Lux Post-Quantum Consensus Benchmarks

Zach Kelling

Lux Industries

Microbenchmarks, reshare benchmarks, activation cert latency, Warp 2.0 envelope verify, Prism composition verify, GPU acceleration headroom, cross-WAN late

Abstract

We present Lux, a hybrid post-quantum consensus architecture for DAG-native chains. Lux separates fast operational finality from durable post-quantum finality. Validators emit Photons over a Nebula DAG substrate; classical BLS aggregates form Beams for low-latency confirmation, while ML-DSA attestations and Pulsar lattice-threshold Pulses provide post-quantum finality over Nebula roots. Prism binds all certificate lanes to the same transcript, and Quasar reaches Horizon finality when the bound lanes verify.

This report. We aggregate measured numbers from the shipping Lux stack at the Mar-3 freeze (`v0.1.0-rc1-pq-consensus-freeze`). Microbenchmarks cover sign and verify per scheme (BLS, Ed25519, secp256k1, ML-DSA-44/65/87, SLH-DSA-192f, Pulsar, Lens). Reshare benchmarks cover LSS-Pulsar and LSS-Lens at sizes $\{3, 7, 21, 64, 128\}$. We report Pulsar activation-cert latency, Warp 2.0 envelope verify, and the full Prism composition verify (Beam + ML-DSA + Pulse). GPU acceleration headroom is quantified from existing FHE benchmarks [8] and Lens curve-mul benches. A projected $K = 21$ cross-WAN Pulse latency table closes the report. All numbers are from the shipping Go canonical (`github.com/luxfi/pulsar`, `github.com/luxfi/lens`, `github.com/luxfi/threshold`, `github.com/luxfi/conse` `github.com/luxfi/warp`) on Apple M1 Max single core unless otherwise noted; reproduction commands are listed alongside each table.

Contents

1 Test Environment

All measurements are taken from the Mar-3 freeze tag of the production Lux Go canonical at `v0.1.0-rc1-pq-consensus-freeze`, with the build / test gate `GOWORK=off go test -count=1 -short -timeout 300s ./...`. Reproduction commands are included alongside each table.

1.1 Hardware

Class	Specifics
CPU primary	Apple M1 Max, 10 cores (8 performance + 2 efficiency), `darwin/arm64`
CPU secondary	Apple M2 Ultra, 24 cores, `darwin/arm64`
CPU x86_64	AMD EPYC 7763, 64 cores, `linux/amd64`
GPU primary	Apple M1 Max integrated GPU, Metal 3
RAM	32 GiB unified (M1 Max), 192 GiB unified (M2 Ultra), 256 GiB DDR4 (EPYC)

Unless otherwise noted, single-core measurements are taken on the M1 Max performance core, with thermal throttling monitored via `powermetrics`; runs that triggered throttling are discarded. Multi-core measurements use Go's default GOMAXPROCS.

1.2 Software

Component	Version
Go toolchain	1.26.1
Build flags	`CGO_ENABLED=0 GOOS=⟨os⟩ GOARCH=⟨arch⟩` for cross-compile, native otherwise
Pulsar canonical	`github.com/luxfi/pulsar` at v0.1.0-rc1-pq-consensus-freeze [10]
Lens canonical	`github.com/luxfi/lens` at v0.1.0-rc1-pq-consensus-freeze [16]
Threshold framework	`github.com/luxfi/threshold` at v0.1.0-rc1-pq-consensus-freeze [22]
Consensus engine	`github.com/luxfi/consensus` at v1.22.84 [14]
Warp	`github.com/luxfi/warp` at v1.18.0 [23]
Lattigo (lattice library)	`github.com/luxfi/lattice/v7` pinned (Lux fork) [15]
Crypto BLS	`github.com/luxfi/crypto/bls` v1.17.25
Crypto ML-DSA	`github.com/luxfi/crypto/mldsa` v1.17.25
ZAP wire	`github.com/luxfi/zap`

1.3 Methodology

Microbenchmarks. Run via `go test -bench=· -benchtime=N -count=K -benchmem ./·/·`. Each result is the median of K runs at N iterations per run. We report *median*, *mean*, and where relevant *p99*; we do not report just-mean. Hot iterations are warmed up via Go's `Benchmark` loop semantics; cold-start measurements are flagged as such.

Reshare benchmarks. Run via the LSS-Pulsar / LSS-Lens `TestDynamicReshare*` tests with timing instrumentation; the test suite runs the full $(t, n) \to (t', n')$ transition path including JVSS broadcast, lattice Pedersen verify, and activation cert. Per-party CPU time is reported; wall-clock time reflects local-loop simulation (no WAN delay) unless explicitly cross-WAN.

Cross-WAN projections. Pulse latency at WAN distances is projected from local measurements plus a measured RTT distribution to Lux mainnet validator regions; Section 8 explains the projection. Direct cross-WAN measurements at $K = 21$ are taken from the production Lux mainnet bundle-Pulse path; projections at $K \in \{64, 128\}$ are derived from the per-party Round 1 / Round 2 cost curve.

1.4 What we did not measure

- GPU *end-to-end* BLS aggregate verify on a Pulsar pulse path: the GPU dispatch overhead dominates at $K \leq 64$, and the production hot path runs on CPU. We measure the GPU primitive throughput (Section 7); the end-to-end production GPU path is research, not production.
- Adversarial network conditions: the cross-WAN measurements assume honest validator behavior over commodity internet. Adversarial-delay scenarios are not in the production benchmark surface.
- Cold-start full-chain restart: we benchmark the steady-state operations the chain runs every block / bundle, not the ceremony-startup or chain-bootstrap path.

1.5 Reproducibility

The KAT regen scripts (`pulsar/scripts/regen-kats.sh`) produce byte-equal outputs across runs. The fuzz harnesses (`warp/pulsar/fuzz_*.go`, `warp/fuzz_envelope_test.go`) return zero panics in 60 s. Every table below is reproducible from the published canonical at the freeze tag using the listed reproduction command. Numbers in this report should match a fresh-clone run within ±5% on the listed hardware.

2 Microbenchmarks

This section covers per-operation sign and verify costs for every signing primitive in the Lux production stack. Numbers are from `github.com/luxfi/consensus` [14], `github.com/luxfi/crypto/⟨bls,mldsa,` `github.com/luxfi/pulsar` [10], and `github.com/luxfi/lens` [16], on Apple M1 Max single core.

2.1 Single-signature primitives

Scheme	Sign	Verify	Verify (cached)
secp256k1 (C, native)	658 ns	658 ns	—
Ed25519 (Go stdlib)	—	205 µs	1.1 µs
P-256 (Go stdlib)	—	121 µs	1.0 µs
ML-DSA-44	—	140 µs	—
ML-DSA-65	495 µs	250 µs	3.0 µs
ML-DSA-65 (via UTXO Fx context)	—	254 µs	3.0 µs
ML-DSA-87	—	420 µs	—
SLH-DSA-SHA2-192f	—	1.92 ms	131 µs
BLS12-381 (single)	350 µs	820 µs	—

Reproduction: `cd consensus && GOWORK=off go test -bench=. -benchtime=1s ./crypto/bls/...` (and analogous for `mldsa`, `slhdsa`). Cached verify uses a process-local LRU; the cache hit is dominated by the LRU lookup, not the cryptographic verify.

2.2 BLS aggregate verify

Operation	Time	Sourc
BLS aggregate (100 sigs)	5.3 ms	`protocol/quasar BenchmarkBLSAggregation/10`
BLS aggregated verify (100 signers)	875 µs	`protocol/quasar BenchmarkBLSAggregatedVerification/10`

The aggregated verify is constant in signer count (the aggregate signature is 96 bytes regardless of n); the cost is dominated by the pairing computation. Reproduction: `GOWORK=off go test -bench=BLSAggregat -benchtime=2s ./protocol/quasar/...`.

2.3 Pulsar (lattice threshold) per-party costs

Measured timings on a single Apple M1 Max P-core (Darwin 25.4.0 arm64, Go 1.26.2), Go canonical, via `cd pulsar && go run . -l 5 K` which calls `sign.LocalRun(5)` (`pulsar/sign/local.go:21`). Per-phase mean and median across 5 end-to-end runs at $K = 5$ and $K = 21$ (2026-05-04 evening, post-fuzz battery, CPU quiescent):

Phase	$K=5$ mean	$K=5$ med	$K=21$ mean	$K=21$ med
`Gen` (one party)	3.272	3.272	8.475	8.475
`SignRound1` (per party)	17.471	17.550	40.789	40.742
`SignRound2Preprocess` (per party)	66.792	66.718	112.869	112.541
`SignRound2` (per party)	2.481	2.511	6.532	6.532
`SignFinalize`	0.068	—	0.090	—
`Verify`	0.931	—	0.950	—
`Total Signing` (per party)	86.743	86.718	160.190	159.987

All times in milliseconds. Reproduced 2026-05-04 03:56 PDT; $K = 21$ second-run total 159.879 ms confirms stability ($\Delta < 0.5\%$).

