Phoenixx: Linear consensus with random sampling

David Chaum[1], Bernardo Cardoso[1,2], William Carter[1], Mario Yaksetig[1,2], and Baltasar Aroso[1,2]

[1] xx network
[2] BitFashioned
info@bitfashioned.com

Abstract. We present Phoenixx, a round and leader based Byzantine fault tolerant consensus protocol, that operates in the partial synchrony network communications model. Phoenixx combines the three phase approach from HotStuff, with a novel *Endorser Sampling*, that selects a subset of nodes, called *endorsers*, to "compress" the opinion of the network.
Unlike traditional sampling approaches that select a subset of the network to run consensus on behalf of the network and disseminate the outcome, Phoenixx still requires participation of the whole network. The endorsers, however, assume a special role as they confirm that at least $2f + 1$ validators are in agreement and issue a compressed certificate, attesting the network reached a decision. Phoenixx achieves linear communication complexity, while maintaining safety, liveness, and optimistic responsiveness, without using threshold signatures.

Keywords: BFT consensus · blockchain · quantum security · scalability

1 Introduction

Since the introduction of Bitcoin [16], blockchain technology became one of the most in demand topics of research in information technology and computer science. In a blockchain network, each participant, also referred to as a node or a validator, maintains a ledger and provides a payment service to clients. A consensus protocol is necessary for these validators to agree on the ordering of client transactions, and ensuring a consistent ledger state is maintained.

Blockchain applications aim to provide the service on a worldwide scale, thus operating in a decentralized, global and open network, which needs to withstand the actions of malicious validators, also referred to as Byzantine failures. This has generated a renewed interest in BFT consensus protocols. The scope, however, is different from traditional algorithms: blockchain networks require a protocol that supports scaling to a large number of validators, deployed on a global network. Due to this network model containing a large number of validators, one of the main goals for an efficient BFT consensus protocol is to provide *linear scalability*.

In this paper, we present Phoenixx, a novel BFT consensus protocol that builds on top of the cutting edge research behind HotStuff [21] and DiemBFT [1]. To the best of our knowledge, Phoenixx is the first BFT replication protocol that achieves linear authenticator complexity, and potential quantum security, *without* the use of threshold signatures. This result is possible due to a novel *Endorser Sampling* approach. Phoenixx provides strong probabilistic guarantees of *safety*, *liveness*, and *optimistic responsiveness*, and constant-sized proofs of block finality. Phoenixx is designed to be flexible, not restricting the kind of cryptography that can be used, and particularly well-suited for the use of quantum secure hash based signatures for future proofing.

Our contributions. Succinctly, the main contributions of this work is Phoenixx, a novel BFT consensus protocol that:

- achieves linear authenticator complexity regardless of the used cryptography (e.g., hash-based, elliptic curve-based, pairing-based, lattice-based, ...).
- does not rely on any Distributed Key Generation (DKG) steps, which are traditionally very expensive in large-scale distributed systems.
- uses a different endorser sampling approach.
- produces constant-sized proofs of finality.

2 Previous Work

Byzantine fault tolerance (BFT) is defined as the ability of a networked system to operate correctly, even when one or more of its components experiences arbitrary failures. The problem of reaching consensus in the presence of byzantine failures was first introduced in the celebrated Byzantine Generals Problem paper by Lamport et al. [15].

The first solution to the problem, under a synchronous network communication model, is proposed in [17], which proves that the problem is only solvable if the size of the system, N is at least $3f + 1$, where f is the number of byzantine components. One of the first applications of BFT consensus algorithms was to solve the state machine replication (SMR) problem. In this setting, each component of the system serves client requests and replicates the same state. A BFT SMR consensus protocol allows non-faulty components to agree on a deterministic ordering of client requests, ensuring that the overall system state is consistent.

When the network is asynchronous, the proven FLP impossibility [9] shows that, under the presence of a single byzantine failure, the problem is unsolvable by a deterministic algorithm.

DLS [8], is one approach developed to sidestep this impossibility, and operates under the partially synchronous communication model, where the network maintains synchrony *most* of the time. The network, however, experiences sporadic periods of asynchrony. DLS guarantees safety at all times and liveness only when the network becomes synchronous. This work pioneered the use of a leader

based, round by round paradigm, which became the standard for most BFT consensus protocols developed since. DLS, however, is very inefficient, having $\mathcal{O}(n^4)$ expected communication complexity.

PBFT [5], is the first practical solution to this communication complexity problem and reduces the complexity to $\mathcal{O}(n^2)$ when an honest leader drives the protocol, and $\mathcal{O}(n^3)$ when the leader experiences a fault. As with other traditional BFT consensus protocols, DLS and PBFT were designed targeting systems deployed on LANs, with $f = 1$ or $f = 2$, i.e. $N = 4$ or $N = 7$.

The latest years of research in decentralized networks have produced many BFT protocols, tailored specifically to blockchain applications, in search of the desired linear scalability.

SBFT [11] improves on PBFT, by applying threshold signatures, reducing the complexity to $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ for the best and worst case, respectively.

Tendermint [13, 14], Casper [4], and GRANDPA [19] borrow the two-phase approach from PBFT, but include a synchronous core that relies on the worst-case message propagation time, achieving complexity of $\mathcal{O}(n^2)$ for all cases. These protocols can be optimized using threshold signatures, thus achieving the desired complexity of $\mathcal{O}(n)$. However, by depending on an upper bound worst-case network delay, these protocols fail to provide a classic property of BFT consensus algorithms: *optimistic responsiveness*.

Looking to address the outstanding issues with BFT consensus protocols used in blockchain networks, HotStuff [21] pioneered a novel three phase approach, using threshold signatures, that achieves both linear complexity $\mathcal{O}(n)$ and optimistic responsiveness. HotStuff also created a framework for representing most BFT consensus protocols, which allows for a clean separation of the mechanisms needed to achieve safety and liveness.

DiemBFT [1] proposes improvements to HotStuff and simplifies the safety argument, encapsulating a set of *safety rules* in a separable system component called the safety module. Additionally, it provides a better synchronization mechanism, which improves liveness, and introduces the concept of a *nil* block, enabling progress even in rounds where the leader is byzantine. HotStuff and Diem rely on the use of threshold signatures to achieve linear scalability.

3 Problem Statement

State of the art BFT consensus protocols rely on the use of aggregate signatures [3] to achieve linear scalability as these types of signatures allow third parties to aggregate public keys and signatures without requiring any interaction between individual signers. Traditionally, a Quorum Certificate (QC) aggregates $2f + 1$ votes, so its size grows linearly with the number of validators in the network. However, using signature aggregation schemes, $2f + 1$ votes can be expressed as a single signature, resulting in a constant-sized QC. This approach, however, is not future proof as these signatures are not quantum secure.

Quantum security. Quantum-secure aggregate signatures are an active area of research. Most of the existing schemes, however, have significant issues in a

conversion into threshold signature schemes using generic MPC techniques [7] We highlight [12] that showcases an approach for constructing a hash-based t-of-n threshold signature scheme using STARKs [2]. This approach, however, comes at the cost of significant time to aggregate signatures, which is unpractical for a real-world distributed system deployment.

