Steelhead: Interleaving Partially Synchronous and Asynchronous Commit Rules on a Shared DAG

George Danezis[1,2], Zeno de Angeli[2], Philipp Jovanovic[1,2], Lefteris Kokoris-Kogias[1], Markus Legner[1], and Alberto Sonnino[1,2]

[1] Mysten Labs
[2] University College London (UCL)

Abstract. Dual-mode consensus protocols are fast when the network is partially synchronous and remain live under asynchrony. We introduce Steelhead, a dual-mode mechanism that composes a partially synchronous and an asynchronous commit rule over one DAG: every k-th round is decided by the asynchronous rule, whose leader a common coin reveals after the votes, and all other rounds by the partially synchronous rule. Every interval, validators replay the committed DAG under each candidate period, adopt the one with the fewest expected message delays, and fall back to $k = 1$ when the output stalls; the asynchronous rule applied to the coin rounds alone keeps the protocol live. Steelhead sends no message beyond the DAG's blocks, not even to agree on the period, and opens a coin only on the rounds that need a hidden leader. It is generic over pairs of DAG commit rules that share a committee; we instantiate it with Mysticeti [3] and Mahi-Mahi [18] at $n \geq 3f + 1$ and with the two variants of BlueBottle [29] at $n \geq 5f + 1$. We prove it safe and live, and provide mechanized proofs in Lean 4. In simulation, Steelhead matches the partially synchronous protocol in a healthy network, stays close to the asynchronous one when network conditions stall the partially synchronous one, and adapts quickly in both directions.

1 Introduction

Partially synchronous BFT consensus [2,3,6,8,26,30] is fast but loses liveness under asynchrony or a targeted leader attack [17]. Asynchronous BFT [1,12,18,19,23] stays live but pays extra message delays and a common coin. Dual-mode protocols [5,10,11,9,16,21,22,27] aim for both.

Existing dual-mode designs all pay for the mode decision, in one of three ways (Table 2). Serial designs, such as Ditto [16] or Bolt-Dumbo Transformer [22], leave the fast path on a local timer and must then agree on where the abandoned path stopped; this reconciliation is slow and fragile, as the attack of Rambaud et al. [25] on Ditto's fallback showed. Parallel designs run both paths at once and pay quadratic messages even under synchrony: ParBFT1 [11] settles the race between them with a separate binary agreement per slot, and Ipotane [9] folds that agreement into its asynchronous path at the price of one extra round per slot. Bullshark [27] instead moves the mode into the blocks, chosen by timeouts,

and opens a coin for every leader whether the network is synchronous or not, which it can afford only because it is already slow: on its certified DAG a commit takes six message delays, against three on an uncertified one [3,20]. Icarus [10] abandons the second path altogether, rotating instead among the validators' chains, but still runs a Byzantine agreement at every switch to align what the previous path committed. We discuss these designs in Section 7.

Steelhead makes no such decision. DAG commit rules only read the DAG and, beyond pacing, never shape it, so two rules can share one DAG: each round holds one leader slot, and the round number picks which rule decides it. Every k-th slot is an asynchronous slot, decided by the asynchronous rule (R_{a}), and the others are synchronous slots, decided by the partially synchronous one (R_{s}). There is no mode vote, no path switch, and no extra message. Because reading the DAG is virtually free, the same blocks can also be read a second time, for a different purpose (Section 4.3). The period k acts as a dial, with $k = 1$ giving an asynchronous protocol and $k = \infty$ giving a partially synchronous one (Fig. 1).

Insights. First, each rule needs a fixed number of rounds to decide a slot, called *wavelength*; the two rules have different wavelengths. As in every DAG protocol, a slot left undecided after its wavelength is decided later by the first committed slot above it, its *anchor*. Steelhead starts the anchor search one wavelength above the undecided slot, using the wavelength of the undecided slot rather than that of the anchor. Any anchor at that height sees evidence of a direct commit, whichever rule produced it. This is the core of the safety argument. It lets Steelhead skip the reconciliation step of related work [16,5,11], agreeing on where the previous rule stopped: the slots one rule leaves pending are decided by the anchors above them, whichever rule decides those.

Second, because a change is completed by the anchors that follow it, a change of rule is safe whenever it happens. Steelhead thus never has to decide a mode switch from the local observation of a number of timeouts, which is how Bullshark [27] and serial designs choose their mode [16,22].

Third, every k-th slot is decided by the asynchronous rule, so the DAG carries a coin every k rounds rather than every round. Under asynchrony an adversary can keep the known-leader slots undecided, and output stops behind the first of them. Validators therefore read the same blocks a second time, applying the asynchronous rule to the coin-carrying slots alone. This *control reading* outputs nothing, but it keeps deciding under asynchrony, and every honest validator obtains from it the same committed block from which to compute the period (Section 4.3). In the opposite direction, at $k = 1$ nobody waits for a known leader, so no evidence about the partially synchronous rule would form; on a few scheduled *canary rounds*, validators still wait for it, and the certificates that form are that evidence (Section 4.2). The period is thus a deterministic function of data every honest validator holds identically, as in Shoal [26] and Hammerhead [28], and drops to 1 when output has stalled.

Finally, validators replay the agreed data under each candidate period and keep the period that would have committed with the lowest latency (Section 4.4). Rounds that ran under the partially synchronous rule opened no coin, so Steelhead

Fig. 1. Slots by round number. Squares are synchronous slots, decided by R_s; circles are asynchronous slots, decided by R_a. At $k = 8$, seven slots are decided by R_s and the eighth carries a coin; at $k = 1$ every slot is an asynchronous slot. A canary round, drawn as both shapes, is an asynchronous slot on which validators still wait for the known leader.

averages over the n possible leaders, which is the expectation a uniform coin would give. This is sound because the replay only ranks the two rules and commits nothing. The asynchronous rule certifies a fixed fraction of every round's blocks under any message schedule, so an adversary cannot drive that average down.

We instantiate Steelhead on two pairs of uncertified DAG protocols. As an example of $n \geq 3f + 1$, we pair Mysticeti [3] with Mahi-Mahi [18]. At $n \geq 5f + 1$, we pair the partially synchronous and asynchronous variants of BlueBottle [29], which that paper presents as two separate protocols.

Contributions.

- An adaptive period selection mechanism allowing validators to recompute the best commit rule at run time from the committed DAG alone, with no extra message and no timer.
- Steelhead, a dual-mode mechanism in which two commit rules interpret one DAG and the round number alone selects which of them decides each slot.
- Proofs of safety and liveness, machine-checked in Lean 4.
- An implementation of Steelhead, over two types of protocols, $n \geq 3f + 1$ (Mysticeti and Mahi-Mahi) and $n \geq 5f + 1$ (BlueBottle) on top of the authors' codebase, and simulations in various network conditions.

2 Background and Model

System model. The system consists of n validators, of which at most f are Byzantine, and every quorum has size $n - f$; each instantiation fixes the resilience (in our evaluation either $n \geq 3f + 1$ or $n \geq 5f + 1$). Validators communicate over point-to-point channels, and the DAG substrate carries the base protocols' standard assumptions of signed blocks and hash-based causal references. Steelhead itself adds no cryptography and no messages.

Under partial synchrony [14], message delays between honest validators are bounded by a known Δ after GST. Under asynchrony [15], an adaptive adversary schedules delivery but must eventually deliver every message. For a coin-based R_a we assume a common coin as a black box [19,18,27], with three properties: agreement, unpredictability until $n - f$ authorities have contributed, and uniformity conditional on everything the adversary has seen before the value can be learned, including earlier coins. The round's leader is a balanced map of the coin's value into the n authorities, so it is uniform and unknown until $n - f$ shares exist. The coin can be instantiated with threshold signatures [7], one share per block of the rounds that need a coin.

Table 1. The four base protocols and the parameters Steelhead reads

Rule	Wavelength (w)	Leader election	Leader wait	Committee
Mysticeti	3	Deterministic	✓	$n \geq 3f+1$
Mahi-Mahi	$\{4,5\}$	Randomized	×	$n \geq 3f+1$
BlueBottle (sync.)	2	Deterministic	✓	$n \geq 5f+1$
BlueBottle (async.)	3	Randomized	×	$n \geq 5f+1$

DAG-based consensus. In an uncertified DAG there is at most one block per honest validator per round, and each block references $n-f$ blocks of the previous round. A validator advances a round after receiving $n-f$ blocks from the previous round, and after a leader wait (in partial synchrony). Equivocation is tolerated rather than prevented: a vote is the first-seen block per author and round in a depth-first search. Each round has a known number of leader slots. The direct rule decides each slot as a *commit* or a *skip* from the rounds of its wave, or leaves it $\perp$. Under the indirect rule, an $\perp$ slot is decided by its *anchor*, the earliest later slot that is committed or $\perp$, passing over skipped slots. The slot commits if and only if the anchor's causal history holds a certificate for it [3]. Output proceeds in slot order, so an $\perp$ slot holds back every slot above it; each committed leader adds to the output, in a deterministic order, the blocks of its causal history not already in that of an earlier committed leader.

A rule of wavelength w decides the slot at round r from the wave of rounds $r, \ldots, r+w-1$ alone: it commits when the blocks of that wave reference the leader block in the pattern the rule prescribes, skips when enough of them fail to, and otherwise leaves the slot $\perp$. Steelhead never inspects that pattern: it calls the rule's own decision predicates and reads only the parameters of Table 1, so any rule of this shape plugs in unchanged. As an example, in Mysticeti and Mahi-Mahi round $r+w-2$ is the vote round and round $r+w-1$ the certify round, a certificate is a certify-round block referencing $n-f$ votes, a direct commit needs $n-f$ certificates and a direct skip $n-f$ non-votes, and the boost rounds fill the gap at $w \geq 4$.

What the two rules must provide. Steelhead requires five (typical) properties from the two rules it is instantiated with:

- **A1**: Both rules run on the same committee, with quorums of size $n-f$, and read the same blocks; an honest author produces one block per round, referencing a quorum of the previous round and every block it holds that its previous block did not cover, and blocks carry whatever either rule needs.
- **A2**: A rule decides the slot at round r from the wave of rounds $r, \ldots, r+w-1$, and a direct commit leaves its evidence in the causal history of every block at round $r+w$ or above.
- **A3**: A direct skip at a slot implies that no certificate for it ever forms.
- **A4**: After GST, with pacing, a synchronous slot whose leader is honest and whose successor round carries a leader wait is directly committed.
- **A5**: Before the round's coin can be learned, the revealed history fixes a set of at least pn candidate authors whose blocks have direct-commit evidence in every admissible continuation that populates the wave, the counting property;

conditional on that history the leader is uniform over all authors, so a direct commit has probability at least p whatever the adversary learned from earlier coins; and each candidate's evidence is checkable on a committed window.

The wave family of Table 1 provides clauses **A1** to **A3** by construction: evidence accumulates in a wave, quorum intersection carries a direct commit's evidence into every block one wave later, and the commit and skip thresholds intersect, so a skipped slot can never be certified. A pair must add one clause each, clause **A4** for the partially synchronous rule and clause **A5** for the asynchronous one, which is why no single protocol satisfies the interface alone. Each clause is discharged for the two pairs we instantiate (Appendix B).

Problem statement. Steelhead composes a rule R_{s}, of wavelength w_{s} and with a known leader, and a rule R_{a}, of wavelength w_{a} and with a coin-elected leader, into one protocol that inherits the latency of R_{s} under periods of synchrony and the liveness of R_{a} under asynchrony. Both rules are used through the interface above. Steelhead implements atomic broadcast.

Definition 1 (Atomic broadcast). *Atomic broadcast satisfies:*
- ***Agreement:*** *If one honest validator commits a block, all honest validators eventually commit it.*
- ***Integrity:*** *Each honest validator commits a block at most once, and only if it was proposed by its author.*
- ***Validity:*** *If an honest validator proposes a block, all honest validators eventually commit it (with probability* 1*).*
- ***Total order:*** *If an honest validator commits a block b before a block b', no honest validator commits b' before b.*

3 Interleaving Two Commit Rules

Steelhead adjusts the period k at run time. For clarity, we first describe the protocol at a fixed k in this section, then describe the full adaptive protocol in Section 4. The reader may keep Mysticeti and Mahi-Mahi in mind throughout, to make the exposition concrete.

3.1 Slots by Round Number

The wavelength function is $w(r) = w_{\mathrm{a}}$ if $r \bmod k = 0$, and w_{s} otherwise, where k is the period in force at round r (Section 4 indexes it by interval, as $k_{j(r)}$, once it adapts). Each slot is therefore decided by exactly one rule, fixed by its round number, and this function is the only thing Steelhead adds to the decision rule. Blocks are identical in every round, aside from a coin share, and the DAG never pauses; the wavelength dictates only which rounds hold a slot's evidence. Fig. 2 is the running example of Sections 3 and 4.

A synchronous slot's leader is known in advance from a deterministic schedule, such as a round-robin or reputation-based schedule [26,28]. An asynchronous

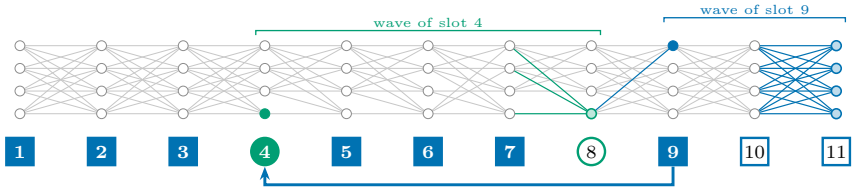


Fig. 2. The two rules deciding one another's slots, on one DAG at $k = 4$ ($n = 4$, $w_s = 3$, $w_a = 5$). Squares are synchronous slots and circles asynchronous slots, and the brackets mark the wave each rule reads, three rounds for R_s and five for R_a; filled is commit, white is undecided. Slots 8, 10 and 11 are left undecided because their waves reach above the drawn rounds. The direct rule leaves slot 4 undecided: only one block of round 8 certifies its coin leader, short of the three a direct commit needs. Slot 9, the first slot at its anchor floor $4 + w_a$, commits directly under R_s, and its causal history holds that certificate (the arrow), so it decides slot 4.

slot's leader is revealed by the coin only once its votes are fixed, at round $r + w_a - 1$. Validators wait, up to a bounded timeout, for a synchronous slot's leader block before proposing at the round above it and, in Mysticeti, for a quorum of votes for it before proposing one round higher. Nobody waits for the hidden leader of an asynchronous slot; the canary rounds of Section 4.2 are the one exception. Clause **A4** rests on these waits, which Steelhead inherits unchanged; a synchronous slot immediately below an asynchronous slot lacks the second one, so it may be decided by its anchor instead of directly, and clause **A4** is stated for synchronous slots whose successor round carries a leader wait, which the canary rounds restore even at $k = 1$.

3.2 Deciding Slots: the Anchor Floor

Each slot is decided by its own rule over its own wave (Algorithm 1), and this interpretation of the DAG, which considers every slot and produces the ledger, is the output reading; Section 4.3 adds a second reading of the same blocks later. The anchor of the slot at round r is searched from round $r + w(r)$, a floor set by the wavelength of the *undecided slot*, not by that of the anchor. A direct commit leaves certificates at round $r + w(r) - 1$, so every block from round $r + w(r)$ on holds one in its causal history. An anchor closer than that, such as an asynchronous slot anchored at the synchronous slot right above it, cannot see those certificates and would skip a slot that another validator committed. Fig. 2 shows this: the direct rule leaves the asynchronous slot at round 4 undecided, and its anchor search starts at round $4 + w_a$, where the synchronous slot at round 9 has committed. Appendix B discusses and illustrates (Fig. 6) the choice of floor in further detail.

