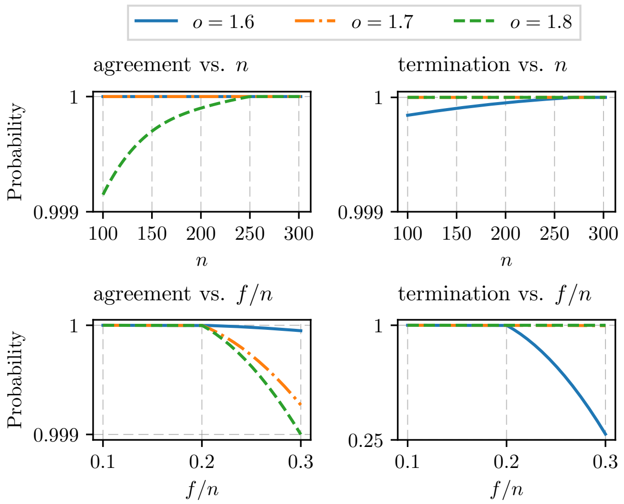


Figure 5: ProBFT agreement and termination probability analysis with $q = 2\sqrt{n}$. For $f/n = 0.2$, the top-left figure depicts the probability of ensuring agreement with faulty leaders in every view, while the top-right figure shows the probability of terminating in a view after GST when the leader is correct. For $n = 100$, the bottom-left figure depicts the probability of ensuring agreement with faulty leaders in every view, while the bottom-right figure depicts the probability of terminating in a view after GST when the leader is correct.

Analysis with fixed system size and a varying number of faulty replicas. In Figure 5, the bottom sub-figures depict similar analyses with $n = 100$ and a varying number of faulty replicas. The bottom-left sub-figure shows that the probability of ensuring agreement increases as we have fewer Byzantine replicas. Similarly, the bottom-right sub-figure shows that the probability of deciding increases as we have fewer Byzantine replicas.

Number of exchanged messages. Figure 1b shows the number of exchanged messages in PBFT, ProBFT (for different values of o), and HotStuff. In the figure, it is possible to see that ProBFT exchanges significantly fewer messages than PBFT despite having the same good-case optimal latency. Taking together with the results of Figure 5, we can see that ProBFT with $o = 1.7$ ensures a high probability of agreement and termination exchanging only 18-25% of the messages required by PBFT.

6 RELATED WORK

Scalable BFT consensus protocols. PBFT [13] is considered the baseline “practical” BFT protocol. It is optimal in terms of resilience ($f < n/3$) and best-case latency (three communication steps) but employs an all-to-all message exchange pattern, which results in a quadratic message complexity, making it very costly in large deployments. With the advent of large-scale systems like blockchains and decentralized payment systems, the scalability of Byzantine consensus protocols has become a hot topic, with numerous contributions from both academia and industry.

There are several approaches to enhance the scalability of BFT consensus protocols. For instance, protocols like HotStuff [56] modify the message exchange pattern from all-to-all to leader-to-all-to-leader, resulting in a linear message complexity. Alternatively, to improve the message complexity and balance the load on the system, some protocols [32, 46] utilize a tree-based message exchange pattern, while other protocols (e.g., [12, 23, 31, 58]) employ a gossip layer for communication. Similarly, there are some works that use expander graphs (and related techniques) to make Byzantine agreement more scalable (e.g., [28]). Other works like RedBelly [15] improve different aspects of the protocol, such as how ordered transactions are disseminated and verified, to achieve better scalability in practical deployments without reducing the message complexity of the base protocol. While most of these works avoid the all-to-all communication pattern, they increase the number of communication steps necessary to achieve consensus.

Randomized consensus protocols. The primary drive for developing randomized consensus protocols was the well-known FLP impossibility result [21]. This result states that for crash-prone asynchronous systems, designing a deterministic consensus protocol is impossible. One of the main approaches to circumvent such an impossibility result involves relaxing deterministic termination to probabilistic termination, assuming processes have access to random numbers.

The randomized consensus protocol by Ben-Or [4] assumes a strong adversary who can observe the entire history of the system and uses a local coin. The protocol operates in rounds, with each round involving $O(n^2)$ message exchanges, and it requires exponential expected time to converge in the worst case. Subsequent works like RITAS [40] and WaterBear [59] showed this type of protocol can be made practical.

