BlueBottle: Fast and Robust Blockchains through Subsystem Specialization

Preston Vander Vos‡, Alberto Sonnino§‡, Giorgos Tsimos*, Philipp Jovanovic§‡, Lefteris Kokoris-Kogias§

§Mysten Labs, *pod Network, ‡University College London (UCL)

***Abstract*—Blockchain consensus faces a trilemma of security, latency, and decentralization. High-throughput systems often require a reduction in decentralization or robustness against strong adversaries, while highly decentralized and secure systems tend to have lower performance. We present BlueBottle, a two-layer consensus architecture. The core layer, BB-Core, is an $n = 5f+1$ protocol that trades some fault tolerance for a much lower finality latency with a medium-sized core validator set. Our experiments show that BB-Core reduces latency by 20–25% in comparison to Mysticeti. The guard layer, BB-Guard, provides decentralized timestamping, proactive misbehavior detection in BB-Core, and a synchronous recovery path. When it observes equivocations or liveness failures in the core–while tolerating up to $f < 3n/5$ faulty nodes in the primary layer–guard validators disseminate evidence, agree on misbehaving parties for exclusion or slashing, and either restart the core protocol (for liveness violations) or select a canonical fork (for safety violations). Together, these layers enable optimistic sub-second finality at high throughput while maintaining strong safety and liveness under a mild synchrony assumption.**

1. Introduction

The design of robust and efficient consensus protocols remains a cornerstone challenge in distributed systems, particularly within the demanding context of blockchain technology. Existing blockchain systems face a trilemma of high-security thresholds, low transaction-finality latency, and extensive decentralization. As an example of these trade-offs, consider Ethereum, Solana, and Avalanche, three of the most influential blockchain systems that have demonstrated their value over the last decade. Ethereum [1] takes a strong stance on extensive decentralization to ensure that thousands of participants can audit the chain's state transitions and even participate in the consensus protocol, and requires high-security thresholds with a theoretical fault-tolerance of $n = 3f + 1$. As a consequence of these two design choices, Ethereum has an average transaction finality latency of several minutes, though. Avalanche [2], on the other hand, prioritizes decentralization and low transaction finality at the cost of weaker probabilistic safety guarantees and a fault tolerance somewhere between $f = \sqrt{n}$ and $n = 5f + 1$ through its scalable Snowflake consensus protocol [3]. Finally, Solana [4] trades decentralization for resilience with $n = 3f + 1$ and performance, i.e., low latency and high throughput, but relies on professional validators[1].

This paper introduces BlueBottle, a novel dual-layer consensus architecture designed to synergistically combine the strengths of different consensus paradigms and thereby mitigate this trilemma. At BlueBottle's core is a partially synchronous consensus protocol that trades off failure tolerance ($n = 5f + 1$) for significantly lower latency than other state-of-the-art consensus protocols and is operated by a medium-sized set of core validators. A secondary synchronous Byzantine Agreement protocol, which is operated by a potentially much larger set of guard validators (or simply "guards"), complements this core engine by providing enhanced decentralization, high resilience, and a robust recovery mechanism. Our approach aims to deliver optimistic sub-second transaction finality while simultaneously harnessing the scale and resilience benefits of large-scale decentralized protocols to quickly detect and recover.

The core layer. Building BlueBottle's core layer requires a high-performance consensus protocol. Our goal is simple: achieve the lowest possible finality latency and the highest experimentally demonstrated throughput. To do so, we introduce BB-Core, which builds on top the state-of-the-art partially synchronous DAG-based consensus protocol Mysticeti [5] and reduces latency by trading fault tolerance from $n = 3f + 1$ to $n = 5f + 1$, where n is the number of validators and f is the number of Byzantine faults tolerated. BB-Core introduces the first $5f + 1$ DAG-based consensus protocol and commits transactions in under 0.5,s while sustaining more than 200,000 tx/s (Section 6)—a performance level unmatched in the BFT consensus literature. We also introduce an asynchronous variant of BB-Core in Appendix C, which tracks the performance of BB-Core with a 33% latency increase.

We use the standard two-step design: first, consensus fixes a single order of operations for everyone, and then identical replicas apply that order ensuring they all end up in the same state (state machine replication). Clients can observe finality in two ways. Clients who follow every block see finality in about $1\,\mathrm{RTT}$; under $n = 5f+1$, this becomes roughly $0.5\,\mathrm{RTT}$ faster than the equivalent $n = 3f + 1$ protocol. Clients who do not require such low finality and prefer to avoid the bandwidth cost of streaming the DAG can

1. The votes of the top 84 validators, in terms of stake, are enough to achieve supermajority. As per https://solanabeach.io/validators (Nov 25).

rely on checkpoints, that is, compact summaries of recent updates that are re-agreed upon through consensus. These clients wait for two steps (one to commit and one to agree on the commit), and this increase in latency comes with increased safety. Specifically, they remain safe if fewer than 60% of validators are malicious. This holds because even if an adversary creates two forks with $f+1$ adversarial validators, only one of the forks can be directly committed through a checkpoint with $4f+1$ support as long as at least 40% of the validators are honest.

The guard layer. The guard protocol in BlueBottle is a highly decentralized auditing and recovery layer, called BB-Guard. BB-Guard operates in synchronous steps and monitors the primary layer for liveness or safety violations (possible up to $f < 3n/5$ faults in the primary layer) in the form of equivocations. If this happens, it enables a synchronous recovery protocol.

Upon detecting such misbehavior, the guard validators transition to a recovery phase. This consists of running Byzantine Broadcast of accumulated evidence and deterministically agreeing on the set of core validators that misbehaved and then excluding them, potentially by slashing their stake. If the violation relates to liveness, then the reduced set of core validators restarts consensus, whereas if it was a safety violation, they first agree on the canonical fork. As a result, this safety mechanism allows BlueBottle to operate with optimistic responsiveness while retaining strong safety and liveness guarantees that rely on a synchrony assumption orders of magnitude longer than the speed of the chain.

Contributions. We make the following contributions:

- We present BlueBottle, a novel dual-layer consensus architecture that simultaneously achieves low latency, high security, and high decentralization.
- We introduce the two main novel building blocks of BlueBottle, namely BB-Core, a DAG-based consensus protocol that reduces latency by lowering fault-tolerance to $n = 5f + 1$ and BB-Guard, a highly decentralized guard protocol for misbehavior monitoring and recovery.
- We provide a rigorous security analysis of both BB-Core and BB-Guard, overall demonstrating that BlueBottle achieves optimal safety and liveness guarantees under standard assumptions.
- We implement and evaluate BB-Core and compare it at scale against Mysticeti, confirming the expected latency gains of around 20–25% while maintaining similar throughput to Mysticeti.

2. System Overview

This section provides an overview on BlueBottle and presents, in particular, the system and threat models, the design goals, and a design overview.

2.1. System model

We assume a total number of nodes n. BlueBottle has a *core* and an *guard* layer and distinguishes between *core* and *guard* validators (or guards). We denote the number of core validators as n_c and the number of guard validators as n_g such that $n = n_c + n_g$. Core validators continuously operate the core consensus protocol and process transactions, while guard validators checkpoint the output and audit the operation of core validators and help with recovery. Both core and guard validators are selected using a Sybil-resistant election mechanism [6] and we usually assume a proof-of-stake approach [7]. We can translate the quantitative assumptions on the nodes to stake assumptions and we often use the two concepts interchangeably. Core validators are chosen similarly to existing quorum-based blockchains, consisting of roughly the 100 entities with the highest stake or those that meet specific criteria, such as owning a minimum percentage of the total stake [8]. Guard validators include all other stakeholding entities not in this core group. The number of guards is assumed to be significantly larger than the number of core validators (multiple hundreds), while their total amount of stake is significantly smaller. In practice, we expect full nodes to operate as guards. We provide a discussion on the stake distribution in the next section. BlueBottle operates as a message-passing system.

2.2. Threat model

BlueBottle assumes a computationally bounded adversary, ensuring that common cryptographic security assumptions, like those for hash functions and digital signatures, hold. We assume there are at most f Byzantine nodes, with $n = 2f + 1$. Byzantine nodes may behave arbitrarily, whereas the remaining $n - f$ nodes are honest and follow the protocol. We provide a security assessment on the stake requirements for the BB-Core and BB-Guard layers at the end of this section.

Core layer. BlueBottle uses the novel BB-Core consensus protocol at its core, which assumes that the total number of core validators n_c satisfies $n_c \geq 5f_c + 1$, where f_c is the maximum number of Byzantine core validators which may deviate from the protocol arbitrarily. The remaining core validators are assumed to be honest following the protocol specification. We further assume that BB-Core operates in a partially synchronous [9] network. We also provide a variant of BB-Core for asynchronous networks in Appendix C. Since the threat model of the core layer is significantly weaker than the global assumption, it can fail. However, it will only fail accountably and, as a result, allow the system to detect the adversarial nodes and exclude them, slowly converging the threat model to the operational requirements of the core layer.

Guard layer. For the guard layer, BlueBottle uses BB-Guard and assumes a synchronous network, which is a stronger networking model. Consequently, this synchrony assumption also applies to BlueBottle overall. However, BlueBottle provides optimistic responsiveness when the assumptions of BB-Core hold and thus faster finality than what is achievable under synchrony. When the BB-Core assumptions are violated, we use the synchrony assumption of BB-Guard to detect accountable faults and recover.

Client Finality Guarantees. The client finality guarantees are a local assumption. This is on par with designs around Flexible BFT [10, 11]. Clients who believe the adversary is not actively compromising BB-Core can get the faster finality, whereas the rest can follow the checkpoint finality.

Stake distribution between validator groups. We show how to translate the quantitative assumptions from our threat model to stake assumptions and determine the stake distribution between core and guard validators. We assume there is a total amount of stake S available such that $S = 2S_f + 1$ where S_f is the stake controlled by the adversary. We denote the stake assigned to core validators by S_c, the stake assigned to guard validators by S_g, and require that $S = S_c + S_g$. Given our initial stake distribution assumptions, the security assumption of the core layer might be violated at some point. However, in this case, we still want to achieve safety of checkpoints. To preserve checkpoint safety, the adversary is allowed to control at most 60% of the core validators' stake. Let S_{c_f} denote the core stake controlled by the adversary, we must thus maintain the invariant that

$$S_{c_f} \leq 3/5 \cdot S_c \ . \tag{1}$$

Since the adversary can concentrate its full corruption budget $S_f = (S-1)/2$ on the core validators, i.e., $S_{c_f} = (S-1)/2$, and considering that invariant 1 must nevertheless hold, we obtain

$$(S-1)/2 \leq 3/5 \cdot S_c \Rightarrow S_c \geq 5(S-1)/6 \ . \tag{2}$$

In other words, we require that the total stake contributed by the core validators S_c is at least $\frac{5(S-1)}{6}$.

Consequently, the stake contributed by the guard validators S_a is $S - S_c \leq \frac{S+5}{6}$. For simplicity, we can set $S_c = S \cdot 5/6$ and $S_g = S \cdot 1/6$.

2.3. Design goals

BlueBottle has the following three primary design goals in terms of performance, security, and decentralization.

- **G1 High performance:** BlueBottle provides significant lower latency and equally high throughput as state-of-the-art (DAG-based) consensus protocols when all BB-Core underlying assumptions hold.
- **G2 High security:** BlueBottle can tolerate up to S_f stake being controlled by the adversary with the total amount of stake being $S = 2S_f + 1$.
- **G3 High decentralization:** BlueBottle enables non-core-validator participants to contribute to the security of the system through BB-Guard.

2.4. Design overview

We present an overview of BB-Core and BB-Guard.

2.4.1. The core layer.

BlueBottle introduces the novel consensus protocol BB-Core for its core layer. BB-Core builds atop an uncertified DAG [5, 12, 13], and is the first $5f + 1$ DAG-based consensus protocol. It trades off the security threshold from $n = 3f + 1$ to $n = 5f + 1$ to achieve finality in 2 message delays instead of the original 3. The BB-Core protocol follows similar ideas to Mysticeti with the following modifications to its decision rules:

- **Direct decision rule:** Commit (skip) a leader in round R, if it has $4f + 1$ votes (blames) from round $R + 1$.
- **Indirect decision rule:** Commit an undecided leader in round R, if there is a committed leader in round $R + 2$ that supports the round-R leader with at least $2f + 1$ round-$R + 1$ votes.

We introduce BB-Core formally in Section 3, analyse its security in Section 4, and show that it indeed achieves lower latency and equally high throughput as Mysticeti in Section 6. BB-Core commit in under 0.5s while sustaining a load of over 200,000 tx/s (Section 6).

2.4.2. The guard layer.

BlueBottle introduces a novel guard protocol, BB-Guard, which serves three goals:

Fault detection. This is the common-case execution of BB-Guard. Guard validators synchronously gossip the blocks output by BB-Core as a commit sequence and make sure that there is no equivocated block proposal. If this is true and no equivocation is detected within the timeout the BB-Core continues uninterrupted. Otherwise, BB-Guard has to proceed to accountability assignment.