Throughout the paper, we use authenticator complexity to measure the scalability of Phoenixx. We borrow the definition from HotStuff [21]: *the sum, over all validators, of the number of authenticators received by each participant to reach a consensus decision after GST*. One of the consequences of using this metric is that the consensus protocol can, at least theoretically, use multi-signatures or threshold signatures, since these provide linear costs. We note that, even though these are different types of signatures, some require overhead during the consensus protocol, as participants must incur in additional communication rounds to ensure the correct signing of data. Moreover, threshold signature schemes require a distributed key generation (DKG) process, with strict availability requirements from participants. In fact, many blockchain consensus protocol proposals require the network membership to change. Consequently, validators must frequently (e.g., daily or even per block) perform a completely new DKG setup to obtain a new network public key.

Existing sampling approaches. The concept of having a group of validators perform special operations on behalf of a network is not novel, as other works employ similar ideas. For example, Algorand [10] uses a cryptographic sortition-based *committee* selection algorithm to choose a subset of the network to participate in each step of the consensus algorithm for each round. Validators not selected by the algorithm do not participate in the protocol, and simply wait for the consensus decision.

Our solution. Phoenixx requires no threshold signatures, and operates using a small endorser sample that confirms they witnessed $N - f$ signatures for the same round proposal. This confirmation signature is called an *endorsement*, and a participant considers a round proposal *certified* once it collects a given number of endorsements, forming a endorser quorum certificate (EQC). Contrasting with other sampling approaches, in Phoenixx, the entire network is required to participate in the protocol, since no honest endorser will produce their *endorsement* without first receiving a network majority of signatures for the same proposal.

4 Phoenixx Preliminaries

4.1 Model & Goals

We assume a fixed size network of $N = 3f + 1$ validators, where f is the number of faulty validators, and the remaining $2f + 1$ are honest. Validators have a fixed ID, or public identity, that is known before the protocol starts. In other words, the "membership list" of the network is known by all participants.

To model the network communications we use the partial synchrony model, which states that there is an unknown Global Stabilization Time (GST), after which two honest validators are able to communicate in a bounded time Δ.

We assume a global adversary capable of simultaneously coordinating all the Byzantine validators in the network and isolate honest validators from the rest of the network, provided that the BFT assumption is not broken. This adversary can also view the state of every honest validator at any time and can instantly modify the state of all Byzantine validators accordingly.

Consensus critical messages are signed by validators, and there are no restrictions on the cryptographic primitives used for this signing process, as long as signatures can be verified using the public identity of the node.

Phoenixx consensus protocol has the following goals:

Goal 1 (Probabilistic Safety). *With overwhelming probability, no two honest validators commit different decisions.*

Goal 2 (Probabilistic Liveness). *With overwhelming probability, all honest validators eventually commit a decision.*

Goal 3 (Optimistic responsiveness). *Under synchrony, an honest leader can drive a decision as fast as allowed by actual message delays.*

Goal 4 (Scalability). *Complexity of the protocol should, in the worst case, increase linearly with respect to the network size. Also, the proof size of a reached decision should remain constant.*

4.2 Phoenixx Overview

The Phoenixx consensus protocol operates in a succession of *rounds*, identified by monotonically increasing *round numbers*. For each round, there is a single *leader* and an unique *endorser set* known by all parties.

During protocol execution, validators create, distribute, execute and vote on ***proposals*** - the core structure of Phoenixx, which encapsulates a $parentEQC$, the current *round* of the protocol, a *block* (containing a batch of *transactions*), and a signature from the author.

Validators maintain a tree of pending proposals, with the root as the latest *committed* proposal. Each proposal contains a link to its parent, which allows computation of a *branch* by following parent links from a leaf proposal to the root. When an EQC is gathered for a proposal, it becomes *certified*. Future proposals can then reference this EQC as a parent.

A proposal P can be committed when it becomes the head of a "3-chain" of certified proposals created in consecutive rounds. Once this happens, any pending proposals part of the branch ending in P are committed in order. For example, consider proposals $\mathcal{P}_0$; $\mathcal{P}_1$, with parent $\mathcal{P}_0$; and $\mathcal{P}_2$ with parent $\mathcal{P}_1$. $\mathcal{P}_0$ becomes committed when an EQC is formed for $\mathcal{P}_2$, iff $round(\mathcal{P}_0) + 1 = round(\mathcal{P}_1) \wedge round(\mathcal{P}_1) + 1 = round(\mathcal{P}_2)$.

Phoenixx provides a probabilistic guarantee that only one branch ever becomes committed. Therefore, there are no *forks*. This guarantee is achieved by using a concise set of ***safety rules***, which are always followed by honest validators.

Rule 1 (Voting). *Validators must only vote for one proposal per round, in strictly increasing rounds.*

Rule 2 (Endorsing). *Validators that are selected as endorsers for a round must only endorse the same proposal they voted for, and only after seeing a network quorum certificate (NQC) of votes for that proposal. Validators must only endorse one proposal per round, in strictly increasing rounds.*

Rule 3 (Locking). *Validators keep track of a* ***preferred round*** *and must only vote for a proposal if the round of its parent is at least the preferred round. When a validator votes for a proposal, it checks if the round of its grandparent is higher than the current preferred round. If so, the preferred round is updated to that value.*

4.3 Phases

There is no explicit definition of phases in Phoenixx. Each round, however, can be split into three logical phases: *Propose*, *Confirm*, and *Endorse*. There is also a *Timeout* mechanism for rounds to be abandoned when no progress is made.

Propose. For the leader of each round, the protocol begins with an attempt to create a new proposal, extending from the highest certified proposal from its local tree, defined as $highProposal$. The leader builds a new block of transactions using $highProposal$ as the parent, and creates a new proposal containing this block. The proposal carries the certificate for $highProposal$ as the $parentEQC$. Then, the leader sends the proposal to every validator in the network.

As a regular validator, each round starts by waiting to receive a proposal from the leader of the round. This waiting period is limited to a configurable timeout, defined as $proposeTimeout$, which is set up at the start of each round.

Confirm. When a validator receives a proposal, it verifies the validity and integrity of the block, processes the $parentEQC$, and speculatively executes the block. Provided no failures occurred, the validator attempts to $confirm$ the proposal. This is done by creating and signing a vote, if safety rules 1 and 3 allow. If a vote is produced, the validator sends it to every endorser for the current round.

Endorse. When an endorser receives $2f + 1$ votes for the same proposal, it combines them into an NQC and the proposal becomes $confirmed$. Upon confirmation, there is an attempt to $endorse$ the proposal if safety rule 2 allows. If an endorsement is produced, the endorser sends it to every validator in the network. When a validator receives a quorum of endorsements $q \cdot |\mathcal{E}|$, it combines them[3] into an EQC, the proposal becomes $certified$, and the validator moves

[3] "Combining" simply represents appending items to a data structure, such as a list.

to the next round. If the EQC contains information to commit a proposal CP, then the branch of proposals ending in CP will be committed.

