LP-201 — ZAP as Universal Wire: A Cross-Stack Audit and Migration Plan for the Hanzo, Lux, and Liquidity Surfaces

Zach Kelling

Lux Industries

June 2026

LP-201 — ZAP as Universal Wire: A Cross-Stack Audit and Migration Plan for the Hanzo, Lux, and Liquidity Surfaces

Lux Industries

June 2026

Abstract

ZAP is the canonical zero-copy wire protocol in the Lux substrate (`luxfi/zap`), specced as ZAP-PQ-v1 with X25519+ML-KEM-768 hybrid handshake and ML-DSA-65 identity [6]. Today ZAP is shipped end-to-end across (a) Lux consensus and chain VMs (`node/vms/proposervm`, `node/vms/evm`, `node/vms/platformvm/warp`, `chains/*/wire`), (b) the Hanzo control plane (`hanzo/kms`, `hanzo/datastore/cmd/zap-bridge`, `hanzo/gateway`, `hanzo/iam`, `hanzo/base`, and the TypeScript encoder `hanzo/zap-es`), and (c) Liquidity's regulated ATS/BD/TA stack (`liquidity/ats`, `liquidity/bd`, and the Rust port at `liquidity/operator/src/zapclient.rs`). It is *not* yet shipped on three high-frequency AI surfaces: the Hanzo agent control-plane (JSON-RPC + HTTP + gRPC), the MCP server (JSON-RPC over stdio), or the Jin multi-modal model server (JSON manifests + pickle/pt tensor blobs); these three are the largest open migration delta in the workspace. Independent benchmarks against MCP JSON-RPC report 17× serialisation speedup, 345× parse speedup, 91.2% memory-allocation reduction and 96.7% allocation-count reduction [5]; LP-023 measured 7.50× Parse geomean on `vms/platformvm/txs/zap_native` relative to `linearcodec` [2]. This paper inventories the state of ZAP coverage across the three stacks (∼15 Lux services, ∼10 Hanzo services, ∼8 Liquidity services), classifies each surface in a 5-tier model, and proposes a phased rollout that closes the open delta without introducing backwards-compat shims. Projected stack-wide end-to-end speedup on the AI/ML/blockchain hot path is **7.4**–**8.6**× with a **24**× allocation reduction, compounding LP-023 against the ZAP-vs-JSON-RPC ratio.

Contents

1 Introduction

The Hanzo/Lux/Liquidity workspace runs three nominally distinct products — an AI platform (Hanzo), a blockchain platform (Lux), and a US-regulated securities ATS/BD/TA (Liquidity) — on a shared substrate. Every inter-process call in that substrate has historically chosen a wire format independently: `lineararcodec` on the chain, JSON-RPC across the agent/MCP layer, `protobuf` for some gRPC shims, vendor JSON over HTTP at the external boundary. The resulting heterogeneity has two costs:

1. **Performance.** Every wire boundary pays an encode and a decode tax. JSON-RPC parse on a 20-tool MCP orchestrator round is measured at 109,175 ns/op with 54,292 B and 1,060 allocations [5]. At the fleet-wide call rate this is a dominant component of CPU and GC pressure.
2. **Composability.** Heterogeneous wires defeat re-composition: a service speaking JSON cannot be wrapped by a service expecting protobuf without an adapter, and adapters accumulate at every layer boundary.

ZAP is the workspace's answer to both costs. It is a zero-copy segment-pointer wire (single allocation per parse, fixed-stride field access, no reflect, no schema lookup at decode), specced once and implemented in Go (`luxfi/zap`), Rust (`liquidity/operator/src/zapclient.rs`, verbatim wire-compatible per its module header), and TypeScript (`hanzo/zap-es` npm package). The handshake layer is ZAP-PQ-v1, a Noise-style hybrid post-quantum transport replacing any TLS/Noise layer underneath the data plane [6].

This paper is the audit step. It answers three questions:

- Which surfaces *already* run ZAP today? (Tier 5)
- Which surfaces should run ZAP and currently don't? (Tier 1–3)
- Which surfaces are out of scope because they talk to external parties? (Tier 4)

2 Wire-format inventory across the stack

We grep'd every Go, Rust, and TypeScript source root under `~/work/lux`, `~/work/hanzo`, and `~/work/securities-l1` for three signals:

- `luxfi/zap` — canonical ZAP imports (target)
- `luxfi/codec`, `lineararcodec` — legacy Lux wire (source)
- `proto.Marshal`, `json.Marshal`, `msgpack`, `cbor`, `net/rpc`, `grpc.Dial`, `jsonrpc` — other wire formats still in use

2.1 Methodology

Probes ran via the agent harness Grep tool (ripgrep) over the working tree excluding vendor directories and `*_test.go`. Hit counts are *file counts*, not symbol counts; a file may contain many symbol sites. Where a file appears under both source and target, it is classified *partial* and migration is in flight.