The consequence is handover (Corollary 1, Appendix B): the indirect step of either rule completes a direct commit of the other that it never witnessed. The arrow in Fig. 2 marks this: the synchronous slot's commit decides a slot the asynchronous rule left undecided. This is the whole cross-rule argument, and the reason a period change never has to pause the protocol to finish pending slots.

```
1: procedure WaveLength(r)
2:     return w_a if r mod k = 0 else w_s                          ▷ k in force for round r

3: procedure TryCommit(DAG)
4:     for each slot s at round r, from the highest undecided downward:   ▷ as Mysticeti
5:         w ← WaveLength(r)
6:         v ← TryDirectDecide(s, w)        ▷ skip on n−f blames or commit on n−f certificates
7:         if v = ⊥: v ← TryIndirectDecide(s, w)
8:     return the maximal decided prefix, in round order

9: procedure TryIndirectDecide(s, w)
10:    a ← FindAnchor(s, w)                 ▷ earliest commit-or-undecided slot at round ≥ r + w
11:    if a is undecided: return ⊥
12:    if a certificate for s's leader block is in a's causal history: return commit
13:    else: return skip
```

Algorithm 1: Decision with a slot-dependent wavelength

```
1: interval, maxPeriod, canary, ϵ          ▷ e.g., 128 rounds, period 64, canary 31, hysteresis 2%
2: k ← maxPeriod                                                  ▷ state: the period in force

3: procedure OnInterval(j)                                  ▷ once the scan of interval j ends
4:     A ← the pivot: the first commit of the control reading in interval j ▷ Section 4.3; none keeps k
5:     W ← causal history of A over the preceding interval rounds   ▷ the window: agreed data
6:     if the ledger up to A committed no slot in W then            ▷ failover on a stalled ledger
7:         k ← 1
8:     else
9:         k* ← arg min_{k'∈K} Replay(W,k') ▷ expected output delay under each candidate (Section 4.4)
10:        if k* beats k by more than ϵ: k ← k*                                  ▷ hysteresis
11:    return k                          ▷ applies to the rounds of interval j + 1 and above
```

Algorithm 2: The adaptation loop, abstract form (runs at every validator)

3.3 What a Fixed Period Cannot Do

In a healthy network, an asynchronous slot costs $w_a - w_s$ extra rounds once every k rounds, and its successor waits at most $\max(0, w_a - w_s - 1)$ rounds for causal ordering. These delays never compound and are negligible when the network is synchronous and k is large. Appendix A discusses a variant that avoids even this, at the price of a longer proof.

Under asynchrony, every synchronous slot could be left $\perp$. The asynchronous slots still commit, because their leader is hidden, but nothing they commit is output: slots are output in round order, and the anchor search of an $\perp$ synchronous slot stops at the next $\perp$ one. No fixed $k > 1$ is therefore live under asynchrony, while $k = 1$ is live but slow when the network is healthy. Consequently, the period must be adjusted dynamically, which is the subject of Section 4.

4 Adapting the Period

The period selects between the two rules from what the committed DAG shows, rather than from a guess about the network, which the adversary controls. Section 4.1 states the loop, and the three subsections after it supply what the loop needs: evidence of both rules in the DAG (Section 4.2), agreement on that evidence even when the ledger is stuck (Section 4.3), and a score that turns it into a period (Section 4.4).

4.1 The Adaptation Loop

Rounds are grouped into intervals of `interval` rounds, each running under one period. In each interval a *pivot* is chosen, the first commit of the control reading, a second reading of the same blocks that applies the asynchronous rule to the rounds that carry a coin and whose verdicts never enter the ledger (Section 4.3). The causal history of that pivot over the last `interval` rounds is the *window*, and the period of the next interval is the candidate whose replay of the window scores best (Section 4.4). Algorithm 2 states the loop. As a failover, if the ledger up to the pivot committed nothing in the window, the next period is 1 outright: the replay ranks periods, while the failover guarantees that an output which has stopped advancing always gets the rule that makes it advance again (and greatly simplifies the liveness proof).

The new period is a deterministic function of the pivot's window and of round numbers, and it applies only from the next interval on. By induction over intervals, every honest validator computes the same sequence of periods, for any deterministic update rule (Theorem 4). No message and no vote is involved.

The remaining parameters and guards are of no consequence to the argument and are deferred to Appendix B: the gating rule, under which a slot is evaluated once its interval's period is known, the one-interval lag, the bounds on `interval` and `maxPeriod`, the warm-up interval, the hysteresis, and the candidate set $\{1, 2, 4, \ldots, \texttt{maxPeriod}\}$.

4.2 Seeing Both Rules: Two-Way Probes

A window shows only what the running rule did. Without a probe, at $k = 1$ nothing would wait for a known leader, so the partially synchronous rule would look dead forever and the period could never climb back; and at $k = \infty$ there would be no coin at all, so nothing would be live when the network turns, which is why ∞ is not a candidate period. Each regime—the synchronous one, at a large k, and the asynchronous one, at $k = 1$—therefore keeps a thin probe of the other rule alive, at a fixed cost. There are two dials, one per direction.

From R_{s} to R_{a}. In the synchronous regime, the asynchronous slot every k rounds (with $k \leq \texttt{maxPeriod}$) is the probe of the asynchronous rule. It carries a coin, it commits under asynchrony, and it keeps the control reading moving, so the period can fall when the network turns.

From R_{a} to R_{s}. In the asynchronous regime, every round r with $r \bmod \texttt{canary} = 0$ is a *canary*: validators keep the leader wait for the known leader of that round even on an asynchronous slot. A canary that gathers a certificate is evidence that the partially synchronous rule would work again, so the period can climb back. The cost is one leader wait every `canary` rounds (although wall-time waits are meaningless in asynchrony), with `canary` odd and hence coprime to every candidate period, so that every candidate is probed.[3] The same wait restores the precondition of clause **A4** at $k = 1$ (Section 3.1), so the canaries serve twice.

[3] A `canary` sharing a factor with some candidate would leave all its canaries on that candidate's asynchronous slots, and the period could never climb to it.

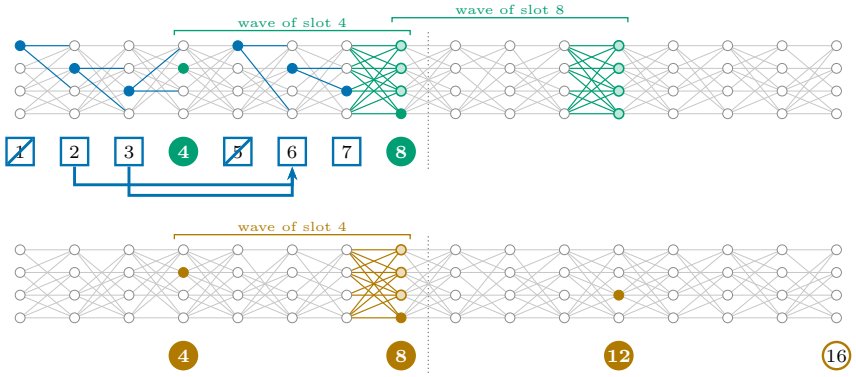


Fig. 3. One DAG read twice ($n = 4$, $w_s = 3$, $w_a = 5$, $k = 4$, an interval of 8 rounds ending at the dotted line, every known leader's block reaching only $f + 1$ validators in time). Squares are synchronous slots and circles asynchronous slots, filled when the reading commits them, struck on a skip and outlined while $\perp$; the brackets mark a wave. **Top:** the output reading; slots 4 and 8 commit, but slots 2 and 3 wait at slot 6 (arrows), so nothing is output. **Bottom:** the control reading of the same blocks considers the coin-carrying rounds alone and resolves anyway; its first commit, slot 4, is the pivot, and an output that committed nothing sets $k = 1$ above round 8. Slots 4 and 8 belong to both readings and are committed directly in both.

Neither regime ever loses sight of the other rule, and switching regimes is reading that evidence. The two probe rows of Fig. 1 show it.

4.3 Agreeing on the Evidence: the Control Reading

The pivot cannot come from the ledger. Under asynchrony the ledger stalls below the first synchronous slot left $\perp$, so a pivot taken from it would never be committed and the period could never fall (Fig. 3). The control reading considers only its own slots, defined next, and searches anchors among them alone. Each carries a coin, so its leader is hidden and asynchrony cannot stall the reading, and its first commit in an interval is the pivot.

Its slots are the interval's asynchronous slots, the rounds with $r \bmod k = 0$, and, above the interval, the multiples of `maxPeriod`. The anchor of a slot near the end of an interval lies above it, where the period is precisely what this scan is computing, so k is no guide there; the multiples of `maxPeriod` are asynchronous slots under every candidate period, hence defined and coin-carrying whatever the scan decides. Because this set is fixed by rule rather than by what a view holds, the reading is agreed by the same lemmas as the ledger, with direct verdicts identical in both readings.

Steelhead opens one coin every k rounds, and the control reading opens none of its own: every candidate divides `maxPeriod`, so the multiples of `maxPeriod` that the reading uses above an interval boundary already carry one. They are what producers fall back on while an interval's period is still being computed and they cannot yet tell which of its rounds are asynchronous slots; a share for such a round is contributed late.

The two readings overlap and need not agree: a round divisible by the period is a slot of both. Where either reading decides directly, both decide alike (Theorem 1

and Corollary 1, Appendix B); only their indirect verdicts can differ, because each searches anchors among its own slots. That divergence is harmless, since the two answer different questions: the output reading says what is committed, and the control reading says only where the period is computed from. A reading is a way to analyze an agreed subset of a DAG whose full interpretation is stuck.

4.4 Scoring Without a Coin: Counterfactual Replay

Because a commit rule only reads the DAG, the window can be reinterpreted under any candidate period, with $w'(r) = w_{\mathrm{a}}$ if $r \bmod k' = 0$ and w_{s} otherwise. The score is the expected delay from a round to the output of its blocks, so the head-of-line blocking caused by $\perp$ slots is already in it and nothing is added by hand. Each rule is scored from the evidence the window holds about it, which is a different kind of evidence for each.

The asynchronous rule, without a coin. Its only random input is a uniform leader, so its expected latency on the window is the plain average over the n possible leaders, each decided from the certificates the window holds. The slot at round r commits directly with probability c_r/n, where c_r is the number of the n candidate leaders that the window shows directly committed, and no coin is reconstructed.

The partially synchronous rule, from the canaries. On an asynchronous slot only the canary rounds waited for the known leader, so the replay takes its evidence from the canary rounds alone. Their success rate stands in for the candidate's other synchronous slots, and the replay is exact when the window holds no probe.

The adversary cannot make the asynchronous rule look slow: by the counting property, c_r/n has a floor under any scheduling (Lemma 3; at least $1/3$ for the $3f + 1$ pair at $w_{\mathrm{a}} = 5$). It can serve the public canaries and starve the other known leaders: if the output then stops, the failover of Section 4.1 takes over, and otherwise the output keeps advancing. Behaving well in one window and badly in the next is a regime change, measured by the next window.

The replay makes two approximations, both deterministic and identical for every candidate. First, where the rule would decide a slot from the causal history of the anchor that the candidate would have used, the replay asks only whether a certificate for the slot is anywhere in the window; the window is the pivot's own causal history, so it stands in for the anchor's. Second, a slot that a candidate never commits within the window counts as committing at the window's top round, so the rounds it holds back are deferred there rather than left undefined. The cost is $O(|K| \cdot |W|)$ for $|K|$ candidates over a window of $|W|$ blocks, and Algorithm 3 in Appendix B gives the procedure in full.

5 Correctness

Steelhead implements atomic broadcast (Definition 1) for any pair of rules satisfying the interface of Section 2. Full statements and proofs are in Appendix B, machine-checked in Lean 4 (Appendix C).[4]

Safety. A direct commit leaves a certificate in every block from round $r + w(r)$ on, and the anchor search starts there, so an indirect decision by either rule agrees with a direct commit by the other. Two indirect decisions agree because validators find the same anchor. This is the one cross-rule argument; everything else is inherited from the base protocols (Theorem 1, Corollaries 1 and 2).

Liveness. After GST, synchronous slots with an honest leader and a leader wait on their successor round commit directly by clause **A4**, and asynchronous slots do not get in the way (Theorem 2). Under asynchrony, the control reading keeps resolving, because every slot on it has a hidden leader. The failover then forces $k = 1$, and at $k = 1$ a run of w_{a} direct commits, which the counting property makes occur with probability 1, decides every slot below it, so the output resumes (Theorem 3).

Agreement on the period. The period is a deterministic function of the pivot's window, and the pivot is agreed because the control reading is. Induction over intervals gives the same sequence of periods at every honest validator, for any deterministic update rule (Theorem 4). At $k = 1$ and $k = \infty$, Steelhead's verdicts are exactly Mahi-Mahi's and Mysticeti's (Theorem 5).

6 Implementation and Evaluation

Our evaluation quantifies what deciding slots by round number buys over each pure protocol on the same DAG, under network conditions that favor one or the other. Benchmarking BFT protocols under actual Byzantine behavior is an open problem [4]; worst-case guarantees are established by the formal proofs of Section 5, while this evaluation measures latency under specific network conditions. We make the following claims:

- **C1**: In a healthy network, Steelhead matches the latency of the partially synchronous protocol.
- **C2**: Under network conditions that stall the partially synchronous protocol, Steelhead matches the latency of the asynchronous one.
- **C3**: Under benign asynchrony, Steelhead matches the latency of the best of the two protocols.
- **C4**: Steelhead adapts promptly to the network: it switches to the best protocol when conditions change and back when they lift.
- **C5**: Claims **C1** to **C4** hold for both classes of target protocols, at $n \geq 3f + 1$ and at $n \geq 5f + 1$.

[4] https://github.com/gdanezis/lean-dag (commit 4228bc5)

6.1 Implementation

We instantiate Steelhead on two pairs of protocols: Mysticeti [3] with Mahi-Mahi [18], at $n \geq 3f + 1$, and BlueBottle's partially synchronous variant with its asynchronous variant [29], at $n \geq 5f + 1$. We implement both in Rust on top of the codebase of Mysticeti, Mahi-Mahi, and BlueBottle [24] written by their respective authors; public leaders use a round-robin schedule.[5] Steelhead adds a per-round wavelength schedule read by the decision rule and the anchor search, restricts the leader wait to synchronous slots and canary rounds, and implements the period update by counterfactual replay of Algorithm 3. The replay module is about 770 lines; integrating it into the committer and the protocol table takes about 700 more. The block format remains untouched, and the DAG layer changes only in its round pacing, which reads the period to select the leader wait, and in its retention horizon: Steelhead adds no protocol messages, no cryptography beyond the coin the asynchronous protocol already carries, and no storage access. It runs purely in memory, and its memory is bounded by the interval: the replay reads the blocks of the last `interval` rounds.