Timeouts. When waiting for a proposal, if *proposeTimeout* is triggered, the validator creates a *nil* proposal, using its local *highProposal* as the parent, and a deterministically generated block that doesn't contain transactions. The *nil* proposal is then used to perform the *Confirm* phase of the protocol. This timeout procedure provides faster agreement on rounds where the leader is offline.

At the beginning of each round, validators start a timer with a configured *roundTimeout*. If this timeout is triggered, the validator creates and signs a *timeout* vote, certifying the intention to skip the round. After timing out, voting is disallowed for the round. The timeout vote is sent to every endorser, and when $2f+1$ timeouts are collected, the endorser aggregates them into a ***timeout certificate*** TC. Then, the endorser creates and signs an *endorseTimeout* vote and sends it to every validator in the network. When a validator receives a quorum of endorser timeouts $(1-q)$, it combines them into an ***endorser timeout certificate*** ETC, and moves to the next round. Validators only advance rounds by receiving either an EQC or an ETC.

5 Phoenixx Spec

We proceed with a detailed description of Phoenixx, and explicitly divide the protocol into logical modules.

5.1 Application Interfaces

To ensure that Phoenixx operates safely and correctly, applications wishing to use the protocol to achieve consensus need to comply with some basic requirements. No restrictions are imposed on the structure and contents of a block. The block, however, must contain the following mandatory information: *round* number, parent $blockID$, proposer ID, *timestamp*.

We group all the application specific functions into the *Application* module. The *RoleSelection* module provides the selection of leaders and endorsers. Access to private keys for signing consensus data is encapsulated in a separate *Signer* module. The implementation of these modules is independent from Phoenixx, but function signatures are provided in Algorithm 1 below.

Algorithm 1 - Application Interfaces

```
1: Module - Application
       ▷ Propose a new block chaining on the given parent for the given round
2:     function Propose(round, parentID)
       ▷ Compute deterministic nil block with given parent and round
3:     function ComputeNil(round, parentID)
       ▷ Validate a block and speculatively execute its transactions
4:     function Validate(block)
       ▷ Commit a branch of blocks using provided EQC as proof
5:     function Commit(blocks, eqc)
       ▷ Drop a list of blocks
```

```
6:    function DROP(blocks)
7: MODULE - Role Selection
      ▷ Get the validator's role for the given round
8:    function GETROLE(round)
      ▷ Get the validator ID that is the leader for the given round
9:    function GETLEADER(round)
      ▷ Get list of validator IDs that are endorsers for the given round
10:   function GETENDORSERS(round)
11: MODULE - Signer
      ▷ Sign provided data with a specified type for a given round
12:   function SIGN(round, type, data)
```

5.2 Safety Module

Inspired by the work from [1], we define the *Safety* module, which maintains the core ***safety rules*** of Phoenixx. Function signatures for this module are described in Algorithm 2.The *Safety* module is the only component that can access the *Signer* module. This allows validators to compartmentalize their system by physically separating these two modules from the rest.

Algorithm 2 - Safety Module

```
1: MODULE - Safety
      ▷ Certify a proposal, if proposing rule allows
2:    function CERTIFYPROPOSAL(proposal)
      ▷ Create a confirm vote for a proposal, if voting and locking rules allow
3:    function ATTEMPTVOTE(proposal)
      ▷ Create an endorsement for a confirmed proposal, if endorsing rule allows
4:    function ATTEMPTENDORSEMENT(proposal, networkQC)
      ▷ Certify a timeout or endorseTimeout for a given round
5:    function CERTIFYTIMEOUT(round, endorser)
```

5.3 Phoenixx Algorithm

The main *Phoenixx* module combines all other modules to execute the protocol, as described in Algorithm 3.

Algorithm 3 - Phoenixx

```
   ▷ Vote collection
1: function COLLECTVOTE(v, quorum)
      // V is a map that collect votes with a matching hash value
2:    V[hash(v)] ← V[hash(v)] ∪ v
3:    if |V[hash(v)]| = quorum then
         // Get the list of vote authors and respective signatures from V
4:       return QC{v.block, v.type, v.commit, V[hash(v)].authors, V[hash(v)].signatures}
5: round ← 1
6: main loop:
      // Setup timeouts (reset timers)
7:    proposalTimeout.Reset
8:    roundTimeout.Reset
      // Round selections
9:    role ← RoleSelection.GETROLE(round)
10:   leader ← RoleSelection.GETLEADER(round)
11:   endorsers ← RoleSelection.GETENDORSERS(round)
      // Leader proposes new block, extending highest known EQC
12:   if role.IsLeader then
13:      block ← Application.PROPOSE(round, highEQC.blockID)
```

```
14:         proposal ← Proposal{highEQC, block}
15:         sig ← Safety.CertifyProposal(proposal)
16:         if sig ≠ nil then
17:             proposal.sig ← sig
18:             broadcast proposal
        // All validators wait for round proposal, with specific timeout
19:     wait for proposal p where (p.round = round) ∧ (p.author = leader)
            // Vote for nil proposal if timeout expires
20:         if proposalTimeout.Expired then
21:             parentID ← highEQC.blockID
22:             nilBlock ← Application.ComputeNil(round, parentID)
23:             nilProposal ← Proposal{highEQC, nilBlock}
24:             vote ← Safety.AttemptVote(nilProposal)
25:             if vote ≠ nil then
26:                 send vote to endorsers
            // Proposal received in time, attempt to vote for it
27:         if received p then
28:             proposalTimeout.Stop
29:             if Application.Validate(p.block) then
30:                 vote ← Safety.AttemptVote(p)
31:                 if vote ≠ nil then
32:                     send vote to endorsers
        // Endorsers collect votes until a network QC or TC is formed
33:     if role.IsEndorser then
34:         wait for votes v where (v.type = confirm) ∧ (v.round = round)
35:             networkQC ←CollectVote(v, 2f + 1)
36:             if networkQC ≠ nil then
37:                 endorse ← Safety.AttemptEndorsement(p, networkQC)
38:                 if endorse ≠ nil then
39:                     broadcast endorse
40:         wait for votes v where (v.type = timeout) ∧ (v.round = round)
41:             networkTC ←CollectVote(v, 2f + 1)
42:             if networkTC ≠ nil then
43:                 endorseTimeout ← Safety.CertifyTimeout(round, true)
44:                 if endorseTimeout ≠ nil then
45:                     broadcast endorseTimeout
        // All validators collect votes until an EQC or ETC is formed
46:     wait for votes v where (v.type = endorse) ∧ (v.round = round)
47:         eqc ←CollectVote(v, q)
48:         if eqc ≠ nil then
49:             highEQC ← eqc
50:             if eqc.commit ≠ nil then
51:                 Application.Commit(blocks through eqc.commitID, eqc)
52:                 Application.Drop(blocks from competing branches)
53:             round ← round + 1
54:     wait for votes v where (v.type = endorseTimeout) ∧ (v.round = round)
55:         etc ←CollectVote(v, (1 − q))
56:         if etc ≠ nil then
57:             round ← round + 1
        // At any time, if round timeout is triggered, send timeout and reset
58:     if roundTimeout.Expired then
59:         timeout ← Safety.CertifyTimeout(round, false)
60:         send timeout to endorsers
61:         roundTimeout.Reset
```

6 Analysis

We now explore a scenario where safety is broken due to endorser sampling, depicted in Figure 1, where $EQCs$ are formed for rounds k and $k+1$, committing proposal $k-1$. Then, $2f+1$ validators vote for proposal $k+2$ and become locked on round k. The endorser set of round $k+2$ creates an EQC for proposal $k+2$, which commits proposal k. Now, we assume that the leader of round $k+3$ is Byzantine, and intentionally creates a proposal that uses $k-1$ as its parent.