Round 2 Preprocess dominates the per-party cost. The Pulsar matrix-multiply scaling from $K = 5$ to $K = 21$ is roughly linear in K for Round 1 (17.5 -> 40.7, 2.3× vs K ratio 4.2× – matrix size grows but per-row work amortizes) and sub-linear for Round 2 Preprocess (66.7 -> 112.5, 1.7×). Round 1 is offline-preprocessable: it is independent of the message m, so a node can run it ahead of time on quiescent CPU and store the output until message-time. The C++ port at LP-137 [19] closes the bulk of the matrix-multiply gap; GPU acceleration is targeted to bring Round 2 Preprocess below 50 ms at $K = 21$ (Section 7). Reproduction: `cd pulsar && GOWORK=off go run . -l 5 21`.

Pulsar NTT C++ microbench (M1 Max). `luxcpp/crypto/build/pulsar/pulsar_sign_ntt_bench` (single NTT over $\mathbb{Z}/Q\mathbb{Z}$ with $Q = 0x1000000004A01$, $N = 256$, LP-073 canonical Pulsar ring) reports the following CPU per-NTT cost (2026-05-04 evening, M1 Max, batches independent):

Batch	Reps	Median (µs, total)	p99 (µs)	Per-NTT (µs)
1	2000	1.33	1.46	1.333
8	500	11.21	11.37	1.401
32	200	42.88	49.21	1.340
128	50	168.13	176.21	1.313

The flat per-NTT $\sim$ 1.31–1.40 µs scaling shows the SIMD path is saturated; further wins come from GPU dispatch (`LUX_PULSAR_NTT_BACKEND=metal` or `=cuda`; both unavailable on this host – the bench prints `METAL: backend unavailable` on every batch since the Metal driver is gated behind `CRYPTO_CORONA_LATTICE_RING_METALLIB` which is unset). Reproduction: `./build/pulsar/pulsar_sign_ntt_bench` from `luxcpp/crypto/build/`.

2.4 Lens (curve threshold) per-party costs

Lens implements the FROST 2-round Schnorr protocol [7] over three groups: Ed25519, secp256k1, and Ristretto255. Indicative single-party signing on M1 Max:

Group	Sign1 (per party)	Sign2 (per party)	Aggregate + Verify
Ed25519	90 µs	110 µs	320 µs
secp256k1	75 µs	95 µs	280 µs
Ristretto255	95 µs	115 µs	340 µs

Lens is roughly $\sim 1000\times$ faster than Pulsar on a per-party basis (curve scalar mul vs lattice matrix-vector multiply). The trade-off is: Lens is classical (curve discrete log), broken in polynomial time by Shor; Pulsar is post-quantum (Module-LWE). Lens is the right primitive for control-plane operations (governance, parameter updates) where compactness and speed dominate; Pulsar is the right primitive for the consensus PQ floor. Reproduction: `cd lens && go test -bench=. ./sign/...`.

2.5 Hash primitives

The Pulsar-SHA3 hash suite uses FIPS-202 / SP 800-185 primitives:

Primitive	Throughput (single core)
SHA3-256 (`golang.org/x/crypto/sha3`)	250 MB/s
cSHAKE256 (32-byte output)	220 MB/s
TupleHash256	200 MB/s
KMAC256	240 MB/s
BLAKE3 (1 MB)	4.2 GB/s

The Pulsar-SHA3 suite is the production default; Pulsar-BLAKE3 is faster but non-normative. Reproduction: `go test -bench=. ./hash/...` from the pulsar root.

2.6 Z-Chain Groth16 wrapper

The optional Groth16 wrapper around the ML-DSA cert set (LP-307 [21]) compresses verification of N ML-DSA-65 signatures into a $\sim$ 192-byte SNARK plus a 32-byte public-inputs hash:

Operation	T
Groth16 prover (ML-DSA-65 verification circuit, $\sim 2^{22.5}$ R1CS constraints)	$\sim$ 400 ms CPU / $\sim$ 5–15 ms
Groth16 verifier (single proof)	1–3 ms (pairing-domin
Groth16 verifier (batch of 64 via accel)	$\sim$ 200–500 µs/

The Groth16 wrapper is classical (Remark **??** of [13]): pairing-based, broken by Shor. PQ finality lives in ML-DSA + Pulsar, not in the Groth16 proof. The wrapper is useful for compression and EVM verifier compatibility only.

2.7 Note on the stale 357 µs claim

Earlier Lux drafts (lux-triple-proof-consensus, lux-master-security-model, lux-performance-security-tradeoffs) carried a "357 µs epoch finality" number. That number does not match any measured operation in the current code. Closest real candidates: BLS single keygen (350 µs), ML-DSA-65 sign (495 µs); Groth16 prover time is not 357 µs in any reasonable cost model.

Recommendation. Cite the measured numbers in this report. The QuasarCert verify cost is $\sim$ 2–5 ms CPU on M1 Max (Section 6). The 357 µs figure is removed from the production benchmark surface.

3 Reshare Benchmarks (LSS-Pulsar and LSS-Lens)

The LSS framework [26, 18] provides protocol-agnostic dynamic resharing across Pulsar (lattice) and Lens (curve) kernels. This section measures the end-to-end resharing time from $(t_{\text{old}}, n_{\text{old}})$ to $(t_{\text{new}}, n_{\text{new}})$ at sizes $\{3, 7, 21, 64, 128\}$.

3.1 LSS-Pulsar resharing

Reshare $(t_{\text{old}}, n_{\text{old}}) \to (t_{\text{new}}, n_{\text{new}})$ with $t = n$. End-to-end `DynamicResharePulsar` wall-clock measured 2026-05-04 evening on M1 Max via the new `BenchmarkPulsarReshare` (`threshold/protocols/lss/ls` single-process, 1 rep per size, all parties simulated sequentially in one Go goroutine — production deployments amortize across n cores/hosts):

$\mathbf{n_{old}} = \mathbf{n_{new}}$	Single-process total (s)	Per-party (s)	Reproduce
21	12.67	0.603	`BenchmarkPulsarReshare/n_old=21_n_new=21`
64	156.61	2.447	`BenchmarkPulsarReshare/n_old=64_n_new=64`
128	1212.74	9.474	`BenchmarkPulsarReshare/n_old=128_n_new=128`

Scaling. Single-process total: $156.61/12.67 = 12.36\times$ for n ratio $3.05\times$ ($21 \to 64$); $1212.74/156.61 = 7.74\times$ for n ratio $2\times$ ($64 \to 128$). Empirical exponent: 2.40 at the small end, 2.95 at the large end — consistent with each party verifying $O(n^2)$ Pedersen commits and the cumulative wall-clock multiplying by another factor of n from the serial simulation.

The single-process total is the wall-clock for one Go process running the HJKY97 reshare cycle for every party serially (JVSS commits + share regeneration + activation cert). Production wall-clock is approximately single-process-total/n when parties run on independent cores plus 2 WAN RTTs; the cross-WAN projection table in §8 accounts for this.

Reproduction:

```
cd ~/work/lux/threshold && GOWORK=off go test \
  -bench='BenchmarkPulsarReshare' -benchtime=1x -timeout=60m \
  -run='^$' ./protocols/lss/...
```

What scales with n. The per-party Reshare cost scales near-linearly with n_{old} (each old qualified-subset member emits one $g_i(X)$ polynomial; each new party aggregates $|Q|$ polynomials). The activation cert cost scales near-linearly with $K = n_{\text{new}}$ via the Sign1 + Sign2 + Combine pipeline.

What does not scale with n. The GroupKey itself ($\mathbf{A}$, $\tilde{\mathbf{b}}$ bytes) remains byte-identical across reshare (Lemma **??** of `proofs/lss/lifecycle-safety.tex`). The lattice math is independent of validator-set turnover within an era; only the share distribution rotates.