Accountability assignment. There are two types of accountable faults in BlueBottle: safety and liveness faults. Safety faults are directly detected through the previous mechanism and, once detected, are flagged to all connected peers. Liveness faults are only detected when a core validator flags a set of $f + 1$ validators as non-responsive and sends this through the gossip network for notarization. At this point, the guard validators start a timer to see if the DAG building has halted. If the timer expires, they locally decide that there are liveness faults.

Synchronous recovery. When a core validator detects a fault, it starts the synchronous recovery protocol. For this, it first decides on an accountable set of faulty validators it wishes to exclude. This set is unique and the core validator cannot change it afterwards. The accountable set comes from either safety faults (i.e., double-spending from the core), or liveness faults as described above. In either event, an honest validator (core or guard) can deduce at least an $f + 1$-sized accountable set.

The validator then Byzantine Broadcasts (BB) this set and all validators sign the set if they believe that the set indeed constitutes Byzantine validators. The first BB that outputs a valid set is considered the last message of the view. The validators exclude the blamed stakeholders from the new view and return to regular operation.

Algorithm 1 BB-Core

```
leadersPerRound                                   ▷ A number between 1 and 4f + 1
waveLength                          ▷ Set to 2 for BB-Core, 3 for BB-Core-Async

procedure TryDecide(r_committed, r_highest)
    S ← [ ]                                                     ▷ Holds decisions
    for r ← r_highest down to r_committed + 1 do
        for l ← leadersPerRound − 1 down to 0 do
            i ← r mod waveLength
            D ← Decider(i, l)
            w ← D.WaveNumber(r)
            s ← D.TryDirectDecide(w)
            if s = ⊥ then s ← D.TryIndirectDecide(w, S)
            S ← s || S
    return S

procedure ExtendCommitSeq(r_committed, r_highest)
    S ← TryDecide(r_committed, r_highest)
    S_commit ← [ ]                                       ▷ Holds committed blocks
    for s ∈ S do
        if s = ⊥ then break
        if s = Commit(b_leader) then S_commit ← S_commit || b_leader
    return LinearizeSubDags(S_commit)                 ▷ Same as DAG-Rider [14]
```

Algorithm 2 Decider Instance

```
waveOffset = i                         ▷ The first parameter of the Decider (i)
leaderOffset = l                      ▷ The second parameter of the Decider (l)
waveLength                  ▷ Set to 2 for BB-Core, 3 for BB-Core-Async

procedure WaveNumber(r)
    return (r − waveOffset)/waveLength

procedure ProposeRound(w)
    return (w ∗ waveLength) + waveOffset

procedure DecisionRound(w)
    return ProposeRound(w)+(waveLength − 1)

procedure StronglyCertifiedLeader(w, b_leader)
    B_decision ← GetDecisionBlocks(w)
    return |{b' ∈ B_decision : IsVote(b', b_leader)}| ≥ 4f + 1

procedure SkippedLeader(w, b_leader)
    B_decision ← GetDecisionBlocks(w)
    return |{b' ∈ B_decision : ¬IsVote(b', b_leader)}| ≥ 4f + 1

procedure TryDirectDecide(w)
    B_leader ← GetLeaderBlocks(w, leaderOffset)
    for b_leader ∈ B_leader do
        if SkippedLeader(w, b_leader) then return Skip
        if StronglyCertifiedLeader(w, b_leader) then return Commit(b_leader)
    return ⊥

procedure WeaklyCertifiedLeader(b_anchor, b_leader)
    w ← WaveNumber(b_leader.round)
    B_decision ← GetDecisionBlocks(w)
    return |{b ∈ B_decision : IsVote(b, b_leader) ∧ Link(b, b_anchor)}| ≥ 2f + 1

procedure TryIndirectDecide(w, S)
    r_decision ← DecisionRound(w)
    s_anchor ← first s ∈ S s.t. r_decision < s.round ∧ s ≠ Skip
    if s_anchor = Commit(b_anchor) then
        B_leader ← GetLeaderBlocks(w, leaderOffset)
        if ∃ b_leader ∈ B_leader s.t. WeaklyCertifiedLeader(b_anchor, b_leader) then return Commit(b_leader)
        else return Skip
    return ⊥
```

3. The BB-Core Protocol

BB-Core is the first $5f + 1$ DAG-based consensus protocol. It is built on top of an uncertified DAG, similarly to Mysticeti [5] and Cordial Miners [12]. It commits in two message delays by lowering the fault threshold to $f < n/5$ ($\approx 20\%$). Algorithm 1 provides the main entry point and is invoked whenever a validator receives a new block. Algorithm 2 specifies the decision process, with each validator instantiating one *Decider* instance per leader slot. Finally, Algorithm 3 defines utility procedures that support common operations used throughout the protocol. The orange-colored lines of the algorithm refer to the asynchronous version of BB-Core which we prove in the Appendix. This is of independent interest and can be ignored.

3.1. The directed acyclic graph

BB-Core builds a *directed acyclic graph (DAG)* of *blocks* that reference each other via cryptographic hashes, similar to Mysticeti. This DAG enables validators to decide (locally) which blocks to commit and in what order. The BB-Core protocol proceeds in a sequence of logical *rounds*. In each round, every honest validator proposes exactly one.

Block creation and validation. A block must include at least the following elements to be valid: (1) the author A of the block together with their valid cryptographic signature on the block's contents; (2) a round number R; (3) a list of transactions; and (4) at least $4f + 1$ distinct hashes of valid blocks from the previous round $R - 1$.

Honest validators store only valid blocks in their local DAG and discard invalid ones. Moreover, they reference a block's hash in their proposals only if the block is valid and after downloading and verifying its entire causal history, thereby ensuring the correctness of the block's lineage.

Rounds and waves. As already mentioned, BB-Core operates in *rounds* and two subsequent rounds R and $R + 1$ form a *wave*. Figure 1 (left) illustrates an example of a single wave with six validators $(v_0, v_1, v_2, v_3, v_4, v_5)$. The first round R (Propose) of a wave contains the blocks that the wave attempts to commit $(P_0, P_1, P_2, P_3, P_4, P_5)$ along with the equivocating block P_1'. In the second round $R+1$ (Decision), each block serves as a *vote* for the Propose blocks it references. In the example of Figure 1, blocks V_0, V_1, V_2, V_3, and V_4 vote for P_0, P_1, P_2, P_3, and P_4 (but not for P_1' or P_5), whereas block V_5 votes for P_1', P_2, P_3, P_4, and P_5 (but not for P_0 or P_1). The procedure $\textsc{IsVote}(\cdot)$ in Algorithm 3 formally defines a vote. A Propose block is considered *strongly certified* if it has at least $4f+1$ votes. We also say that the voting blocks form a *strong certificate* for the proposed block. In the example, blocks P_0, P_1, P_2, P_3, and P_4 are strongly certified, while P_1' and P_5 are not. If a block receives at least $2f+1$ votes but fewer than $4f+1$, it is considered *weakly certified*. As before, we also say that the voting blocks form a *weak certificate* for the proposed block. As shown in Figure 1 (right), BB-Core initiates a new wave every round: round R is a Propose round for wave 1, round $R+1$ is a Decision round for wave 1 and a Propose round for wave 2, etc. Algorithm 2 formally defines a wave.

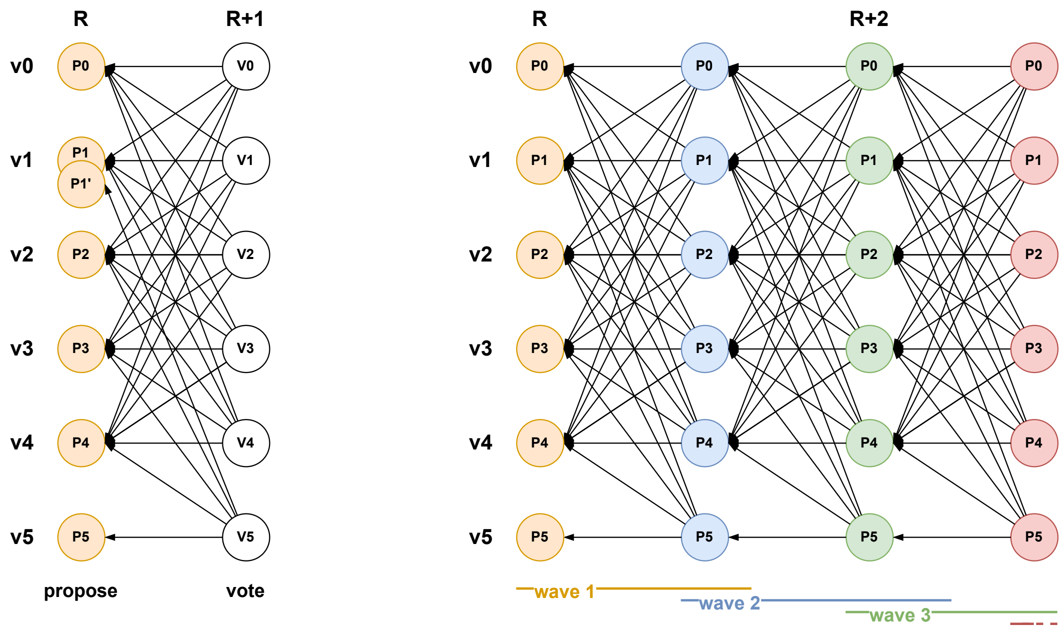


Figure 1: The structure of the BB-Core DAG. Left: A wave consisting of two rounds (Propose and Decision). Right: Wave patterns in the BB-Core protocol (each round initiates a new overlapping wave).

Algorithm 3 Helper Functions

`validators` ▷ The set of validators
`async` ▷ Whether the protocol is asynchronous

procedure GETDECISIONBLOCKS(w)
  $r_{decision} \leftarrow$ DECISIONROUND(w)
  return $DAG[r_{decision}]$

procedure GETLEADERBLOCKS($w, rank$) ▷ Validators may equivocate
  $r_{propose} \leftarrow$ PROPOSEROUND(w)
  if `async` **then**
    $r_{decision} \leftarrow$ DECISIONROUND(w)
    $s \leftarrow$ COMBINECOINSHARES($\{b.share$ s.t. $b \in DAG[r_{decision}]\}$)
  else
    $s \leftarrow r_{propose}$
  $leader \leftarrow \texttt{validators}[(s + rank) \bmod |\texttt{validators}|]$
  return $\{b \in DAG[r_{propose}] : b.author = leader\}$

procedure ISVOTE($b_{support}, b_{leader}$)
  /* Note: If $\texttt{waveLength} = 2$, equivalent to LINK($b_{leader}, b_{support}$) */
  function VOTEDBLOCK(b, id, r) ▷ Depth-first search
    if $r \geq b.round$ **then return** $\perp$
    for $b' \in b.parents$ **do**
      if $(b'.author, b'.round) = (id, r)$ **then return** b'
      $res \leftarrow$ VOTEDBLOCK(b', id, r)
      if $res \neq \perp$ **then return** res
    return $\perp$
  $(id, r) \leftarrow (b_{leader}.author, b_{leader}.round)$
  return VOTEDBLOCK($b_{support}, id, r$) $= b_{leader}$

procedure LINK(b_{old}, b_{new})
  return $\exists$ a sequence of $k \in \mathbb{N}$ blocks $b_1, \ldots, b_k$ s.t.
   $b_1 = b_{old} \wedge b_k = b_{new} \wedge \forall j \in [2, k] : b_j \in \bigcup_{r \geq 1} DAG[r] \wedge b_{j-1} \in b_j.parents$

3.2. Proposers and anchors

BB-Core's leader slots represents a pair (validator, round) and can be either empty or contain the validator's proposal for the respective round. If the validator is Byzantine, they may have equivocated, in which case the slot would contain more than one block.

Multiple leader slots can be instantiated per round, enabling parallel leader block proposals. Each slot is in one of three states: `commit`, `skip`, or `undecided`. All slots begin in the `undecided` state, and the protocol's objective is to classify them as either `commit` or `skip`. The commit state indicates that the slot's block should be included in the total ordering, while the `skip` state allows the protocol to exclude slots from crashed or Byzantine validators. Crucially, the `undecided` state forces subsequent leader slots to wait, preventing unsafe commitments. Similarly to related work [5, 15, 13, 16], the number of leader slots per round is a system parameter. Before advancing round, validators wait up to 2Δ time to receive the blocks from the leaders slots of the previous round, if they have not already.

3.3. The decision rule

We present the decision rule of BB-Core through an example protocol run. Figure 2 shows the local view of a BB-Core validator in a system with six validators $(v_0, v_1, v_2, v_3, v_4, v_5)$, parameterized with two leader slots per round. Following standard notation, we denote a block as $B_{(v_i,R)}$, where v_i is the issuing validator and R is the block's round.

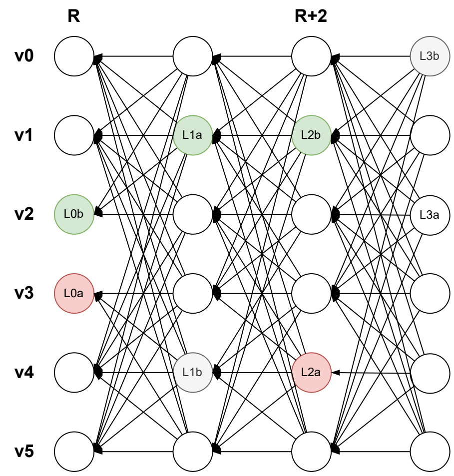


Figure 2: Example with six validators and two leader slots per round.