6.2 Experimental Setup

The network conditions under test must be identical across protocols, switch on and off at scripted times, and remain reproducible. This is not possible in a cloud deployment, where asynchrony cannot be injected on demand and never repeats identically. We therefore use a deterministic discrete-event simulator with seeded runs, virtual time, and the real consensus code. We use a committee of $n = 50$ validators ($f = 16$ for the $3f + 1$ pair, $f = 9$ for the $5f + 1$ pair) in a full mesh; Appendix E repeats the runs at $n = 10$. Per-link latency is uniform in 25–50 ms. The leader timeout is 100 ms.

Each timeline runs a healthy network for 30 s, applies a specific network condition until 150 s, and restores the healthy network for a final 60 s. Plateau figures are taken over the windows between 60 and 150 s. We test six conditions: (i) a small leader delay of 30 ms, below the timeout; (ii) a large leader delay of 125 ms, above it; (iii) a permanent crash of f validators; (iv) a partial random delay, where each message is delayed with probability 0.3 by a duration drawn uniformly at random in 100–150 ms; (v) a full random delay, the same with probability 1; and (vi) a high jitter, an exponentially distributed delay of mean 75 ms capped at 400 ms. The leader delays target the public round-robin schedule and are blind to the coin; the random delays instantiate the random asynchronous model of Danezis et al. [13].

We compare the two pure protocols of each pair with adaptive Steelhead, and report end-to-end latency from submission to commit, as the mean per 5 s window and the median over seven seeded runs. Appendix E lists every parameter of Steelhead and of the simulator, and the per-panel numbers.

[5] `github.com/asonnino/mysticeti` (commit `1ae2aa7`)

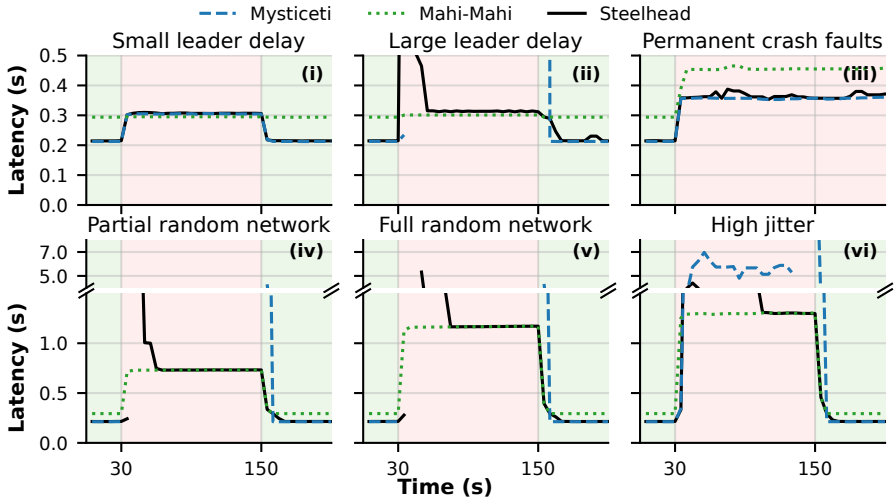


Fig. 4. Latency over time of Mysticeti, Mahi-Mahi, and adaptive Steelhead ($3f + 1$ pair, $n = 50$) under six network conditions applied from 30 s to 150 s (shaded). The bottom row breaks its axis: Mysticeti runs off the chart under large leader delay and full random delay.

6.3 Healthy Network

In the first 30 s and the last 60 s of every panel in Fig. 4, most legibly in (i), Steelhead commits at 215 ms, Mysticeti at 213 ms, and Mahi-Mahi at 294 ms. Steelhead is within 1% of Mysticeti and 27% below Mahi-Mahi. The period sits at its maximum, so one slot in 64 operates as an asynchronous slot and pays the two extra rounds described in Section 3, while everything else runs the Mysticeti rule. The counterfactual replay never lowers the period in a healthy window. This confirms Claim **C1**.

6.4 Degraded Network and Recovery

Mysticeti stalls in panels (ii), (iv), and (v) of Fig. 4. In (ii), every known leader's block arrives after the timeout, so every slot is directly skipped and nothing commits. In (iv) and (v) it commits sporadically, by the mechanism of Appendix D, and under the jitter of (vi) it degrades to a 5.7 s plateau. Mahi-Mahi is barely affected, at 301, 731, 1167, and 1300 ms.

Steelhead descends to period 1 within 10–20 s of the onset in (ii), 15 s in (iv), 20 s in (v), and 20–45 s in (vi). Until the period drops, Steelhead behaves as Mysticeti, causing the latency spike at the start of the phase. It then sits on the Mahi-Mahi line: 313 vs 301 ms in (ii), 732 vs 731 ms in (iv), and 1168 vs 1167 ms in (v). Panel (vi) takes longer to settle, at 1473 vs 1300 ms over the plateau window: under jitter the period passes through 2 for some 40 s before it reaches 1, and the last 25 s of the condition run at 1301 ms. This confirms Claim **C2**.

In (i) Mysticeti survives: all three protocols are within 5% (305, 295, 308 ms). Steelhead stays at period 64 but for at most one one-interval drop. In (iii),

Table 2. Dual-mode designs: what triggers a switch, and the dominant price of having one.

Design	Mode-change trigger	Dominant cost
Ditto [16]	quorum of timeouts	timeouts; MVBA per switch
BDT [22]	quorum of timeouts	timeouts; binary agreement per switch
Abraxas [5]	fast path lags slow chain	$O(n^2B)$ bits per block, always
ParBFT1 [11]	first path to finish	a VABA every slot
Ipotane [9]	first path to finish	an extra round per slot
TockOwl+ [21], ParBFT2 [11]	hedging delay	slow-track messages when hedged
Bullshark [27]	quorum of timeouts	timeouts; a coin every wave
Icarus [10]	backlog threshold	alignment (40–60% when switching)
Steelhead (this work)	round number	a coin every k rounds

Steelhead records 363 ms vs Mysticeti 356 ms vs Mahi-Mahi 455 ms, a 2% cost. Probes landing on a crashed round-robin leader read as starved slots, so the period briefly dips and climbs back every few intervals. Steelhead can naturally be composed with reputation-based leader-election [26,28] to remove crashed leaders from the schedule and with it this effect (Section 4.2). This confirms Claim **C3**.

Steelhead switches in both directions within a few intervals. In every panel it is back at period 64 within 10 s of the condition lifting, one interval at the healthy round rate; latency follows the period each time (Appendix E). This confirms Claim **C4**.

6.5 The $5f + 1$ Pair

We evaluate the $5f+1$ pair using the same code and parameters, with BlueBottle's two variants as the pure protocols (Fig. 5). In a healthy network, Steelhead commits at 169 ms, BlueBottle-PS at 168 ms, and BlueBottle-Async at 212 ms.

BlueBottle-PS stalls only in panel (ii) of Fig. 5, where Steelhead drops to period 1 within 10 s and matches BlueBottle-Async (229 vs 220 ms). Under the network-wide conditions (iv) to (vi) it survives but no longer ties its asynchronous variant at this committee size, running 48%, 41% and 9% slower; Steelhead follows the faster variant within 1% (705 vs 702, 837 vs 835, and 944 vs 935 ms). The link probability of Appendix D explains the gap: at $n = 50$ a validator with minimum-quorum references still references the leader with probability 0.82, but every commit now needs 41 votes, and the one-layer rule pays for it under random delays. In (iii) Steelhead follows BlueBottle-PS within 2% (221 vs 217 ms), and in (i) it follows it exactly (238 ms), which leaves it 11% above the asynchronous variant, the faster of the two in that panel. At $n = 10$ (Appendix E) the picture holds with one change: BlueBottle-PS ties its asynchronous variant under the network-wide conditions, and Steelhead sits between them. This confirms Claim **C5**.

7 Related Work

Table 2 compares these designs. Ditto [16], the Bolt-Dumbo Transformer [22] and Bullshark [27] leave the partially synchronous path once enough parties have reported several timeouts, typically a quorum of parties timing out on multiple successive leaders. Sizing that count is the known difficulty of these designs [11]:

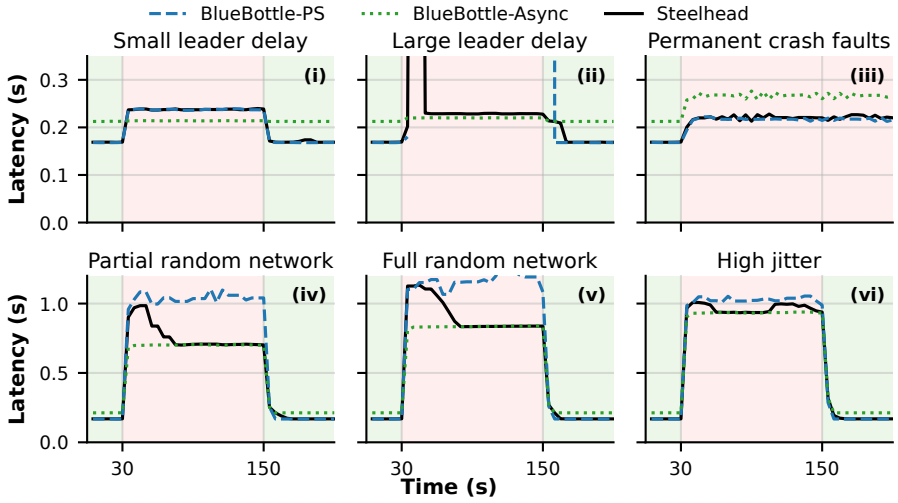


Fig. 5. Latency over time of BlueBottle's partially synchronous variant (BlueBottle-PS), its asynchronous variant (BlueBottle-Async), and adaptive Steelhead ($5f + 1$ pair, $n = 50$) under six network conditions applied from 30 s to 150 s (shaded).

too few flips a benign but poorly connected network, and too many stalls the partially synchronous path before the fallback starts. The flip is their dominant source of latency when the network degrades. The way back is not symmetric: these designs return to the fast path once the fallback commits regularly, or after a fixed number of rounds. Steelhead has no timeout count to wait or size, because the round number alone says which rule decides a slot, and k adapts from the committed DAG. The flip itself is also paid for: Ditto runs an MVBA and the Bolt-Dumbo Transformer a binary agreement, both on the critical path, at every switch. Bullshark instead opens a coin in every wave, so every party computes and verifies a coin share for every leader whether or not the protocol runs in asynchrony. Steelhead only needs a coin once every k rounds, with $k = 64$ or 128 in typical deployments.

A second family avoids the timeout by never abandoning either path, and pays for the second path instead. Abraxas [5] runs a slow protocol at all times alongside a fast one and uses it as the detector: parties fall back when a slow-chain block goes λ blocks without confirmation certificates. The $O(n^2B)$ bits per block are paid in a healthy network too. ParBFT1 [11] runs an optimistic and a pessimistic path concurrently and takes the first to finish, resolved by an asynchronous binary agreement per slot; its pessimistic path, a validated asynchronous agreement on every slot, costs a quadratic number of messages even under synchrony. Ipotane [9] does the same at the cost of one extra round per slot. TockOwl+ [21] and ParBFT2 [11] start their slow track after a hedging delay rather than immediately, and pay the slow track's messages only on the slots they hedge. Icarus [10] removes the second path instead: the chains of all parties take turns as the single optimistic path, and a switch is triggered once another

chain accumulates a backlog of uncommitted blocks. Parties observe that backlog locally and may switch at different heights, so a Byzantine agreement runs on the critical path at every switch to align them, which accounts for 40–60% of a switch. Steelhead runs no second path and wastes no block: one slot in k pays $w_a - w_s$ extra rounds, and nothing else.

Acknowledgements

This work is partially funded by Mysten Labs.

References

1. Abraham, I., Malkhi, D., Spiegelman, A.: Asymptotically optimal validated asynchronous Byzantine agreement. In: ACM Symposium on Principles of Distributed Computing (PODC) (2019)
2. Arun, B., Li, Z., Suri-Payer, F., Das, S., Spiegelman, A.: Shoal++: High throughput DAG BFT can be fast and robust! In: USENIX Symposium on Networked Systems Design and Implementation (NSDI) (2025)
3. Babel, K., Chursin, A., Danezis, G., Kichidis, A., Kokoris-Kogias, L., Koshy, A., Sonnino, A., Tian, M.: Mysticeti: Reaching the latency limits with uncertified DAGs. In: Network and Distributed System Security Symposium (NDSS) (2025)
4. Bano, S., Sonnino, A., Chursin, A., Perelman, D., Li, Z., Ching, A., Malkhi, D.: Twins: BFT systems made robust. In: International Conference on Principles of Distributed Systems (OPODIS) (2021)
5. Blum, E., Katz, J., Loss, J., Nayak, K., Ochsenreither, S.: Abraxas: Throughput-efficient hybrid asynchronous consensus. In: ACM Conference on Computer and Communications Security (CCS) (2023)
6. Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on BFT consensus. arXiv:1807.04938 (2018), https://arxiv.org/abs/1807.04938
7. Cachin, C., Kursawe, K., Shoup, V.: Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. Journal of Cryptology **18**(3) (2005)
8. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. In: USENIX Symposium on Operating Systems Design and Implementation (OSDI) (1999)
9. Dai, X., Ding, C., Jin, H., Loss, J., Ren, L.: Ipotane: Balancing the good and bad cases of asynchronous BFT. In: Network and Distributed System Security Symposium (NDSS) (2026)
10. Dai, X., Yu, Y., Duan, S., Hao, R., Xiao, J., Jin, H.: Icarus: Achieving performant asynchronous BFT with only optimistic paths. In: Network and Distributed System Security Symposium (NDSS) (2026)
11. Dai, X., Zhang, B., Jin, H., Ren, L.: ParBFT: Faster asynchronous BFT consensus with a parallel optimistic path. In: ACM Conference on Computer and Communications Security (CCS) (2023)
12. Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.: Narwhal and Tusk: A DAG-based mempool and efficient BFT consensus. In: European Conference on Computer Systems (EuroSys) (2022)
13. Danezis, G., Komatovic, J., Kokoris-Kogias, L., Sonnino, A., Zablotchi, I.: Byzantine consensus in the random asynchronous model. In: International Symposium on Distributed Computing (DISC) (2025)

14. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM **35**(2) (1988)
15. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Journal of the ACM **32**(2) (1985)
16. Gelashvili, R., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A., Xiang, Z.: Jolteon and Ditto: Network-adaptive efficient consensus with asynchronous fallback. In: Financial Cryptography and Data Security (FC) (2022), revised version: arXiv:2106.10362v4, 2024
17. Giuliari, G., Sonnino, A., Frei, M., Streun, F., Kokoris-Kogias, L., Perrig, A.: An empirical study of consensus protocols' DoS resilience. In: ACM Asia Conference on Computer and Communications Security (AsiaCCS) (2024)
18. Jovanovic, P., Kokoris-Kogias, L., Kumara, B., Sonnino, A., Tennage, P., Zablotchi, I.: Mahi-Mahi: Low-latency asynchronous BFT DAG-based consensus. In: IEEE International Conference on Distributed Computing Systems (ICDCS) (2025)
19. Keidar, I., Kokoris-Kogias, L., Naor, O., Spiegelman, A.: All you need is DAG. In: ACM Symposium on Principles of Distributed Computing (PODC) (2021)
20. Keidar, I., Naor, O., Poupko, O., Shapiro, E.: Cordial Miners: Fast and efficient consensus for every eventuality. In: International Symposium on Distributed Computing (DISC) (2023)
21. Li, M., Wu, Q., Wang, Z., Qin, B., Wei, B., Ruan, H., Xiong, S., Ding, Z.: TockOwl: Asynchronous consensus with fault and network adaptability. In: USENIX Security Symposium (USENIX Security) (2025)
22. Lu, Y., Lu, Z., Tang, Q.: Bolt-Dumbo Transformer: Asynchronous consensus as fast as the pipelined BFT. In: ACM Conference on Computer and Communications Security (CCS) (2022)
23. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The Honey Badger of BFT protocols. In: ACM Conference on Computer and Communications Security (CCS) (2016)
24. Mysticeti contributors: Mysticeti, Mahi-Mahi, and BlueBottle reference implementation. https://github.com/asonnino/mysticeti (2026)
25. Rambaud, M., Dai, X., Ding, C.: Faster asynchronous blockchain consensus. Cryptology ePrint Archive, Paper 2024/1108 (2024), https://eprint.iacr.org/2024/1108
26. Spiegelman, A., Arun, B., Gelashvili, R., Li, Z.: Shoal: Improving DAG-BFT latency and robustness. In: Financial Cryptography and Data Security (FC) (2024)
27. Spiegelman, A., Giridharan, N., Sonnino, A., Kokoris-Kogias, L.: Bullshark: DAG BFT protocols made practical. In: ACM Conference on Computer and Communications Security (CCS) (2022)
28. Tsimos, G., Kichidis, A., Sonnino, A., Kokoris-Kogias, L.: HammerHead: Leader reputation for dynamic scheduling. In: IEEE International Conference on Distributed Computing Systems (ICDCS) (2024)
29. Vander Vos, P., Sonnino, A., Tsimos, G., Jovanovic, P., Kokoris-Kogias, L.: BlueBottle: Fast and robust blockchains through subsystem specialization. arXiv:2511.15361 (2025), https://arxiv.org/abs/2511.15361
30. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: HotStuff: BFT consensus with linearity and responsiveness. In: ACM Symposium on Principles of Distributed Computing (PODC) (2019)

A Variant Without Asynchronous Slots in the Ledger

While $k > 1$, the output reading could apply R_{s} to every round so that no asynchronous slot ever enters the ledger. The rounds that carry a coin would be read as asynchronous slots by the control reading alone. The failover of Section 4.1 would still drop the period to 1, and at $k = 1$ the output reading would be R_{a} on every round as before.

The liveness argument of Theorem 3 uses only the control reading and the failover; the asynchronous slots of the output reading play no part in it. Under asynchrony nothing they commit is output anyway (Section 3.3), since the anchor search of an $\perp$ synchronous slot below stops at the next $\perp$ one. The variant would therefore save what those slots cost in a healthy network: the $w_{\mathrm{a}} - w_{\mathrm{s}}$ rounds of one slot in k, and the one round its successor waits for causal ordering, which our evaluation measures at about 1% of latency at $k = 64$ (Section 6.3).

We keep the protocol as it is for three reasons. The output reading would no longer be a single rule with a slot-dependent wavelength, so the conservativity of Theorem 5 (Appendix B) would become a statement about a reading rather than about what the protocol outputs. The two readings would no longer consider the same slots, as the same round would have a known leader in one and a coin leader in the other, so the agreement argument could no longer reuse Corollary 1 (Appendix B) and would have to be made once per reading. Nothing would be saved beyond that latency, since the coin is opened for the control reading either way. We prefer the smaller proof to saving a percent of latency, particularly because k is large in any deployment.

B Full Proofs

This appendix states every result of Section 5 and proves it. Each proof names the clause of Section 2 at the step that relies on it; every argument that needs neither clause **A4** nor clause **A5** holds for any pair of rules of the wave family, at any wavelength function. Theorem 1 also holds beyond the wave family, for any family of rules that satisfy clauses **A2** and **A3** and the uniqueness of Lemma 1, and that agree on the tie-break and on the number of links the indirect step tries (Appendix C.1): composed so that each slot is decided by the rule of its kind, they form one rule with the same properties, and Steelhead is that composite for $\{R_{\mathrm{s}}, R_{\mathrm{a}}\}$. The Lean development of Appendix C machine-checks these statements, and the arguments here follow the machine-checked ones.

Correctness rests on the interface of Section 2: a schedule of wavelengths $w : \mathbb{N} \to \{w_{\mathrm{s}}, w_{\mathrm{a}}\}$ with $w(r) \geq 2$, and two rules satisfying clauses **A1** to **A5**. Statements apply to local views of one universe of blocks, and the properties established are those of Definition 1. The two lemmas below rely only on clause **A1**, that is, on a shared committee with quorums of $n - f$ and one block per honest author and round, and on two rules of Section 2: every block references $n - f$ blocks of the round below it, and a block votes for the first block it sees from the leader; they discharge clause **A3** (Lemma 1) and clause **A2** (Lemma 2) for both

pairs; for the $3f + 1$ pair, Lemma 1 is Mahi-Mahi's certificate lemma read at the slot's own wavelength. They read a rule through two parameters. Its *support* for a candidate of the slot at round r is the set of authors whose block at the decision round $r + w(r) - 1$ carries the rule's evidence for it: for the $3f + 1$ pair a certificate, a block referencing $n - f$ votes for the candidate from round $r + w(r) - 2$, at every $w \geq 3$; for BlueBottle's pair a vote for the candidate, cast directly at $r + 1$ by the synchronous variant and through the causal history at $r + 2$ by the asynchronous one. Its *indirect threshold* is the number of supporters an anchor's causal history must hold to commit the slot indirectly, one for the $3f + 1$ pair and $n - 3f$ for BlueBottle's. Under every rule a direct commit needs $n - f$ supporters, and a direct skip $n - f$ *blames*, blocks of one round of the wave that vote for no candidate, the vote round for the $3f + 1$ pair and the decision round for BlueBottle's. Table 3, at the end of this appendix, maps each clause to each base protocol.

Lemma 1 (Support uniqueness). *For the $3f + 1$ pair, at most one candidate of a slot is ever certified, and none if the slot is directly skipped. For BlueBottle's pair, a candidate with $n - f$ supporters leaves every rival below $n - 3f$ supporters, and a directly skipped slot leaves every candidate below $n - 3f$.*

Proof. An honest author has one block per round (clause **A1**), and that block votes for at most one candidate of a slot, the first block it sees from the leader (Section 2). Two sets of authors whose blocks at one round vote for two different candidates, or one for a candidate and one for none, therefore share only Byzantine authors and together number at most $n + f$; if one set has $n - f$ authors, the other has at most $2f$.

The $3f + 1$ pair. Count at the vote round. A certified candidate has $n - f$ votes there, since a certificate references $n - f$ votes, and a directly skipped slot has $n - f$ blames there. A second certified candidate, or a certified candidate of a skipped slot, would need $n - f$ votes against the $2f$ that are left, and $n - f > 2f$ at $n \geq 3f + 1$.

BlueBottle's pair. Count at the decision round, where the supporters are the voters themselves. Against a candidate's $n - f$ supporters every rival keeps at most $2f$ supporters, and against the $n - f$ blames of a direct skip every candidate does; $2f < n - 3f$ because $n \geq 5f + 1$. □

Lemma 2 (Quorum intersection across the wave). *If a candidate of the slot at round r has $n - f$ supporters, the causal history of every block at round $r + w(r)$ or above reaches the slot's indirect threshold for it: it holds a certificate for the candidate under the $3f + 1$ pair and the supporting blocks of at least $n - 3f$ of its supporters under BlueBottle's.*

Proof. A block B at round $r + w(r)$ references $n - f$ blocks of the decision round $r + w(r) - 1$ (Section 2). Their authors and the $n - f$ supporters both lie among the n authors of one round, so at least $n - 2f$ authors are in both sets, and at least $n - 3f$ of those are honest. An honest author has one block at the round (clause **A1**), its supporting block, so B references it outright; for a Byzantine

author in the intersection the block B references may be a twin carrying no support, which is why the count stops at the honest ones. At $n \geq 3f + 1$ this is at least one certificate, and under BlueBottle's pair it is the $n - 3f$ supporters its indirect step asks. A block above round $r + w(r)$ references a block of the round below it (Section 2), so by induction on its round it reaches a block at round $r + w(r)$, and causal history is transitive. □

Theorem 1 (Agreement). *For every slot, no two honest validators hold conflicting verdicts, regardless of their local views and whether they decided directly or indirectly. At the adaptive schedule this holds for validators that have each derived the period of every interval up to the highest round of the (finite) universe of blocks.*

Proof. Fix a wavelength function w and the schedule it induces; the last paragraph extends the claim to validators running their own derived periods. Let s be the slot at round r and let two honest validators u and v hold verdicts for it. The proof is by induction on the derivation of u's verdict: the indirect step derives a verdict for s from verdicts for slots strictly above it, and the induction hypothesis is that each of those agrees with every verdict v holds for the same slot.

Two direct verdicts. Two direct commits name the same block, and a direct commit and a direct skip cannot coexist, both by Lemma 1.

One direct and one indirect verdict. This case uses no induction hypothesis, so let v be the validator that decides directly, and u decide s through its anchor A. If v directly commits s, its candidate has $n - f$ supporters at the decision round $r + w(r) - 1$. The anchor search starts at round $r + w(r)$ (Section 3.2), so A lies at that round or above, whichever rule decides A and whatever wavelength A's own slot carries. By Lemma 2 (clause **A2**) the causal history of A's leader block reaches the slot's indirect threshold for v's candidate, and by Lemma 1 no rival does, so u commits the same candidate. If instead v directly skips s, Lemma 1 (clause **A3**) leaves every candidate below the indirect threshold, so u's anchor holds none at it and u skips as well.

Two indirect verdicts. Both searches start at round $r + w(r)$, a function of r and of the schedule alone, and both anchors are commits, since an $\perp$ anchor yields no verdict. A slot u's search passes over is a skip in u's derivation, which by the induction hypothesis v does not commit, so v's anchor does not lie below u's; and u's anchor is a commit in u's derivation, which by the induction hypothesis v does not skip, so v's search does not pass over it. Both validators therefore land on the same slot A, where by the induction hypothesis they commit the same block. A block determines its own causal history, so both examine the same supporting blocks, apply the same tie-break, and reach the same verdict.

At the adaptive schedule. Let u and v each run their own derived periods, and let each have derived the period of every interval up to the highest round of the universe of blocks. By Theorem 4 the two sequences of periods agree on those intervals, so every slot either derivation reads has the same wavelength at both, and the argument above applies on one schedule. There is no circularity: the proof of Theorem 4 invokes the present theorem only at the control slots of

one scan, whose schedule is fixed by the interval's period, the period bound and the interval boundary. □

Corollary 1 (Handover). *If some honest validator directly commits the slot at round r, every honest validator whose anchor for that slot is committed commits it too, whichever rule decides the anchor, and no honest validator ever skips it.*

Proof. An honest validator whose anchor for the slot is committed holds a verdict for the slot, its direct one or else the indirect step's, which returns a commit or a skip, and by Theorem 1 that verdict agrees with the direct commit; the same theorem excludes a skip at every honest validator. □

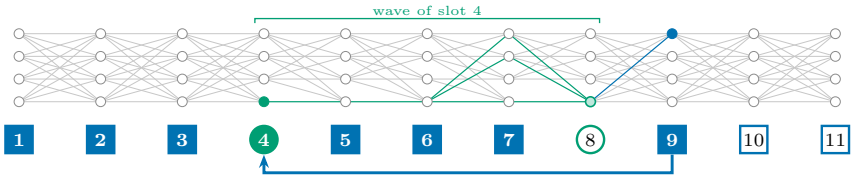
Where the two rules meet. The case of one direct and one indirect verdict in the proof of Theorem 1, whose commit half Corollary 1 states, is the only step where the two rules interact: an anchor committed by one rule completes a direct commit of the other. Two consequences are used later. A period change never has to pause the protocol to finish pending slots: the anchors of the next period decide the slots left $\perp$ under the previous one. And the output reading and the control reading of Section 4.3 read a direct verdict identically, so their verdicts on a shared slot can differ only where both are indirect.

Why the floor is the slot's own wavelength. Lemma 2 reads the wavelength of the slot, $w(r)$, not that of the slot at the round of the block whose causal history it reads; this is what clause **A2** asks of a rule, and it sets the anchor floor at $r + w(r)$. Neither $r + 1$ nor $r + w(\text{anchor})$ will do. Consider an asynchronous slot L at round r with $w_\text{a} = 5$: its certificates sit at round $r + 4$, and a validator holding $n - f$ of them commits L directly. An anchor below round $r + w_\text{a} - 1$ has its entire causal history below the round those certificates sit at, so it holds none of them and its indirect step skips L, contradicting the direct commit; an anchor at round $r + w_\text{a} - 1$ itself is one block of that round, which need not be among the certifiers. Only from round $r + w_\text{a}$ on does Lemma 2 guarantee a certificate. A synchronous slot P at round $r + 1$ is the extreme case, its blocks referencing only round-r blocks, but a synchronous slot at round $r + w_\text{s}$ is no better. Fig. 6 draws both verdicts on one DAG: the rule's floor commits the slot through the synchronous slot at round 9, while a floor read from the anchor's own wavelength lands on the synchronous slot at round 7 and skips it.

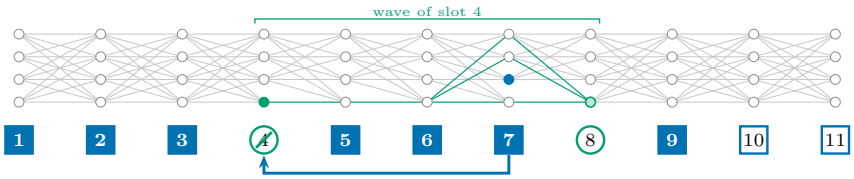
Corollary 2 (Total order and integrity). *Slots are output in round order, and each block is output at most once.*

Proof. Total order. Output proceeds in slot order and stops at the first $\perp$ slot (Section 2), so what a validator has output is the sequence of commit verdicts over a settled prefix of slots, and settling further slots only appends. By Theorem 1 two validators agree on the verdict of every slot both have settled, so their outputs agree as far as both reach and neither reorders what it already released.

Integrity. Both the direct and the indirect step commit only a candidate of the slot under decision, that is, a block of the slot's round authored by that



(a) The floor of Section 3.2, $4 + w_a = 9$. The search lands on slot 9, and the edge from its leader block to the certificate at round 8 is why slot 4 commits.



(b) A floor read from the anchor's own wavelength, $4 + w_s = 7$. The search lands on slot 7, whose causal history lies entirely below round 8, so no edge reaches the certificate and slot 4 is skipped.