Proposal $k+3$ can never be confirmed, because $2f+1$ validators are locked in round k, and $parent(k+3) = k-1 < k$. However, if enough Byzantine validators are selected as endorsers for round $k+3$, they can form a quorum without any votes from honest endorsers. Furthermore, they can disregard the endorsing safety rule completely, and create an EQC for proposal $k+3$, without requiring any votes from honest validators during the confirm phase. Then, the leader of $k+4$ creates a proposal using $k+3$ as a parent, and $2f+1$ validators vote for it, as the locking rule is now met since $parent(k+4) = k+3 > k$. Then, $EQCs$ are formed for round $k+4$ and $k+5$, causing proposal $k+3$ to commit, creating a fork and breaking safety.

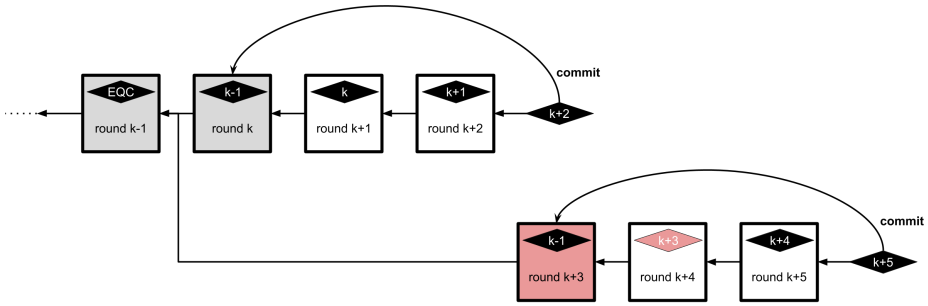


Fig. 1: Safety violation with fully Byzantine endorser quorum

To analyze the impact of endorser sampling in the Phoenixx protocol, we calculate the probability that a given endorser sample contains a specific number of Byzantine validators. The number of Byzantine validators selected on each endorser set can be upper bounded by a binomial distribution. This is noted as the function B, and depends only on the assumed percentage of Byzantine validators in the network (fixed to $1/3$) and the size of the endorser sample $|\mathcal{E}|$. The distribution of Byzantine validators in a endorser set is given by $Byz \sim B(|\mathcal{E}|, 1/3)$.

Probabilistic Safety. From the analysis above, we know that safety is broken if an endorser set contains at least a quorum of Byzantine validators. Therefore, the probability of breaking safety is given by $P_f = P(Byz \geq q \cdot |\mathcal{E}|)$

We can make this probability negligible by selecting appropriate values of $|\mathcal{E}|$ and q. Therefore, Phoenixx is probabilistically safe as, with overwhelming probability, only a single growing branch of blocks becomes committed. Furthermore, Phoenixx maintains safety at all times, even during periods of network asynchrony, such as GST.

Probabilistic Liveness. In Phoenixx, rounds can be abandoned when no progress is observed. When *roundTimeout* triggers, validators send timeout votes to the endorsers and reset the timer. This means that validators keep sending timeout votes for a round until they can make progress by receiving either an EQC or an ETC. Since the endorser set is responsible for endorsing timeouts,

progress might not be possible if there are not enough honest endorsers. This is a "dual" scenario to the one of breaking safety, since liveness is lost if an endorser set does not contain a timeout quorum of honest endorsers. By selecting the endorser set quorum for an ETC as $(1-q)$, we ensure both scenarios are equally unlikely. We note that a proposal cannot extend an ETC, and therefore, safety is not violated if both an EQC and an ETC are formed during the same round.

The probability of loss of liveness is given by $P_l = P(Honest < (1-q) \cdot |\mathcal{E}|) \iff P(Byz \geq q \cdot |\mathcal{E}| + 1)$. This probability is at most equal to the one of breaking safety, so in practice, liveness never breaks, since breaking safety is the worst case, and happens first.

Optimistic responsiveness. Phoenixx has no explicit step where an introduced waiting period depends on an estimated network delay. Therefore, Phoenixx has optimistic responsiveness. Due to endorser sampling, however, an honest leader alone might not be able to drive a round to a decision. This can happen if the endorser set does not contain enough honest validators to form a quorum and certify the proposal. Byzantine endorsers can refuse to endorse a confirmed proposal, causing the round to be abandoned by a timeout. In this case, optimistic responsiveness is not achieved, and the probability is given by $P_{resp} = P(Honest < q \cdot |\mathcal{E}|) \iff P(Byz \geq (1-q) \cdot |\mathcal{E}| + 1)$.

Game theory remarks. A side effect of relying on the endorser set to create timeout certificates is that Byzantine endorsers might be able to create an ETC without needing any honest endorsers, and without receiving any timeout votes from honest validators. By creating a "fake" ETC, Byzantine endorsers can force a round to be skipped, even if progress could be possible. Since the dual quorum of $(1-q)$ is used for $ETCs$, the probability of this scenario occurring is not negligible, and is given by: $P(Byz \geq (1-q) \cdot |\mathcal{E}|)$ An interesting takeaway if we assume the Byzantine endorsers always attempt to cause maximum disruption, is that the situation that disrupts optimistic responsiveness is a subset of the one where a "fake" ETC can be created. This means that Byzantine endorsers can only stand to gain by exploring the second situation if: $Byz = (1-q) \cdot |\mathcal{E}|$.

In the remaining cases, where $Byz \geq (1-q) \cdot |\mathcal{E}| + 1$, Byzantine endorsers are encouraged to let the round have an actual timeout, since creating a "fake" ETC would only advance the round faster.

6.1 Scalability

We assume one authenticator is a single cryptographic signature and use Table 1 to show how many signatures each participant in the network receives for each phase of a round of Phoenixx, depending on their role. We assume, w.l.o.g., that the leader sends the proposal to itself and is not an endorser.