Initially, all proposer slots are in the `undecided` state. The validator examines the portion of the DAG shown in Figure 2 and attempts to classify as many leader-slot blocks as possible into either `commit` or `skip`.

Step 1: Determine the leader slots. The sequence of leader slots for each round is predefined and known to all validators. In the example of Figure 2, every validator knows in advance that the leader slot sequence is [L_{0a}, L_{0b}, L_{1a}, L_{1b}, L_{2a}, L_{2b}, L_{3a}, L_{3b}]. This deterministic mechanism ensures that even if validators have divergent DAG views, they still agree on the leader slots (and their order) for a given round—regardless of whether a block has been observed in that slot. This design also enables low latency by allowing multiple leaders per round, using these slots to order the causal history.

Step 2: Direct decision rule. The validator attempts to classify each slot (even those without a block) as either `commit` or `skip`. To do so, it processes each slot individually, starting with the lowest slot that is part of a complete wave (L_{2b}), applying the BB-Core *direct decision rule*. The validator classifies a block B in a slot as `skip` if it observes $4f+1$ blocks from the subsequent Decision round that do not vote for it, and as `commit` if it is strongly certified. As discussed in Section 3.1, a block B is strongly certified if there are $4f+1$ blocks voting for it. Otherwise, the validator leaves the slot as `undecided`.

In this example, the validator first analyzes L_{2b} and observes that $B_{(v_0,R+3)}$, $B_{(v_1,R+3)}$, $B_{(v_2,R+3)}$, $B_{(v_3,R+3)}$, and $B_{(v_5,R+3)}$ strongly certify it. Therefore, it classifies L_{2b} as `commit`. Section 6 shows that this scenario is the most common (in the absence of an adversary) and results in the lowest latency. The validator then analyzes L_{2a} and observes that $B_{(v_0,R+3)}$, $B_{(v_1,R+3)}$, $B_{(v_2,R+3)}$, $B_{(v_3,R+3)}$, and $B_{(v_5,R+3)}$ do not vote for it. Therefore, it classifies L_{2b} as `skip`. The presence of $4f+1$ blocks from the Decision round that do not vote for a block ensures that it will never be certified, and will thus never be committed by other validators with a potentially different local view of the DAG. Section 6 shows that this rule allows BB-Core to promptly skip (benign) crashed leaders to minimize their impact on the protocol's performance.

Step 3: Indirect decision rule. In the case where the direct decision rule cannot classify a slot, the validator uses the BB-Core *indirect decision rule*. This rule looks at future slots to decide about the current one. First, it finds an *anchor*. This is the earliest leader slot with a round number $R' > R+1$ that is either still classified as `undecided` or already classified as `commit`. If the anchor is `undecided`, the validator marks the current slot as `undecided`. If the anchor is `commit`, the validator checks if it references at least one weak certificate over the current slot. If it does, the validator marks the current slot as `commit`. If it does not, the validator marks the current slot as `skip`. Section 4 shows the direct and indirect decision rules are consistent, namely if one validator direct commits a block no honest validators will indirect skip it and vice versa.

In this example, the validator fails to classify L_{0b} using the direct decision rule as it is neither strongly certified nor are there enough blocks from the subsequent Decision round that do not reference it to classify it as `skip`. It thus searches for its anchor. Since L_{2a} has been classified as `skip`, it cannot serve as an anchor; therefore, L_{2b} is the anchor for L_{0a}. Given that L_{2b} references $B_{(v_0,R+1)}$, $B_{(v_1,R+1)}$, and $B_{(v_2,R+1)}$, which form a weak certificate for L_{0a}, the validator classifies L_{0a} as `commit`.

Finally, L_{0a} cannot be classified using the direct decision rule as well. Its anchor is again L_{2b}, but it does not reference a weak certificate over L_{0a}. Therefore, the validator classifies L_{0a} as `skip`.

Step 4: Commit sequence. After processing all slots, the validator derives an ordered sequence of the leader-slot blocks. It then iterates over this sequence, committing all slots marked as `commit` and skipping all slots marked as `skip`, until it encounters the first `undecided` slot. As shown in Section 4, this commit sequence is safe, and eventually, every slot is classified as either `commit` or `skip`. In the example of Figure 2, the validator's leader sequence is [L_{0b}, L_{1a}].

Following the approach introduced by DagRider [14], the validator derives the final commit sequence by linearizing the blocks within the sub-DAG defined by each leader block using a depth-first search. If a block has already been included by a previous leader slot, it is not re-linearized. Leader slots are processed sequentially, ensuring that all blocks appear in the final commit sequence exactly once and in an order consistent with their causal dependencies. The procedure LINEARIZESUBDAGS$(\cdot)$ (Algorithm 1) formalizes this step. In the running example, the commit sequence is [L_{0b}, $B_{(v_0,R)}$, $B_{(v_1,R)}$, $B_{(v_4,R)}$, $B_{(v_5,R)}$, L_{1a}].

4. BB-Core Security and Liveness

Lemma 1. There will never be a block that an honest authority directly commits while another honest authority directly skips.

Proof 1. Assume for the sake of contradiction that such a block exists and call it B. Thus, there are $4f+1$ authorities which support B and $4f+1$ authorities which blame B. Since f authorities are Byzantine, there are $3f+1$ honest authorities that support B and a distinct set of $3f+1$ honest authorities that blame B. This means there are $7f+2$ authorities overall contradicting our assumption of $n=5f+1$.

Lemma 2. There will never be a block that an honest authority directly skips while another honest authority indirectly commits.

Proof 2. Assume for the sake of contradiction that such a block exists and call it B. Thus, there are $4f+1$ authorities which blame B and $2f+1$ authorities which support B. Since f authorities in the network are Byzantine, there are $3f+1$ honest authorities that blame B and a distinct set of $f+1$ honest authorities that support B. This means there are $5f+2$ authorities overall contradicting our assumption of $n=5f+1$.

Lemma 3. If at a round R, $4f+1$ blocks from distinct authorities support a block B, then all blocks at future rounds $R' > R$ will link to $2f+1$ supports for B from round R.

Proof 3. Each block links to $4f+1$ blocks from the previous round. For the sake of contradiction, assume that a block B' in round $R' > R$ does not link to $2f+1$ supports for B from round R.

Case: $R' = R+1$. B' links to $4f+1$ blocks from round R. Since $4f+1$ blocks in round R support B, by the standard quorum intersection, the minimum overlap between B' parents and B supports is $2f+1$. Thus, the only way for B' to not link to $2f+1$ supports is if an honest validator equivocated in round R. This is a contradiction.

Case: $R' > R+1$. B' links to $4f+1$ blocks from round $R'-1$. At least $3f+1$ of these blocks are produced by honest authorities. Honest authorities always link to their own blocks, which means they will eventually link to their block from round $R+1$. The above case proves how these blocks from round $R+1$ link to $2f+1$ supports for B. Thus, the only way for B' to not link to $2f+1$ supports for B is if all of these honest authorities do not link to their own block in round $R+1$. This is a contradiction.

As a result of Lemma 3, we have the following corollary.

Corollary 1. There will never be a block which an honest authority directly commits while another honest authority indirectly skips.

Lemma 4. All honest authorities who have decided on a leader block, agree on the decision.

Proof 4. Let B_n and B_m be the highest committed leader blocks according to authorities X and Y respectively. Without loss of generality, let $n \leq m$. Note that leader blocks decided by X which are higher than B_n are direct skips which according to Lemma 1 and Lemma 2 will be consistent with Y's decision. The proof continues by induction on the statement for $0 \leq i \leq n$, if both X and Y decide on leader block B_i, then they either both commit or both skip the block.

Case: $i = n$. By definition, X directly commits B_i and from Lemma 1 and Corollary 1, Y will also commit B_i.

Case: Assuming the statement is true regarding B_i for $k+1 \leq i \leq n$, we prove it is true for B_k. This is done by enumerating decision possibilities.

1) If either authority directly commits B_k, then by Lemma 1 and Corollary 1, the other will commit.
2) If either authority directly skips B_k, then by Lemma 1 and Lemma 2, the other will skip.
3) Both X and Y indirectly decide B_k. Let A_c^X and A_d^Y be the anchors used by X and Y to indirectly decide B_k. Since $k+1 < c \leq n$, it follows from the induction hypothesis that $A_c^X = A_d^Y$. Thus, both X and Y use the same anchor to decide B_k. The indirect decision rule solely depends on the causal history of the anchor. By using the same anchor, X and Y will agree on the decision for B_k.

Theorem 1. BB-Core maintains safety.

Proof 5. The final state of consensus is a total ordering of all blocks. In BB-Core, the total ordering is maintained until an undecided leader block is encountered. From Lemma 4, all honest validators agree upon the decisions of all leader blocks until an undecided leader block. The total ordering of all blocks is a deterministic algorithm run on the sequence of committed leader blocks. Since all honest validators have the same sequence of committed leader blocks, the total ordering will be the same.

Lemma 5. After GST, all honest authorities will enter the same round within Δ.

Proof 6. Messages sent before GST will deliver in Δ after GST commences. Thus, the valid block of the highest round that any authority sent before GST will be delivered to all authorities in GST + Δ. Upon receiving this block, all honest authorities will enter the round.

Lemma 6. After GST, leader blocks from honest authorities will receive support from all honest authorities.

Proof 7. By Lemma 5, all honest authorities will enter the same round within Δ after GST. When an honest authority sends its leader block for this round, it is delivered to all authorities within Δ. Since the protocol sets the timeout to $2 \cdot \Delta$, any honest authority that has entered the round will receive the leader block and cast support for it before timing out, regardless of when exactly they entered the round relative to other honest authorities.

As a result of Lemma 6, we have the following corollary. Recall that there are $4f+1$ honest authorities.

Corollary 2. After GST, leader blocks from honest authorities will be (directly) committed.

Lemma 7. The round-robin schedule of leader-block proposers ensures that, within a window of $2f + 2$ rounds, there are two consecutive rounds in which an honest authority is the proposer of the highest-ranked leader block.

Proof 8. The network contains f Byzantine authorities. In $2f + 2$ rounds, there are $2f + 1$ sets of two consecutive rounds. Due to the schedule being round robin, in at least $f + 1$ of the rounds, an honest authority will be the proposer of the highest-ranked leader block. These blocks are the highest-ranked leader block in exactly two of the sets. By the pigeonhole principle, one set must contain $\lceil \frac{2\cdot(f+1)}{2f+1} \rceil = 2$ honest authorities proposing the highest-ranked leader block.

Lemma 8. After GST, undecided leader blocks will eventually be decided.

Proof 9. Let B be an undecided leader block in round r. By Lemma 7, after GST, there will be two consecutive rounds, j and $j+1$ with $j > r$, where honest authorities propose the highest-ranked leader block. By Corollary 2, their leader blocks will be committed. The proof continues by induction on the statement for rounds earlier than j, all leader blocks are decided.
Case: All undecided leader blocks in rounds $j-1$ and $j-2$ will be decided as they now have decided anchors in rounds $j+1$ and j respectively.
Case: For undecided leader blocks in round $i < j - 2$, j is higher than the decision round of the wave that i is in. Thus, by the induction hypothesis, there are no undecided leader blocks between i and j. Hence, the leader block in round i will also be decided.

Theorem 2. BB-Core maintains liveness.

Proof 10. By Lemma 5, all honest authorities synchronize to the same round within Δ after GST. By Lemma 7, within any window of $2f+2$ rounds, there exist two consecutive rounds where honest authorities propose the highest-ranked leader blocks. By Corollary 2, these honest leader blocks will be directly committed.
Furthermore, by Lemma 8, any previously undecided leader blocks will eventually be decided once we have committed blocks from honest authorities in consecutive rounds. This creates a cascading effect where the commitment of new honest blocks triggers the decision of older undecided blocks.
Since we can guarantee that honest leader blocks are committed every $2f+2$ rounds, and each such commitment resolves all pending undecided blocks, the protocol makes continuous progress. No block remains undecided indefinitely, and new blocks are regularly committed, satisfying the liveness property.

Theorem 3. BB-Core implements Byzantine Atomic Broadcast.

Proof 11. **Validity**: If an honest authority broadcasts a block, it will be included in the DAG and processed by the consensus protocol. Leader blocks are eventually decided (either committed or skipped), as shown in Theorem 2. Non-leader blocks are included in the linearization process of committed leader blocks, ensuring all honest authorities eventually deliver the same decision regarding all blocks in the system.
Agreement: By Lemma 4, all honest authorities reach identical decisions for all leader blocks. Since the total ordering is constructed deterministically from these agreed-upon decisions, if any honest authority delivers a block in the final ordering, all other honest authorities will deliver the same block.
Integrity: The protocol's block validation and cryptographic signatures ensure that blocks are only accepted from legitimate authors. The deterministic ordering algorithm processes each decided block exactly once, preventing duplicate delivery in the total ordering.
Total Order: From Theorem 1, all honest authorities construct identical total orderings from the same sequence of committed leader blocks. Since all blocks are delivered according to their position within these identical orderings, all honest authorities deliver blocks in the same sequential order.