Rabin [49] showed that the same type of protocol could achieve termination in the expected constant time by resorting to a common coin. The construction of this common coin typically requires strong cryptographic primitives, which can negatively affect the performance of the protocol. Further works like Cachin et al. [10] leveraged threshold signatures to devise a protocol with constant expected time and message complexity of $O(n^2)$. More recently, Mostéfaoui et al. [41] proposed a similar protocol that is signature-free. HoneyBadgerBFT [39] is a practical randomized consensus protocol that has at least an $O(n^3)$ message complexity. Follow-up works (e.g., [18, 22, 34]) improved different aspects of practical randomized protocols, including their message complexity, but never below $O(n^2)$. The same can be said about DAG-based protocols (e.g., [27, 52]) that use local rules to commit blocks (i.e., decide consensus values) and resort to a kind of randomized consensus only when such rules are not enough.

Synchronizer. Byzantine fault-tolerant consensus protocols designed for partially synchronous systems typically structure their execution in a sequence of views, with the premise that there will be a view in which all correct replicas will overlap with enough time to reach a consensus if there is a correct leader. Designing these protocols is challenging, and researchers usually pay more attention to guaranteeing the system's safety rather than liveness [13, 25, 56]. The problem with the partially synchronous model and designing the protocols in a sequence of views is that replicas may diverge indefinitely before the GST, reaching GST in different views. This problem is typically not addressed in commonly used Byzantine fault-tolerant protocols [6, 13, 44] in which the liveness is argued based on the assumption that after the system reaches GST, all the correct replicas will eventually converge to the same view. By separating the mechanism used for view synchronization in a distinct component, Bravo et al. [7] formally defined the synchronizer abstraction, which we employ in ProBFT. Notice that using such abstraction does not incur added message complexity as there are constructions with linear message complexity [30, 45].

Probabilistic quorum systems. Malkhi et al. [37] introduced probabilistic quorum systems to enhance the efficiency of data replication by relaxing strict quorum requirements and allowing for probabilistic guarantees of consistency. These quorum systems operate under the implicit assumption that any chosen quorum will be accessible without taking into account the potential effects of failures or asynchrony [2, 54, 57]. In other words, it does not account for the impact of an adversarial scheduler (also known as an active adversary [2] or an asynchronous scheduler [57]) that could potentially delay the delivery of messages. Yu [57] introduced an alternative concept termed signed quorum systems, aiming to address the challenges posed by network scheduling. Nevertheless, Yu's method remains susceptible to manipulation by an adversarial scheduler [2].

7 CONCLUSION

We presented ProBFT, a Byzantine fault-tolerant consensus protocol that ensures safety and liveness with high probability in permissioned partially synchronous systems. This protocol's message complexity is $O(n\sqrt{n})$ in a system with n replicas, and it has an optimal number of communication steps. ProBFT introduces a novel paradigm for designing scalable Byzantine-resilient protocols for less pessimistic contexts. We believe the same techniques used in ProBFT can be employed in other types of quorum-based protocols. As future work, we are particularly interested in leveraging ProBFT for constructing a scalable state machine replication protocol and a streamlined blockchain consensus, eliminating the need for a view-change sub-protocol.

ACKNOWLEDGMENTS

This work was partially supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 446811880 (BFT2Chain), and by FCT through the SMaRtChain project (2022.08431.PTDC) and the LASIGE Research Unit (UIDB/00408/2020 and UIDP/00408/2020).

REFERENCES

[1] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. 2021. Good-case Latency of Byzantine Broadcast: a Complete Categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC '21).*

[2] Amitanand Aiyer, Lorenzo Alvisi, and Rida A. Bazzi. 2005. On the Availability of Non-strict Quorum Systems. In *Proceedings of th 19th International Symposium on Distributed Computing (DISC '05).*

[3] Diogo Avelãs, Hasan Heydari, Eduardo Alchieri, Tobias Distler, and Alysson Bessani. 2024. Probabilistic Byzantine Fault Tolerance (Extended Version). arXiv:2405.04606 [cs.DC]

[4] Michael Ben-Or. 1983. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. In *Proceedings of the 2nd Symposium on Principles of Distributed Computing (PODC '83).*

[5] Alysson Bessani, Eduardo Alchieri, João Sousa, André Oliveira, and Fernando Pedone. 2020. From Byzantine Replication to Blockchain: Consensus is only the Beginning. In *Proceedings of the 50th IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN '20)*.