The sum of authenticators received by all participants is the sum of authenticators received by each validator and each endorser, which is given by:

$$\begin{aligned}&(N - |\mathcal{E}|) \times [(q \cdot |\mathcal{E}| + 1) + 0 + |\mathcal{E}|] + |\mathcal{E}| \times [(q \cdot |\mathcal{E}| + 1) + N + |\mathcal{E}|] \\ &= \boldsymbol{N \times [(2+q) \cdot |\mathcal{E}| + 1]}\end{aligned}$$

Role	Propose	Confirm	Endorse
Validator	$q \cdot \|\mathcal{E}\| + 1$	0	$\|\mathcal{E}\|$
Endorser	$q \cdot \|\mathcal{E}\| + 1$	N	$\|\mathcal{E}\|$

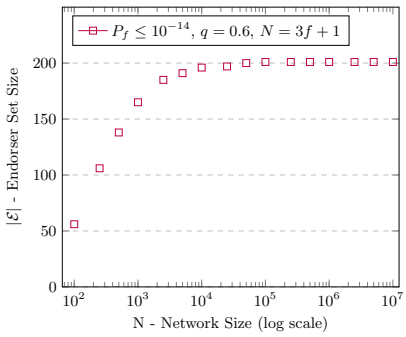
Table 1: Authenticator Complexity

The sum depends linearly on the size of the network N. Therefore, Phoenixx has linear authenticator complexity: $\mathcal{O}(n)$.

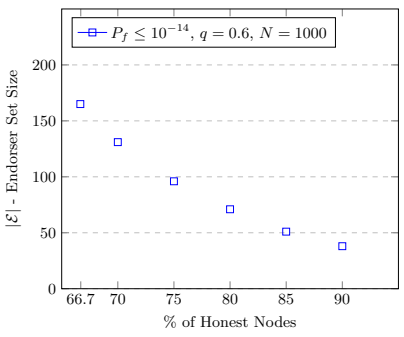
The proof of finality is defined as the cryptographic proof that a given block has been committed. In Phoenixx, the *EQC* provides proof of commitment of a block, and contains $q \cdot |\mathcal{E}|$ signatures from endorsers. When the network size increases, if the endorser set remains constant, so does the *EQC* size. This way, Phoenixx achieves constant-sized proofs of finality, regardless of the type of cryptography used.

6.2 Parameter Selection

Phoenixx has an upper bound on the value of $|\mathcal{E}|$ as if $|\mathcal{E}| = N$, the endorser sample equals the network size and the protocol becomes quadratic. Below, we expose two charts that describe the scaling properties of Phoenixx [4]. Both graphs represent a setting with a maximum failure probability of $P_f = 10^{-14}$ and $q = 0.6$



(a) Endorser set size $|\mathcal{E}|$ upper bound as the network size increases.



(b) Fixed N and varying percentage of honest validators in the network.

Timeouts. Based on empirical results from currently deployed blockchain networks [20], [18], [6], we recommend the following timeout values: $roundTimeout = 6s$ and $proposeTimeout = 4s$.

[4] Note: Phoenixx has over one order of magnitude less endorsers than Algorand under the same 80% honest node assumption, while providing stronger safety properties.

7 Safety proof

Lemma 1. *Under the BFT assumption, given two blocks $\mathcal{B}$, $\mathcal{B}'$, which are confirmed by two respective network quorum certificates NQC, NQC', then:*

$$round(\mathcal{B}) = round(\mathcal{B}') \Rightarrow \mathcal{B} = \mathcal{B}'$$

This is equivalent to saying there can only be one confirmed block per round.

Proof. By definition, network quorum certificates have at least $2f + 1$ votes. Due to the BFT assumption, any certificate has at most f votes from byzantine validators, meaning that it has at least $f + 1$ votes from honest validators.

Let $honest(NQC)$ be the set of honest validators that contributed to a certificate. We compute the intersection of honest validators in two certificates as:

$$|honest(NQC) \cap honest(NQC')| \geq (f + 1) + (f + 1) - (2f + 1) \geq 1$$

Consequently, both certificates contain at least one vote from the same honest validator, h. Since $round(\mathcal{B}) = round(\mathcal{B}')$, due to Rule 1, h could only have voted for one block in this round, which implies that $\mathcal{B} = \mathcal{B}'$.

□

Definition 1 (Sampling Assumption). *For each round, the leader and endorser set are selected in a way that can't be manipulated by byzantine actors.*

Lemma 2. *Under the sampling and BFT assumptions, given an endorser set $\mathcal{E}$, the number of byzantine endorsers $byz(\mathcal{E})$ can be upper bounded by the following binomial distribution:*

$$P(byz(\mathcal{E}) = k) = \binom{|\mathcal{E}|}{k} (1/3)^k (2/3)^{|\mathcal{E}|-k}$$

Proof. Under the sampling assumption, all validators have equal probability of endorser selection. Therefore, the endorser selection is effectively a random sample of the network with multiple draws, without replacement, so $byz(\mathcal{E})$ follows an hypergeometric distribution.

Let us assume the number of validators in the network is much larger than the size of the endorser set. This means that after each draw, the number of byzantine validators in the network remains virtually the same. This implies that $byz(\mathcal{E})$ can be upper bounded by a binomial distribution.

Then, $byz(\mathcal{E})$ follows a probability mass function $\mathcal{F}(k, n, p)$, where n is the size of the endorser sample $|\mathcal{E}|$ and p is the probability of a selected validator being byzantine, which, under the BFT assumption, is upper bounded by $1/3$.

□

Lemma 3. *Under the sampling and BFT assumptions, given a block $\mathcal{B}$, which is certified by an endorser quorum certificate EQC, then, with overwhelming probability, there exists a network quorum certificate NQC for the same block. This is equivalent to saying that, with overwhelming probability, a block being* ***certified*** *implies it was also* ***confirmed*** *in the same round.*

Proof. Let $honest(EQC)$ be the set of honest endorsers that contributed to an endorser certificate. For a single endorser certificate, we have either of the possibilities: $honest(EQC) \geq 1 \quad \vee \quad honest(EQC) = 0$

It suffices to prove that the first possibility only occurs if the block is ***confirmed***, and that the second possibility occurs with negligible probability. We start by proving the first possibility.

Proof. Due to Rule 2, any honest endorser that contributed to a certificate *must* have voted and then seen a network quorum certificate NQC for the same block, meaning the block was confirmed. Using Lemma 1, we know that there is only one confirmed block per round, which implies that EQC and NQC certify the same block $\mathcal{B}$. ■

The second possibility occurs if enough byzantine validators are selected to be endorsers. We prove that the probability of this happening is negligible.

Proof. Using Lemma 2, we know that $byz(\mathcal{E})$ can be upper bounded by the probability mass function $\mathcal{F}(k, |\mathcal{E}|, 1/3)$. Let q be the quorum percentage of endorsers needed to form a certificate.