5. The BB-Guard protocol

As we discussed in the previous section, BB-Core achieves low latency, but in a setting that assumes fewer corruptions, which introduces the practical risk of a weaker resilience threshold. We now present an approach to further mitigate any safety concerns, by introducing an additional (slow) layer of higher resilience. That is BB-Guard, a protocol that monitors the operation of BB-Core in a synchronous pace, and corrects potential violations via two guarantees; (i) any safety or livenesss violation will be caught and (ii) at least $f + 1$ violating core validators will be provably identified by all participating validators. In practice, BB-Guard will operate in a synchronous, slower pace than BB-Core, while verifying that the main protocol is operating correctly, and recovering liveness and safety in case any of the two fails due to more than f corruptions. We present the main BB-Guard functionality in Algorithm 4 and some helper functions in Algorithm 5.

5.1. Motivation

The BB-Guard protocol fulfils a crucial function to ensure safety and liveness of BlueBottle when the system is under attack. We argue via two constructive examples.

Example 1: Liveness failure. Assume that during the operation of BB-Core, the resilience assumption breaks, and an adversary $\mathcal{A}$ takes control of $f + 1$ validators. In such a case, it is simple to see that our protocol loses liveness indefinitely, even via crash-faults only. The protocol requires $4f + 1$ votes for each round's slots to be determined and for it to move forward on the next round. With fewer than $4f+1$ honest validators participating, the protocol could be skipping rounds or make no progress indefinitely.

Example 2: Safety failure. Again, assume that during the operation of BB-Core, the resilience assumption breaks, and an adversary $\mathcal{A}$ takes control of $3f$ core validators. The adversary is then able to present to some validators a directly committed leader block B_L ($4f+1$ support), whereas for others it will be undecided ($f+1$ support). The next leader will also be presented the latter case, so in its view B_{L+1} will not have enough support to indirectly commit B_L. When B_{L+1} is directly committed, the validators will diverge views (some will already have B_L committed whereas other will skip it) and safety will break.

Both examples showcase the importance of maintaining the resilience assumption for BB-Core to operate correctly. However, in practice, it is possible that an adversary could temporarily break the resilience assumption. In such a case, we would like to ensure that our protocol can recover both safety and liveness, which has not been the case so far. This is what we aim to achieve with BB-Guard.

5.2. BB-Guard setting

Before diving into the protocol details, we recall the overall setting: We assume a set of n validators who in total hold stake S such that each validator maintains stake equivalent to S/n.[2] We assume that there exists a polynomially bounded adversary $\mathcal{A}$ who can corrupt up to a total of $S_f \leq (S-1)/2$ stake in a static fashion, i.e. $\mathcal{A}$ picks which validators to corrupt (up to the corruption threshold) before the protocol starts, and cannot change corruptions after. Furthermore, we separate the validators into two sets, core and guard, such that $\sum_{v_i \in \mathsf{core}} \mathsf{stake}_i \geq 5S/6$. [3]

For this part of our construction, we assume that validators communicate over a synchronous, point-to-point protocol, with a known network delay Δ_{net}. We also denote by Δ, the *network delay for reliable broadcast*. This is the theoretical delay with which we will analyze our guard protocol[4].

This is also the Δ timeout used for the liveness proofs of BB-Core, however, in the practical deployment we optimistically wait a much smaller timeout before blaming a leader. This practice has been introduced by Shoal [17], where the real Δ timeout is only used when many consecutive leaders are skipped. It balances well the practical needs with the theoretical requirements and is what most production systems follow.

2. This model can be easily simulated even in a setting where validators hold unequal units of stake, by having each validator simulate a separate Sybil validator per each unit of stake it holds.

3. As already mentioned earlier, this implies that $\mathcal{A}$ can corrupt at most $t \cdot |\mathsf{core}|$ validators, where $t < 3/5$.

4. In practice, we will set Δ to a sufficiently large time so that all messages are guaranteed to be delivered by then. Since this construction is going to be part of the slow path, we can set Δ to a large number.

5.3. Monitoring failures

Block handling. Upon receiving a block, each guard verifies that the block is valid with respect to BB-Core, by executing the predetermined validity checks defined by the BB-Core protocol. Furthermore, guards check for equivocations with respect to every received block. They do so, by maintaining state of what blocks they have received from which core validators for each round of BB-Core. If a block is equivocated by its authoring validator, this is proof of misbehavior for that party. In the current construction, guards simply do not consider any such blocks at all. However, more sophisticated designs could allow such blocks to contribute to the provable equivocations of the system, so that malicious parties are identified via such equivocations as well.

Blamesets. Blamesets are the main gadget of our construction. A *valid blameset* is a set of at least $f+1$ core validators, that have provably misbehaved. There are two types of blamesets, namely *liveness* and *safety* blamesets, according to the type of misbehavior the validator has been proven to have performed. Each valid blameset is accompanied by a valid proof of misbehavior for each validator in it. Once an honest guard has established a new valid blameset, it can request a recovery process, via which, guards will agree on one blameset, and slash the validators in that blameset.

Liveness failure detection. Guards are also tasked with restoring the core protocol's liveness. To check for potential liveness failures, for each new round, they maintain a timer Δ_{live} and a set asleep that contains inactive core validators and is initialized to core, the set of all core validators. Upon receiving a valid block from some core validator v_i for round r, the guard will remove v_i from $\mathsf{asleep}(r)$, indicating that the validator is active for the round. After the timer expires, any validator remaining in asleep, will be blamed by the guard. After the blaming phase, validators have a chance to respond within a grace period Δ_{grace}. During the grace period, blamed (or not) validators can respond by providing a valid block for the round produced by the the blamed validator, to convince all others that the blamed party is live. After the grace period , any validator who is not convinced, votes against the blamed party. If enough (majority) votes are gathered, validators can use them as proof for the inclusion of the party in their liveness blameset.

Safety failure detection. Honest validators are guaranteed to not equivocate, and to not vote for equivocating blocks. However, if the safety assumption of the BB-Core protocol breaks, i.e., the adversary can corrupt more than f validators, then equivocating blocks can be committed (still, no two equivocating blocks can both be directly committed, as long as the corruptions remain $\leq 3f$). However, as we described before there could be an equivocation through the indirect skip path. In this case, BB-Guard will come in play; every time a guard observes a new set of blocks that are being committed to the DAG, they compare existing already committed, leader blocks to the new ones. In case a guard finds two equivocating blocks, it compares their support, to find the set of overlapping parties supporting the two. This overlapping set, which as we show will be provable and will contain at least $f+1$ misbehaving parties, can then be used for recovery/slashing. The term equivocating blocks here is broad and can refer to any type of equivocation: we focus on BB-Core but this generalized to any consensus protocol where equivocating blocks or votes can exist.

5.4. Recovery

In this construction, we show how guards can catch liveness/safety issues, and in such cases, agree on a blameset of at least $f+1$ Byzantine parties. Honest guards can agree simply by running Byzantine agreement on the recovery set of each guard, which they can do since they maintain honest majority. After that, they can deterministically choose the first (in order) valid blameset, as the set of core validators to be removed.

After honest guards agree on such a blameset, they can provably notify the core validators. They in turn can then disregard the participants proven to be malicious (or in practice, slash their stake, which leads to future research towards cryptoeconomic/incentive-based security of such systems). The core validators can, at that point, execute an honest-majority agreement protocol, since they will have regained majority (by removing $f+1$ malicious out of $5f+1$ total, the new split is $2f+1$ honest out of $4f+1$ total). For this reason, we focus on showing how to regain honest majority in the core validators in cases of misbehaviour, and not on the specifics of how to utilize the honest majority to recover the protocol afterwards.

5.5. BB-Guard Security and Liveness

Lemma 9. Assume there are $n = 5f+1$ core authorities and up to $3f$ of them are malicious. No two honest authorities can directly commit different leader blocks (or directly commit and directly skip) for the same leader and round.

Proof 12. Assume that this can occur. Then, there are $4f+1$ authorities which vote $B_{(l,r)}$ and $4f+1$ authorities which vote (or skip) $B'_{(l,r)}$. Since there are $3f$ malicious authorities, there are still $f+1$ honest votes needed for $B_{(l,r)}$ and another $f+1$ honest votes needed for $B'_{(l,r)}$ (or to skip B'). This means that an honest authority must do both, a contradiction.

Corollary 3. Lemma 1 holds even under $3f$ corruptions.

Proof 13. Same argument as in the respective proof, but with $5f+2$ authorities required, instead of $7f+2$.

Lemma 10. If there exists a block which an honest core authority directly commits while another honest core authority indirectly skips, then there exist at least $f+1$ authorities that have provably equivocated.

Proof 14. Assume that such a block exists and call it B. Each block links to $4f+1$ blocks from the previous round. Assume that a block B' in round $R' > R$ does not link to $2f+1$ support for B from round R (otherwise, from lemma 3, corollary 1 would apply and B would never be indirectly skipped). In that case, focus on $R' = R+1$. B' links to $4f+1$ blocks from round R. Since $4f+1$ blocks in round R support B, and the total of distinct core authorities is $5f+1$, the minimum number of equivocations is $f+1$, and it is provable by their

Algorithm 4 BB-Guard Main Functions

```
procedure OnRound(r)
  Set now = r − 1
  Set timers Δ_live = 4Δ, Δ_leader = 2Δ
  if Δ_leader fires then
    Set now = r
    for l ∈ GetLeaders(r − 1) : l ∈ asleep(r − 1) do
      BC(LBlame, i, l, r − 1)
  if Δ_live fires then
    for core_j ∈ asleep(r) do
      BC(LBlame, i, core_j, r)
    for l ∈ GetLeaders(r − 1) : l ∉ asleep(r − 1) do
      for core_j : ¬IsVote(b_(j,r), b_(l,r−1)) do
        BC(LBlame, i, core_j, r)
  Set timer Δ_grace = 2Δ
  if Δ_grace fires and |LBlamed(r)| ≥ f + 1 then
    Recover(i, LBlamed(r), π_r)
procedure OnBlock(b := B_(u,r))
  if Valid(b) ∧ Equivocates(b) = ⊥ ∧ now ≤ r then
    asleep(r) ← asleep(r) \ c;
    BC((i, b, σ_i(b)))
procedure OnCoreUpdate(S)            ▷ ⊥ ≠ S ← TryDecide(...)
  (Set, π) ← CheckEquivocation(S)
  if Set ≠ ⊥ then
    Recover((i, Set, π))
  else
    for b ∈ S do
      BC((i, b, σ_i(b)))
procedure OnLBlame((j, core_k, r))   ▷ blames(k, r) initialized to ∅
  blames(k, r) ← blames(k, r) ∪ {j}
  if Maj(blames(k, r)) then
    LBlamed(r) ← LBlamed(r) ∪ {k}, π_r ← π_r || blames(k, r)
procedure OnRecover((j, S_B, π))
  if isValidBlameSet(S_B, π) and S_i = ⊥ then
    S_i ← S_B
    Recover(i, S_i, π)
```

Algorithm 5 BB-Guard Helper Functions

```
validators                                   ▷ The set of validators

procedure GetLeaders(r)
  Let l = leadersPerRound, V = |validators|
  return {validators[(r + d) mod V] : d = 0, ..., l − 1}
procedure ResolveEquivocation(b, b')
  Set ← {v ∈ validators : supporting b and b'}
  π ← {respective blocks showing support from Set}
  return (Set, π)
procedure CheckEquivocation(S)
  if ∃b ∈ S : B ← Equivocates(b, LocState()) ∧ B ≠ ⊥ then
    (Set, π) ← ResolveEquivocation(B, b)
    return (Set, π)
  return ⊥
procedure Recover(j, BS, π)
  BC(i, S, σ_i(S, recover))
  Set RCVec_i = ⟨S_j⟩_{j∈[n]}, where S_j is:

    { S_j,  if received unique: (j, S_j, σ_j(S_j, recover)) within Δ
    { 1,    else.

  BA(j, S_j) → SR_j, for all j ∈ [n]
  Order({SR_j}_{j∈[n]}) → O
  return the first valid SR_j ∈ O
```

difference in votes.

Lemma 11. If there exists a block that an honest core authority directly skips while another honest core authority indirectly commits, then there exist at least $f+1$ authorities that have provably equivocated.

Proof 15. Assume that such a block exists and call it B. Thus, there are $4f+1$ authorities which blame B and $2f+1$ authorities which support B. This requires a total of $6f+2$ votes, out of which only $5f+1$ can be distinct. As a results $6f+2-(5f+1)=f+1$ need to both support and blame B which constitutes a proof of equivocation.

Lemma 12. If there exist two honest core authorities who have decided on a leader block and do not agree on the decision, then there exist at least $f+1$ core authorities that have provably equivocated.