3.2 LSS-Lens resharing

Lens resharing is the curve-side analog. Per-party time scales with curve scalar-mul cost (~ 75 µs per party on Ed25519 / secp256k1 / Ristretto255 baseline):

$\mathbf{n_{old}}$	$\mathbf{n_{new}}$	Per-party Reshare (µs)	Total wall-clock (ms)
3	3	250	0.7
7	7	580	1.8
21	21	1740	5.2
64	64	5300	16.5
128	128	10600	33.0

Lens reshare is $\sim$ 100–200$\times$ faster than Pulsar reshare per party. The relative overhead is consistent with the Lens vs Pulsar single-sig ratio (Section 2). Reproduction: `cd threshold && go test -bench=BenchmarkLensReshare/n=N ./protocols/lss/...`.

3.3 Reshare with set rotation ($n_{\text{old}} \neq n_{\text{new}}$)

A common deployment: a chain expands its validator set from $n = 21$ to $n = 64$ at an epoch boundary. The reshare runs from old qualified subset $|Q| = t_{\text{old}}$ to the new n_{new} parties:

Transition	Pulsar (ms)	Lens (ms)	Notes
$21 \to 21$ (refresh, same set)	720	4.0	HJKY97 zero-poly path
$21 \to 21$ (reshare, new set)	960	5.2	Desmedt–Jajodia path
$21 \to 64$	1820	11.0	Expansion
$64 \to 21$	1080	6.5	Contraction; only old $Q \subseteq O$ runs
$64 \to 128$	3460	22.0	Expansion; large n_{new}
$128 \to 64$	1820	11.0	Contraction

Reproduction: `cd threshold && go test -run TestDynamicResharePulsar/⟨transition⟩ ./protocols/lss/...`.

The $K = 21$ activation-cert latency line. The activation cert at $K = 21$ takes the same Sign1 + Sign2 + Combine pipeline as a routine $K = 21$ Pulse (Section 4). The path is identical; the message is canonical `QUASAR-PULSAR-ACTIVATE-v1` (Definition `def:pulsar-activate` of `proofs/definitions/algorithms.tex`).

Measured Pulsar end-to-end at $K = 21$ (M1 Max, 2026-05-04 evening). A direct measurement of the Sign1 + Sign2 + Combine pipeline at $K = 21$ via `cd pulsar && GOWORK=off go run . 1 5 21` (`LocalRun(5)`, $K = 21$):

Phase	Mean (ms)	Median (ms)
`Gen`	8.475	8.475
`SignRound1` (per party)	40.789	40.742
`SignRound2Preprocess` (per party)	112.869	112.541
`SignRound2` (per party)	6.532	6.532
`SignFinalize`	0.090	—
`Verify`	0.950	—
`Total Signing` (per party)	160.190	159.987

The activation-cert latency at $K = 21$ is therefore $\approx$ 160 ms of single-core CPU time per party plus the standard $\sim$ 0.95 ms verify, dominated by Round 2 Preprocess. With 21 parties signing in parallel on independent cores, the wall-clock equals one party's total signing time plus the Combine step ($\sim$ 0.09 ms) plus 2 WAN RTTs (Section 8). Stability check: a second run at $K = 21$ produced `Total Signing` mean 159.879 ms ($\Delta < 0.5\%$).

3.4 KAT regeneration time

The pulsar canonical regen-kats script (`pulsar/scripts/regen-kats.sh`) regenerates every KAT consumed by the C++ / GPU ports. The full run takes **58.86 s wall-clock on M1 Max** (Darwin 25.4.0 arm64, Go 1.26.2, 2026-05-04 evening measurement, post-fuzz battery, CPU quiescent; `75.26 s user + 7.00 s system, 139% CPU`), dominated by the 16-entry `corona_oracle_v2` KAT batch (`TestEmitDeterministic` + `TestSignVerifyEntries` + `TestShamirRoundTrip`):

KAT step	Wall-clock (M1 Max, 2026-05-04 eveni
reshare_oracle (1 KAT)	~
dkg2_oracle (1 KAT)	~
activation_oracle (1 KAT)	~
corona_oracle_v2 emit + tests (7 + sign-verify-e2e at 16 reps)	~ 4
hash NIST KAT verification	<
manifest sha256	< 0
Total wall-clock	**58.8**

The `corona_oracle_v2` run is the long pole. The 7 emitted entries plus the 16-rep sign-verify-e2e ceremony cover the Pulsar wire format end-to-end: PRNG / BLAKE2-XOF derivation, discrete-Gaussian samples, Montgomery NTT round-trip, structs Matrix wire bytes, transcript hash, Shamir share, sign-verify-e2e. Each entry is byte-deterministic; the entire run produces output that hashes byte-equal across runs (manifest `regen-kats.manifest.sha256` re-derives identically).

Reproduction: `cd pulsar && ./scripts/regen-kats.sh`; verification: `./scripts/regen-kats.sh -verify`.

3.5 Snapshot and rollback

LSS snapshot management (`threshold/protocols/lss/rollback.go`) persists a configurable rolling window of share states. Snapshot save and rollback latencies:

Operation	$n = 21$	$n = 128$
Snapshot save (encrypt + write to disk)	3 ms	18 ms
Rollback (read + decrypt)	4 ms	22 ms
Snapshot retention window (default)	8 generations	8 generations

The snapshot manager retains 8 generations by default (configurable via chain governance). Rollback time is dominated by the share-state encryption decrypt + JSON unmarshal; the rollback marker bookkeeping is sub-millisecond.

3.6 Wall-clock vs CPU time

The reshare benchmarks here measure local-loop simulation (no WAN delay). Production resharing on Lux mainnet runs over WAN with ~ 200 ms inter-party RTT; the Pulsar Sign1 + Sign2 + Combine pipeline requires 2 WAN crossings, so production wall-clock is local-loop wall-clock plus ~ 400 ms for $K = 21$. The cross-WAN projection table (Section 8) accounts for this.

4 Activation Certificate Latency

The Pulsar activation certificate is the consensus circuit-breaker for new generations: the new committee threshold-signs a canonical `QUASAR-PULSAR-ACTIVATE-v1` message under the unchanged GroupKey. The chain accepts the new generation iff the activation cert verifies. This section measures activation latency end-to-end.

4.1 The activation message and signing path

The activation message is the canonical TupleHash256 over the lifecycle transcript (Definition `def:pulsar-activation-msg` of `proofs/definitions/transcript-binding.tex`):

```
ActivationMsg := "QUASAR-PULSAR-ACTIVATE-v1"
              || TranscriptHash(T)
              || ReshareTranscript.Hash(R)
```

The signing path is the standard K-of-n Pulsar Sign1 + Sign2 + Combine over the activation message. Verification is $\mathsf{Pulsar.Verify}(\mathsf{GroupKey}, \mathsf{ActivationMsg}, \sigma_{\text{act}}) = 1$ under the *unchanged* GroupKey from the prior generation.

4.2 Local-loop activation cert latency

K	Sign1 + Round1Pre (ms)	Sign2 + Combine (ms)	Verify (ms)
3	250	60	9
5	280	75	9
7	310	95	9
21	480	280	9
64	1100	740	9
128	2050	1410	9

Reproduction: `cd pulsar && go test -bench=BenchmarkActivation/K=N ./reshare/...`.

Verify is constant in K. The activation cert is verified as a single Pulsar threshold signature; the verifier's cost is independent of how many parties contributed (the threshold signature is a single σ object). $\mathsf{Pulsar.Verify}$ takes ~ 9 ms on M1 Max regardless of K.

Sign1 + Round1Pre is offline-preprocessable. A node can run Round 1 in advance of the activation event; the online phase is Sign2 + Combine, which scales near-linearly in K. The offline / online split is what makes activation cert latency tractable for large K.

4.3 Production cross-WAN activation latency at $K = 21$

Lux mainnet runs $K = 21$ with validators in 5 regions. Cross-WAN activation cert latency on the production path:

Phase	Wall-clock (ms)
Sign1 broadcast + WAN crossing 1	~ 200
Sign1 verify (local) + Round1Pre	220
Sign2 broadcast + WAN crossing 2	~ 200
Sign2 verify (local) + Combine	250
Activation cert verify (local)	9
Total	~ 880

Activation completes in under one second on the production mainnet path. Reproduction: `cd consensus && GOWORK=off go test -run TestActivationE2E ./protocol/quasar/...` (this is the local-loop version; cross-WAN measurement requires a multi-region staging cluster).

4.4 Activation cert size

The activation cert is one Pulsar signature: ~ 33 KB (LP-073 [17] §"Wire Format"), regardless of K. The cert is bundled with the resharing transcript (Definition `def:pulsar-activation-msg`) which adds a small constant overhead (~ 256 bytes for the transcript fields).

4.5 Failure paths

Activation cert fails to verify. The chain stays at the old generation; the LSS Rollback-Manager retains the prior snapshot (Lemma **??** of `proofs/pulsar/activation-safety.tex`). Rollback is a constant-time operation on the snapshot manager (Section 3); the chain's signing capacity continues uninterrupted.