Fig. 6. One DAG read with two anchor floors ($n = 4$, $w_s = 3$, $w_a = 5$, $k = 4$; the DAG of Fig. 2). Highlighted in the DAG are the leader block of slot 4 at round 4, the one block that certifies it at round 8, every route between the two, of which the three top edges are the round-7 votes that make that block a certificate, and the leader block of the anchor the search lands on. The routes funnel through the leader's own chain, since no other author's block at rounds 5 or 6 references it, which is why slot 4 has exactly one certificate. The bracket marks the wave of slot 4, and the arrow runs from the anchor to the slot it decides. In the strip below each panel, squares are synchronous slots and circles asynchronous slots, filled on a commit and struck on a skip. One view yields both verdicts. No validator commits slot 4 directly here, but the low floor can skip a slot that another validator has committed directly, which the floor of the rule excludes (Lemma 2).

slot's leader. Steelhead places one slot at each round, so a committed leader block enters the output only at the slot of its own round, which carries a single verdict. Every other block enters with the first committed leader whose causal history holds it, and a later commit does not output it again. □

Theorem 2 (Liveness under partial synchrony). *After GST, with bounded leader timeouts, every synchronous slot whose leader is honest and whose successor round carries a leader wait is decided as a commit by the direct rule within w_s rounds, and every asynchronous slot whose elected leader is honest (or crashed) is decided directly within w_a rounds, and partial dissemination alone does not defer the decision of an asynchronous slot. Any slot the direct rule leaves $\perp$, such as one whose leader equivocates, is decided by its anchor once some slot above its floor commits and every slot in between is decided.*

Proof. Synchronous slots. After GST and under the pacing of Section 3.1, a synchronous slot whose leader is honest and whose successor round carries a leader wait is directly committed within w_s rounds. This is clause **A4**.

Asynchronous slots with an honest or crashed leader. After GST, under the reference condition of Appendix C.2, a block held by one honest validator is referenced, transitively, by every honest block two rounds later. An honest leader's block therefore lies in the causal history of every honest block from round $r + 2$ on, the honest voters vote for it, and every honest block at the certify round

references all of them and so certifies it: the slot is directly committed within w_a rounds, at $w_a \geq 4$. BlueBottle's asynchronous variant needs no certify round: its voters two rounds up reach the leader's block through their causal history, so the slot commits at $w_a = 3$. A crashed or silent leader collects $n - f$ blames at the vote round and is directly skipped. Partial dissemination therefore does not defer the decision: it suffices that one honest validator holds the leader block in time, and the leader may be Byzantine as long as it did not equivocate.

The wait above an $\perp$ slot. What remains is a slot the direct rule leaves $\perp$, for instance one whose leader equivocates or a synchronous slot whose successor round carries no leader wait, and its wait is set by the leader schedule rather than by the decision rule. Consider the *floor chain* of the slot: hop from the slot to the first slot at or above its floor that the validator does not skip, then hop again from there. Crashed and silent leaders are skipped, so every landing is a slot whose leader did propose, and a landing the first two paragraphs cover, an honestly led asynchronous slot or an honestly led synchronous slot whose successor round carries a leader wait, commits directly. Going back down the chain, a landing whose own successor landing commits is decided by the indirect step and not skipped, hence commits in turn and anchors the landing below it. It is therefore enough that the chain reach a committed landing; if some slot above the floor commits and every slot in between is decided, the first landing is already a commit, which is the second claim. □

Round counts. The wait above an $\perp$ slot has a bound at the two ends of the dial. At $k = \infty$, under a round-robin schedule with $w_s f < n$, no two landings share a residue class of the rotation. Were two landings at one residue, they would bound a whole number of cycles; an honestly led round commits and so is never one of the skips a hop passes over, hence honest leaders could sit only in the $w_s - 1$ rounds between a landing and its own floor, and counting the f Byzantine residues against the cycles gives $n \leq w_s f$. The landings' leaders being distinct, the $b \leq f$ Byzantine validators occupy at most b of them and one of the first $b + 1$ landings is honestly led. An honestly led round lies within f rounds above any floor, so a hop climbs at most $w_s + f$ rounds, and every decision round the argument reads lies within $(b + 1)(w_s + f)$ rounds of the slot.

At $k = 1$ the wait is the first run of w_a coin rounds naming directly committed candidates, since such a run decides every slot below it (the proof of Theorem 3). Cut the coin rounds into blocks of w_a, and call a block good if every one of its coins names a directly committed candidate. Conditioning on the coins already revealed, clause **A5** gives each round probability at least p, so a block is good with probability at least p^{w_a}, and the expected number of blocks up to and including the first good one is at most $1/p^{w_a}$. No bound per hop of the search holds under the coin: the coin that commits an anchor above a pending slot both skips that slot and moves the landing, so the landings are not independent draws, and a two-hop bound of $(b/n)^2$ fails on a concrete DAG (Appendix C). In between, the rotation leads only the synchronous slots and the coin the rest, and the residue count needs slack in the committee that $n = 3f + 1$ does not have.

Theorem 3 (Liveness under asynchrony). *Assume* $\texttt{maxPeriod} \leq \texttt{interval}$, $w_a \leq \texttt{interval}$ *and* $w_s \leq w_a$, *that every round above* $2\,\texttt{interval}$ *is populated (clause* **A1***), and that the counting bound of clause* **A5** *holds at every round with* $p > 0$. *Under asynchrony, with probability* 1, *above every interval some interval finds a pivot (Section 4.3), and every slot is eventually decided. While a slot is* $\perp$, *every pivot at least two intervals above it hands the next interval* $k = 1$ *through the failover.*

Proof. Good blocks. Fix a slot s and take blocks of $w_a \cdot \texttt{maxPeriod}$ consecutive rounds, each opening an interval: the first at the second interval above that of s, and one more every $\lceil w_a \cdot \texttt{maxPeriod}/\texttt{interval} \rceil$-th interval, so that no two overlap. Read the coin as a draw at every round, of which the protocol reveals those at its asynchronous slots. A block is *good* if the coin names a directly committed candidate at each of its rounds. Conditioning on the coins already revealed, clause **A5** gives each round probability at least p, so a block is good with probability at least $p^{w_a \cdot \texttt{maxPeriod}}$ whatever the earlier blocks did.

Some interval finds a pivot. A good block holds w_a consecutive multiples of `maxPeriod`, which every scan of an earlier interval reads as consecutive control slots above its boundary (Section 4.3). They commit directly and, as the run of the next paragraph does at the control reading's fixed wavelength w_a, decide every control slot below them, so every earlier scan closes and the period of the interval the block opens is derived. That period is at most $\texttt{maxPeriod} \leq \texttt{interval}$, so the interval's first control slot lies among the block's first `maxPeriod` rounds; it commits directly and is the interval's pivot. Good blocks occur with probability 1, by the bound at the end of this proof.

Every slot is eventually decided. Let a good block give an interval at least two past that of s its pivot, and let a later good block, which settles every scan below its own interval, supply in its first w_a rounds, which lie in that interval since $w_a \leq \texttt{interval}$, a run of coins naming directly committed candidates. If s is still $\perp$, the agreed output's last commit lies below s and the pivot more than `interval` rounds above s, so the failover hands the next interval $k = 1$, and every later interval up to the run keeps it while s stays $\perp$. The run's slots are then asynchronous slots that commit directly, and since no slot's wavelength exceeds w_a, the run decides every slot below it: the highest $\perp$ slot below the run has its floor at or below the run's last slot, the first slot at or above that floor that is not a skip is a commit, and the indirect step returns a verdict. Two good blocks, one in each half of M blocks, occur except with probability at most $2(1 - p^{w_a \cdot \texttt{maxPeriod}})^{\lfloor M/2 \rfloor}$, which vanishes. The slots being countably many, with probability 1 every slot is eventually decided, with the run's commits above it, and by Corollary 2 the ledger grows with them. □

Period updates while output is stalled. A direct commit reaches a lower slot only through a decided stretch: from the lower slot's floor up to the first commit, every slot must be decided, because the anchor search stops at an $\perp$ one. Suppose no synchronous slot is ever certified or directly skipped, as an adversary can arrange under asynchrony (Section 3.3), so that no synchronous slot commits.

At $k \geq w_s$ the stretch above a slot at a round r with $r \bmod k = k - 1$ then ends at an asynchronous slot above round $r + k$ and so holds the slot at round $r + k$, which is in the same position; no such slot is therefore ever decided, since each needs the one a period up decided first. Take $k = 4$, $w_s = 3$, and $w_a = 5$. The synchronous slot at round 3 searches from round 6, and since no synchronous slot commits, only a committed asynchronous slot at round 8 or above can decide it; the search reaches round 8 only if slots 6 and 7 are skipped (in Fig. 3 it waits at slot 6), and slot 7 is in the same position one period up, and so on. Meanwhile the asynchronous slots at 8 and 12 commit directly and are never output. Because a directly committed asynchronous slot is read identically in both readings, the two readings differ only when an asynchronous slot is not directly committed. Suppose the direct rule leaves the asynchronous slot at round 4 $\perp$. In the output reading, its anchor search begins at round 9 and, since no synchronous slot commits, only a committed asynchronous slot at round 12 or above can decide it, which needs slot 11 skipped; slot 11 is never decided, so slot 4 stays $\perp$. In the control reading, whose slots above round 8 are the multiples of `maxPeriod` $= 4$ as in Fig. 3, the search visits only control slots and reaches slot 12, whose causal history decides it. The pivot of an interval, the slot whose causal history determines the next period, must be identical for validators that observed a direct commit and those that did not. Only the control reading lets the latter resolve it, which is why the period update evaluates R_a at the control slots rather than at the output.

The counting lemma underpinning the asynchronous bound requires $w_a \geq 4$: at $w_a = 4$ it guarantees only one directly committed candidate, so a commit has probability at least $1/n$, and at $w_a = 5$ the stronger $n - f - b$ bound holds (both derived under "Instantiating the theorems" below). The $3f + 1$ pair inherits this trade-off from Mahi-Mahi; BlueBottle's pair does not, its counting lemma holding at $w_a = 3$. The bound also shows what the sparse control slots cost. An interval contains $c = \texttt{interval}/k$ control slots, with at least two slots when $k = \texttt{maxPeriod}$. Since an interval retains its period when all of its control slots are skipped, counting direct commits alone, a scan finds its pivot within an expected $1/(1 - (1 - p)^c)$ intervals, which is approximately 1.8 intervals for $p = 1/3$ and $c = 2$, but approximately $n/2$ intervals for the $p = 1/n$ of $w_a = 4$. A deployment at $w_a = 4$ should therefore keep the ratio $\texttt{interval}/\texttt{maxPeriod}$ well above two.

Theorem 4 (Agreement of the period). *Under any deterministic update rule, two honest validators that derive the period of interval j derive the same one.*

Proof. By induction on configurations. Configuration j is interval j, the rounds $j\,\texttt{interval} + 1$ through $(j + 1)\,\texttt{interval}$ (Section 4), and its state is the period k_j that the scan of interval $j - 1$ fixed, the agreed output's next slot, and the round of its last committed leader. Interval 0 runs at the initial period. Assume that any two honest validators holding a state for interval j hold the same one.

The scan's slots agree. The control slots of the scan of interval j are the multiples of k_j up to the boundary $(j + 1)\,\texttt{interval}$ and the multiples of `maxPeriod`

above it (Section 4.3). They are a function of k_j, of `maxPeriod` and of the boundary, all agreed, and of nothing that a view holds.

The scan's verdicts agree. The control reading is R_{a} on the sub-schedule those slots name, at the fixed wavelength w_{a}. Theorem 1 applies to it unchanged, its proof reading only Lemmas 1 and 2 and a common anchor search, none of which mentions the output's slots. Each scan has its own control slots, and verdicts of different scans are never compared.

The pivot agrees. The pivot is the first control slot of interval j whose control verdict is a commit, every earlier one being a skip. Both components are agreed by the previous step, and so is the case of no pivot, in which the state carries over unchanged.

The next state agrees. The agreed output is the output rule applied to the pivot's causal history, continued from the state's cursor. A block determines its own causal history, so both validators read the same blocks and, the output rule being deterministic, derive the same verdicts, and they move the cursor and the last commit alike. Every verdict that advance reads lies at or below the pivot's round, where the periods are agreed, so the induction closes at each interval. The failover then compares the round of the agreed output's last commit with the pivot's round, both agreed, and the update rule is a deterministic function of the pivot block, the window it determines, and the current period. □

Theorem 5 (Conservativity). *With a constant rule that never changes k, setting $k = 1$ yields exactly the verdicts of R_{a} and setting $k = \infty$ exactly those of R_{s} (Mahi-Mahi's and Mysticeti's, in the $3f + 1$ pair), on a given DAG.*

Proof. At $k = 1$ every round satisfies $r \bmod k = 0$, so $w(r) = w_{\mathrm{a}}$ everywhere, every slot is an asynchronous slot with a coin-elected leader, and both the direct and the indirect step are R_{a}'s at w_{a}. At $k = \infty$ no round satisfies it, so $w(r) = w_{\mathrm{s}}$ everywhere and every slot is a synchronous slot. In both cases the anchor floor $r + w(r)$ collapses to the constant floor of the base rule, so the anchor searches coincide as well, and the verdicts agree slot by slot. □

Lemma 3 (The replay's direct-commit weight). *For the $3f + 1$ pair at $w_{\mathrm{a}} = 5$ and for BlueBottle's pair, let r be a round of a window that holds the blocks of a quorum, honest for BlueBottle's pair, at the rounds the counting lemma reads for r: the vote and decision rounds of r's wave for Mahi-Mahi, the two rounds above r for BlueBottle's asynchronous variant. Then c_r/n is at least the counting fraction of R_{a}'s clause **A5** under any message scheduling ($c_r \geq n - f - b$ for Mahi-Mahi).*

Proof. Read the window as a record of its own: its support is that of the universe restricted to it, and by hypothesis a quorum populates within it the rounds the counting lemma reads for r. The counting property of clause **A5** then applies to every candidate at once and names at least pn authors whose blocks carry direct-commit evidence inside the window, which for Mahi-Mahi at $w_{\mathrm{a}} = 5$ is $c_r \geq n - f - b$ under any scheduling. By construction c_r/n is the share of the n candidate leaders that the replay marks committed at round r, which is the probability that a uniform coin names a directly committed leader there. □

Corollary 3 (Atomic broadcast). Steelhead *implements atomic broadcast (Definition 1) for any pair of rules satisfying clauses **A1** to **A5**.*

Proof. Agreement, total order, and integrity. By Theorem 1 and Corollary 2, honest validators agree on every slot both have settled, output settled slots in order, and output each block at one slot only, so a block one of them delivers lies in the output of every other whose settled prefix reaches as far, and with probability 1 every honest validator's settled prefix eventually reaches as far (Theorems 2 and 3). Every output block is a block of the DAG and so carries its author's signature (Section 2).

Validity. An honest block lies in the causal history of every honest block from some later round on: after GST the round above its own, and under asynchrony the round by which the reference rule of clause **A1** has spread it to every honest validator. Every block above that round, whatever its author, references $n - f$ blocks of the round below (Section 2), at least one of them honest, and so holds it too. The first slot committed above that round therefore delivers it, whichever leader it has, once every slot below it is decided. With probability 1 every slot is decided (Theorems 2 and 3), and a slot above that round commits: after GST an honestly led one (Theorem 2), and under asynchrony a slot of the run in the proof of Theorem 3. □

Parameters and guards. The loop of Section 4.1 defers six details here. None of them enters a safety argument, and Theorem 4 holds whatever they are set to, since each is a function of round numbers and of agreed state.

The gating rule. A slot is evaluated once its interval's period is known. This needs no separate mechanism: a validator that has not closed the scan of interval j derives no state for interval $j+1$, so it has no wavelength for those rounds and decides nothing there. *The one-interval lag.* A period computed from the scan of interval j applies from interval $j+1$ on, never to interval j itself. Without the lag the period of a round would be needed in order to decide the very slots whose verdicts fix it.