Proof 16. Let $B_X \neq B_Y$ be the committed (or skipped) leader blocks according to authorities X and Y respectively for leader l in round r. By Lemma 9, no two honest authorities could have both directly committed conflicting blocks. Similarly, no two honest authorities could have directly committed and directly skipped a block respectively(Corollary 3). The only viable cases are the following:

Case: X directly committed B_X, while Y directly skipped B_Y (and vice versa): From Lemma 10 (and Lemma 11 respectively), there will exist a set of at least $f+1$ provably misbehaving parties.

Case: X directly committed B_X, while Y indirectly committed B_Y (and vice versa): Similar to previous arguments, this requires at least $(4f+1)+(2f+1)$ distinct votes, i.e., at least $f+1$ provable equivocations.

Case: X indirectly committed B_X, while Y indirectly committed B_Y: Let A^X and A^Y be the directly-committed anchors used by X and Y to indirectly commit B_X and B_Y, respectively. Since no honest party could have voted for both anchors, there are not enough votes for both anchors to be directly committed (total votes needed $8f+2$, but total votes on the protocol are $2f+1+6f$).

Case: X indirectly committed B_X, while Y indirectly skipped B_Y: In that case, let A denote the directly-committed anchor used to indirectly commit B_X, and A' denote the directly-committed anchor used to indirectly skip B_Y. Again, since no honest party could have voted for both anchors, there are not enough votes for both anchors to be directly committed.

Lemma 13. No honest core validators will ever be included in a valid safety-blameset.

Proof 17. Honest validators receive blocks and vote based on whether the blocks preserve the rules of the protocol. The validators' votes cannot be equivocated, and honest validators will never vote for two conflicting blocks. As such, honest validators can not be included in any valid blameset.

Lemma 14. No honest core validators will ever be included in a valid liveness-blameset.

Proof 18. Assume that an honest core validator v is correctly included in a liveness-blame for some round r, i.e., at least S_f+1 guards have attested that they blamed v for round r. At least one of the guards is honest, say g. Then, according to Algorithm 4, either i) v did not vote for a leader that g considers live, or ii) v did not vote on time, or iii) v was a leader and did not send on time. For the first case, g considers leaders on time, only if it receives a valid leader block for the previous round within 2Δ after the round has started. Even if the previous round progressed instantly, within Δ, v must have also progressed and received the leader's block. If v then voted for the leader, within another Δ, v's block voting for the leader would arrive to g, on time (within the 4Δ duration of Δ^g_{live}. As such, v must have seen a leader block on time, and still not voted for it, which contradicts the honest behavior.

Similarly for the second case, g considers on time any valid vote that arrives within 4Δ after g enters the round. Since v will enter the round by at most a Δ delay after g, if v were to follow the protocol, it would wait by at most another 2Δ for the leader's block and then submit its vote. Since it takes at most an additional Δ for v's vote to arrive at g, v must have entered the round on time, and still not voted for it on time, which contradicts the honest behavior.

Finally, if v was a leader for a round, it enters the round within at most Δ after the first honest party. An honest leader immediately proposes a block on entering a round for which it is the leader. Since g would wait for 2Δ for the leader's block after entering the round, even if v entered the round Δ time after g, it would still be on time to send its block to g. Thus, v must have entered the round on time, but not have sent its block on time, which also contradicts the honest behavior.

Corollary 4. If the number of corruptions is $\leq f$, then no Recover call occurs.

Proof 19. Implied by Lemmas 13, 14, since there are not enough misbehaving parties to construct blamesets.

Lemma 15 (Safety Violations). Any safety violation (equivocation) on BB-Core will be identified, and restored within $2\Delta+\Delta_{BA}$ of the moment the first honest party observes it.

Proof 20. Assume that an honest party (validator or guard) observes two equivocating blocks (according to its view) at time t. Due to synchrony, after Δ, all honest parties will be observing a superset of the validator's view, and can also see the equivocation. Such equivocations can always construct an $f+1$ blameset (per Lemma 12) and as such by time $t+2\Delta$, all honest guards will be running BA, some with input the blameset of the aforementioned equivocation. Within another Δ_{BA}, all guards will agree on a blameset of at least $f+1$ validators, (since there exists at least one such valid blameset), and will recover the safety of the protocol.

Lemma 16 (Liveness). For every round r, BB-Core progresses from r within 6Δ of entering r, or at least $f+1$ parties are blamed in BB-Guard.

Proof 21. First, parties *enter* round r of BB-Core, once they receive $4f+1$ blocks (from distinct senders) from round $r-1$. Every guard will also enter rounds according to this logic. From synchrony and broadcast, it is guaranteed that once a guard enters round r, every participant will enter round r at most within Δ as well (similar to Lemma 5). Each guard sets a timer $\Delta_{\text{live}} = 4\Delta$, by which it expects to receive blocks from at least $4f+1$ core validators; this is because in the worst-case, it would take at most an additional Δ for the slowest validator to enter the round, it would wait for the leaders of the past round for 2Δ and would take an additional Δ for its block to return to the guard. Each guard expects to have received blocks from the leaders of the past round already before the current round starts, or within 2Δ of the current round start; this is because in the worst-case, it would take an additional Δ for the slowest leader to enter the past round, and it would take another Δ for its block to arrive at the guard.

Say g^* is the first honest guard that enters round r, and let $\mathsf{asleep}_{g^*}(r)$ denote the set once $\Delta_{\text{live}}^{g^*} = 4\Delta$ expires. Within another Δ after $\Delta_{\text{live}}^{g^*}$ (5Δ total), every other guard's (say g) timer will also expire and they will be blaming each of the asleep core validators $v \in \mathsf{asleep}_g(r)$. If $|\mathsf{asleep}_g(r)| \leq f$ for any $g \in H$, with H denoting the set of all honest guards, then it is guaranteed that all guards can progress the round after another Δ (6Δ total). The same holds if $|\cap_{g\in H}\mathsf{asleep}_g(r)| \leq f$, since locally, every guard by that time will have blocks from all $v \notin \cap_{g\in H}\mathsf{asleep}_g(r)$, which are sufficient to progress the BB-Core round. Otherwise, by 6Δ total, all guards will have at least $|H| = S_f + 1$ blames for all $v \in \cap_{g\in H}\mathsf{asleep}_g(r)$, where $|\cap_{g\in H}\mathsf{asleep}_g(r)| \geq f+1$.

6. Evaluation

We implement a BB-Core validator[5] in Rust by forking the Mysticeti codebase [18]. Section A provides more details about our implementation and testing methodology. We evaluate the throughput and latency of BB-Core through experiments conducted on Amazon Web Services (AWS) on a geo-distributed testbed. Section B describes the experimental setup in detail.

We focus our evaluation on comparing BB-Core with Mysticeti [5], one of the state-of-the-art BFT consensus protocols in terms of both latency and throughput. The scope of this evaluation is to show that BB-Core can achieve lower latency than Mysticeti without sacrificing throughput, but at the cost of a lower fault tolerance of up to 20% Byzantine faults instead of 33%. Unfortunately, there is no $n = 5f+1$ BFT consensus protocol that has a publicly available implementation [19, 20, 21, 22]. We leave to future work a further evaluation, when such artifacts become available.

Our evaluation demonstrates the following claims:

- **C1** BB-Core has similar throughput and lower latency than Mysticeti when operating in fault-free and synchronous network conditions.
- **C2** BB-Core scales as well as Mysticeti by maintaining high throughput and low latency as the number of validators increases.
- **C3** BB-Core has a similar throughput to and lower latency than Mysticeti when operating in the presence of (benign) crash faults.

Note that evaluating the performance of BFT protocols in the presence of Byzantine faults is an open research question [23], and state-of-the-art evidence relies on formal proofs (presented in Section 4).

6.1. Benchmark under ideal conditions

We evaluate the performance of BB-Core under normal, failure-free conditions in a wide-area network (WAN). Figure 3 (Left) reports results from a geo-replicated deployment with a small committee of 10 validators and a larger committee of 50 validators. For cost reasons, we cap the input load at 100,000 tx/s (10 validators) and 300,000 tx/s (50 validators). These rates are two orders of magnitude above the observed peak throughput of any existing blockchain and we believe sufficient to assess system behavior under load.

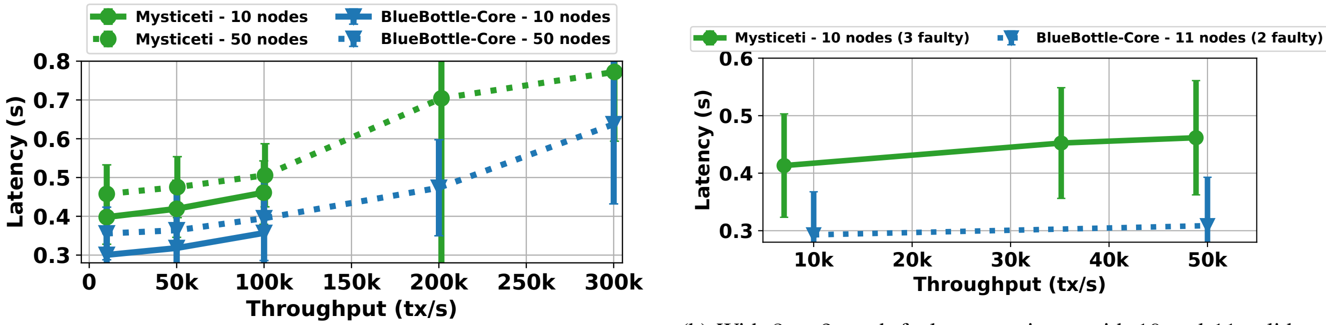
The figure shows that BB-Core successfully trades some fault tolerance for lower latency. BB-Core lowers the commit path to two message delays. Each round of BB-Core must, however, wait for a larger parent quorum (80% of validators) than Mysticeti (67%). Despite this larger quorum, BB-Core reduces end-to-end latency by about 20–25% in the WAN setting across all input loads, confirming claim **C1**. Concretely, with 10 validators at 100,000 tx/s, BB-Core attains about 357 ms latency versus about 461 ms for Mysticeti; with 50 validators at 100,000 tx/s, BB-Core reaches about 395 ms versus about 505 ms, and at 300,000 tx/s, BB-Core reaches about 637 ms versus about 772 ms. We observe a consistent proportional latency reduction at both committee sizes, confirming claim **C2**.

6.2. Benchmark under faults

Figure 3 (Right) shows both protocols under 3 crash faults in committees of 10 and 11 validators (the minimum sizes used for this fault level in our experiments). We cap the offered load at 50,000 tx/s for cost reasons.

As expected, both systems sustain the input load with no material latency inflation relative to the fault-free runs. Mysticeti's latency lies between 400 and 500 ms, whereas BB-Core remains between 300 and 350 ms, preserving a 20–25% latency advantage (again at the cost of lowering fault tolerance) and confirming claim **C3**. Both systems handle benign crashes gracefully by rapidly skipping faulty leaders via the direct skip rule described in Section 3.

5. https://github.com/phvv/mysticeti/tree/odontoceti (commit `e02aeba`)



(a) Committees with 10 and 50 validators, no validator faults.

(b) With 3 or 2 crash faults; committees with 10 and 11 validators (minimum to tolerate 3 or 2 faults).

Figure 3: WAN throughput-latency performance comparison of BB-Core ($n = 5f + 1$) and Mysticeti ($n = 3f + 1$) with a 512B transaction size. The y-axis starts at 300 ms to zoom in on the latency difference between the systems.

7. Making the Core Asynchronous

We study BB-Core under partial synchrony and also introduce an asynchronous variant. The asynchronous algorithm mirrors the partially synchronous one, except for the orange-highlighted additions; Appendix C provides formal correctness proofs. Composing BlueBottle with its asynchronous counterpart introduces a subtle effect: BB-Guard must detect liveness faults via a different mechanism.

In the partially synchronous design, liveness monitoring first checks whether the round leader is faulty and then verifies that the received blocks correctly reject a faulty leader. This procedure uses a timer of 3Δ, where Δ denotes the network delay bound. The asynchronous variant removes leader timeouts, so liveness detection completes in 2Δ.

This yields two implications. The first one is straightforward: the partially synchronous protocol trades one round of optimistic-case latency for faster liveness recovery in the asynchronous protocol. The second one is more significant: the asynchronous variant removes the dependence on the network delay Δ between BB-Core and BB-Guard. This decoupling lets operators choose a conservative Δ (even tens of seconds) without harming BB-Core's performance under crash faults. Operators can adjust this value at every epoch. Moreover, a faulty-leader detection and exclusion mechanism, such as HammerHead [24], further mitigates the impact of faulty leaders.

8. Related Work

Byzantine fault-tolerant state machine replication (BFT-SMR) is a foundational abstraction in distributed systems, and its latency limits have been studied extensively [20, 25, 26, 27, 28, 29]. Classical work emphasizes worst-case latency, with state-of-the-art protocols committing in three communication steps while tolerating up to $f < n/3$ Byzantine faults [27, 30, 31].