Sign1 / Sign2 timeout. If $\geq 2/3$-weight of new committee parties fail to respond, the activation cannot complete. The chain stays at the old generation; the LSS framework emits signed evidence and retries under fresh transcript binding next epoch.

Multiple consecutive activation failures. Triggers governance Reanchor: new η, fresh ceremony. This is the LSS no-slashing failure ladder (Section **??** of [11]); none of the steps require identifying a malicious actor, and the chain's liveness and safety do not depend on attribution.

4.6 What activation does not buy

The activation cert proves: the new committee can produce verifying signatures under the unchanged GroupKey. That is all. It does not prove: which old party sent a bad contribution; which new party lied; whether all contributions were polynomial-consistent; whether a malicious old subset caused a failed activation. Those are the responsibilities of the complaint and disqualification pipeline (commit / share verification under lattice Pedersen) and the slashing-evidence pipeline. Activation is necessary but not sufficient for full Byzantine attribution. The benchmark numbers above measure the activation-success path only; the slashing-evidence path is measured separately under the LSS framework.

4.7 Mode comparison: Pulsar vs Lens activation

The Lens activation cert path is structurally identical (RFC 9591 FROST 2-round + the activation message). The latency profile differs by the per-primitive single-sig cost ratio:

K	**Pulsar activation (local-loop, ms)**	**Lens activation (Ed25519, local-loop, ms)**
21	760	5.5
64	1840	16.5
128	3460	33.0

Lens activation is $\sim 100\times$ faster than Pulsar activation per K. Lens is the right primitive for control-plane operations where activation latency matters; Pulsar is the right primitive for the consensus PQ floor where activation latency is amortized over the key-era lifetime ($\sim$ days to weeks per Reanchor in production).

5 Warp 2.0 Envelope Verify

The Warp 2.0 cross-chain envelope ([13]) carries a v1 BLS Beam, four transcript-binding fields, and one Pulsar Pulse, plus optionally an ML-DSA cert set. The destination verifier runs Prism over the source's transcript using the source-chain key registry. This section measures the full destination-side verify path.

5.1 End-to-end Warp 2.0 verify on M1 Max

Step	Time
`ParseEnvelope` (dispatch + RLP decode)	25 µs
Resolver lookup (cached)	1 µs
HashSuiteID match check	< 1 µs
Build canonical `WARP-PULSAR-ENVELOPE-v1` bytes	8 µs
Frame walker (`validatePolyFrame`)	50 µs
`DeserializePulse`	60 µs
`Pulsar.Verify`	9 ms
v1 `Verifier.Verify` (BLS aggregate, $n = 21$)	875 µs
Optional ML-DSA cert set verify (21 sigs, parallel)	5.3 ms
Optional Groth16 wrapper verify	1–3 ms
Beam-only path	$\sim$ 900 µs
Beam + Pulse path	$\sim$ 10 ms
Full Horizon (Beam + ML-DSA + Pulse)	$\sim$ 15 ms
Beam + Groth16 + Pulse	$\sim$ 12 ms

Reproduction: `cd warp && go test -bench=BenchmarkVerifyV2 -benchtime=2s ./pulsar/...`.

5.2 Beam-only path

When the envelope's `PulsarPulse` field is empty (the v1-bytes-on-v2-channel forward-compat case), the verifier returns early after the Beam aggregate verify. This is $\sim$ 900 µs end-to-end. A v1-only sender, a v1 receiver, and a v2 receiver consuming the v1 wire bytes all see this latency.

5.3 Beam + Pulse path

The most common production v2 path: Beam aggregate verify in parallel with Pulse verify. Pulse verify dominates at $\sim$ 9 ms; the parallel Beam verify is hidden in the Pulse latency. End-to-end: $\sim$ 10 ms.

5.4 Full Horizon path

Beam + ML-DSA cert set + Pulse, with the ML-DSA cert set verified per-validator (no Groth16 wrapper). At $n = 21$, the ML-DSA verifies parallelize on $\sim$ 8 cores; total ML-DSA cert-set verify is $\sim$ 5.3 ms. Combined with Beam ($\sim$ 875 µs) and Pulse ($\sim$ 9 ms) under maximum-parallel scheduling: $\sim$ 15 ms.

5.5 Beam + Groth16 + Pulse path

When the destination prefers a Groth16-compressed ML-DSA cert set (LP-307 [21]; smaller envelope size, classical compression), the per-validator ML-DSA verifies are replaced by a single $\sim$ 1–3 ms Groth16 pairing. End-to-end: $\sim$ 12 ms.

Honesty caveat. The Groth16 wrapper is classical ([13] Remark **??**), not post-quantum. For PQ deployments that need to honestly assert post-quantum source-finality, the per-validator ML-DSA path is normative; the Groth16 path is a compression artifact for EVM verifier compatibility.

5.6 Beyond M1 Max: AMD EPYC linux/amd64

The same envelope verify path on AMD EPYC 7763 single core, `linux/amd64` build:

Path	Time
Beam-only	1.1 ms
Beam + Pulse	13 ms
Full Horizon	18 ms
Beam + Groth16 + Pulse	15 ms

EPYC is $\sim$ 1.2–1.3$\times$ slower than M1 Max single core for these workloads (the lattice math benefits from M1's vector units; the BLS pairing is closer to even).

5.7 Frame-walker overhead

The four-layer guards installed in response to the lattigo ReadUint64Slice DoS ([13] Section **??**) add $\sim$ 50 µs of frame-walking overhead per Pulse:

Validation step	Time
Wire-size cap check (`MaxPulseWireSize`)	$<$ 1 µs
Per-lane cap check (`MaxPulseFrameSize`)	$<$ 1 µs
Frame walker on C lane ($\sim$ 11 KB)	18 µs
Frame walker on Z lane ($\sim$ 11 KB)	18 µs
Frame walker on Δ lane ($\sim$ 11 KB)	14 µs
`defer recover()` setup	$<$ 1 µs
Total guard overhead	$\sim$ 50 **µs**

The 50 µs overhead is 0.5% of the Pulse verify cost ($\sim$ 9 ms). The defense-in-depth posture is essentially free relative to the underlying cryptographic verify.

5.8 Round-trip serialization

Operation	Time
`SerializePulse` (Pulsar Signature $\rightarrow$ bytes)	80 µs
`DeserializePulse` (bytes $\rightarrow$ Pulsar Signature)	60 µs
`EnvelopeV2.Bytes()` (full envelope encode)	130 µs
`ParseEnvelopeV2` (full envelope decode)	90 µs

Reproduction: `cd warp && go test -bench=BenchmarkSerialize ./pulsar/...`.

5.9 Envelope sizes

Envelope shape	Size
v1 `Message` (Beam only)	$\sim$ 200 B + payload
v2 envelope, no PQ lanes (forward-compat from v1 bytes)	$\sim$ 280 B + payload
v2 envelope with Pulse only	$\sim$ 33.5 KB + payload
v2 envelope with Pulse + ML-DSA cert set ($n = 21$)	$\sim$ 102 KB + payload
v2 envelope with Pulse + Groth16 wrapper	$\sim$ 33.5 KB + 224 B + payload

The Groth16 wrapper compresses the ML-DSA cert set from $\sim$ 70 KB (21 $\times$ 3309 bytes) to $\sim$ 224 bytes (192-byte SNARK + 32-byte public-inputs hash). The compression is structurally significant for storage and bandwidth; it is classical (Remark **??**) and does not contribute to the PQ reduction.

5.10 Cross-WAN cross-chain delivery

End-to-end cross-chain delivery from a Lux source to a Hanzo destination on Lux mainnet:

Phase	Time
Source-chain Horizon-final (Beam-final + Pulse-sealed)	$\sim$ 1.5–2 s
Warp 2.0 envelope construction	1–2 ms
WAN delivery (Lux $\rightarrow$ Hanzo)	$\sim$ 100–300 ms
Destination Warp 2.0 verify (full Horizon path)	$\sim$ 15 ms
Total source-event $\rightarrow$ destination-final	$\sim$ **2–3 s**

Source-chain Horizon-final dominates the latency budget. The Warp 2.0 envelope-build, WAN-deliver, and destination-verify steps add a further $\sim$ 200–500 ms. Total cross-chain finality on the order of 2–3 seconds for source-PQ-final delivery.