[6] Alysson Bessani, João Sousa, and Eduardo E. P. Alchieri. 2014. State Machine Replication for the Masses with BFT-SMaRt. In *Proceedings of the 44th International Conference on Dependable Systems and Networks (DSN '14)*.

[7] Manuel Bravo, Gregory Chockler, and Alexey Gotsman. 2022. Making Byzantine Consensus Live. *Distributed Computing* 35, 6 (2022).

[8] Vitalik Buterin and Virgil Griffith. 2019. Casper the Friendly Finality Gadget. arXiv:1710.09437 [cs.CR]

[9] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and Efficient Asynchronous Broadcast Protocols. In *Proceedings of the 21st Annual International Cryptology Conference (Crypto '01)*.

[10] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2000. Random Oracles in Constantipole: Practical Asynchronous Byzantine Agreement Using Cryptography. In *Proceedings of the 19th Symposium on Principles of Distributed Computing (PODC '00)*.

[11] Christian Cachin, Giuliano Losa, and Luca Zanolini. 2022. Quorum Systems in Permissionless Network. In *Proceedings of the 26th International Conference on Principles of Distributed Systems (OPODIS '22)*.

[12] Daniel Cason, Enrique Fynn, Nenad Milosevic, Zarko Milosevic, Ethan Buchman, and Fernando Pedone. 2021. The Design, Architecture and Performance of the Tendermint Blockchain Network. In *Proceedings of the 40th International Symposium on Reliable Distributed Systems (SRDS '21)*.

[13] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI '99)*.

[14] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *Symposium on Operating Systems Design and Implementation*.

[15] Tyler Crain, Christopher Natoli, and Vincent Gramoli. 2021. Red Belly: A Secure, Fair and Scalable Open Blockchain. In *Proceedings of the IEEE Symposium on Security and Privacy (SP '21)*.

[16] Tobias Distler. 2021. Byzantine Fault-Tolerant State-Machine Replication from a Systems Perspective. *Comput. Surveys* 54, 1 (2021).

[17] John R Douceur. 2002. The Sybil Attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*.

[18] Sisi Duan, Michael K Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*.

[19] Devdatt P Dubhashi and Alessandro Panconesi. 2009. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press.

[20] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *J. ACM* 35, 2 (1988).

[21] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32, 2 (1985).

[22] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Dumbo-NG: Fast Asynchronous BFT Consensus with Throughput-Oblivious Latency. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*.

[23] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*.

[24] Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Včelák. 2022. *Verifiable Random Functions (VRFs)*. Technical Report draft-irtf-cfrg-vrf-11. Internet Engineering Task Force.

[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 *Proceedings of the 49th International Conference on Dependable Systems and Networks (DSN '19)*.

[26] Vassos Hadzilacos and Sam Toueg. 1994. *A modular approach to fault-tolerant broadcasts and related problems*. Technical Report. Cornell University.

[27] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. 2021. All You Need is DAG. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC '21)*.

[28] Valerie King and Jared Saia. 2011. Breaking the $O(n^2)$ bit barrier: Scalable Byzantine agreement with an adaptive adversary. *J. ACM* 58, 4 (2011).

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

[30] Andrew Lewis-Pye and Ittai Abraham. 2023. Fever: Optimal Responsive View Synchronisation. In *Proceedings of the 27th International Conference on Principles of Distributed Systems (OPODIS '23)*.

[31] Peilun Li, Guosai Wang, Xiaoqi Chen, Fan Long, and Wei Xu. 2020. Gosig: A Scalable and High-Performance Byzantine Consensus for Consortium Blockchains. In *Proceedings of the 11th Symposium on Cloud Computing (SoCC '20)*.

[32] Wenyu Li, Chenglin Feng, Lei Zhang, Hao Xu, Bin Cao, and Muhammad Ali Imran. 2020. A Scalable Multi-Layer PBFT Consensus for Blockchain. *IEEE Transactions on Parallel and Distributed Systems* 32, 5 (2020).

[33] Xiao Li, Eric Chan, and Mohsen Lesani. 2023. Quorum Subsumption for Heterogeneous Quorum Systems. In *Proceedings of the 37th International Symposium on Distributed Computing (DISC '23)*.