Motivated by practical deployments (e.g., blockchains), recent work increasingly targets good-case latency under partial synchrony, i.e., the latency to commit when the designated leader is correct and the network satisfies partial synchrony assumptions [32, 33]. In this setting, protocols aspire to two-round commitment. SBFT [28] achieves two rounds in the absence of Byzantine faults and reverts to a slower path otherwise. FAB [20] removes this restriction but was later shown to suffer from a liveness issue [34]. Kudzu [21] commits in three rounds in general and in two rounds when the number of Byzantine faults is small ($f < n/5$). Hydrangea [22] adopts a generalized fault model that distinguishes Byzantine from crash faults; while this model is weaker than a purely Byzantine model for a fixed total fault budget, it requires careful parameterization for practical deployment. Alpenglow [19] claims two-round commitment while simultaneously tolerating less than 20% Byzantine and less than 20% crash faults; however, this guarantee this is impossible according to [22] and after careful analysis it is actually a 3δ protocol as Rotor requires 2δ and Votor an extra one. Optimistic fast paths have also been explored in synchronous and asynchronous settings [35, 36, 37, 38, 39, 40, 41].

For comparison, BlueBottle commits in two rounds under partial synchrony while tolerating up to f Byzantine faults with $n = 5f + 1$ replicas. Two-round finality with $n = 5f - 1$ replicas is achievable when the protocol explicitly identifies and ignores faulty leaders [33, 42]. BlueBottle could be extended with such leader-exclusion mechanisms to attain the $5f - 1$ bound; however, we deliberately avoid this design point because the incremental gain in fault tolerance is modest relative to the additional protocol design and engineering complexity it introduces.

Accountable safety [43, 44, 45] and accountable liveness [46, 47] strengthen classical safety and liveness via crypto-economic mechanisms. In addition to preserving agreement (safety) and eventual decision (liveness) below a fault threshold, these notions enable attribution of provable misbehavior—e.g., through slashing conditions—when safety or liveness is violated, thereby deterring equivocation or censorship in permissioned and permissionless settings.

References

[1] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," 2014.

[2] K. Sekniqi, D. Laine, S. Buttolph, and E. G. Sirer, "Avalanche Platform," 2020.

[3] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, "Scalable and Probabilistic Leaderless BFT Consensus through Metastability," 2020.

[4] A. Yakovenko, "Solana: A New Architecture for a High Performance Blockchain," 2017.

[5] K. Babel, A. Chursin, G. Danezis, L. Kokoris-Kogias, and A. Sonnino, "Mysticeti: Low-Latency DAG Consensus with Fast Commit Path," 2024.

[6] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[7] A. Kuçi, C. Cachin, and G. A. Marson, "Proof-of-stake blockchain: Ouroboros," 2021.

[8] The Sui team, "The sui blockchain," http://sui.io, 2023.

[9] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[10] D. Malkhi, K. Nayak, and L. Ren, "Flexible byzantine fault tolerance," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 1041–1053.

[11] J. Neu, S. Sridhar, L. Yang, and D. Tse, "Optimal flexible consensus and its application to ethereum," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3885–3903.

[12] I. Keidar, O. Naor, O. Poupko, and E. Shapiro, "Cordial Miners: Fast and Efficient Consensus for Every Eventuality," in *37th International Symposium on Distributed Computing (DISC 2023)*, 2022.

[13] P. Jovanovic, L. K. Kogias, B. Kumara, A. Sonnino, P. Tennage, and I. Zablotchi, "Mahi-Mahi: Low-Latency Asynchronous BFT DAG-Based Consensus," in *2025 IEEE 45th International Conference on Distributed Computing Systems*, ser. ICDS '25, 2025.

[14] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All You Need is DAG," in *PODC'21: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, 2021.

[15] B. Arun, Z. Li, F. Suri-Payer, S. Das, and A. Spiegelman, "Shoal++: High throughput dag bft can be fast!" *arXiv preprint arXiv:2405.20488*, 2024.

[16] N. Shrestha, R. Shrothrium, A. Kate, and K. Nayak, "Sailfish: Towards improving latency of dag-based bft," *Cryptology ePrint Archive*, 2024.

[17] A. Spiegelman, B. Arun, R. Gelashvili, and Z. Li, "Shoal: Improving dag-bft latency and robustness," *arXiv preprint arXiv:2306.03058*, 2023.

[18] M. Labs, "Mysticeti: Low-latency dag consensus with fast commit path," https://github.com/asonnino/mysticeti, 2024.

[19] Q. Kniep, J. Sliwinski, and R. Wattenhofer, "Alpenglow," https://www.anza.xyz/alpenglow-1-1, 2025.

[20] J.-P. Martin and L. Alvisi, "Fast byzantine consensus (fab paxos)," in *Proceedings of the International Conference on Dependable Systems and Networks*, ser. DSN '05, 2005.

[21] V. Shoup, J. Sliwinski, and Y. Vonlanthen, "Kudzu: Fast and Simple High-Throughput BFT," 2025.

[22] N. Shrestha, A. Kate, and K. Nayak, "Hydrangea: Optimistic two-round partial synchrony," Cryptology ePrint Archive, Paper 2025/1112, 2025. [Online]. Available: https://eprint.iacr.org/2025/1112

[23] S. Bano, A. Sonnino, A. Chursin, D. Perelman, Z. Li, A. Ching, and D. Malkhi, "Twins: Bft systems made robust," *arXiv preprint arXiv:2004.10617*, 2020.

[24] G. Tsimos, A. Kichidis, A. Sonnino, and L. Kokoris-Kogias, "Hammerhead: Leader reputation for dynamic scheduling," in *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2024, pp. 1377–1387.

[25] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync hotstuff: Simple and practical synchronous state machine replication," in *2020 IEEE Symposium on Security and Privacy*, ser. SP '20, 2020.

[26] I. Abraham, K. Nayak, L. Ren, and Z. Xiang, "Byzantine Agreement, Broadcast and State Machine Replication with Near-optimal Good-case Latency," arXiv cs.CR 2003.13155, 2020.

[27] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, M. I. Seltzer and P. J. Leach, Eds. USENIX Association, 1999, pp. 173–186.

[28] G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "SBFT: A Scalable and Decentralized Trust Infrastructure," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019.

[29] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," in *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07. Association for Computing Machinery, 2007.

[30] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on BFT consensus," arXiv cs.DC 1807.04938, 2019.

[31] B. Y. Chan and R. Pass, "Simplex Consensus: A Simple and Fast Consensus Protocol," in *Theory of Cryptography: 21st International Conference*, ser. TCC 2023. Springer-Verlag, 2023.

[32] I. Abraham, K. Nayak, L. Ren, and Z. Xiang, "Good-case Latency of Byzantine Broadcast: A Complete Categorization," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, ser. PODC'21, 2021.

[33] P. Kuznetsov, A. Tonkikh, and Y. X. Zhang, "Revisiting Optimal Resilience of Fast Byzantine Consensus," in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, ser. PODC'21, 2021.

[34] I. Abraham, G. Gueta, D. Malkhi, and J.-P. Martin, "Revisiting Fast Practical Byzantine Fault Tolerance: Thelma, Velma, and Zelma," arXiv CS.DC 1801.10022, 2018.

[35] P.-L. Aublin, R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The next 700 bft protocols," *ACM Trans. Comput. Syst.*, vol. 32, no. 4, 2015.

[36] F. V. Brasileiro, F. Greve, A. Mostéfaoui, and M. Raynal, "Consensus in one communication step," in *Proceedings of the 6th International Conference on Parallel Computing Technologies*, ser. PaCT '01. Springer-Verlag, 2001.

[37] R. Friedman, A. Mostefaoui, and M. Raynal, " Simple and Efficient Oracle-Based Consensus Protocols for Asynchronous Byzantine Systems ," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 1, pp. 46–56, 2005.

[38] K. Kursawe, "Optimistic byzantine agreement," in *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, ser. SRDS '02. IEEE Computer Society, 2002.

[39] R. Pass and E. Shi, "Thunderella: Blockchains with optimistic instant confirmation," in *Advances in Cryptology – EUROCRYPT 2018*. Springer International Publishing, 2018.

[40] N. Shrestha, I. Abraham, L. Ren, and K. Nayak, "On the optimality of optimistic responsiveness," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. Association for Computing Machinery, 2020.

[41] N. Shrestha, Q. Yu, A. Kate, G. Losa, K. Nayak, and X. Wang, "Optimistic, Signature-Free Reliable Broadcast and Its Applications," 2025.

[42] I. Abraham and L. Ren, "2-round bft in simplex style for $n = 5f - 1$," https://decentralizedthoughts.github.io/2025-08-06-5fminus1-simplex/, Aug. 2025, decentralized Thoughts. Accessed: 2025-11-13. [Online]. Available: https://decentralizedthoughts.github.io/2025-08-06-5fminus1-simplex/

[43] V. Buterin and V. Griffith, "Casper the Friendly Finality Gadget," arXiv cs.CR 1710.09437, 2019.

[44] J. Neu, E. N. Tas, and D. Tse, "The Availability-Accountability Dilemma and Its Resolution via Accountability Gadgets," in *Financial Cryptography and Data Security: 26th International Conference*, ser. FC '22, 2022.

[45] P. Sheng, G. Wang, K. Nayak, S. Kannan, and P. Viswanath, "BFT Protocol Forensics," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. Association for Computing Machinery, 2021.

[46] A. Lewis-Pye, J. Neu, T. Roughgarden, and L. Zanolini, "Accountable Liveness," in *Proceedings of the 2025 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '25, 2025.

[47] E. N. Tas, D. Tse, F. Gai, S. Kannan, M. A. Maddah-Ali, and F. Yu, "Bitcoin-Enhanced Proof-of-Stake Security: Possibilities and Impossibilities," in *2023 IEEE Symposium on Security and Privacy*, ser. SP '23, 2023.

[48] T. T. Team, "Tokio," https://tokio.rs, 2024.

[49] H. de Valence, "Ed25519 for consensus-critical contexts," https://crates.io/crates/ed25519-consensus, 2024.

[50] RustCrypto, "Rustcrypto: Hashes," https://github.com/RustCrypto/hashes, 2024.

[51] Die.Net, "writev(3) - linux man page," https://linux.die.net/man/3/writev, 2024.

[52] Meta, "Sapling (minibytes)," https://github.com/facebook/sapling/tree/main/eden/scm/lib/minibytes, 2024.

[53] T. S. Team, "Validator deployment amd configuration," https://docs.sui.io/guides/operator/validator/validator-config, 2025.

[54] E. Blum, J. Katz, C.-D. Liu-Zhang, and J. Loss, "Asynchronous byzantine agreement with subquadratic communication," in *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part I 18*. Springer, 2020, pp. 353–380.

[55] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography," in *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, 2000, pp. 123–132.

[56] J. Loss and T. Moran, "Combining asynchronous and synchronous byzantine agreement: The best of both worlds," *Cryptology ePrint Archive*, 2018.

[57] R. Bacho and J. Loss, "On the Adaptive Security of the Threshold BLS Signature Scheme," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022.

[58] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *International conference on the theory and application of cryptology and information security*. Springer, 2001, pp. 514–532.

[59] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, and G. Stern, "Bingo: Adaptivity and Asynchrony in Verifiable Secret Sharing and Distributed Key Generation," in *Advances in Cryptology – CRYPTO 2023*, 2023.

[60] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu, "Reaching Consensus for Asynchronous Distributed Key Generation," *Distributed Computing*, vol. 36, 2023.

Appendix A. Implementation

We implement a networked, multi-core BB-Core validator in Rust by forking the Mysticeti codebase [18]. Our implementation leverages `tokio` [48] for asynchronous networking, utilizing raw TCP sockets for communication without relying on any RPC frameworks. For cryptographic operations, we rely on `ed25519-consensus` [49] for asymmetric cryptography and `blake2` [50] for cryptographic hashing. To ensure data persistence and crash recovery, we employed the Write-Ahead Log (WAL). Our WAL optimizes I/O operations through vectored writes [51] and efficient memory-mapped file usage with the `minibytes` [52] crate, minimizing data copying and serialization.

In addition to regular unit tests, we inherited and utilized two supplementary testing utilities from the Mysticeti codebase. First, a simulation layer replicates the functionality of the `tokio` runtime and TCP networking. This simulated network accurately simulates real-world WAN latencies, while the `tokio` runtime simulator employs a discrete event simulation approach to mimic the passage of time. Second, a command-line utility (called *orchestrator*) which deploys real-world clusters of BB-Core on machines distributed across the globe.

We are open-sourcing our BB-Core implementation, along with its simulator and orchestration tools, to ensure reproducibility of our results[6].

Appendix B. Experimental Setup

This section complements Section 6 by detailing the experimental setup used to evaluate BB-Core.

We deploy BB-Core on AWS, using `m5d.8xlarge` instances across 13 different AWS regions: Northern Virginia (us-east-1), Oregon (us-west-2), Canada (ca-central-1), Frankfurt (eu-central-1), Ireland (eu-west-1), London (eu-west-2), Paris (eu-west-3), Stockholm (eu-north-1), Mumbai (ap-south-1), Singapore (ap-southeast-1), Sydney (ap-southeast-2), Tokyo (ap-northeast-1), and Seoul (ap-northeast-2). Validators are distributed across those regions as equally as possible. Each machine provides 10 Gbps of bandwidth, 32 virtual CPUs (16 physical cores) on a 3.1 GHz Intel Xeon Skylake 8175M, 128 GB memory, and runs Linux Ubuntu server 24.04. We select these machines because they provide decent performance, are in the price range of "commodity servers", and match the minimal specifications of modern quorum-based blockchains [53].