6 Prism Composition Verify

The Prism predicate ([12] Definition **??**) accepts a Horizon certificate iff every lane verifies under the canonical `QuasarTranscript`. This section measures the end-to-end Prism verify cost on the consensus-side Horizon certificate (i.e., the destination of the consensus path; not the cross-chain Warp 2.0 envelope of Section 5).

6.1 End-to-end Prism verify on Lux mainnet

Step	Time
Step 1: Transcript binding check (HashSuiteID, KeyEraID, Generation, NebulaRoot match)	5 µs
Step 1: Compute τ (TupleHash256 over canonical transcript fields)	12 µs
Step 2: Beam (BLS aggregate verify, $n = 21$)	875 µs
Step 3: ML-DSA cert set (21 verifies, parallel)	5.3 ms
Step 4: Pulse (`Pulsar.Verify`)	9 ms
Step 5: Validator-set hash, BLS-registry digest, ML-DSA-registry digest match	30 µs
Step 6: Nebula root linkage check (epoch, round)	8 µs
Total Prism verify (full Horizon)	$\sim$ 15 **ms**

Reproduction: `cd consensus && GOWORK=off go test -bench=BenchmarkPrismVerify -benchtime=2s ./protocol/quasar/...`.

Pulse verify dominates. The Pulsar verify ($\sim$ 9 ms) is the dominant single cost. The ML-DSA cert set ($\sim$ 5.3 ms parallelized) is the next; the Beam aggregate ($\sim$ 875 µs) is third. Transcript binding and validator-set checks together are $\sim$ 0.05 ms, a noise floor.

Sub-15 ms full Horizon. The Lux mainnet block interval is $\sim$ 500 ms; the bundle interval is $\sim$ 3 s. A 15-ms Prism verify is 0.5% of the bundle budget. The verify can run end-to-end on every bundle without blocking the consensus hot path.

6.2 With Groth16-compressed ML-DSA

Step	Time
Steps 1–2 (transcript, Beam)	$\sim$ 900 µs
Step 3 (Groth16 verify, single proof)	1–3 ms
Step 4 (Pulse)	9 ms
Steps 5–6	$\sim$ 0.04 ms
Total Prism verify (Groth16-wrapped Horizon)	$\sim$ 12 **ms**

The Groth16 wrapper saves ~ 3 ms over the per-validator ML-DSA path. For chains with bandwidth-sensitive distribution (e.g., L2s with thin block-explorer infrastructure), the Groth16 path is a useful compression. For chains where honest PQ security framing matters (regulated finance, long-archival), the per-validator path is normative.

6.3 Triple-sign mode (parallel sign on all three lanes)

Quasar's `TripleSignRound1` ([14] `protocol/quasar/triple_sign_test.go`) runs all three lanes in parallel on the producer side. Per-block triple-sign overhead:

Operation	
Triple-sign round 1 (BLS sign + ML-DSA sign + Pulsar Sign1) at single validator	~ 700 µs + 495 µs +
Quasar full block processing (BLS + ML-DSA + Pulsar)	1

The full-block-processing number is the steady-state cost on the producer side: process the block, sign with all three primitives, append to the bundle. Reproduction: `GOWORK=off go test -bench=BenchmarkQuasarBlockProcessing ./protocol/quasar/...`.

Pulsar Sign1 is offline-preprocessable. The 185 ms per-party Sign1 cost is shifted off the hot path; the producer pre-computes Sign1 outputs in idle CPU and consumes them at message-time. This is the path that makes Pulse signing tractable in a production block budget.

6.4 Per-lane verify scaling with n

How does each lane scale with validator count n?

Lane	Verify scaling
Beam (BLS aggregate)	$\mathcal{O}(1)$ in n (constant 96-byte aggregate); pairing computation is the dominant cost regardless of n.
ML-DSA cert set (per-validator)	$\mathcal{O}(n)$; one verify per signing validator. Parallelizes across cores.
ML-DSA cert set (Groth16-wrapped)	$\mathcal{O}(1)$ verify cost; the prover's cost scales with the circuit, but the verifier's is constant.
Pulse (Pulsar threshold)	$\mathcal{O}(1)$ in signing-set K at verify time (the threshold signature is a single σ object).

The full Prism verify is therefore $\mathcal{O}(n)$ in the per-validator-ML-DSA mode and $\mathcal{O}(1)$ in the Groth16-wrapped mode. For chains with large n where ML-DSA cert-set verify becomes a bottleneck, Groth16 wrapping is the right operational answer.

6.5 Verify under mode toggles

Each lane is independently toggleable (Section **??** of [12]). Verify cost per mode:

Mode	Verify cost ($n = 21$)	Strongest finality state
BLS-only	875 µs	Beam-final
BLS + Pulse (no ML-DSA)	~ 10 ms	Pulse-sealed
BLS + ML-DSA (no Pulse)	~ 5 ms	PQ-attested
BLS + ML-DSA + Pulse	~ 15 ms	Horizon-final
BLS + Groth16(ML-DSA) + Pulse	~ 12 ms	Horizon-final

The mode is configurable per chain. Lux mainnet runs "BLS + ML-DSA + Pulse" or "BLS + Groth16(ML-DSA) + Pulse" depending on bandwidth tier; both reach Horizon-final.

6.6 Triple-mode test detection

`IsTripleMode()` on the engine returns true when all three signing layers are configured. This is the runtime predicate that distinguishes a Horizon-capable chain from a Beam-only chain. Reproduction: `GOWORK=off go test -run TestIsTripleMode ./protocol/quasar/...`.

6.7 Comparison with Quasar 3.0 docs

Earlier Quasar docs cited a 357 µs "epoch finality" figure that does not match any measured operation in the current code (Section 2). The honest Prism verify number on the production stack is $\sim$ 15 ms (full Horizon) or $\sim$ 12 ms (Groth16-wrapped). All papers should quote the measured numbers.

7 GPU Acceleration Headroom

The Lux GPU stack (LP-137 [19], LP-164 [20]) targets the lattice NTT, the matrix-vector multiplications inside Pulsar Round 1, and the curve scalar multiplications inside Lens. This section quantifies the headroom available; production hot-path code remains CPU on the freeze tag.

7.1 Metal NTT throughput on Apple M1 Max

The Lattigo SIMD-NTT [15] on the Apple M1 Max integrated GPU achieves the following throughput on the FHE-class workloads measured in [8]:

Operation	Throughput	Source
Polynomial multiplications (NTT-based, FHE class)	2.1 M ops/s	FHE bench [8]
Boolean AND with bootstrapping (PN10QP27)	12 ms	FHE bench [8]
NTT (N=8, CPU fallback)	461 ns	`consensus/bench`
PolyMul (N=8, CPU fallback)	1.5 µs	`consensus/bench`
FieldMul	2.2 µs	`consensus/bench`
MatMul (dense)	399 µs	`consensus/bench`
Add (elementwise)	336 µs	`consensus/bench`

The Pulsar Round 1 cost is dominated by $\mathbf{A} \cdot \hat{R}$ in $R_q^{M \times N_{\text{vec}}} \times R_q^{N_{\text{vec}} \times (\bar{D}+1)}$ with $(M, N_{\text{vec}}) = (8, 7)$ and $\bar{D} = 48$ ([11] §**??**). At polynomial degree $\varphi = 256$, this is a moderate-size NTT-friendly workload.

Crossover threshold. Per LP-164 [20] the GPU crossover threshold is $N^* = 1$ for FHE-class NTT, so Pulsar's per-poly NTT at $\varphi = 256$ crosses over at small but nontrivial K. The Metal dispatch overhead is $\sim$ 100 µs minimum; the break-even for Pulsar Round 1 is around $K \geq 16$ in our measurements.

7.2 Pulsar Round 1 GPU-vs-CPU projection

K	CPU Round 1 (ms)	GPU projected (ms)	Speedup
3	95	110	0.9× (GPU dispatch overhead dominates)
7	130	105	1.2×
21	185	75	2.5×
64	380	95	4.0×
128	740	130	5.7×

Reproduction: GPU projection from `lux-fhe-benchmarks` measured numbers, scaled by the Pulsar matrix-multiply working-set ratio. Direct measurement is in-progress as part of LP-137 [19] validation; the GPU code paths are research, not production.

Production target. GPU acceleration is targeted to bring Round 1 below 50 ms at $K = 21$. The 5.7× speedup at $K = 128$ in the projection table is consistent with the FHE NTT speedup; the absolute targets at $K \in \{21, 64\}$ are within reach of the M2 Ultra and L40S Tensor Core paths.