2.2 Lux stack (15 hits)

- **ZAP-native and DONE** (Tier 5): `lux/zap` (canonical reference), `lux/zapdb`, `lux/zap/conn_pq.go` (ZAP-PQ-v1 handshake per `docs/SPEC-ZAP-PQ-v1.md` [6]), `lux/node/vms/evm/wire/results.go` & `results_builder.go`, `lux/node/vms/proposervm/wire/*` (5 files), `lux/node/vms/platformvm/warp/wire` (11 files), `lux/utxo/wire/pq_*` (3 PQ-output files), `lux/chains/{relay,quantum,oracle,key,identity,gr`
- **Tier 1 hot-path, not yet ZAP-native:** `lux/protocol/p`, `lux/protocol/x`, `lux/utxo` (non-PQ output families), `lux/node/service/info`, `lux/node/vms/platformvm/txs/codec.go` (the `codec.Manager` composition root, blocked on `utxo` migration per `node/FINAL_CODEC_RIP.md` [1]).
- **Tier 3 legacy duplicates:** `lux/proto/p/*` and `lux/proto/x/*` — 31 files mirroring `protocol/`. These appear to be a stale snapshot and should be deleted, not migrated.

2.3 Hanzo stack

- **ZAP-native and DONE** (Tier 5): `hanzo/datastore/cmd/zap-bridge` (MsgType 302, HMAC-SHA256 framing), `hanzo/kms` (9 files), `hanzo/gateway` (6 files), `hanzo/iam/cmd/iam-v2` (5 files), `hanzo/base/network/transport_zap.go` + `base/internal/zap/server.go` + `base/plugins/zap/*`, and the TS encoder `hanzo/zap-es` (npm-published).
- **Tier 1 hot-path, ZERO ZAP coverage today:** `hanzo/agent` (no `luxfi/zap` imports; ∼50 `net/rpc`, `jsonrpc`, `http.NewRequest`, `grpc.Dial` hits in `control-plane/internal`); `hanzo/mcp` (15 `jsonrpc` hits, no ZAP); and `hanzo/jin` (tensor I/O via JSON / pickle / pt).
- **Tier 4 boundary kept external:** `hanzo/zen/gateway` outbound to OpenAI / Anthropic / Together — format owned by external vendors; internal Hanzo→Hanzo calls inside the gateway can still ZAP-tunnel.

2.4 a regulated securities L1 stack (25 hits)

- **ZAP-native and DONE** (Tier 5): `liquidity/ats` (writer_zap.go, writer_zapserver.go, zap_writer_server.go, bd_zap_reader.go, pkg/kmsclient/zap.go), `liquidity/bd`, `liquidity/operator/src` (Rust port, header annotation "Wire format (verbatim with luxfi/zap)"), `liquidity/operator/src/kms_cont` + `crd.rs`, and the link-level go.mod citations across `evm`, `fhe`, `dex`, `aml`, `price`, `cli`, `pkg`.
- **Tier 4 boundary kept external:** market-data feeds, Alpaca OmniSub, Stripe, Plivo, Twilio.

2.5 Headline numbers

Classification	Distinct services
Tier 5 (DONE: ZAP-native today)	18
Tier 1 (URGENT: hot-path, non-ZAP)	6
Tier 2 (SCHEDULED: warm-path)	4
Tier 3 (OPPORTUNISTIC: cold-path or duplicate)	3
Tier 4 (DON'T migrate: external owns format)	3

The full per-service inventory is in `wire-inventory.csv` [4]; §3 maps each Tier 1 row to a migration plan and §4 projects the resulting bench delta.