In Section 6, *latency* refers to the time elapsed from the moment a client submits a transaction to when it is committed by the validators, and *throughput* refers to the number of transactions committed per second. We instantiate several geo-distributed benchmark clients within each validator submitting transactions in an open loop model, at a fixed

6. https://github.com/phvv/mysticeti/tree/odontoceti (commit `e02aeba`)

rate. We experimentally increase the load of transactions sent to the systems, and record the throughput and latency of commits. As a result, all plots Section 6 illustrate the steady-state latency of all systems under low load, as well as the maximal throughput they can provide after which latency grows quickly. Transactions in the benchmarks are arbitrary and contain 512 bytes. We configure both BB-Core and Mysticeti with 2 leaders per round, and a leader timeout of 1 second.

Appendix C. Asynchronous BB-Core

We present BB-Core-Async, the asynchronous variant of BB-Core. It builds upon the same core ideas as BB-Core, but operates in a fully asynchronous network model and leverages a threshold common coin to achieve liveness. BB-Core-Async is the first $5f + 1$-validator BFT consensus protocol to achieve both safety and liveness in a fully asynchronous network. It achieves low latency through its shortened commit path. BB-Core-Async algorithm is Algorithm 1, Algorithm 2, and Algorithm 3 when we also include the orange-colored lines.

C.1. Additional Assumptions

First, the communication network is asynchronous and messages can be delayed arbitrarily, but messages among honest validators are eventually delivered.

Additionally, we employ a global perfect coin to introduce randomization, similar to previous work [54, 55, 14, 56]. This coin can be constructed using an adaptively secure threshold signature scheme [57, 58], with the distributed key setup performed under asynchronous conditions [59, 60].

Each validator v_k broadcasts messages by invoking $\textsc{Bcast}km, q$, where m is the message and $q \in \mathbb{N}$ is a sequence number. Every validator v_i has an output $\mathsf{Deliver}_i(m, q, v_k)$, where m is the message, q is the sequence number, and v_k is the identity of the validator that initiated the corresponding $\textsc{Bcast}km, q$. BlueBottle implements a BAB protocol and guarantees the following [14]:

- **Validity:** If an honest participant v_k calls $\textsc{Bcast}km, q$, then every honest participant v_i eventually outputs $\mathsf{Deliver}_i(m, q, v_k)$, with probability 1.
- **Agreement:** If an honest participant v_i outputs $\mathsf{Deliver}_i(m, q, v_k)$, then every honest participant v_j eventually outputs $\mathsf{Deliver}_j(m, q, v_k)$ with probability 1.
- **Integrity:** For each sequence number $q \in \mathbb{N}$ and participant v_k, an honest participant v_i outputs $\mathsf{Deliver}_i(m, q, v_k)$ at most once, regardless of m.
- **Total Order:** If an honest participant v_i outputs $\mathsf{Deliver}_i(m, q, v_k)$ and $\mathsf{Deliver}_i(m', q', v'_k)$ where $q < q'$, all honest participants output $\mathsf{Deliver}_j(m, q, v_k)$ before $\mathsf{Deliver}_j(m', q', v'_k)$.

We only provide the lemmas with only a few of them proven due to lack of space. We plan to release a full version online.

C.2. Safety Lemmas

We start by proving the Total Order and Integrity properties of BAB. A crucial intermediate result towards these properties is that all honest validators have consistent commit sequences, i.e., the committed sequence of one honest validator is a prefix of another's, or vice-versa. This is shown in Lemmas Lemma 21 and Lemma 22, which the following lemmas and observations build up to.

Definition 1 (Weak Certificate). A *weak certificate* for a block b in round r is a set of $2f + 1$ distinct *valid* round-$(r+2)$ blocks, each from a *distinct validator*, that support b.

Definition 2 (Strong Certificate). A *strong certificate* for a block b in round r is a set of $4f + 1$ distinct *valid* round-$(r+2)$ blocks, each from a *distinct validator*, that support b.

Lemma 17 (Strong-to-Weak Propagation). Let b be a round-r block with a *strong certificate* S (i.e., $|S| = 4f + 1$ distinct round-$(r + 2)$ blocks supporting b, Definition 2). Then every valid block at any future round $r' \geq r + 3$ has paths to at least $2f + 1$ blocks from S; equivalently, it indirectly references a *weak certificate* for b (Definition 1).

Proof 22. We prove the claim by induction on r', starting at $r' = r + 3$.
Base case ($r' = r + 3$): Let x be a valid round-$(r + 3)$ block. By construction, x references $4f + 1$ distinct valid round-$(r + 2)$ blocks from distinct validators; thus at most f of these can be Byzantine, so x includes at least $3f+1$ round-$(r+2)$ blocks from honest validators. Likewise, the strong certificate S contains at least $3f+1$ blocks from honest validators (since at most f of its $4f+1$ members can be Byzantine). Restricting attention to honest validator identities (a universe of size $4f+1$), these two sets intersect in at least $(3f+1)+(3f+1)-(4f+1) = 2f+1$ blocks. Hence x directly references at least $2f+1$ members of S, so x has paths to a weak certificate for b.
Induction step: Assume that every valid round-r' block, for some $r' \geq r + 3$, has paths to at least $2f + 1$ members of S. Consider any valid round-$(r' + 1)$ block y. By construction, y references $4f + 1$ distinct round-r' blocks. By the induction hypothesis, each of those round-r' blocks has paths to at least $2f+1$ members of S. Therefore, y references at least one round-r' block that has paths to at least $2f + 1$ members of S; by transitivity of paths, y itself has paths to at least $2f+1$ members of S, i.e., to a weak certificate for b.

Observation 1. A block cannot support for more than one block proposal from a given validator, in a given round.

Proof 23. This is by construction. Honest validators interpret support in the DAG through deterministic depth-first traversal. So even if a block b in the vote round has paths to multiple leader round blocks from the same validator

v (i.e., equivocating blocks), all honest validators will interpret b to vote for only one of v's blocks (the first block to appear in the depth-first traversal starting from b).

Lemma 18 (Strong certificate exclusivity). Fix a validator v and round r. If some round-r block b of v has a strong certificate, then no other round-r block $b' \neq b$ of v can have even a weak certificate. In particular, at most one round-r block of v can have a strong certificate.

Proof 24. Suppose, for contradiction, that b has a strong certificate S (so $|S| = 4f + 1$ decision-round blocks at round $r+2$ support b), and some other block $b' \neq b$ from the same validator and round has a weak certificate W (so $|W| = 2f + 1$ vote-round blocks at round $r + 2$ support b'). Both S and W are over the universe of $5f + 1$ validator identities and contain one valid block per identity.
Let $\mathrm{Id}(S)$ and $\mathrm{Id}(W)$ be the sets of validator identities appearing in the certificates S and W (one valid round-$(r + 2)$ block per identity). Then $|\mathrm{Id}(S)| = 4f + 1$ and $|\mathrm{Id}(W)| = 2f + 1$. By quorum intersection over identities,

$$|\mathrm{Id}(S)\cap\mathrm{Id}(W)| \geq (4f+1)+(2f+1)-(5f+1) = f+1 > 0.$$

Since at most f validators are Byzantine overall, this intersection contains at least one honest identity. Pick such an honest validator $u \in \mathrm{Id}(S)\cap\mathrm{Id}(W)$, and let x be u's unique valid decision-round (round $r + 2$) block (by the uniqueness-per-round assumption). Because certificates record one valid block per identity, the same block x is the representative of u in both S and W, hence x supports both b and b'. This contradicts Observation 1. Therefore, b' cannot have a weak certificate. Taking W to be a strong certificate S' gives the special case that two distinct blocks of v in the same round cannot both have strong certificates.

Observation 2. If an honest validator v directly or indirectly commits a block b, then v's local DAG contains a weak certificate for b.

Proof 25. This follows immediately from our direct and indirect commit rules.

Observation 3. Honest validators agree on the sequence of leader slots.

Proof 26. This follows immediately from the properties of the common coin, see Section C.1.

Observation 4 (Unique valid block per (validator, round)). For any validator w and round r, at most one block signed by w is counted as valid in round r. Since commit rules apply only to valid blocks, any commitment for a slot with leader w in round r must be that unique valid block.

Proof 27. Immediate from the DAG construction and the fact that commit rules consider only valid blocks.

Lemma 19. If an honest validator v commits some block b in a slot s, then no other honest validator decides to directly skip the slot s.

Proof 28. Assume by contradiction that some honest validator v' decides to directly skip s. By the direct-skip rule, this means that in v''s local DAG there exists a set N of $4f + 1$ distinct valid round-$(r + 2)$ blocks (one per validator identity) whose support does not go to b (a direct-skip witness for s).
Since v commits b at s, by Observation 2 there exists a weak certificate W for b at s in v's local DAG: a set of $2f + 1$ distinct valid round-$(r + 2)$ blocks (one per validator identity) that support b (Definition 1). Consider the identity sets $\mathrm{Id}(N)$ and $\mathrm{Id}(W)$. We have $|\mathrm{Id}(N)| = 4f + 1$ and $|\mathrm{Id}(W)| = 2f + 1$, hence by quorum intersection over the $5f + 1$ validator identities,

$$|\mathrm{Id}(N)\cap\mathrm{Id}(W)| \geq (4f+1)+(2f+1)-(5f+1) = f+1.$$

As at most f validators are Byzantine, the intersection contains at least one honest identity. Pick such an honest validator $u \in \mathrm{Id}(N) \cap \mathrm{Id}(W)$, and let x be u's unique valid round-$(r + 2)$ block.
Because certificates/witnesses record one valid block per identity, the same block x is the representative of u in both N and W. By definition of W, x supports b; by definition of N, the very same x is counted as not supporting b at v'. By Observation 1 (uniqueness of support per validator/round under the deterministic rule), x cannot both support and not support b. Hence v' cannot have a valid direct-skip witness N, a contradiction.

Lemma 20. If an honest validator directly commits some block in a slot s, then no other honest validator decides to skip the slot s.

Proof 29. Assume by contradiction that an honest validator v directly commits block b in slot s while another honest validator v' decides to skip s. By Lemma 19, v' cannot directly skip s; therefore v' must attempt to skip s via the indirect decision rule. Let r be the round of s.
Since v directly commits b, there exists a strong certificate S for b at s (i.e., $|S| = 4f + 1$ distinct valid round-$(r + 2)$ blocks supporting b). By Lemma 17 (Strong-to-Weak Propagation), every valid block at any round $r' \geq r + 3$ has paths to at least $2f + 1$ members of S, i.e., it carries a weak certificate for b. In particular, any valid anchor block for deciding s (which necessarily lies at some round $r' \geq r+3$) has paths to a weak certificate for b.
But the indirect skip rule for s requires an anchor whose deterministic support excludes b (equivalently, with no weak certificate for b). This contradicts the propagation property above. Hence v' cannot skip s indirectly either. Contradiction.

Lemma 21. If a slot s is committed at two honest validators, then s contains the same block at both validators.

Proof 30. Let v and u be honest validators and suppose v commits block b at slot s. If u commits s as well, we show that u commits b.

Let w be the validator identity of the leader for s (i.e., the creator of b). By Observation 3, all honest validators agree on the leader identity per slot, so u also agrees that s must contain a block by w.

By Observation 4, there is a unique valid round-r block by w that can be committed in s.

Since v commits b and b is by w, b is this unique valid block by w for round r. Hence if u commits s, it must also commit b. Thus s contains the same block at both validators.

We say that a slot is *decided* at a validator v if s is committed or skipped, that is, if it is categorized as `commit` or `skip`. Otherwise, s is *undecided*.

Lemma 22. If a slot s is decided at two honest validators v and v', then either both validators commit s, or both validators skip s.

Proof 31. Assume by contradiction that there exists a slot s such that v and v' decide differently at s. We consider a finite execution prefix and assume *wlog* that s is the highest slot at which v and v' decide differently (*). Further assume *wlog* that v commits s and v' skips s. By Lemma 19 and Lemma 20, neither v nor v' could have used the direct decision rule for s; they must both have used the indirect rule. Consider now the anchor of s: v and v' must agree on which slot is the anchor of s, since by our assumption (*) above, they make the same decisions for all slots higher than s, including the anchor of s. Let s' be the anchor of s; s' must be committed at both v and v'. Thus, by Lemma 21, v and v' commit the same block b' at s'. But then v and v' cannot reach different decisions about slot s using the indirect decision rule. We have reached a contradiction.

We have proven the consistency of honest validators' commit sequences: honest validators commit (or skip) the same leader blocks, in the same order. However, we are not done: we also need to prove that non-leader blocks are delivered in the same order by honest validators. We show this next.

Causal history and delivery conditions Consider an honest validator v. We call the *causal history* of a block b in v's DAG, the transitive closure of all blocks referenced by b in v's DAG, including b itself. In BB-Core-Async, a block b is delivered by an honest validator v if (1) there exists a committed leader block l in v's DAG such that b is in l's causal history (2) all slots up to l are decided in v's DAG and (3) b has not been delivered as part of a lower slot's causal history. In this case we say b is *delivered at* slot s, or *delivered with* block l.