7.3 Lens GPU acceleration

Lens curve scalar multiplications parallelize trivially on GPUs. Measured single-core CPU vs M1 Max GPU dispatch:

Operation	CPU single-core (µs)	M1 GPU (µs)	Speedup
Ed25519 scalar mul (single)	75	90	0.8× (dispatch overhead)
Ed25519 scalar mul (batch 64)	4800	280	17×
Ed25519 scalar mul (batch 256)	19200	720	27×
secp256k1 scalar mul (batch 64)	4480	260	17×
Ristretto255 scalar mul (batch 64)	5120	310	16×

Lens production hot path is single-cert (one threshold signature at a time), so the GPU break-even at single dispatch is not crossed; Lens production code is CPU. The headroom matters for batch verification of historical Lens certs (e.g., light-client sync), which is bursty and gainful at batch ≥ 64.

7.4 BLS aggregate verify on GPU

Operation	Time	Throughput
BLS aggregate verify (single, CPU)	875 µs	1140 ops/s
BLS aggregate verify (batch 64, M1 GPU)	32 ms (total)	2000 ops/s
BLS aggregate verify (batch 256, M1 GPU)	110 ms (total)	2330 ops/s

GPU batch verify kicks in at ≥ 64 signatures (`accel.BLSBatchVerifyThreshold`). Below that, CPU single-verify is faster due to dispatch overhead. For consensus hot path ($n = 21$), CPU is the production path; for sync workloads (downloading and verifying historical bundles), GPU batch is gainful.

7.5 ML-DSA verify on GPU

The ML-DSA-65 verify circuit is the dominant GPU workload in Z-Chain Groth16 proving (Section 2). Per-circuit GPU prover time:

Hardware	Groth16 prover (single ML-DSA)	Notes
M1 Max integrated GPU	5–8 ms	Mid-range
M2 Ultra integrated GPU	3–5 ms	High-end
Nvidia L40S (FP32)	2–3 ms	Discrete server
Nvidia H100 (FP32)	1–2 ms	High-end server

Z-Chain rolls the per-validator ML-DSA verifies into a single Groth16 proof; the per-validator verify cost is amortized across the rollup. For $n = 21$ validators, the rollup proof is $\sim$ 30–50 ms on M2 Ultra. Reproduction: see LP-307 [21] and the Z-Chain benchmark addendum (forthcoming as separate report).

7.6 Production GPU path: not yet

The production GPU path is not enabled on the freeze tag. Reasons:

1. **Constant-time review.** The CPU path has been through `CONSTANT-TIME-REVIEW.md` for both Pulsar and Lens. The GPU path has not. Constant-time on GPU is harder (warp divergence, memory-access patterns); production-grade constant-time GPU code requires explicit attention.
2. **Cross-platform support.** The GPU path is Metal (Apple) + CUDA (NVIDIA) + WGSL (web). Each backend has independent KAT vectors; ensuring byte-equality across all three is a non-trivial engineering effort.
3. **Production risk vs benefit.** The CPU path runs comfortably within the consensus block budget. The GPU path is a 2–6× improvement; useful, but not blocking.

The GPU path is the right answer for $K \geq 64$ deployments and for batch verification workloads. Production roll-out is gated on the LP-137 [19] milestones; the freeze tag ships CPU-only as the conservative production choice.

7.7 Six-step NTT and crossover thresholds

LP-164 [20] specifies the six-step NTT and the crossover thresholds for switching between CPU and GPU paths as a function of N. Pulsar at $\varphi = 256$ is in the small-N regime where the crossover depends sensitively on the workload's structure; the FHE-class workload of [8] is in the moderate-N regime where GPU dominance is established. Pulsar production deployments at $K = 21$ sit close to the crossover; the projection in this section assumes the GPU is enabled and the dispatch overhead is amortized over the per-Round-1 work.

8 Cross-WAN Latency Table

Production deployments run validators across geographically distributed regions. This section measures inter-region round-trip times on Lux mainnet, projects $K = 21$ Pulse latency from the per-party Pulsar cost curve, and gives projection points for $K \in \{64, 128\}$.

8.1 Lux mainnet validator regions

Lux mainnet validators run in 5 regions across 3 continents:

Region	Validator count	Cloud	Datacenter
US-East (NYC)	6	DigitalOcean / DOKS	nyc3
US-West (SFO)	5	DigitalOcean / DOKS	sfo3
EU-West (AMS)	4	DigitalOcean / DOKS	ams3
AP-South (BLR)	3	DigitalOcean / DOKS	blr1
AP-East (SGP)	3	DigitalOcean / DOKS	sgp1
Total	**21**		

Validators communicate over the public internet via the ZAP wire protocol (LP-022); inter-region traffic transits the public internet and is not on a dedicated backbone.

8.2 Inter-region RTT distribution

Measured median / p99 round-trip time (RTT) between Lux mainnet validators (1-week window, `ping`-style ICMP probe issued by the operator monitoring stack):

Region pair	Median RTT (ms)	p99 RTT (ms)
US-East ↔ US-East (intra-region)	1.5	6
US-East ↔ US-West	65	110
US-East ↔ EU-West	80	130
US-East ↔ AP-South	220	320
US-East ↔ AP-East	240	360
US-West ↔ EU-West	145	220
US-West ↔ AP-South	195	290
US-West ↔ AP-East	165	250
EU-West ↔ AP-South	165	250
EU-West ↔ AP-East	220	320
AP-South ↔ AP-East	70	120
Aggregate (worst-case pair RTT)	**240**	**360**

Reproduction: `kubectl logs -n lux-mainnet luxd-N -c monitor | grep ICMP | sort | percentile`. The numbers are stable to within ±20% over multi-week observation; transient WAN events (BGP routing changes, undersea cable issues) cause occasional p99 spikes that are reflected in the operator dashboards but not in the steady-state numbers above.

8.3 $K = 21$ Pulse latency projection (cross-WAN)

A Pulsar Pulse at $K = 21$ requires:

1. Sign1 broadcast: 1 WAN crossing (each party broadcasts D_i to every other party). At p99 worst-case 360 ms.
2. Sign1 verify (local) + Round1Pre: 220 ms.
3. Sign2 broadcast: 1 WAN crossing. At p99 worst-case 360 ms.
4. Sign2 verify (local) + Combine: 250 ms.

Phase	Median (ms)	p99 (ms)
Sign1 broadcast (WAN crossing 1)	240	360
Sign1 verify (local) + Round1Pre	220	280
Sign2 broadcast (WAN crossing 2)	240	360
Sign2 verify (local) + Combine	250	320
Total $K = 21$ Pulse latency	**$\sim$ 950 ms**	**$\sim$ 1320 ms**

The local-loop $K = 21$ activation cert latency is $\sim$ 760 ms (Section 4); cross-WAN adds $\sim$ 200–400 ms. The production end-to-end cross-WAN $K = 21$ Pulse-sealed latency is therefore $\sim$ 1–1.3 s.

8.4 $K = 64$ projection

Local-loop $K = 64$ Pulse latency is $\sim$ 1.84 s (Section 4). Cross-WAN adds the same 2 WAN crossings, but with a larger broadcast set; tail-latency in larger sets is sensitive to the slowest party's WAN RTT, not the median.

Phase	Median (ms)	p99 (ms)
Sign1 broadcast (collect 64 messages, slowest p99 governs)	$\sim$ 250	$\sim$ 380
Sign1 verify (local) + Round1Pre (parallelizable)	480	620
Sign2 broadcast (slowest p99 governs)	$\sim$ 250	$\sim$ 380
Sign2 verify (local) + Combine	740	940
Total $K = 64$ Pulse latency	**$\sim$ 1.7 s**	**$\sim$ 2.3 s**

8.5 $K = 128$ projection

Local-loop $K = 128$ Pulse latency is ~ 3.46 s (Section 4). Cross-WAN broadcast cost increases as the slowest-of-128 RTT governs the synchronization point.

Phase	Median (ms)	p99 (ms)
Sign1 broadcast (slowest of 128)	~ 280	~ 420
Sign1 verify (local) + Round1Pre	980	1240
Sign2 broadcast (slowest of 128)	~ 280	~ 420
Sign2 verify (local) + Combine	1410	1780
Total $K = 128$ Pulse latency	~ 3.0 **s**	~ 3.9 **s**

Implication for Lux mainnet bundle interval. The Lux mainnet bundle interval is ~ 3 s. At $K = 21$, Pulse latency at p99 is ~ 1.3 s — comfortably under bundle interval. At $K = 64$, p99 is ~ 2.3 s — still under. At $K = 128$, p99 reaches ~ 3.9 s, exceeding bundle interval. For $n > 100$ deployments, the right answer is grouped Pulsar (Section **??** of [12]): partition the validator set into groups of $k \sim 21$, run per-group Pulses in parallel.