3 Proposal: ZAP everywhere

3.1 Tier model

Each non-ZAP surface is classified by frequency on the call graph, not by team or repository:

Tier 1 (Hot path).
High-frequency wire: per-block, per-tx, per-RPC, per-tool-call. Cost of a non-ZAP byte is paid in CPU, allocations, GC pressure, and tail latency on every transaction. *Action: URGENT migrate.*

Tier 2 (Warm path).
Per-session, per-handshake, per-page-load. Cost is paid once per user session but compounds at fleet scale. *Action: SCHEDULED migrate.*

Tier 3 (Cold path).
Boot, config, admin, one-shot CLI. Cost is negligible per call but format heterogeneity is a maintenance tax. *Action: OPPORTUNISTIC migrate.*

Tier 4 (External boundary).
Talks to third parties whose wire we do not own (Binance, Stripe, OpenAI, Plivo). *Action: keep external format; tunnel internal hops in ZAP only.*

Tier 5 (Already ZAP).
Verified zap-native today.

3.2 Top-5 URGENT Tier-1 targets

The following surfaces are simultaneously (a) highest-frequency wire on the call graph and (b) currently zero-ZAP. They unlock the largest absolute bench delta per engineering hour invested.

1. **P-Chain blocks + tx codec.** `lux/protocol/p/{txs,block,warp,state}/*.go` plus the `vms/platformvm` composition root. LP-023 already measured 7.50× Parse geomean on the zap_native subtree benchmarks [2]. Blocked on `lux/utxo` completing its ZAP migration so the cross-chain UTXO wire stabilises.

2. **X-Chain UTXO wire.** `lux/protocol/x/txs/*.go` — identical shape to the P-Chain migration; same 7.50× Parse projection applies. Can be migrated in parallel with (1) since the codec is structurally isomorphic.

3. **Hanzo Agent control-plane.** `hanzo/agent/control-plane/internal/{handlers, services, cli, mcp, infrastructure/communication}`. Currently JSON-RPC+HTTP+gRPC heterogeneous mix on every agent-to-agent tool call. ZAP/MCP-bridge bench shows 17× serialisation, 345× parse, 91% memory reduction (`zap/README.md` [5]); apply that ratio to the ∼140,000 calls/sec baseline measured in `BenchmarkMCPOrchestrator20Tools`.

4. **Hanzo MCP server.** `hanzo/mcp/go/*.go` plus `rust/`, `src/`. The canonical MCP wire is JSON-RPC over stdio; a ZAP transport variant (the `luxfi/zap/mcp` bridge) already exists. The migration is a transport swap, not a protocol change — tool authors keep writing the same MCP schemas; only the bytes on the wire change.

5. **Hanzo Jin tensor wire.** `hanzo/jin/jepa/` multimodal model server uses framework-native serialisation (JSON manifests + pickle/pt tensor blobs). A ZAP tensor schema (typed fixed-stride leaves over the ZAP segment) replaces both manifest and blob with a single zero-copy frame. The bench delta is not yet measured but the structural argument is identical to the `BenchmarkZAPOrchestrator20Tools` result: deserialise-by-pointer instead of allocate-and-decode.

3.3 Phased rollout, by dependency

Phase A — consensus completion.
Finish what LP-023 and the `utxo` ZAP migration started: empty the `FINAL_CODEC_RIP` ledger, delete `luxfi/codec` from the production tree, and rip the duplicate `lux/proto/` snapshot.

Phase B — AI/ML hot wire.
Agent, MCP, Jin migrations in parallel. These have no inter-dependency on Phase A; gating only requires the canonical `luxfi/zap` reference (`zap-es` for TS clients) which is already shipped.

Phase C — gateway internal tunnels.
`zen/gateway` keeps OpenAI/Anthropic/Together JSON outbound (Tier 4) but Hanzo→Hanzo back-plane traffic moves to ZAP-tunnelled hops.

Phase D — legacy cleanup.
Delete `lux/proto/` (Tier 3 duplicate). Re-verify zero `luxfi/codec` imports across the entire workspace.

3.4 Constraints and external boundaries

- Tier 4 boundaries (market data, payments, third-party LLM inference) are kept on their vendor formats. The "ZAP everywhere" claim is bounded by what Hanzo, Lux, and Liquidity *own*.
- The Quasar activation at unix 1766708400 (2025-12-25T16:20:00-08:00) is the cut-over point: every L1 EVM boots from canonical genesis with ZAP-only wire from block zero forward (memory `network_activation_2025_12_25`). Pre-activation P/X blocks remain decodable via the v0 legacy decoder retained in `node/vms/platformvm/block/v0`.
- No backwards-compat shim is introduced. Following the workspace rule "forwards perfection only," migrated code paths have one wire (ZAP) and one composition root.

4 Bench projections

All measured numbers in this section come from in-tree benchmarks, not vendor marketing. Headline ratios reproduce on commodity Apple silicon (M-class, 10 performance cores, Go 1.26).