Lemma 23. If a block b is delivered by two honest validators v and v', then b is delivered at the same slot s, and b is delivered with the same leader block l, at both v and v'.

Proof 32. Let s be the slot at which b is delivered at validator v, and l the corresponding leader block in s, also at validator v. Consider now the slot s' at which b is delivered at validator v', and l' the corresponding leader block. Assume by contradiction that $s' \neq s$. If $s' < s$, then v would have also delivered b at slot s', since by Lemma 21 must commit the same leader blocks in the same slots, so v could not have delivered b again at slot s; a contradiction. Similarly, if $s < s'$, then v' would have already delivered b at slot s, since by Lemma 21 v and v' must have committed the same block in slot s; contradiction. Thus it must be that $s = s'$, and by Lemma 21, $l = l'$.

We can now prove the main safety properties of BlueBottle: Total Order and Integrity.

Theorem 4 (Total Order). BlueBottle satisfies the total order property of Byzantine Atomic Broadcast.

Proof 33. This property follows immediately from Lemma 23 and the fact that honest validators order the causal histories of committed blocks using the same deterministic function, and deliver blocks in this order.

Theorem 5 (Integrity). BlueBottle satisfies the integrity property of Byzantine Atomic Broadcast.

Proof 34. This is by construction: a block b is delivered as part of the causal history of a committed leader block only if b has not been delivered along with an earlier leader block (see "Causal history & delivery conditions" above). So an honest validator cannot deliver the same block twice.

C.3. Liveness Lemmas

Block inclusion.. The following two lemmas establish that blocks broadcast by honest validators are eventually included in all honest validators' DAGs.

Lemma 24. If a block b produced by an honest validator v references some block b', then b' will eventually be included in the local DAG of every honest validator.

Proof 35. This is ensured by the synchronizer subcomponent in each validator: if some validator w receives b from v, but does not have b' yet, w will request b' from v; since v is honest and the network links are reliable, v will eventually receive w's request, send b' to w, and w will eventually receive b'. The same is recursively true for any blocks from the causal history of b', so w will eventually receive all blocks from the causal history of b' and thus include b' in its local DAG.

Lemma 25. If a honest validator v broadcasts a block b, then b will eventually be included in the local DAG of every honest validator.

Proof 36. Since network links are reliable, all honest validators will eventually receive b from v. By Lemma 24, all honest validators will eventually receive all of b's causal history, and so will include b in their local DAG.

Main structural lemmas.. These are the main structural lemmas that we will use to prove liveness. The key idea is that the reference rule creates significant overlap among honest blocks across consecutive rounds, which we can leverage to ensure that a randomly chosen leader has sufficient honest support to be directly committed.

Lemma 26 (Reference Honesty Lower Bound). Any *valid* round-$(R+1)$ block (whether created by an honest or Byzantine validator) references at least $3f+1$ honest round-R blocks.

Proof 37. In any round there are $5f+1$ total validators of which at most f are Byzantine, so at least $4f+1$ are honest. A valid round-$(R+1)$ block, by definition of the protocol's formation rule, references at least $4f+1$ distinct round-R blocks (one per validator identity). At most f of these can be Byzantine, hence at least $4f+1-f=3f+1$ are honest.

Lemma 27 (Many Heavily Referenced Round-R Blocks). There are at least $2f+1$ honest round-R blocks each referenced by at least $f+1$ honest round-$R+1$ blocks.

Proof 38. By Lemma 26 every valid round-$(R+1)$ block references at least $3f+1$ *honest* round-R blocks. There are $4f+1$ honest validators, hence $4f+1$ honest round-$(R+1)$ blocks in total. Consider the bipartite graph whose left vertices A are the honest round-$(R+1)$ blocks and whose right vertices B are the honest round-R blocks; connect $a \in A$ to $b \in B$ if a references b.

Let E be the total number of edges. Each $a \in A$ has degree at least $3f+1$, so

$$E \geq (4f+1)(3f+1).$$

Suppose for contradiction that fewer than $2f+1$ honest round-R blocks have degree at least $f+1$ (i.e., are referenced by $\geq f+1$ honest round-$(R+1)$ blocks). Let $X \subseteq B$ be the (assumed) set of degree $\geq f+1$ blocks with $|X| \leq 2f$. Any block in $B \setminus X$ then has degree at most f.

We upper bound E under this hypothesis:

$$\begin{aligned} E &\leq |X|(4f+1) + (|B|-|X|)f \\ &= |X|(4f+1-f) + f(4f+1) \\ &= |X|(3f+1) + f(4f+1). \end{aligned}$$

Using $|X| \leq 2f$ we get

$$\begin{aligned} E &\leq 2f(3f+1) + f(4f+1) \\ &= (6f^2+2f) + (4f^2+f) \\ &= 10f^2+3f. \end{aligned}$$

Yet

$$(4f+1)(3f+1) = 12f^2+7f+1 > 10f^2+3f,$$

a contradiction. Therefore, $|X| \geq 2f+1$, establishing the claim.

Lemma 28 (Common Honest Ancestors). Any set of $4f+1$ round-$R+2$ blocks collectively references at least $2f+1$ common honest round-R blocks.

Proof 39. Let H denote the set of (at least) $2f+1$ honest round-R blocks guaranteed by Lemma 27: every $h \in H$ is referenced by at least $f+1$ honest round-$(R+1)$ blocks.

Consider any multiset S of $4f+1$ round-$(R+2)$ blocks (they may be arbitrary, honest or Byzantine). By applying Lemma 26 to round $R+1$ vs. $R+2$, each valid round-$(R+2)$ block includes at least $3f+1$ *honest* round-$(R+1)$ blocks among the $4f+1$ distinct references it must carry.

Fix some $h \in H$. Suppose, for contradiction, that a particular $s \in S$ fails to (indirectly) reference h. Then s must omit every honest round-$(R+1)$ block that references h. However, h has at least $f+1$ such honest round-$(R+1)$ children, while s can omit at most $(4f+1)-(3f+1)=f$ honest round-$(R+1)$ blocks (because it necessarily includes at least $3f+1$ of the $4f+1$ honest ones). Since $f+1>f$, omitting them all is impossible. Therefore, every $s \in S$ (indirectly) references h.

The argument holds for each $h \in H$, so all blocks in S commonly (indirectly) reference every element of H. Thus, their intersection over round-R ancestors contains H, and has size at least $|H| \geq 2f+1$.

Liveness theorems.. We now leverage the structural overlap to obtain probabilistic liveness via the common coin.

Definition 3 (Core Set of Round R). Let C_R be the set of honest round-R blocks each referenced by at least $f+1$ honest round-$(R+1)$ blocks. By Lemma 27, $|C_R| \geq 2f+1$.

Lemma 29 (Persistence of Core Support). Every valid round-$(R+2)$ block (indirectly) references every block in C_R.

Proof 40. Immediate from Lemma 28 since $C_R \subseteq H$ for the set H used there.

We denote by $l \leq 5f+1$ the number of leader slots per round.

Lemma 30 (Direct Commitment via Core Intersection). Fix a round r and let $n=5f+1$. When l leader slots are sampled uniformly at random without replacement from the n validators, the probability that at least one slot can be directly committed is

$$p^\star = 1 - \frac{\binom{n-|C_R|}{l}}{\binom{n}{l}}.$$

Moreover, if $l > n-|C_R|$ (in particular, if $l > 3f$ using $|C_R| \geq 2f+1$) then $p^\star = 1$ (deterministic success).

Proof 41. By Lemma 29, any selected leader whose block lies in C_R is (eventually) directly commit-able by every honest validator. Thus a successful direct commitment in round r occurs iff the sampled set intersects C_R. The

probability that it does not intersect C_R is exactly the hypergeometric zero-success probability $\binom{n-|C_R|}{l}/\binom{n}{l}$; subtracting from 1 yields $p^\star$.
If $l > n - |C_R|$, it is impossible to choose all l leaders outside C_R, so the intersection is certain and $p^\star = 1$. Using only $|C_R| \geq 2f + 1$, we have $n - |C_R| \leq 5f + 1 - (2f + 1) = 3f$, hence any $l > 3f$ suffices for determinism.

Lemma 31 (Eventual Slot Resolution). Fix a slot s. Every honest validator eventually either commits or skips s, with probability 1.

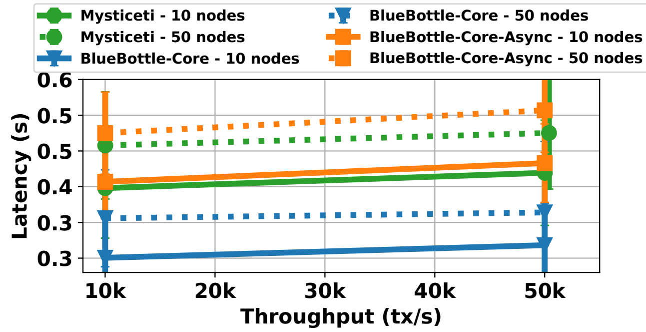
Proof 42. We prove the lemma by showing that the probability of s remaining undecided forever at some honest validator is 0. In order for s to remain undecided forever, s cannot be committed or skipped directly. Furthermore, s cannot be decided using the indirect rule. This means that the anchor s' of s must also remain undecided forever, and therefore the anchor s'' of s' must remain undecided forever, and so on. The probability of this occurring is at most equal to the probability of an infinite sequence of rounds with no directly committed slots, equal to $\lim_{t\to\infty}(1 - p^\star)^t = 0$, where $p^\star > 0$ is the probability from Lemma 30.

Theorem 6 (Validity). BB-Core-Async satisfies the validity property of Byzantine Atomic Broadcast.

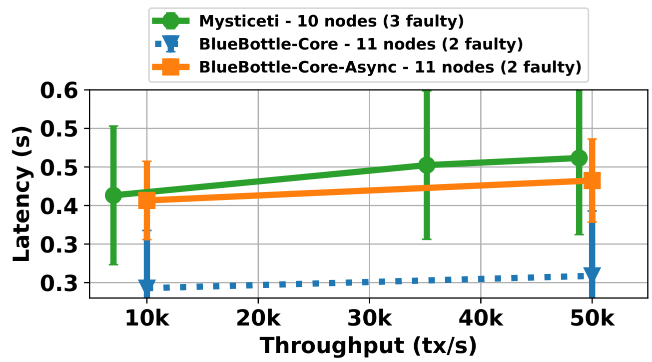
Proof 43. Let v be an honest validator and b a block broadcast by v. We show that, with probability 1, b is eventually delivered by every honest validator. Lemma 25 b is eventually included in the local DAG of every honest validator. So every honest validator will eventually include a reference to b in at least one of its blocks. Let r be the highest round at which some honest validator includes a reference to b in one of its blocks. By Lemma 30, with probability 1, eventually some block b' at a round $r' > r$ will be directly committed. Block b' must reference at least $4f+1$ blocks, thus at least $3f+1$ blocks from honest validators. Since all validators have b in their causal histories by round r, b' must therefore have a path to b. Lemma 31 guarantees that all slots before b' are eventually decided, so b' is eventually delivered. Thus, b will be delivered at all honest validators at the latest when b' is delivered along with its causal history.

Theorem 7 (Agreement). BB-Core-Async satisfies the agreement property of Byzantine Atomic Broadcast.

Proof 44. Let v be an honest validator and b a block delivered by v. We show that, with probability 1, b is eventually delivered by every honest validator. Let l be the leader block with which b is delivered, and s the corresponding slot. By Lemma 31, all blocks up to and including s are eventually decided by all honest validators, with probability 1. By Lemma 23, all honest validators commit l in s. Therefore, all honest validators deliver b eventually.



(a) Committees with 10 and 50 validators with no validator faults.



(b) With 3 or 2 crash faults; committees with 10 and 11 validators (minimum to tolerate 3 or 2 faults).

Figure 4: WAN throughput-latency performance comparison of BB-Core ($n = 5f + 1$), BB-Core-Async ($n = 5f + 1$), and Mysticeti ($n = 3f+1$) with a 512B transaction size. The y-axis starts at 300 ms to zoom in on the latency difference between the systems.

C.4. BB-Core-Async Performance

We implement BB-Core-Async by extending our BB-Core prototype (Appendix A) and evaluate it using the same setup as in Section 6. We compare its performance with BB-Core ($n = 5f + 1$) and Mysticeti ($n = 3f + 1$).

Figure 4 presents the WAN throughput–latency results for BB-Core, BB-Core-Async, and Mysticeti under two scenarios: (a) committees of 10 and 50 validators without faults (Figure 4a), and (b) committees of 10 and 11 validators (enough to tolerate 3 and 2 faults) under corresponding crash failures (Figure 4b). We cap throughput at 50,000 tps for cost reasons. BB-Core-Async matches Mysticeti's throughput–latency performance across both scenarios and committee sizes, as both commit in three rounds in the common case. This highlights a key trade-off: for equivalent performance, one can choose between an asynchronous protocol with $n = 5f + 1$ or a partially synchronous one with $n = 3f + 1$.