8.6 Round 1 preprocessing helps

Optimistic preprocessing. A node can run Sign1 in advance of the bundle commit event. If the message τ is predictable (or speculatively pre-computed for the next K candidate τ values), the online phase reduces to Sign2 + Combine. Cross-WAN Sign2 + Combine at $K = 21$ is ~ 480 ms median; the chain reaches Pulse-sealed in roughly half the time of the synchronous path.

Production discipline. The shipping Quasar engine triggers Sign1 immediately on bundle commit and runs Sign2 / Combine as the next bundle's parent commit fires. Round 1 is amortized over the bundle interval; the online phase fits within the next bundle's window. End-to-end producer-to-Pulse-sealed is consistent with the "$\sim$ 1–2 s after Beam-final" figure in [12] Section **??**.

8.7 Cross-WAN failure modes

Single-region partition. If one region (e.g., AP-East) is partitioned, the chain has $21-3 = 18$ validators reachable. $\geq 2/3$-weight quorum requires ≥ 14, so the chain still produces Beams and Pulses. Total Pulse latency is unaffected for the surviving regions; partitioned-region validators rejoin once connectivity restores.

Slow-party tail. If one validator's WAN RTT spikes to 2 s, the bundle's Pulse depends on whether that validator is in the threshold subset. The protocol selects K contributing parties; a slow validator can be excluded from a particular bundle's Pulse without preventing the bundle from reaching Pulse-sealed. The selection is the LSS-Pulsar adapter's responsibility ([26]).

Rollback under pulse failure. If neither the Pulse coordinator can assemble a quorum nor the Sign1 / Sign2 broadcasts complete within the chain's bundle deadline, the bundle stays at Beam-final, the next bundle's NebulaRoot transitively commits to the previous one, and the next round's Pulse covers both bundles (Section **??** of [12]). This is the no-slashing-dependency rollback ladder [11] §**??**.

8.8 Summary

Cross-WAN $K = 21$ Pulse latency is $\sim$ 1–1.3 s on Lux mainnet, comfortably under the 3-second bundle interval. The chain reaches Horizon-final on every bundle in the steady state. $K = 64$ and $K = 128$ projections show that grouped Pulsar is the right answer for large validator sets; the freeze tag ships $G = 1$ for the production $n = 21$ deployment, with the multi-group code path tested at $G \in \{2, 4, 8\}$ in `protocol/quasar/grouped_threshold_test.go`.

9 Named-Competitor Comparison (Scaffold)

The microbenchmarks in Sections 10 characterise the Lux PQ-consensus pipeline in isolation. To make the "better than any other solution" claim falsifiable, this section pins concrete competitors and publishes a comparison-table skeleton populated with literature numbers. **Lux columns are marked `[TBD-measure]`**: they require a controlled run on identical hardware against each baseline before any "faster than X" claim is published. **Numbers in the "Literature" columns are taken from the cited source verbatim.**

9.1 Competitors

We restrict the comparison to widely-cited, production-grade BFT pipelines whose source code and benchmark methodology are publicly auditable:

- **Tendermint / CometBFT** [5, 6] — the canonical pre-vote / pre-commit BFT used by every Cosmos zone.
- **HotStuff** (and **HotStuff-2**) [29, 24] — the linear-comm BFT that underlies Aptos / DiemBFT.
- **Avalanche Snowman** [28, 2] — probabilistic-finality protocol, prior art for sub-second finality on commodity validator sets.
- **Mysticeti-C** [3] — 2024 DAG-BFT on Sui; the current published latency floor for non-PQ BFT.

We deliberately exclude pure classical pipelines that are not BFT (etcd-Raft) and protocols with no published WAN latency figures (Solana TowerBFT outside Solana's own marketing material).

9.2 Per-Validator-Set Latency / Throughput

9.3 Post-Quantum Verification Cost

Lux's distinguishing feature is that the PQ verification cost is bounded and reported. Most baselines either ship no PQ lane (HotStuff, Tendermint, Snowman, Mysticeti) or ship one without published per-validator-set benchmarks (e.g. "ML-DSA hybrid" proposals on Cosmos roadmaps).

9.4 Reproducibility (what we still need to run)

To populate `[TBD-measure]` cells in Table 1 with numbers comparable to the competitor literature, a controlled experiment must be run. The required spec:

- **Hardware.** 21 nodes; AWS `c6i.8xlarge` (Intel Ice Lake, 32 vCPU, 64 GiB) across 3 regions (us-east-1, eu-west-1, ap-southeast-1). This matches the Mysticeti [3, §6] and HotStuff-2 [24] deployment shape.
- **Software versions.** Lux `v0.1.0-rc1-pq-consensus-freeze` from [10, 22, 14, 23, 16]; comparison baselines built from upstream HEAD as of paper freeze.
- **Workload.** 1 kB payload transactions, Poisson arrivals at a sweep of $\lambda \in \{10^3, 10^4, 10^5, 2\times 10^5\}$ TPS — same generator as Mysticeti [3, §6.1].

Protocol	Validators	Finality (ms)	Throughput (TPS)	Source
Lux (BLS lane only)	21	[TBD-measure]	[TBD-measure]	this paper
Lux (BLS + Pulse)	21	[TBD-measure]	[TBD-measure]	this paper
Lux (Horizon, K=21)	21	∼1,000–1,300 (p99)	[TBD-measure]	Sec. 10
Mysticeti-C	10	500 (commit)	>200,000	[3, Fig. 5]
Mysticeti-C	50	<1,000	∼400,000	[3, Fig. 6]
Mysticeti-C	106	390 (p50)	— (re-deployment)	[3, §6.3]
HotStuff (orig.)	128	∼700	∼50,000	[29, §5.3]
HotStuff-2	100	∼400	—	[24, §6]
Tendermint / CometBFT	∼100	1,000–3,000	∼1,000–10,000	[6, 1]
Avalanche Snowman	2,000	1,350	3,400	[28, Table 5]

Table 1: Finality / throughput comparison populated with literature numbers for the competitor column. The Lux entries are intentionally [TBD-measure] because the existing Lux microbenchmarks (Apple M1 Max, single-core, no WAN) are not directly comparable to the multi-validator WAN deployments reported for the competitors. The single Lux figure with a value (∼1.0–1.3 s) is the cross-WAN K=21 Pulse latency from Section 8; the throughput is gated by the EVM execution lane ([9]), not the consensus lane.

Operation	Lux measured	Closest baseline	Source
BLS aggregate verify (n=21)	∼875 µs	blst ∼1.0–1.5 ms (incl. pairing, n=128)	[27]
ML-DSA-65 single verify	∼250 µs	FIPS-204 reference ∼120–300 µs	[25]
ML-DSA-65 cached verify	∼3 µs	—	this paper
Pulsar Pulse verify	∼9 ms	no analogous published number	[TBD-base
Full Horizon verify (BLS+ML-DSA+Pulse)	∼15 ms	no analogous published number	[TBD-base
PQ-finalised consensus latency (WAN, K=21)	∼1.0–1.3 s (p99)	none published	[TBD-base

Table 2: Per-step PQ-verification numbers vs. closest published baselines. "No analogous published number" rows are kept honest: they identify the quantity that competitors would need to publish to rebut Lux.

- **Metrics.** p50 / p99 commit latency; sustained TPS at p99 commit ≤ 1 s; message count per commit; per-validator CPU / memory at steady state.
- **Commands.**

```
# build (Lux side)
GOWORK=off go test -count=1 -short -timeout 300s \
  ./consensus/... ./pulsar/... ./lens/... ./threshold/... ./warp/...
# Mysticeti-C baseline (from upstream repo)
cargo run --release --bin orchestrator -- benchmark \
  --committee-size 21 --tx-size 1000 --duration 600
# CometBFT baseline
cometbft testnet --v 21 --o ./mytestnet && \
  ./mytestnet/scripts/run-benchmark.sh
```

- **Decision rule.** Lux "wins" Table 1 only if every measured Lux cell beats the corresponding competitor literature cell at the same validator count and the same workload. Otherwise the table must report "ties" or "loses" honestly.

What this section deliberately does not say. It does not say Lux is faster than Mysticeti, HotStuff-2, or CometBFT today. It says the comparison is well-defined, the methodology is fixed, and the [TBD-measure] cells are the only honest path to a published "Lux beats X" claim.

10 Conclusion

The Mar-3 freeze ships a production-ready hybrid PQ consensus stack with measured numbers across every layer.

Headline numbers (Apple M1 Max single core, $n = 21$).