We want to know the probability that at least $q \cdot |\mathcal{E}|$ byzantine validators are selected in a given endorser set. This is computed from the binomial CDF:

$$P(byz(\mathcal{E}) \geq q \cdot |\mathcal{E}|) = \sum_{k=q\cdot|\mathcal{E}|}^{|\mathcal{E}|} \binom{|\mathcal{E}|}{k} (1/3)^k (2/3)^{|\mathcal{E}|-k}$$

By selecting appropriate values of q and $|\mathcal{E}|$, the probability that a quorum of byzantine validators is selected to be endorsers can be made negligible. ■

□

Notation 1. *For any block $\mathcal{B}$, let "$\leftarrow$" denote parent relation, i.e. $\mathcal{B}.parent \leftarrow \mathcal{B}$. Let "$\overset{*}{\leftarrow}$" denote ancestry, that is, the reflexive transitive closure of the parent relation.*

Definition 2 (Probabilistic Safety). *Phoenixx is **probabilistically safe** if given two blocks $\mathcal{B}$, $\mathcal{B}'$, such that $\mathcal{B}$ is committed at round k and $\mathcal{B}'$ is committed at round $k' > k$, then, with overwhelming probability, $\mathcal{B}'$ extends $\mathcal{B}$ (i.e., $\mathcal{B} \overset{*}{\leftarrow} \mathcal{B}'$).*

Theorem 1. *Phoenixx is **probabilistically safe** under the sampling and BFT assumptions.*

Proof. Two blocks $\mathcal{B}_0, \mathcal{B'}_0$ are committed if there exist two chains of certified blocks, proposed in contiguous rounds:

$$\mathcal{B}_0 \leftarrow EQC_0 \leftarrow \mathcal{B}_1 \leftarrow EQC_1 \leftarrow \mathcal{B}_2 \leftarrow EQC_2$$

$$\mathcal{B'}_0 \leftarrow EQC'_0 \leftarrow \mathcal{B'}_1 \leftarrow EQC'_1 \leftarrow \mathcal{B'}_2 \leftarrow EQC'_2$$

Without loss of generality, we can assume that $round(\mathcal{B'}_0) \geq round(\mathcal{B}_0)$. We want to prove that for any two committed blocks $\mathcal{B}_0, \mathcal{B'}_0$, such that $round(\mathcal{B'}_0) \geq round(\mathcal{B}_0)$ then, with overwhelming probability, $\mathcal{B}_0 \overset{*}{\leftarrow} \mathcal{B'}_0$.

We have the following two possibilities for the round of block $\mathcal{B'}_0$:

$$round(\mathcal{B}_0) \leq round(\mathcal{B'}_0) \leq round(\mathcal{B}_2) \quad \vee \quad round(\mathcal{B'}_0) > round(\mathcal{B}_2)$$

We start by proving the first possibility.

Proof. In this case, we know that $round(\mathcal{B'}_0)$ must be one of: $round(\mathcal{B}_0)$, $round(\mathcal{B}_1)$ or $round(\mathcal{B}_2)$. Then, using Lemma 3, with overwhelming probability, $\mathcal{B'}_0$ must be one of $\mathcal{B}_0$, $\mathcal{B}_1$ or $\mathcal{B}_2$, which implies that $\mathcal{B}_0 \overset{*}{\leftarrow} \mathcal{B'}_0$. ■

In the second case, let $\mathcal{B'}_i$ be a block such that $round(\mathcal{B'}_i) > round(\mathcal{B}_2)$ and let $\mathcal{B'}_{i-1}$ be the parent of $\mathcal{B'}_i$ i.e., $round(\mathcal{B'}_i) > round(\mathcal{B'}_{i-1})$. Then either $round(\mathcal{B}_0) \leq round(\mathcal{B'}_{i-1}) \leq round(\mathcal{B}_2)$ and by the previous statement, $\mathcal{B'}_{i-1}$ is one of $\mathcal{B}_0$, $\mathcal{B}_1$ or $\mathcal{B}_2$, which implies $\mathcal{B}_0 \overset{*}{\leftarrow} \mathcal{B'}_{i-1}$ **or** $round(\mathcal{B'}_{i-1}) > round(\mathcal{B}_2)$.

Then, we need to prove that $round(\mathcal{B'}_i) > round(\mathcal{B}_2)$ implies $round(\mathcal{B'}_{i-1}) \geq round(\mathcal{B}_0)$.

Proof. We know that, due to Rule 2 and Lemma 3, with overwhelming probability, two endorser quorum certificates are backed by two respective network quorum certificates.

Using Lemma 1, the intersection of two network QCs contains at least one honest validator. This way, there exists an honest validator, **h**, which has voted for both blocks $\mathcal{B}_2$ and $\mathcal{B'}_i$.

Due to Rule 3, after **h** votes for $\mathcal{B}_2$, which includes an EQC for $\mathcal{B}_1$, it sets $preferred\ round = round(parent(\mathcal{B}_1)) \geq round(\mathcal{B}_0)$.

Since $round(\mathcal{B'}_i) > round(\mathcal{B}_2)$, due to Rule 3, **h** could only have voted for $\mathcal{B'}_i$ if

$$round(parent(\mathcal{B'}_i)) \geq preferred\ round \iff round(\mathcal{B'}_{i-1}) \geq round(\mathcal{B}_0)$$

■

By induction, with overwhelming probability: $\mathcal{B}_0 \overset{*}{\leftarrow} \mathcal{B'}_0$ □

8 Conclusion

Recent exponential growth of interest in blockchain technology caused the resurgence of BFT consensus protocols. Protocols such as HotStuff and DiemBFT are a reference, as they achieve linear scalability and optimistic responsiveness at the cost of requiring threshold signatures to achieve linear complexity. Phoenixx breaks free from this paradigm, and takes advantage of a novel *Endorser Sampling* approach to achieve linear complexity without any type of threshold signatures and the associated complex DKG steps.

Phoenixx ensures safety and liveness under partial synchrony, while maintaining optimistic responsiveness. Due to *Endorser Sampling*, these properties are guaranteed probabilistically, but a correct selection of parameters results in negligible probability of failure, safeguarding the practical use of the protocol.

When paired with hash-based signatures, we hope Phoenixx can influence the next generation of scalable and (quantum) secure blockchain applications.

References

1. Baudet, M., Ching, A., Chursin, A., Danezis, G., Garillot, F., Li, Z., Malkhi, D., Naor, O., Perelman, D., Sonnino, A.: State machine replication in the libra blockchain (2019)
2. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018), https://ia.cr/2018/046
3. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology. p. 514–532. ASIACRYPT '01, Springer-Verlag, Berlin, Heidelberg (2001)
4. Buterin, V., Griffith, V.: Casper the friendly finality gadget. CoRR **abs/1710.09437** (2017)
5. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation. p. 173–186. OSDI '99, USENIX Association, USA (1999)
6. Cosmos: The internet of blockchains, https://cosmos.network
7. Cozzo, D., smart, N.P.: Sharing the luov: Threshold post-quantum signatures. Cryptology ePrint Archive, Report 2019/1060 (2019), https://ia.cr/2019/1060
8. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM (JACM) **35**(2), 288–323 (1988)
9. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (Apr 1985)
10. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp. 51–68. ACM (2017)
11. Gueta, G.G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M.K., Seredinschi, D.A., Tamir, O., Tomescu, A.: Sbft: a scalable and decentralized trust infrastructure (2019)
12. Khaburzaniya, I., Chalkias, K., Lewi, K., Malvai, H.: Aggregating hash-based signatures using starks. Cryptology ePrint Archive, Report 2021/1048 (2021), https://ia.cr/2021/1048
13. Kwon, J.: Tendermint: Consensus without mining. Draft v. 0.6, fall **1**, 11 (2015)
14. Kwon, J., Buchman, E.: A network of distributed ledgers. Cosmos pp. 1–41 (2018)
15. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS) **4**(3), 382–401 (1982)
16. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
17. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM (JACM) **27**(2), 228–234 (1980)
18. Polkadot: Decentralized web 3.0 blockchain interoperability platform, https://polkadot.network/
19. Stewart, A., Kokoris-Kogia, E.: Grandpa: a byzantine finality gadget (2020)
20. xx network: A quantum leap in privacy, https://xx.network
21. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: HotStuff: BFT consensus with linearity and responsiveness. In: Proceedings of 2019 ACM Symposium on Principles of Distributed Computing. pp. 347–356. PODC '19, ACM (2019)