The bounds on `interval`. Two are needed, and the first does not imply the second. $\texttt{interval} \geq 2\,\texttt{maxPeriod}$ makes every interval, and every window of a pivot at round `interval` or above, hold at least two asynchronous slots of every candidate period. $\texttt{interval} \geq \texttt{maxPeriod} + w_{\mathrm{a}} - 2$ makes a window of $\texttt{interval} + 1$ rounds hold the *decision* round of at least one asynchronous slot of every candidate; under the first bound alone, a window resolves such a slot at some pivots and none at others, and the replay then scores a candidate on no evidence. The garbage-collection horizon must retain at least the last $\texttt{interval} + 1$ rounds, which is what a window reads.

The warm-up interval. Interval 0 hands interval 1 its own period whatever the replay answers, because its window holds the start-up rounds, where no rule has had a full wave to commit anything. The failover cannot fire there either: the pivot lies at round `interval` or below and the agreed output's last commit at round 0 or above, so the test $\ell + \texttt{interval} < \mathrm{round}(A)$ of Algorithm 3 fails.

The interval scan reads R_a at the control slots (Section 4.3): the asynchronous slots of interval j and, above its boundary, the multiples of `maxPeriod`. Evidence is the support of Lemma 1; $\beta(w)$ is the offset of the rule's blame round, $w-2$ for the $3f+1$ pair and $w-1$ for BlueBottle's.

procedure UpdatePeriod(j) ▷ scan interval j's control slots upward; wait at undecided
  $A \leftarrow$ the first control slot with a commit verdict in the scan ▷ earlier slots are skipped
  if there is none: **return** ▷ no commit keeps k
  $W \leftarrow$ GetSubDag(A, `interval`) ▷ causal history of A at rounds round(A) − `interval` and above
  extend the agreed output from its last slot: ▷ identical at every validator
    TryCommit on the causal history of A
  $\ell \leftarrow$ round of the agreed output's last commit, or 0 if none
  if $\ell +$ `interval` $<$ round(A): $k \leftarrow 1$; **return** ▷ failover; startup grace through round `interval`
  for $k' \in K$: $L[k'] \leftarrow$ Replay(W, k') ▷ expected output delay under period k'
  $k^\star \leftarrow \arg\min_{k' \in K} L[k']$ ▷ ties keep k, then favor the larger candidate
  if $L[k^\star] < (1-\epsilon) \cdot L[k]$: $k \leftarrow k^\star$ ▷ hysteresis; applies above round $(j+1)$ `interval`

procedure ProbeRate(W, k') ▷ canary rounds that k' replays as synchronous slots
  $P \leftarrow \{r \in W : r \bmod k' \neq 0,\ r \bmod$ `canary` $= 0,\ r + w_s - 1 \leq \text{top}(W)\}$
  $s \leftarrow |\{r \in P : W$ holds $n-f$ supporters of GetLeader(r) at $r + w_s - 1\}|$
  return $(s, |P|)$ ▷ successes, probes

procedure Replay(W, k') ▷ sum over W of the expected delay from a round to its output
  $(s, t) \leftarrow$ ProbeRate(W, k')
  for each round r of W, from the top downward: ▷ pass 1: decide every slot under k'
    $w \leftarrow w_a$ **if** $r \bmod k' = 0$ **else** w_s
    if $r + w - 1 > \text{top}(W)$: $dec_r \leftarrow com_r \leftarrow \text{top}(W)$; **continue** ▷ same penalty for every k'
    if $w = w_s$ **and** $r \bmod$ `canary` $\neq 0$ **and** $t > 0$: ▷ unprobed synchronous slot
      $dec_r \leftarrow \big(s\,(r + w_s - 1) + (t-s)(r + \beta(w_s))\big)/t$
      $com_r \leftarrow \big(s\,(r + w_s - 1) + (t-s)\,\text{top}(W)\big)/t$; **continue** ▷ commits w.p. s/t, else skips
    $a \leftarrow$ FindAnchor(r, w) ▷ earliest slot at round $\geq r + w$ with $com_a < \text{top}(W)$
    if $w = w_s$: $cand \leftarrow$ {GetLeader(r)} ▷ known leader: one candidate, exact
    else: $cand \leftarrow$ all n authors ▷ hidden leader: every author is a candidate, weight $1/n$ each
    for each $v \in cand$, with B the block of v at round r in W (possibly none):
      if $n-f$ blocks at $r + \beta(w)$ blame B: $dec_v \leftarrow r + \beta(w)$; $com_v \leftarrow \text{top}(W)$ ▷ direct skip
      else if $n-f$ blocks at $r + w - 1$ support B: $dec_v \leftarrow com_v \leftarrow r + w - 1$ ▷ direct commit
      else: $dec_v \leftarrow dec_a$ ▷ indirect, by the anchor
        $com_v \leftarrow com_a$ **if** W holds the indirect threshold of supporters of B **else** top(W)
    $dec_r \leftarrow \text{mean}_{v \in cand}\, dec_v$; $com_r \leftarrow \text{mean}_{v \in cand}\, com_v$ ▷ expectation under a uniform coin
  for each round r of W, from the top downward: ▷ pass 2: output at the first commit $\geq r$
    $C_r \leftarrow \min(com_r, C_{r+1})$
  for each round r of W, from the bottom upward: ▷ pass 3: wait for every lower decision
    $G_r \leftarrow \max(G_{r-1}, dec_{r-1})$
  return $\sum_r \big(\max(C_r, G_r) - r\big)$

Algorithm 3: Period update on the committed DAG

Hysteresis and ties. The selection leaves the current period only for a candidate scoring below $(1-\epsilon)$ times the current period's, and when it does leave, it takes the best candidate. The best candidate scores no worse than the current period and no worse than any candidate, is the current period whenever that one scores as well, and is otherwise the largest candidate attaining the best score, in whatever order the candidates are listed. *The candidate set.* $K = \{1, 2, 4, \ldots, \texttt{maxPeriod}\}$, the powers of two up to the bound. Every candidate divides `maxPeriod`, which is what makes the multiples of `maxPeriod` asynchronous slots under every candidate period, hence defined and coin-carrying whatever the scan decides (Section 4.3), and what keeps every derived period a divisor of the bound. Because `canary` is odd it is coprime to every candidate, so a window holding two canary rounds whose decision round it retains holds a probe for every candidate of at least two: two consecutive multiples of the canary spacing cannot both be multiples of the period.

The replay, in full. Algorithm 3 states the update as the implementation runs it. The scan walks the control slots of the interval upward, waits at an $\perp$ control

verdict, takes the first commit as the pivot A, and keeps the period if every control slot is a skip. The control slots are fixed by rule and not read off the coin shares a validator holds: sets read from two views can differ, and two validators can then take their pivot at different rounds of one DAG (Appendix C). The scan then extends the agreed output over A's causal history and applies the failover before consulting the score, so that a stalled output always gets $k = 1$, whatever the score. REPLAY scores a candidate k' by the expected delay from a round of the window to the output of its blocks, in three passes: the first decides every slot under k', exactly for a probed synchronous slot, at the probes' success rate for an unprobed one, and averaged over the n candidate leaders for an asynchronous slot; the second gives each round the first commit at or above it; the third gates that on the decision of every lower slot, which accounts for head-of-line blocking. The cost is $O(|K| \cdot |W|)$ for $|K|$ candidates over a window of $|W|$ blocks.

Two approximations remain, both deterministic and identical for every candidate. Where the rule would decide a slot from the causal history of the anchor that the candidate would have used, the replay asks only whether the slot's indirect threshold of supporters lies anywhere in the window, the window being the pivot's own causal history and so standing in for the anchor's. And a slot that a candidate never commits inside the window is charged the window's top round, so the rounds it holds back are deferred there rather than left undefined. At the wavelengths of Lemma 3, the first pass cannot be starved on the asynchronous side, provided the window holds a quorum's blocks, honest for BlueBottle's pair, at the rounds the counting lemma reads: the counting property counts support on the DAG, while the replay reads the window. On the synchronous side the probes carry the estimate. A probe succeeds only on a support quorum the DAG actually holds (Section 4.2), so the adversary can suppress evidence of the partially synchronous rule but never manufacture it; what it can do is serve the canary rounds' leaders alone, which lifts the rate the replay extends to the unprobed synchronous slots above what those slots would have shown. And Lemma 3 bounds a rate, not a latency: it says how often a uniformly chosen leader would have committed directly, not that the replay selects the fastest period. The failover of Section 4.1 covers that gap, which is why a window in which only the canaries were served goes to the failover and not to the score.

Instantiating the theorems. Table 3 maps each interface clause to the result that discharges it for each of the four base protocols, clause **A4** being Corollary 2 of BlueBottle for its synchronous variant; the control verdicts rely on clause **A5** at the control slots. The counting fraction p of clause **A5** is bounded per rule. For Mahi-Mahi at $w_a = 5$ the counting lemma (Lemmas 12 and 13 of that paper) gives $n - f$ directly committable blocks per populated round at $n = 3f + 1$; counting only the $n - f - b$ honest ones among them bounds p below by $(n - f - b)/n$, at least $1/3$ at $n \geq 3f + 1$; at $w_a = 4$ it promises only one committed candidate per populated round (its Lemma 15), so $p \geq 1/n$. BlueBottle's asynchronous variant counts at $w_a = 3$ (Lemmas 28 to 31 of that paper): on a quorum of honest validators populating the two rounds above a round, at least $n - 3f$ honest blocks of that round are directly committed, so $p \geq (n - 3f)/n$.

Table 3. Discharge table mapping interface clauses to their discharging lemma or base-paper result. In Lean (Appendix C), clause **A1** is the model, except for block production and the reference rule, which the liveness results and validity take as hypotheses (Appendix C.2); clauses **A2** and **A3** are proved for both pairs; and clause **A4** and the counting floor of clause **A5** are proved on a given DAG for the base protocols whose cells in those rows cite a base paper, and enter the generic results as hypotheses.

Clause	Mysticeti ($w_s = 3$) ($n \geq 3f+1$)	Mahi-Mahi ($w_a \in \{4,5\}$) ($n \geq 3f+1$)	BlueBottle (sync.) ($w_s = 2$) ($n \geq 5f+1$)	BlueBottle (async.) ($w_a = 3$) ($n \geq 5f+1$)
A1 (substrate)	shared $3f+1$ DAG		shared $5f+1$ DAG	
A2 (waved evidence)	Lemma 2	Lemma 2	Lemma 2	Lemma 2
A3 (skips exclude)	Lemma 1	Lemma 1	Lemma 1	Lemma 1
A4 (synchronous)	base paper [3]	–	base paper [29]	–
A5 (asynchronous)	–	base paper [18]	–	base paper [29]

C Lean Formalization

Every statement of Appendix B is machine-checked in Lean 4, on the open-source formalization of Mysticeti-family rules,[6] with the base protocols' clauses as hypotheses (Appendix C.2). The development contains no `sorry` and uses no axioms beyond Lean's three standard ones: propositional extensionality, quotient soundness, and choice. Every assumption is a hypothesis of the theorem that needs it, so each result is either proved outright on the DAG model or proved under the hypotheses that Appendix C.2 lists.

The development spans about 21k lines, partitioned so that a human reader audits only definitions and theorem statements, about 7k lines. The proofs, about 10k lines, are AI-generated, checked by the kernel and need no reading. A further 3k lines of executable witnesses evaluate every definition on concrete DAGs before any theorem uses it, so that a definition admitting no instance fails the build instead of making the theorems above it hold for free. A script enforces the partition: statement files are proof-free, model files carry no theorems, and no synchrony hypothesis reaches the rule's own properties.

The witnesses include the DAGs on which a claim fails: a fully connected DAG on which a floor read from the anchor's wavelength derives both a direct commit and an anchored skip of one slot (Fig. 6 draws the skip on a sparser DAG); a rotating-leader family, at every horizon, on which the selector of Algorithm 3 keeps $k = 4$ and no block above round 2 is ever output while the asynchronous slots commit on schedule, together with the recovery once the failover hands the period to 1 and a run of the coin decides the stalled slot; a DAG on which two hops of the anchor search are both led by the one Byzantine validator with probability at least $19/256$, against the $(b/n)^2 = 1/16$ that a per-hop bound would claim; and two views of one DAG that, reading the control slots off the coin shares they hold rather than by rule, take their pivot at different rounds.

C.1 The Model and the Rule

The model is the DAG and a clock that stamps each block: blocks carry a round, an author and references, a view is a set of blocks closed under reference, and a

[6] `https://github.com/gdanezis/lean-dag` (commit `4228bc5`)

schedule assigns each slot a round, a leader and a kind. There are no validators and no messages; Appendix C.2 lists the hypotheses that stand in for them.

The wavelength function is a parameter. One map sends a kind to its wave, w_{s} at the synchronous kind and w_{a} at the asynchronous one, and a second sends a round to its kind, asynchronous at every k-th round and synchronous elsewhere. For the $3f+1$ pair, safety holds at any wavelength function of at least two rounds per kind, the paper's $w(r)$ being one; the liveness results bound it as each pair needs. The $3f+1$ pair's instances of Theorem 2 ask $w \geq 3$: at $w = 2$ the vote round $r + w - 2$ is the slot's own round, where no block but the candidate itself votes for it, so none is certified and nothing commits. BlueBottle's pair reads no certify round, so $w_{\mathrm{s}} = 2$ suffices for it.

For the $3f+1$ pair, Steelhead is one anchored rule whose data at a slot of kind κ are Mahi-Mahi's at wave $w(\kappa)$: the certificate-quorum direct commit, the slot blame as a direct skip, one certified link, and a wave offset of $w(\kappa) - 1$, so that an anchor sits at round $r + w(\kappa)$ or above. At a constant wavelength the rule *is* Mahi-Mahi's at that wave, by definition, and at the constant wave 3 its verdicts are Mysticeti's by a proved equivalence (Theorem 5).

The interface of Section 2 is formalized as a composite: given a family of anchored rules, one per kind, a slot of kind κ takes its wave offset, direct predicates and links from the rule of that kind, while the link count and tie-break, which the relation reads without reference to a slot, come from the family. If every rule of the family satisfies the laws, which include clauses **A2** and **A3** and the uniqueness of Lemma 1 in the relation's terms, and the family agrees on those two parameters, then the composite satisfies them, each law at a slot being that slot's own rule's and the anchor's rule never entering. Theorem 1 is that statement, and both pairs the paper instantiates are instances of it: the $3f+1$ one by definition, since its two rules are one rule read at two waves, and BlueBottle's from the two variants' own laws, which agree on the link count and the tie-break.

The period is a configuration-sequence model written for this paper. Rounds are grouped into intervals; the control slots of a scan are the multiples of the interval's period up to its boundary and the multiples of `maxPeriod` above it; the state a scan carries is the period, the agreed output's next slot and the round of its last committed leader; and the update rule is any function of the pivot block and the current period. The derivation is a relation, so the gating rule of Section 4.1 holds by construction: a state for interval $j+1$ exists only once the scan of interval j has closed, with a pivot or with none. Waiting is the absence of a derivation, and a validator that has not closed a scan has no wavelength for the rounds above it.