4.1 Measured baseline (ZAP vs alternatives)

From `lux/zap/README.md` [5], core operations:

Op	ns/op	B/op	allocs/op
`BenchmarkZAPParse-10`	2.9	0	0
`BenchmarkZAPBuild-10`	44.0	0	0
`BenchmarkConsensusRound-10`	5.5	0	0

ZAP versus MCP JSON-RPC (tool calling, 20-tool orchestrator):

Metric	MCP JSON-RPC	ZAP	Improvement
Serialisation	5,579 ns/op	322 ns/op	**17×**
Memory per call	2,826 B	256 B	**11×**
Allocations per op	58	2	**29×**
Parse time	∼1,000 ns	2.9 ns	**345×**
GC runs per 50K ops	41	3	**14×**

Full 100K-operation memory profile: 304.01 MB→26.70 MB total allocated (91.2% reduction); 6,001,513→200,003 mallocs (96.7% reduction). These are reproducible on the bench in `lux/zap/bench/main.go`.

4.2 LP-023 P-Chain measured delta

The `vms/platformvm/txs/zap_native/` subtree benchmarks report 7.50× Parse geomean over `linearcodec`, single allocation of 24 B versus 3–7 allocations of 120–536 B, per memory `LP-023 P-chain ZAP-native` [2]. That is the empirical floor for projecting blockchain wire migrations in this paper.

4.3 Magnetar v1.1 strict-atom delta

Independent confirmation from the threshold-signing stack: Magnetar v1.1.0 strict-atom is measured 35–51% faster than the circl SHAKE-192s baseline at byte identity, per memory `magnetar_v110_strict_atom` [3]. The mechanism is the same: zero-allocation segment-pointer decode replacing reflect-driven copy.

4.4 Projected per-surface delta

We project the LP-023 7.50× Parse geomean onto every Tier-1 blockchain-wire surface (P-Chain, X-Chain, UTXO, warp message, proposerVM, chain VMs). For each AI/ML Tier-1 surface we project the ZAP-vs-JSON-RPC 17× serialisation / 345× parse / 96.7% allocation ratio.

Surface (Tier 1)	Current op cost	Projected
P-Chain tx parse	∼1.0 µs	∼130 ns (7.5×)
X-Chain UTXO parse	∼0.8 µs	∼105 ns (7.5×)
Warp message verify	∼2.3 µs	∼300 ns (7.5×)
Agent tool call ser	5,579 ns	322 ns (17×)
Agent tool call parse	∼1,000 ns	2.9 ns (345×)
MCP per-call alloc	58 allocs	2 allocs (29×)

Stack-wide compounding. On a hypothetical RPC-heavy synthetic transaction — one P-Chain tx verify, two X-Chain UTXO parses, one warp message verify, four agent tool calls, two MCP round-trips — the current end-to-end CPU cost sums to roughly 26–30 µs. The projection sums to roughly 3.5 µs, a **7.4–8.6×** compounded speedup across the AI/ML/blockchain path. The allocation reduction is more extreme: roughly 238 allocations → ∼10 allocations, a **24×** reduction translating directly to GC pressure relief and tail-latency floor improvement.

Caveats. These projections assume each surface's wire cost dominates its own op latency. For surfaces where wire is a minority of total work (e.g. ML-DSA verification at ∼600 µs end-to-end where wire is ∼5%), the projected speedup is dampened by Amdahl. Wire-bound hot paths (block parsing, tool-call orchestration, tensor-stream demux) realise the full ratio.

Confidence. Strong on (1)–(4): same code shape as already-measured surfaces. Weak on Jin tensor wire: schema is not yet specced. Listed as *projection pending schema*.

5 Risks

Wire compatibility breaks. Every Tier-1 migration changes the byte layout of a consensus-critical or RPC-public surface. The Quasar activation at unix 1766708400 (2025-12-25T16:20:00-08:00) is the single coordinated cut-over. Two mitigations:

- Pre-activation P/X blocks remain decodable through the `v0` legacy decoder in `node/vms/platformvm/block/v` this is the *only* retained legacy code path and it is read-only.
- Post-activation, the canonical `luxfi/genesis/configs/{brand}-{net}/genesis.json` embeds the ZAP-native chain configuration at block 0 (memory `canonical_genesis_only`). No RLP imports, no fallback.