A Endorser Sampling analysis

The endorser sampling approach used in Phoenixx is essentially a selection of a fixed number of validators from the network to perform special functions. We define the following:

- Number of validators in the network: n
- Sampled set: $\mathcal{E}$, with size $|\mathcal{E}|$
- Assumed fraction of byzantine validators in the network: b
- Random variable counting the number of byzantine validators selected: Byz

We assume that the algorithm used to sample is unmanipulatable. Therefore, all validators have equal probability of being selected. Thus, the selection is effectively a random sample of the network with multiple draws, without replacement, so Byz follows an hypergeometric distribution:

$$Byz \sim HyperGeo(n, b \cdot n, |\mathcal{E}|) = \frac{\binom{b \cdot n}{k} \times \binom{(1-b) \cdot n}{|\mathcal{E}|-k}}{\binom{n}{|\mathcal{E}|}}$$

We are interested in calculating the probability of failure, which happens when a large number, given by k, of byzantine validators are selected. This probability can be computed using the cumulative distribution function of the hypergeometric as follows:

$$\begin{aligned} P(Byz \geq k) &= 1 - P(Byz \leq k-1) \\ &= 1 - \mathsf{CDF}_{HyperGeo}(n, b \cdot n, |\mathcal{E}|, k-1) \\ &= 1 - \sum_{i=0}^{k-1} \frac{\binom{b \cdot n}{i} \times \binom{(1-b) \cdot n}{|\mathcal{E}|-i}}{\binom{n}{|\mathcal{E}|}} \end{aligned}$$

To provide better comparison between different sampling approaches, we need to eliminate the influence of the size of the network, n. Without loss of generality, we assume that the size of the network is much larger than the size of the sample (i.e., $n \gg |\mathcal{E}|$). Using this assumption, we can see that after each draw, the number of byzantine validators in the network remains virtually the same. This means that the cumulative hypergeometric degrades to a binomial distribution, i.e., sampling with replacement, which provides an upper bound for the probability of failure, given by:

$$\begin{aligned} P_{\mathsf{fail}}(Safety) = P(Byz \geq k) &\approx 1 - \mathsf{CDF}_{Binom}(|\mathcal{E}|, b, k-1) \\ &\approx 1 - \sum_{i=0}^{k-1} \binom{|\mathcal{E}|}{i} b^i (1-b)^{|\mathcal{E}|-i} \end{aligned}$$

Let us assume the sample is selected from the network with the sole purpose of running a BFT consensus algorithm, i.e., only the validators that are selected participate in consensus. For the BFT assumption to stand in the selection, only

up to 1/3 of selected validators can be byzantine. It is easy to see that if the assumption of byzantine validators in the whole network is also 1/3, the selected sample will never be safe:

$$P\left(Byz \geq \frac{|\mathcal{E}|}{3}\right) \approx 1 - \mathsf{CDF}_{Binom}\left(|\mathcal{E}|, 1/3, \frac{|\mathcal{E}|}{3} - 1\right)\Bigg|_{\lim_{|\mathcal{E}|\to n}} = 0.5$$

Of course, if $|\mathcal{E}| = n$, the whole network is selected, and the algorithm degrades into a traditional BFT consensus where safety is preserved at all times. This means that, under the BFT assumption of 1/3 byzantine validators, it is impossible to exclusively use a sample of the network to run consensus. This provides the motivation for relaxing that assumption in other projects to 20% or 25% byzantine validators, in order to maintain scalability. We take Harmony as an example, where each validator is assigned voting shares based on their stake in the protocol. Then, multiple selections of 600 voting shares are performed, one for each shard, with the assumption that up to 25% of voting shares are malicious. The probability of failure of a shard in this setup is given by:

$$\begin{aligned} P_{\mathsf{fail}}(Safety) &= P(Byz \geq 200) \\ &\approx 1 - \mathsf{CDF}_{Binom}(600, 1/4, 199) \approx \mathbf{3 \times 10^{-6}} \end{aligned}$$

In order to represent failure probability in a protocol agnostic manner, we use the Mean Time To Failure (MTTF) metric, which is calculated in years as:

$$MTTF = \frac{SelectionDuration}{3.154 \times 10^7 \cdot P_{\mathsf{fail}}(Safety)}$$

where *SelectionDuration* is the duration of a single selection in seconds.

For Harmony, assuming that fresh selections are run every 24 hours, this yields a MTTF of:

$$MTTF_{Harmony} = \frac{24 \times 3600}{3.154 \times 10^7 \cdot 3 \times 10^{-6}} \approx \mathbf{913\ years}$$

When designing Phoenixx, one of the goals was to provide the best resilience possible to byzantine actors, so we want to keep the traditional BFT assumption of 1/3 of byzantine validators in the network. From the result above, we know the whole network needs to participate in consensus. This was the main motivation behind searching for a novel way of selecting validators to perform special functions, which led to the use of the endorser sampling approach that Phoenixx relies on.

The *Confirm* phase is of paramount importance to the safety of Phoenixx, because it is a step of the protocol that always depends on the full network. Since honest endorsers only endorse a proposal after receiving a full network *QC* of $2f+1$, and there is only one proposal per round that can achieve this, there is no possibility that a network attacker can "trick" honest endorsers into endorsing another proposal. This allows Phoenixx to withstand the influence of more than

a $1/3$ percentage of byzantine validators in a single endorser set without breaking safety. We define q as the quorum percentage of endorsers needed to create an EQC. Safety breaks when the number of byzantine validators selected is $q \cdot |\mathcal{E}|$ or larger. This way, the probability of breaking safety in Phoenixx is given by:

$$P(Byz \geq q \cdot |\mathcal{E}|) \approx 1 - \mathsf{CDF}_{Binom}(|\mathcal{E}|, 1/3, q \cdot |\mathcal{E}| - 1)$$

We can give an example of sensible values of q and $|E|$ that achieve an MTTF, of let's say, at least one million years, assuming each round of Phoenixx takes 2 seconds and, of course, one sample is selected per round:

$$\frac{2}{3.154 \times 10^7 \cdot P_{\mathsf{fail}}(\mathsf{Safety})} \geq 10^6 \iff P_{\mathsf{fail}}(\mathsf{Safety}) \leq \mathbf{6.34 \times 10^{-14}}$$