What the arc reuses from the existing formalization is the DAG core, the decision rule at a fixed wavelength, the coin modeled by its effect, and the counting lemma. What is new for this paper is the decision rule with a per-slot wavelength, the anchor floor at $r + w(r)$, agreement across slot kinds at each validator's own derived schedule, the control verdicts as R_{a} on the sub-schedule of one scan's control slots, the coin as a uniform distribution against an adversary that adapts to every earlier draw, the handover corollary as the one point where

the two rules interact, the configuration-sequence model of the period with its agreement theorem for any deterministic update rule, and the liveness results, all but two stated for any pair of lawful rules, read through a clause per rule for the synchronous commit, the silent leader's skip and the counting floor.

C.2 What the Model Assumes

Because the model has no execution, five of the paper's assumptions enter as hypotheses rather than as derived facts. Three of them the base protocols already make, in their own formalizations as much as in their papers. The other two are new to the formalization: one states how the coin is drawn and what the DAG may depend on, the other the round by which clause **A1**'s reference rule has spread a block.

Inherited from the base protocols. *Block production and delivery* enter as populated waves, wherever a liveness result derives a commit from the DAG: a claim asks that a quorum's blocks occupy the rounds its argument reads. Mahi-Mahi's counting lemma takes the same hypothesis and no network hypothesis besides, so nothing is added to it here. *Partial synchrony* enters as a condition on references rather than as a clock, namely that from some round on every block of the reliable quorum references every such block of the round below. That condition can be guaranteed only after GST and is what Mysticeti's post-GST liveness becomes when stated on references alone; the honest-leader commit of Theorem 2, and hence clause **A4**, is proved under it. The development also checks that commit under two pacing disciplines, both with a timeout that comes to clear $2\Delta + \text{proc}$ from GST on. The timed one establishes the condition. The reactive one does without it and asks instead for the waits of Section 3.1: the leader wait at the rounds that carry one and, at wave 3, the vote wait, in which a certifier either references the votes or waits its timeout out. *A fair leader schedule* is what Mysticeti's liveness rests on as well. The round count after Theorem 2 needs the sharper form $w_s\,(n - |T|) < n$ for the reliable quorum T, because the hops of the floor chain interact: under it the chain reaches a reliably led landing within $n - |T|$ hops, within b once the remaining faulty validators have crashed, and hence within $(b+1)(w_s + f)$ rounds. At a finite period the model checks the count too, under $w_s\,(n - |T|) + w_s\lceil n/k \rceil < n$, with $n - 1$ rounds to an honestly led synchronous slot in place of f and every asynchronous slot above the floor decided; with $|T| = n - f$ at $w_s = 3$ and $n = 3f + 1$ no period meets that condition.

New for this paper. No base formalization states either of the two new hypotheses. *The coin's unpredictability* (clause **A5**) enters as uniform draws, and the per-round floor of committed candidates that clause **A5** supplies is a hypothesis the model does not derive: the floor may depend only on the draws already revealed, and the adversary may rebuild the DAG at every draw, so long as it does not shrink that floor with the round's own coin. Mahi-Mahi's formalization models its coin by its effect alone, as a clause saying that the leader keeps landing on committed candidates, and leaves uniformity in prose;

the bounds of Theorem 3 and of the round counts after Theorem 2 are the first that need the floor written down, which is why clause **A5** states it. *Validity under asynchrony* rests on the consequence of clause **A1**'s reference rule, that a block every honest validator holds by some round stays in every later honest block's causal history. The substrate's references sit one round back, so the rule and the round it yields lie outside the model, and the claim takes that round as its hypothesis.

Deriving block production, the reactive half of the pacing, and the reference rule from the assumptions themselves needs an execution model; the proofs of Appendix B take the same steps informally. Under these hypotheses the model does check the quantitative claims: the expected wait above an asynchronous slot floor at $k = 1$, as $1/p^{w_a}$ blocks of w_a coin rounds at both waves the pair accepts, and the expected $1/(1-(1-p)^c)$ intervals before a scan finds its pivot at a given count c of control slots.

The "with probability 1" of Theorem 3 needs one more step, since a finite record populates finitely many rounds and can carry only a tail that vanishes with the horizon. It is checked twice: as a tail over blocks of $w_a \cdot \texttt{maxPeriod}$ coin rounds, one block opening every $\lceil w_a \cdot \texttt{maxPeriod}/\texttt{interval} \rceil$-th interval, and then almost surely over a sequence of records with the coin drawn as a process on the infinite product. A block is $w_a \cdot \texttt{maxPeriod}$ rounds long because above an interval boundary a scan reads only the multiples of `maxPeriod`, so that is what it takes to hold w_a consecutive control slots and settle the scans below. The tail asks of the interval only $\texttt{maxPeriod} \leq \texttt{interval}$ and $w_a \leq \texttt{interval}$: at the implementation's $\texttt{interval} = 128$ and $\texttt{maxPeriod} = 64$, a block opens every third interval at $w_a = 5$ and every second at $w_a = 4$.

C.3 Results

Table 4 maps each statement of Appendix B to its machine-checked counterpart. Each Lean result is an audited statement file with a separate kernel-checked proof.

What is not checked. For BlueBottle's pair everything is checked except the agreement of the output reading and the control reading on direct verdicts (Appendix B) and Appendix D, which concerns the certificate layer that pair lacks. The replay's window is read through each pair's support, the decision-round votes for BlueBottle's, and Lemmas 1 and 2 are checked per rule, at the slot's wave for the $3f + 1$ pair and inside each variant's own development for BlueBottle's. Outside the model altogether are the evaluation of Section 6 and any reading of the schedule's clock as wall-clock time.

D Partially Synchronous Protocols Under a Mistimed Leader Timeout

A commit rule that decides on one round of evidence survives asynchrony better than one that needs two. DAG-protocol designers know this as folklore; to the

Table 4. Statements of Appendix B and their machine-checked counterparts.

Statement	Lean result	Content
Lemma 1	Safety	A directly skipped slot has no certificate for any candidate, and two certified candidates of one author and round coincide, at the slot's own wave; for BlueBottle's pair, per variant, a candidate with $n - f$ supporters at its decision round leaves every rival below $n - 3f$, and a direct skip leaves every candidate below it.
Lemma 2	Safety	A directly committed candidate of kind κ at round r is certified in the causal history of every block at round $r + w(\kappa)$ or above, whatever wave that block's own slot carries; for BlueBottle's pair, per variant, $n - 3f$ of the candidate's supporters lie there.
Theorem 1	Safety, Interface	Two views deciding one slot reach the same verdict by any routes; at the interface level, a family of rules whose laws hold and that agree on the link count and tie-break composes into one whose laws hold, and both pairs the paper instantiates are such a composite, the $3f + 1$ one by definition and BlueBottle's from the two variants' own laws.
Corollary 1	Safety	A slot directly committed in one view is committed by every view that finds a committed anchor for it, whichever rule decides that anchor, and no view skips it.
Corollary 2	Ledger	Over a prefix each view has settled, the committed-leader sequences and the ledgers coincide, and a ledger never drops a block and delivers each block at one slot.
Theorem 2	Liveness	The honest-leader commit, the crashed-leader skip from $n - f$ blames, partial dissemination not deferring an asynchronous slot's decision, and the descent of the floor chain from a commit above; the round counts at $k = \infty$ and $k = 1$. Stated for any lawful rule except partial dissemination and the reactive pacing, which are stated per pair; checked at both pairs.
Theorem 3	Period, Coin	Above every interval some interval finds a pivot almost surely, the control verdicts of a scan settling once w_a consecutive control slots commit directly; the failover hands the next interval $k = 1$; at $k = 1$ a run of w_a commits decides every slot below it; and the tail vanishes, almost surely and for every slot at once. Stated for any pair of lawful rules with $w_s \leq w_a$ and the counting floor a parameter: $n - f - b$ for the $3f + 1$ pair at $w_a = 5$, 1 at $w_a = 4$, and $n - 3f$ for BlueBottle's.
Theorem 4	Period	The control slots of a scan are a function of the period, the period bound and the boundary, their verdicts agree across views, and the state is agreed under any update rule and at each validator's own derived schedule, for any lawful rules and at both pairs.
Theorem 5	Safety	At a schedule of one kind the composite decides exactly as that kind's rule, for any family that agrees on the link count and tie-break, and so at both pairs; for the $3f+1$ pair the rule at a constant wavelength is Mahi-Mahi's at that wave by definition, and at the constant wave 3 its verdicts are exactly Mysticeti's.
Lemma 3	Coin, Replay	c_r/n is at least the counting fraction on the DAG under any scheduling, $(n - f - b)/n$ for the $3f + 1$ pair at $w_a = 5$ and $(n - 3f)/n$ for BlueBottle's, and at both pairs also on the window once a quorum, honest for BlueBottle's pair, has populated within it the rounds the counting lemma reads; a probe's success is a support quorum the DAG holds.
Corollary 3	Broadcast	Definition 1 clause by clause over settled prefixes: agreement, integrity and total order, validity after GST with the first commit two rounds up and, under asynchrony, with the first commit above the round by which the reliable validators have referenced the block.
Algorithm 3	Replay	The window's evidence at both pairs, read through the rule's support, the three passes in exact rationals, the hysteresis and the tie rule, the odd canary probing every candidate, and every answer a divisor of `maxPeriod` within $[1, \texttt{maxPeriod}]$.
Parameters and guards	Period, Replay	Gating rule; both bounds on `interval`; the weaker one alone leaves a window resolving an asynchronous slot at some pivots and none at others; the warm-up interval; every derived period in range.
Appendix D	Timeout	A certificate forms with probability 1 on a unanimous vote round and f/n when one block abstained, and the $1/P_2$ wait is the layer cake of the tail probabilities.

best of our knowledge it has never been written down. We formalize it here. We consider the two partially synchronous protocols of this paper, Mysticeti and BlueBottle-PS, when their leader timeout T is smaller than the link delay D. The commit rule of BlueBottle-PS ($w_s = 2$) keeps committing with a constant probability. The commit rule of Mysticeti ($w_s = 3$) stalls. Two observations explain the difference: a larger quorum increases the probability of referencing the leader block, and requiring a single layer of evidence increases the probability of a direct commit. This is not a liveness statement: the delay D is bounded, so configuring $T \geq D$ restores both rules (Section 3). Instead, we explain a latency collapse under a too-small timeout, and show why it depends on the number of quorum layers.

D.1 Observation 1: Minimum-Quorum References

A validator waits up to T for the leader block before proposing, and the timer is armed at its own proposal. With every link delayed by $D > T$, the timer fires before any block of the round arrives, so the validator proposes the instant its threshold clock reaches $n - f$ blocks. As a consequence, every block carries minimum-quorum references. It includes exactly $n - f$ previous-round blocks, which are the earliest to arrive, and never more. We verified this on the DAG of the fixed-delay runs at $n = 10$: every block of both rules references exactly $n - f$ blocks.

We model this by assuming that at each validator the arrival order of a round's blocks is uniformly random. We assume equal stake and that a validator always references its own block. A validator references its own block plus the $n-f-1$ earliest blocks of the other $n-1$. A leader block L is therefore referenced with probability

$$p = \frac{n - f - 1}{n - 1}. \tag{1}$$

The chance of referencing the leader block is thus larger when the quorum threshold required to advance a round, $n - f$, is larger. At $n = 10$, BlueBottle-PS waits for $n - f = 9$ blocks and references the leader with probability 0.889, while Mysticeti waits for $n - f = 2f + 1 = 7$ blocks and gets 0.667. The seemingly harder quorum helps, because the same quorum sets how long a validator waits before proposing.

D.2 Observation 2: Layers of Evidence

One layer of evidence. The commit rule of BlueBottle-PS, at $n \geq 5f+1$, operates with $w_s = 2$. The votes of the single decision round are the certificates (Section 2). With p from (1), the total votes V for L follow a binomial distribution, with the leading 1 being L's author voting for itself:

$$V = 1 + \mathrm{Bin}(n - 1,\, p). \tag{2}$$

A direct commit requires at least $n - f$ votes for the leader block, so the commit probability is

$$P_1 = \Pr[V \geq n - f] = \sum_{k=n-f-1}^{n-1} \binom{n-1}{k} p^k (1-p)^{n-1-k}. \tag{3}$$

Two layers of evidence. The commit rule of Mysticeti, at $n \geq 3f + 1$, operates with $w_s = 3$. A certificate is a certify-round block referencing $n - f$ votes, and a direct commit needs $n - f$ certificates. With minimum-quorum references, a certifier holds exactly $n - f$ references. So all of them must be votes. With v votes in the round, the chance of a certificate forming is hypergeometric:

$$\pi(v) = \frac{\binom{v}{n-f}}{\binom{n}{n-f}}. \tag{4}$$

This probability is equal to 1 at $v = n$ and f/n at $v = n - 1$, and is negligible below. At $n = 10$ the $v = n - 1$ term contributes 0.001 to the commit probability, hence

$$P_2 \approx \Pr[V = n]. \tag{5}$$

The second layer turns the rule into an all-or-nothing filter: the slot commits directly only if every vote-round block voted.

Under independent arrival orders, $\Pr[V = n] = p^{n-1}$. This yields 0.026 at $n = 10$ and $4 \cdot 10^{-9}$ at $n = 50$. Independence is the one optimistic assumption. Arrival orders are positively correlated across validators because a block proposed early arrives early everywhere. On the fixed-delay DAG two validators' reference sets overlap in 3.7 blocks against 3.1 under independence, and V has fatter tails with the same bulk. P_1 reads the bulk and is unaffected, with a measured $\Pr[V \geq n - f] = 0.652$ against 0.650. P_2 reads the extreme tail, so p^{n-1} is a floor. We measured $\Pr[V = n] = 0.10$ against 0.026 at $n = 10$.

The consequence for progress follows. An undecided slot is decided only by the next direct commit above it, its anchor, and that commit decides every slot below it at once. Nothing accumulates; every slot simply waits for the next direct commit, which takes $1/P_2$ rounds in expectation. At $n = 50$ it never comes.

D.3 Experimental Validation

Fixed delay. We tested a setup where every link is delayed by a fixed $D = 800\,\text{ms}$, with $T = 100\,\text{ms}$. We ran Mysticeti and BlueBottle-PS at $n \in \{10, 50\}$ across three seeds. Every run advances at 1.2 rounds per second. The direct-commit rates per round are detailed in Table 5. At $n = 10$ the one-layer rule commits at the predicted rate. The two-layer rule commits directly in about one round in ten. This is four times the independent-order floor, but decisions arrive in bursts separated by 20–170 s in which no slot is decided. Transactions wait for the next burst, with a mean latency of 16–90 s, and one seed decides nothing in 180 s. At $n = 50$ the one-layer rule again matches P_1 within 3%. The two-layer rule decides

Table 5. Direct commits per round and latency with minimum-quorum references ($D = 800$ ms on every link, $T = 100$ ms), measured over three seeds against the closed forms (3) and (5).

Rule	n	Direct commits per round	Model	Latency
Mysticeti ($w_s = 3$)	10	0.10 (one seed: none)	$P_2 = 0.027$	16–90 s, in bursts
Mysticeti ($w_s = 3$)	50	0 in all seeds	$P_2 \approx 4 \cdot 10^{-9}$	stalled, no decision
BlueBottle-PS ($w_s = 2$)	10	0.68–0.75	$P_1 = 0.736$	3.9–4.1 s
BlueBottle-PS ($w_s = 2$)	50	0.57–0.62	$P_1 = 0.588$	4.9–5.4 s

nothing in any seed. The DAG advances at the same pace of about 210 rounds and 1700 leader timeouts, but the commit rule never fires, since without a direct commit there is no anchor either.