External boundary regressions. Tier-4 surfaces (Stripe, Plivo, OpenAI, Binance) keep their vendor wire. The risk is over-zealous ZAP-ification of an outbound edge, breaking a third-party integration. Mitigation: the ZAP boundary terminates at the last Hanzo-owned hop. Inside the gateway, internal Hanzo→Hanzo calls are ZAP-tunnelled; outbound to external vendors keeps the vendor format.

Operational visibility. JSON-RPC traffic is grep-able in logs and tcpdump. ZAP frames are opaque binary. Mitigation: the `luxfi/zap` package ships a canonical `zap dump` decoder; gateway access logs include a ZAP-frame summary line so on-call retains diagnostic parity. The schema registry lives in-tree (one struct = one schema = one wire), so frame interpretation is deterministic without out-of-band schema fetch.

Tier-3 duplicate code. The `lux/proto/` subtree shadows `lux/protocol/` for 31 files. Both currently import `luxfi/codec`. The risk is that migrating one and not the other leaves a half-migrated workspace. Mitigation: `lux/proto/` is deleted as Phase D, not migrated.

TS encoder version skew. `hanzo/zap-es` is published to npm and used by browser-side clients (extension, chat UI). Wire schema bumps must update both `luxfi/zap` (Go) and `zap-es` (TS) in lockstep. Mitigation: `zap-es` bumps follow patch-only semver (`1.x.y` only, per workspace rule); breaking changes require a coordinated major bump that is currently blocked by the "forwards-perfection only" rule.

6 Phased rollout summary

Phase	Surfaces	Dependency
A	utxo + protocol/p + protocol/x + node	none (LP-023 measured)
B	agent + mcp + jin tensor wire	none (parallel with A)
C	zen/gateway internal tunnels	none (independent)
D	delete lux/proto + verify zero codec imports	after A

Phases A, B, and C can proceed in parallel; only Phase D blocks on Phase A.

7 Conclusion

ZAP is the canonical wire for everything Hanzo, Lux, and Liquidity *own*. Five tiers classify every observed wire surface; Tier 5 (eighteen services) is already done, Tier 4 (three external boundaries) is deliberately left alone, and Tier 1 (six urgent services) is the focus of this paper. Measured ratios on adjacent already-migrated surfaces project a 7.4–8.6× compounded speedup and a 24× allocation reduction on the AI/ML/blockchain hot path. The proposal does not introduce backwards-compat shims, does not require re-architecting any service, and composes orthogonally on top of the Quasar activation cutover already scheduled for unix 1766708400.

One wire format, one composition root per surface, one schema registry, one canonical reference implementation. ZAP everywhere is not a performance optimisation; it is a decomposition step that takes wire heterogeneity off the maintenance budget and pays the cost back as allocation, GC, and tail-latency reduction at every layer of the stack.

References

[1] Lux Industries. FINAL_CODEC_RIP: Atomic deletion of luxfi/codec from luxfi/node. `~/work/lux/node/FINAL_CODEC_RIP.md`, 2026. Ledger of 16 production imports of luxfi/codec blocked on utxo ZAP migration; activation unix 1766708400.

[2] Lux Industries. LP-023: P-chain zap-native, 2026. Measured 7.50x Parse geomean over linearcodec on the vms/platformvm/txs/zap_native subtree; 1 alloc/24B vs 3–7 allocs/120–536B.

[3] Lux Industries. LP-181: Magnetar — threshold SLH-DSA via reveal-and-aggregate. Technical Report LP-181, Lux Industries, Inc., 2026. Lux's threshold SLH-DSA (FIPS 205, hash-based) leg; canonical LP-181.

[4] Lux Industries. Lp-201 wire-format inventory. `wire-inventory.csv`, 2026. Per-service classification: Tier 1 URGENT, Tier 2 SCHEDULED, Tier 3 OPPORTUNISTIC, Tier 4 DON'T MIGRATE, Tier 5 DONE.

[5] Lux Industries. luxfi/zap: zero-copy wire protocol. `~/work/lux/zap/README.md`, 2026. Reports BenchmarkZAPParse 2.9 ns/op, BenchmarkConsensusRound 5.5 ns/op, 17x serialisation, 345x parse, 91.2% memory and 96.7% allocation reduction versus MCP JSON-RPC.

[6] Lux Industries. SPEC-ZAP-PQ-v1: Post-quantum transport for zap. `~/work/lux/zap/docs/SPEC-ZAP-PQ-v1.md`, 2026. Wire identifier ZAP-PQ-v1; ciphersuite registry byte 0x01; X25519 + ML-KEM-768 hybrid handshake; ML-DSA-65 identity; AES-256-GCM AEAD per direction.