After iterating different possible solutions, we reach values of $q = 60\%$ and $|\mathcal{E}| = 200$, which allow us to calculate the probability of failure:

$$\begin{aligned} P_{\mathsf{fail}}(\mathsf{Safety}) &= P(Byz \geq 120) \\ &\approx 1 - \mathsf{CDF}_{Binom}(200, 1/3, 119) \approx \mathbf{1.11 \times 10^{-14}} \end{aligned}$$

which results in an MTTF of:

$$MTTF_{Phoenixx} = \frac{2}{3.154 \times 10^7 \cdot 1.11 \times 10^{-14}} \approx \textbf{5.7 million years}$$

This example provides evidence that, due to endorser sampling, Phoenixx can scale better than other BFT based solutions that rely on sampling a subset of the network, while maintaining the standard BFT assumption of $1/3$ byzantine validators. Continuing with this example, we can compute concrete probabilities for all the relevant scenarios explored in the analysis section:

$$\begin{aligned} P_{\mathsf{fail}}(\mathsf{Liveness}) &= P(Byz \geq 121) \\ &\approx 1 - \mathsf{CDF}_{Binom}(200, 1/3, 120) \approx \mathbf{3.65 \times 10^{-15}} \end{aligned}$$

$$\begin{aligned} P_{\mathsf{fail}}(Responsiveness) &= P(Byz \geq 81) \\ &\approx 1 - CDF_{Binom}(200, 1/3, 80) \approx \mathbf{0.02} \end{aligned}$$

$$\begin{aligned} P_{game}(ETC) &= P(Byz = 80) \\ &\approx PDF_{Binom}(200, 1/3, 80) \approx \mathbf{8.24 \times 10^{-3}} \end{aligned}$$

Finally, we note that using the binomial distribution to upper bound the calculations above is useful to provide comparisons with other systems and to

remove the influence of the size of the network from the analysis of endorser sampling. However, we expect the size of networks using Phoenixx to stay in the thousands of validators, for example $n = 1000$. In this case, n is not much larger than $|\mathcal{E}| = 200$, so it is more accurate to use the cumulative hypergeometric distribution directly to calculate the above probabilities and MTTF:

$$P_{\mathsf{fail}}(\mathsf{Safety}) = P(Byz \geq 120)$$
$$= 1 - \mathsf{CDF}_{HyperGeo}(1000, 333, 200, 119) \approx \mathbf{2.59 \times 10^{-18}}$$

$$P_{\mathsf{fail}}(\mathsf{Liveness}) = P(Byz \geq 121)$$
$$= 1 - CDF_{HyperGeo}(1000, 333, 200, 120) \approx \mathbf{6.15 \times 10^{-19}}$$

$$P_{\mathsf{fail}}(Responsiveness) = P(Byz \geq 81)$$
$$= 1 - \mathsf{CDF}_{HyperGeo}(1000, 333, 200, 80) \approx \mathbf{0.01}$$

$$P_{game}(ETC) = P(Byz = 80)$$
$$= \mathsf{PDF}_{HyperGeo}(1000, 333, 200, 80) \approx \mathbf{5.5 \times 10^{-3}}$$

$$\mathsf{MTTF}_{Phoenixx} = \frac{2}{3.154 \times 10^7 \cdot 2.59 \times 10^{-18}} \approx \mathbf{24.5\ billion\ years}$$

This means we can adjust the q parameter to provide lower MTTF, but drastically improve the responsiveness of the system, or reduce the size of the endorser sample $|\mathcal{E}|$, which improves scalability.

B Safety Module implementation

Algorithm 4 - Safety Module implementation

▷ **Safety variables**

$lastProposeRnd, lastVoteRnd, lastEndorseRnd, preferredRnd \leftarrow 0$
$latestVote, latestEndorse \leftarrow nil$

▷ **Helper functions**

function VerifyVotingRule($round$)
    if $round \leq lastVoteRnd$ **then**
        return $false$
    $lastVoteRnd \leftarrow round$
    return $true$
function VerifyEndorsingRule($networkQC$)
    $round \leftarrow networkQC.round$
    $blockID \leftarrow networkQC.blockID$
    if $round \leq lastEndorseRnd$ **then**
        return $false$
    if $latestVote = nil$ **then**
        return $false$
    if $(latestVote.round \neq round) \vee (latestVote.blockID \neq blockID)$ **then**
        return $false$
    $lastEndorseRnd \leftarrow round$
    return $true$
function VerifyLockingRule(eqc)
    if $eqc.round < preferredRnd$ **then**
        return $false$
    if $eqc.parentRound > preferredRnd$ **then**
        $preferredRnd \leftarrow eqc.parentRound$
    return $true$
function VerifyProposeRule($round$)
    if $round \leq lastProposeRnd$ **then**
        return $false$
    $lastProposeRnd \leftarrow round$
    return $true$
function VerifyCommitRule($eqc, round$)
    $twoRound \leftarrow eqc.parentRound$
    $oneRound \leftarrow eqc.round$
    if $(twoRound + 1 = oneRound) \wedge (oneRound + 1 = round)$ **then**
        return $eqc.parentBlock$
    return nil

▷ **Safety Module functions**

function CertifyProposal($proposal$)
    // Check locking rule as it might update preferred round
    VerifyLockingRule($proposal.parentEQC$)
    // Certify proposal by signing the blockID if propose rule allows

```
    rnd ← proposal.round
    signature ← nil
    if VerifyProposeRule(rnd) then
        signature ← Signer.Sign(rnd, propose, proposal.blockID)
    return signature
function AttemptVote(proposal)
    parentEQC ← proposal.parentEQC
    round ← proposal.round
    if (latestVote ≠ nil) ∧ (latestVote.round = round) then
        return latestVote
    if VerifyLockingRule(parentEQC) = false then
        return nil
    if VerifyVotingRule(round) = false then
        return nil
    commit ←VerifyCommitRule(parentEQC, round)
    // Create, certify and store confirm vote for the proposal
    vote ← Vote{proposal.block, confirm, commit, validatorID}
    vote.sig ← Signer.Sign(round, confirm, vote.data)
    latestVote ← vote
    return vote
function AttemptEndorsement(proposal, networkQC)
    parentEQC ← proposal.parentEQC
    round ← proposal.round
    if (latestEndorse ≠ nil) ∧ (latestEndorse.round = round) then
        return latestEndorse
    if VerifyEndorsingRule(networkQC) = false then
        return nil
    commit ←VerifyCommitRule(parentEQC, round)
    // Create, certify and store endorsement for the proposal
    endorsement ← Vote{proposal.block, endorse, commit, validatorID}
    endorsement.sig ← Signer.Sign(round, endorse, vote.data)
    latestEndorse ← endorsement
    return endorsement
function CertifyTimeout(round, endorser)
    if round < lastVoteRnd then
        return nil
    // Create and certify timeout vote with correct type for the round
    type ← endorser ? endorseTimeout : timeout
    timeout ← Vote{round, type, nil, validatorID}
    timeout.sig ← Signer.Sign(round, type, timeout.data)
    // Disallow voting for the round
    if round > lastVoteRnd then
        lastVoteRnd ← round
    return timeout
```