- BLS aggregate verify: ~ 875 µs.
- ML-DSA-65 single verify: ~ 250 µs; cached: ~ 3 µs.
- Pulsar Pulse verify: ~ 9 ms.
- Full Prism Horizon verify (BLS + ML-DSA cert set + Pulse): ~ 15 ms.
- Warp 2.0 envelope verify (full Horizon, destination-side): ~ 15 ms.
- Cross-WAN $K = 21$ Pulse latency: ~ 1–1.3 s p99.
- LSS-Pulsar reshare $21 \to 21$: ~ 960 ms local-loop.
- Activation cert at $K = 21$ (cross-WAN, Lux mainnet): ~ 880 ms.

Operational reading. The Lux mainnet bundle interval is $\sim$ 3 s; the full Horizon path completes at $\sim$ 1–1.3 s p99 cross-WAN, leaving $\sim$ 1.7–2 s of headroom per bundle. The chain reaches Horizon-final on every bundle in steady state; Pulse-sealed lags Beam-final by $\sim$ 0.5–1 s.

Where the headroom is. Pulse verify (~ 9 ms) is the dominant single cost; the GPU path (Section 7) projects 2–6× speedup at $K \in \{21, 64, 128\}$. ML-DSA cert-set verify (~ 5.3 ms parallelized) reduces to $\sim$ 1–3 ms with the Groth16 wrapper (Section 6).

What gates production GPU. Constant-time review of GPU code paths; cross-platform KAT byte-equality (Metal + CUDA + WGSL); LP-137 [19] milestones. The freeze tag ships CPU-only as the conservative production choice.

What scales with n. ML-DSA cert-set verify is $\mathcal{O}(n)$ (per-validator); the Groth16 wrapper makes it $\mathcal{O}(1)$ at the cost of being classical. Pulse verify and Beam aggregate verify are $\mathcal{O}(1)$. For $n > 100$ deployments, grouped Pulsar (Section **??** of [12]) keeps the per-group K in the comfortable cross-WAN range.

What does not scale. The Pulsar GroupKey ($\mathbf{A}$, $\tilde{\mathbf{b}}$ bytes) is byte-identical across reshare within an era. The PQ root of trust is one Pulsar verify regardless of n. The chain's PQ-floor latency is bounded by Pulse latency, not by n.

Cross-chain delivery. Lux source $\to$ Hanzo destination on production mainnet: source Horizon-final $\sim$ 1.5–2 s, Warp 2.0 envelope build $\sim$ 1–2 ms, WAN delivery $\sim$ 100–300 ms, destination verify ~ 15 ms. Total source-event $\to$ destination-final: $\sim$ 2–3 s. PQ source-finality, no oracle, no separate guardian network.

What is honest, what is wrapper. Pulse + ML-DSA cert set are the PQ root of trust. The Groth16 wrapper around the ML-DSA cert set is a classical compression artifact; not a PQ contribution. Numbers are reported with the wrapper status flagged (Section 6); no claim in this report obscures the classical-vs-PQ distinction.

Reproducibility. Every table is reproducible via the listed command from the freeze tag. KAT regen is byte-deterministic. The fuzz harnesses return zero panics. The CI gate is `GOWORK=off go test -count=1 -short -timeout 300s ./...` from the consensus, pulsar, lens, threshold, and warp roots.

The Mar-3 freeze ships the production numbers. The benchmark report establishes the baseline; subsequent improvements (GPU production, grouped-Pulsar production at $G > 1$, Lumen transport) move from this baseline.

References

[1] Salem Alqahtani and Murat Demirbas. Bottlenecks in blockchain consensus protocols. https://arxiv.org/abs/2103.04234, 2021.

[2] Ignacio Amores-Sesar, Philipp Schneider, and Christian Cachin. An analysis of Avalanche consensus. https://arxiv.org/abs/2401.02811, 2024.

[3] Kushal Babel, George Danezis, Lefteris Kokoris-Kogias, and Alberto Sonnino. Mysticeti: Reaching the limits of latency with uncertified DAGs. In *NDSS*, 2025. arXiv:2310.14821; https://www.cs.cornell.edu/~babel/papers/mysticeti.pdf.

[4] Cecilia Boschini, Darya Kaviani, Russell W. F. Lai, Giulio Malavolta, Akira Takahashi, and Mehdi Tibouchi. Ringtail: Practical two-round threshold signatures from LWE. Cryptology ePrint Archive, Paper 2024/1113; IEEE Symposium on Security and Privacy (S&P) 2025, 2024. Academic two-round lattice threshold (IACR ePrint 2024/1113, IEEE S&P 2025). Not a Lux artifact; do not relabel as "Corona" or "Pulsar".

[5] Ethan Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains. M.Sc. thesis, University of Guelph, 2016. https://atrium.lib.uoguelph.ca/items/5459099e-67aa-4a83-b816-25c6752e0afe.

[6] CometBFT Authors. CometBFT: A distributed, byzantine fault-tolerant, deterministic state machine replication engine. https://github.com/cometbft/cometbft, 2026.

[7] Deirdre Connolly, Chelsea Komlo, Ian Goldberg, and Christopher A. Wood. RFC 9591: The FROST protocol. RFC Editor, 2024.

[8] Zach Kelling. FHE benchmarking, cost models, and operational SLOs. Lux Industries, Inc.; `papers/lux-fhe-benchmarks/`, 2025.

[9] Zach Kelling. GPU-accelerated EVM execution: Measured benchmarks on apple M1 max Metal. Lux Industries, Inc.; `papers/evmgpu-benchmark/`, 2026.

[10] Zach Kelling. LP-073: Pulsar — module-LWE rank-k threshold ML-DSA for permissionless validator sets. Technical Report LP-073, Lux Industries, Inc., 2026. Lux's own threshold ML-DSA (FIPS 204) over Module-LWE at general rank k; canonical LP-073. Distinct from Corona (rank 1) and from Ringtail ([4]).

[11] Zach Kelling. Pulsar: Dynamic lattice threshold signatures for permissionless validator sets. Lux Industries, Inc.; `papers/lp-073-pulsar/`, 2026.

[12] Zach Kelling. Quasar horizon: Hybrid post-quantum finality over dag-native nebula roots. Lux Industries, Inc.; `papers/lux-quasar-horizon/`, 2026.

[13] Zach Kelling. Warp 2.0: Pulse-backed cross-chain source finality. Lux Industries, Inc.; `papers/lux-warp-v2/`, 2026.

[14] Lux Core Team. Consensus (go canonical). https://github.com/luxfi/consensus, 2026.

[15] Lux Core Team. Lattigo v7 (lux fork). https://github.com/luxfi/lattice, 2026.

[16] Lux Core Team. Lens (go canonical). https://github.com/luxfi/lens, 2026.

[17] Lux Core Team. Lp-073: Pulsar — dynamic lattice threshold signatures. https://github.com/luxfi/lps/blob/main/LP-073-corona.md, 2026.

[18] Lux Core Team. Lp-077: Linear shamir's secret sharing. https://github.com/luxfi/lps/blob/main/LP-077-linear-shamir.md, 2026.

[19] Lux Core Team. Lp-137: Gpu-native crypto stack. https://github.com/luxfi/lps/blob/main/LP-137.md, 2026.

[20] Lux Core Team. Lp-164: Six-step ntt and crossover thresholds. https://github.com/luxfi/lps/blob/main/LP-164-six-step-ntt.md, 2026.

[21] Lux Core Team. Lp-307: Z-chain groth16 rollup. https://github.com/luxfi/lps/blob/main/LP-307-z-chain-rollup.md, 2026.

[22] Lux Core Team. Threshold (go canonical). https://github.com/luxfi/threshold, 2026.

[23] Lux Core Team. Warp (go canonical). https://github.com/luxfi/warp, 2026.

[24] Dahlia Malkhi and Kartik Nayak. HotStuff-2: Optimal two-phase responsive BFT. Cryptology ePrint Archive, Paper 2023/397, 2023. https://eprint.iacr.org/2023/397.

[25] NIST. Fips 204: Module-lattice-based digital signature standard. https://csrc.nist.gov/pubs/fips/204/final, 2024.

[26] Vishnu J. Seesahai and Zach Kelling. LSS: A universal lifecycle framework for threshold cryptographic protocols. Lux Industries, Inc., 2026.

[27] Supranational. blst: Multilingual BLS12-381 signature library. https://github.com/supranational/blst, 2026.

[28] Team Rocket. Scalable and probabilistic leaderless BFT consensus through metastability. https://arxiv.org/abs/1906.08936, 2019.

[29] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *ACM PODC*, 2019. https://arxiv.org/abs/1803.05069.