[34] Shengyun Liu, Wenbo Xu, Chen Shan, Xiaofeng Yan, Tianjing Xu, Bo Wang, Lei Fan, Fuxi Deng, Ying Yan, and Hui Zhang. 2023. Flexible Advancement in Asynchronous BFT Consensus. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*.

[35] Marta Lokhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafał Malinowsky, and Jed McCaleb. 2019. Fast and Secure Global Payments with Stellar. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP '19)*.

[36] Shyh-Wei Luan and Virgil D Gligor. 1990. A Fault-tolerant Protocol for Atomic Broadcast. *IEEE Transactions on Parallel & Distributed Systems* 1, 3 (1990).

[37] Dahlia Malkhi, Michael Reiter, Avishai Wool, and Rebecca Wright. 2001. Probabilistic Quorum Systems. *Information and Computation* 170, 2 (2001).

[38] Silvio Micali, Michael Rabin, and Salil Vadhan. 1999. Verifiable Random Functions. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS '99)*.

[39] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The Honey Badger of BFT Protocols. In *Proceedings of the 23rd Conference on Computer and Communications Security (CCS '16)*.

[40] Henrique Moniz, Nuno Ferreria Neves, Miguel Correia, and Paulo Verissimo. 2011. RITAS: Services for Randomized Intrusion Tolerance. *IEEE Transactions on Dependable and Secure Computing* 8, 1 (2011).

[41] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. 2015. Signature-free Asynchronous Binary Byzantine Consensus with $t < n/3$, $O(n^2)$ Messages, and $O(1)$ Expected Time. *J. ACM* 62, 4 (2015).

[42] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press.

[43] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System.

[44] Oded Naor, Mathieu Baudet, Dahlia Malkhi, and Alexander Spiegelman. 2021. Cogsworth: Byzantine View Synchronization. *Cryptoeconomic Systems* 1, 2 (2021).

[45] Oded Naor and Idit Keidar. 2020. Expected Linear Round Synchronization: The Missing Link for Linear Byzantine SMR. In *Proceedings of the 34th International Symposium on Distributed Computing (DISC '20)*.

[46] Ray Neiheiser, Miguel Matos, and Luís Rodrigues. 2021. Kauri: Scalable BFT Consensus with Pipelined Tree-based Dissemination and Aggregation. In *Proceedings of the 28th Symposium on Operating Systems Principles (SOSP '21)*.

[47] Rafael Pass and Elaine Shi. 2017. The Sleepy Model of Consensus. In *Proceedings of the 23rd International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT '17)*.

[48] Fernando Pedone and André Schiper. 1998. Optimistic Atomic Broadcast. In *Proceedings of the 12th International Symposium on Distributed Computing (DISC '98)*.

[49] Michael O. Rabin. 1983. Randomized Byzantine Generals. In *Proceedings of the 24th Proceedings of the 24th Symposium on Foundations of Computer Science (FOCS '83)*.

[50] Fred B. Schneider. 1990. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *Comput. Surveys* 22, 4 (1990).

[51] João Sousa and Alysson Bessani. 2012. From Byzantine Consensus to BFT State Machine Replication: A Latency-Optimal Transformation. In *Proceedings of the 9th European Conference on Dependable Computing (EDCC'12)*.

[52] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. 2022. Bullshark: DAG BFT Protocols Made Practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*.

[53] Robin Vassantlal, Hasan Heydari, and Alysson Bessani. 2023. On the Minimal Knowledge Required for Solving Stellar Consensus. In *Proceedings of the 43rd International Conference on Distributed Computing Systems (ICDCS '23)*.

[54] Marko Vukolić. 2013. The Origin of Quorum Systems. *Bulletin of EATCS* 2, 101 (2013).

[55] Marko Vukolić. 2015. The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In *Proceedings of the International Workshop on Open Problems in Network Security (iNetSec '15)*.

[56] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 38th Symposium on Principles of Distributed Computing (PODC '19)*.

[57] Haifeng Yu. 2006. Signed Quorum Systems. *Distributed Computing* 18, 4 (2006).

[58] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. RapidChain: Scaling Blockchain via Full Sharding. In *Proceedings of the 2018 Conference on Computer and Communications Security (CCS '18)*.

[59] Haibin Zhang, Sisi Duan, Boxin Zhao, and Liehuang Zhu. 2023. WaterBear: Practical Asynchronous BFT Matching Security Guarantees of Partially Synchronous BFT. In *Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23)*.