Partial asynchrony. We apply the random asynchronous model of Danezis et al. [13], in which messages follow a random rather than an adversarial schedule, as instantiated in Section 6: each message is delayed by 100–150 ms, past $T = 100$ ms, with probability ϕ. We evaluated $n = 10$ and $n = 50$ across three seeds. Fig. 7 plots latency and the direct-commit rate. Both rules interpolate between their healthy value and their minimum-quorum value of Table 5. BlueBottle-PS's direct-commit rate settles on the P_1 plateau of 0.74 from $\phi \approx 0.3$. Mysticeti's rate falls toward its measured $\Pr[V = n]$ of 0.10, which is above the independent-order floor. There is no threshold. Mysticeti's latency grows about 1.4 times per five points of ϕ. Sampling windows without any commit appear at $\phi = 0.25$ and dominate from $\phi = 0.40$. BlueBottle-PS pays at most five times its healthy latency and stays live throughout. The partial and full random-network panels of Section 6 are the $\phi = 0.3$ and $\phi = 1.0$ points. At $n = 50$ the same shape appears one step earlier: BlueBottle-PS saturates at 1.1 s with a direct-commit rate of 0.55–0.60, on its closed form $P_1 = 0.59$, while Mysticeti's rate has fallen to 0.17 by $\phi = 0.2$ and its latency risen to 2.7 s. Beyond that no slot is decided, and the runs cannot complete, as the simulator's cost per round grows with the undecided slots; Fig. 7 marks these points as off-scale. The closed forms describe only the minimum-quorum regime of Table 5, so they do not predict where the stall begins.

E Evaluation Details

We present the full setup behind Section 6. We detail the network conditions and explain how to read the figures. We then give the per-panel numbers for both pairs of protocols at $n = 10$ and $n = 50$, and report the runs at $n = 10$.

E.1 Setup

Parameters. Table 6 lists the parameters for the simulator and the load generator in its top block. The quorum timeout defines the wait for stragglers past the quorum on rounds without a leader wait. The load generator batches transactions every 50 ms. The bottom block of Table 6 lists the four parameters of Steelhead. The `interval` defines the replay window and the reaction horizon.

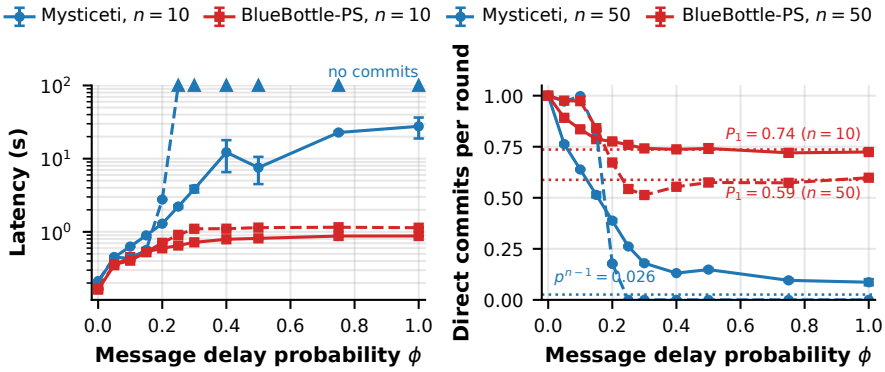


Fig. 7. Mysticeti and BlueBottle-PS against the probability ϕ that a message is delayed past the leader timeout ($n = 10$ solid, $n = 50$ dashed, three seeds, $\phi = 0$ is the runs' healthy phase). Left: mean latency, log scale. Right: direct commits per round, with the closed forms P_1 (3) at both committee sizes and the independent-order floor p^{n-1} (5) at $n = 10$ as guides. Mysticeti at $n = 50$ commits nothing beyond $\phi = 0.2$; those points are drawn off-scale at the top of the latency panel and at zero commits (see text).

Table 6. Simulator, load, and Steelhead parameters common to every run at $n = 10$.

Parameter	Value
Committee size n	10 ($f = 3$ for the $3f + 1$ pair, $f = 1$ for the $5f + 1$ pair)
Topology, per-link latency	full mesh, uniform in 25–50 ms per message
Leader timeout, quorum timeout	100 ms, 40 ms
Offered load, transaction size	1000 tx/s system-wide (100 tx/s per validator), 512 B
Load generator batching	every 50 ms
Run	450 s: healthy 0–30, condition 30–330, healthy 330–450
Sampling, seeds	5 s windows, 7 seeds
Leader schedule	round-robin, one leader per round
`interval`	128 rounds
`maxPeriod` (initial period)	64
Hysteresis ϵ	10%
Canary	every 31st asynchronous round

This spans about 6 s at the healthy round rate. The `maxPeriod` is the starting period, setting one asynchronous slot in 64. The hysteresis is the threshold of improvement a candidate period must show to be adopted. The canary keeps the leader wait on every 31st asynchronous round, so the committed DAG records the synchronous rule's evidence at any period; 31 is coprime to the candidate periods, so the probes fall on synchronous slots of every candidate.

We configure the base protocols as shown in Table 1. We run Mahi-Mahi at $w_a = 5$.

Conditions. Table 7 details the six conditions of Section 6 with their parameters. Every model adds delay on top of the link latency and never drops a message. The leader-delay adversary delays only the current round-robin leader's outbound blocks. This operates as a mute rather than an eclipse, as the targeted validator still receives blocks. The adversary tracks the round from the blocks it sees and is blind to the coin. The hidden leaders of asynchronous slots are therefore never hit. Crashes are permanent. The last f validators stop at 30 s. The random-delay

Table 7. The six network conditions of Figs. 8 and 9, applied from 30 s to 330 s.

Panel	Condition	Parameters
(i)	small leader delay	leader's outbound blocks delayed by 30 ms, below the timeout
(ii)	large leader delay	leader's outbound blocks delayed by 125 ms, above the timeout
(iii)	permanent crash faults	the last f validators stop at 30 s (3 / 1), no recovery
(iv)	partial random network	each message delayed with probability 0.3, uniformly in 100–150 ms
(v)	full random network	the same with probability 1
(vi)	high jitter	each message delayed by an exponential draw of mean 75 ms, capped at 400 ms

conditions instantiate the random asynchronous model of Danezis et al. [13]. Delays are drawn per message independently.

Measurement. We measure latency from a transaction's submission to the commit of its block, per replica, as the mean over each 5 s window. A panel plots the median across the seven seeds after a 3-tick rolling median. This rolling median removes isolated single-window spikes of a backlogged protocol. Windows without a commit appear as gaps. The bottom row of each figure breaks its axis when the partially synchronous protocol runs off the lower band. The period trace is the per-replica period averaged across the committee. A value between two candidates indicates replicas mid-transition.

E.2 Per-Panel Results

Table 8 provides the degraded plateau of the three lines per pair and panel. We calculate the plateau as the median of the 5 s windows between 70 s and 330 s. The table lists the time for Steelhead to reach period 1 after the onset and the time to return to 64 after the condition lifts. The healthy rows represent the first 30 s. "Stalled" marks a protocol that commits in fewer than one window in ten. Mysticeti in panel (v) of Fig. 8 (Section 6.4) commits in 43% of the windows at a 30 s median.

We detail two features of the period trace highlighted in Section 6. In the crash panel (iii) of Fig. 8, the period alternates between 64 and a value in $\{1, 2, 4, 8\}$ every second or third interval. This occurs identically in every seed because a probe landing on a crashed round-robin leader reads as a starved slot. Each dip costs a few windows of Mahi-Mahi latency and causes the 7% gap to Mysticeti. After recovery in panels (ii), (iv), and (v) of Fig. 8 (Section 6.4), the period drops to 2 for one interval 20–70 s after the condition lifts, and one 5 s window commits at about 0.25 s. In panel (i) the period drops to 1 or 2 for one interval once or twice during the condition, each time costing one window at about 0.3 s.

E.3 Committee of 10

We repeat the six conditions with the same parameters at $n = 10$ ($f = 3$ for the $3f + 1$ pair, $f = 1$ for the $5f + 1$ pair) in Figs. 8 and 9, on a longer timeline, which the smaller committee affords: 30 s healthy, the condition until 330 s, and healthy again until 450 s. Table 8 gives the per-panel numbers.

Table 8. Per-panel results at $n = 10$: degraded-plateau latency (ms) of the partially synchronous protocol, the asynchronous protocol, and Steelhead; Steelhead's time to reach period 1 after the onset and to return to 64 after the condition lifts.

Pair	Panel	Sync	Async	Steelhead	To period 1	Back to 64
$3f+1$	healthy	206	287	208	stays at 64	
	(i) small leader delay	303	293	306	1–2 one-interval drops	
	(ii) large leader delay	stalled	334	344	10–20 s	10 s
	(iii) crash faults	238	334	255	dips, see text	
	(iv) partial random network	3710	685	684	15–20 s	10 s
	(v) full random network	29 800	1121	1122	20 s	5–15 s
	(vi) high jitter	3114	1017	1022	15–25 s	10 s
$5f+1$	healthy	162	208	163	stays at 64	
	(i) small leader delay	236	218	238	a few one-interval drops	
	(ii) large leader delay	stalled	371	374	10 s	10 s
	(iii) crash faults	169	218	198	dips, see text	
	(iv) partial random network	714	672	685	wanders	0–10 s
	(v) full random network	872	810	806	wanders	0–10 s
	(vi) high jitter	929	908	915	wanders	0–10 s

Table 9. Per-panel results at $n = 50$: degraded-plateau latency (ms) of the partially synchronous protocol, the asynchronous protocol, and Steelhead; Steelhead's time to reach period 1 after the onset and to return to 64 after the condition lifts.

Pair	Panel	Sync	Async	Steelhead	To period 1	Back to 64
$3f+1$	healthy	213	294	215	stays at 64	
	(i) small leader delay	305	295	308	0–1 one-interval drops	
	(ii) large leader delay	stalled	301	313	10–20 s	10 s
	(iii) crash faults	356	455	363	dips, see Appendix E.2	
	(iv) partial random network	stalled	731	732	15 s	10 s
	(v) full random network	stalled	1167	1168	20 s	10 s
	(vi) high jitter	5713	1300	1473	20–45 s	10 s
$5f+1$	healthy	168	212	169	stays at 64	
	(i) small leader delay	238	214	238	stays at 64	
	(ii) large leader delay	stalled	220	229	10 s	10 s
	(iii) crash faults	217	268	221	dips, see Appendix E.2	
	(iv) partial random network	1041	702	705	10–30 s	5–15 s
	(v) full random network	1176	835	837	15–45 s	5–15 s
	(vi) high jitter	1023	935	944	wanders, 15–80 s	0–10 s

Claims C1 to C4 hold at $n = 10$ as well. In the healthy phases Steelhead stays within 1% of the partially synchronous protocol (208 vs 206 ms and 163 vs 162 ms) and 22–28% below the asynchronous one. It reaches period 1 within 10–25 s of the onset in every panel where a protocol stalls and returns to 64 within 5–15 s of the condition lifting, and it tracks the better protocol within 3% under the large leader delay (344 vs 334 ms) and within 1% under the network-wide conditions: 684 vs 685 ms under the partial random delay, 1122 vs 1121 ms under the full one, and 1022 vs 1017 ms under jitter. The crash dips of panel (iii) cost 7% and 17% for the two pairs, against 2% at $n = 50$. Two things differ from $n = 50$. Mysticeti survives jitter at a 3.1 s plateau rather than 5.7 s, and BlueBottle-PS ties its asynchronous variant under the network-wide conditions instead of falling behind it, so Steelhead sits between the two variants or just below both (685, 806 and 915 ms) while its period wanders across the candidates, which costs nothing.

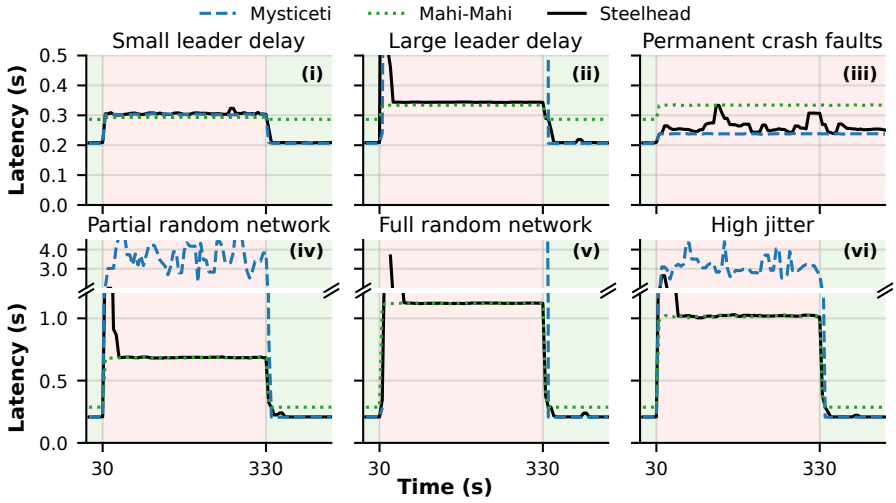


Fig. 8. Latency over time of Mysticeti, Mahi-Mahi, and adaptive Steelhead ($3f + 1$ pair, $n = 10$) under the six network conditions of Section 6.2, each applied from 30 s to 330 s (shaded).

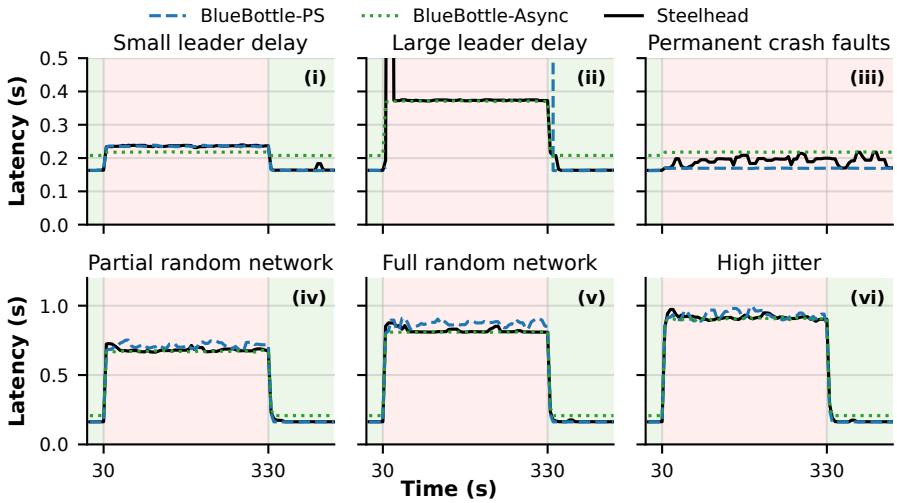


Fig. 9. Latency over time of BlueBottle-PS, BlueBottle-Async, and adaptive Steelhead ($5f + 1$ pair, $n = 10$) under the six network conditions of Section 6.2, each applied from 30 s to 330 s (shaded).