ZAP: Zero-Allocation Binary Wire Protocol for Consensus Transport

Zach Kelling

Lux Industries

Original: 2023 — Revised: February 2026

Abstract

We present ZAP, a zero-allocation binary wire protocol designed for inter-node consensus communication. ZAP achieves 20.26 million transactions per second in batch mode (50 connections, batch size 1000) and 376,000 TPS in parallel streaming, with 8.7μs per-message latency on a single connection. The protocol uses fixed-size message framing with type-length-value encoding, Ed25519 message authentication, and optional PQ TLS 1.3 (X25519MLKEM768) for transport encryption. We prove that ZAP's message ordering guarantees preserve consensus safety under partial synchrony, and that the zero-allocation design eliminates GC pauses that cause consensus timeouts in garbage-collected runtimes. ZAP is deployed in production across the MPC threshold signing cluster, DEX matching engine, and Quasar consensus protocol.

Keywords: wire protocol, binary encoding, consensus transport, zero allocation, post-quantum TLS

1 Introduction

Blockchain consensus protocols require a transport layer with three properties:

1. **Low latency**: Sub-millisecond message delivery for fast finality
2. **Deterministic performance**: No GC pauses or allocation spikes
3. **Authentication**: Every message cryptographically bound to its sender

Existing solutions fall short: gRPC adds $\sim$100μs of protobuf serialization overhead. JSON-RPC wastes 60%+ of bandwidth on field names. NATS introduces a broker hop. WebSocket adds HTTP upgrade negotiation.

ZAP eliminates all intermediaries. Nodes connect directly via TCP, authenticate with Ed25519 identity keys, and exchange fixed-format binary messages with zero heap allocation on the hot path.

2 Protocol Specification

2.1 Message Format

Definition 1 (ZAP Frame)**.** *A ZAP frame is a fixed-layout binary message:*

$$\underbrace{1}_{type} \| \underbrace{4}_{length} \| \underbrace{32}_{sender\ ID} \| \underbrace{N}_{payload} \| \underbrace{64}_{Ed25519\ sig}$$

Total overhead: 101 bytes per message (independent of payload size).

2.2 Message Types

2.3 Zero-Allocation Design

Theorem 1 (Zero-Allocation Hot Path)**.** *The ZAP read/write path performs zero heap allocations for messages $\leq$ 8192 bytes. All data is read into pre-allocated buffers and returned as sub-slices.*

Proof. The buffer buf is allocated once at connection establishment (not per-message). The `Read` syscall writes directly into this buffer. Payload and sender ID are returned as slice headers pointing into buf—no `make`, `append`, or `new` on the hot path. Go's escape analysis confirms: the slice headers remain on the stack. The Ed25519 verification uses a pre-allocated scratchpad. Verified with `go test -benchmem`: 0 allocs/op for messages $\leq$ buffer size. □

Type Byte	Name	Usage
0x01	Identity	Connection handshake
0x02	Ping/Pong	Liveness check
0x3C (60)	MPCBroadcast	MPC pub/sub
0x3D (61)	MPCDirect	MPC point-to-point
0x3E (62)	MPCReady	Peer registry
0x40 (64)	MPCKeygen	DKG protocol
0x41 (65)	MPCSign	Signing protocol
0x42 (66)	MPCReshare	Key resharing
0x43 (67)	MPCResult	Session result
0x50 (80)	ConsensusVote	Consensus voting
0x51 (81)	ConsensusPropose	Block proposal
0x52 (82)	ConsensusCommit	Commit certificate

Algorithm 1 ZAP Message Read (Zero-Allocation)

Require: TCP connection conn, pre-allocated buffer buf[8192]
Ensure: Parsed message type, payload slice, verified sender
1: Read 5 bytes into buf[0 : 5] {type + length}
2: msgType ← buf[0]
3: payloadLen ← BigEndian(buf[1 : 5])
4: Read 32 + payloadLen + 64 bytes into buf[5 :]
5: senderID ← buf[5 : 37] {slice, no copy}
6: payload ← buf[37 : 37 + payloadLen] {slice, no copy}
7: sig ← buf[37 + payloadLen :]
8: **if** ¬Ed25519Verify(senderID, buf[0 : 37 + payloadLen], sig) **then**
9: **return** (error, nil, nil)
10: **end if**
11: **return** (msgType, payload, senderID)

Lemma 2 (GC Pause Elimination)**.** *With zero allocations on the hot path, the Go garbage collector has no new objects to trace during consensus rounds. GC pauses are bounded by the allocation rate of non-consensus goroutines (logging, metrics), typically <100µs.*

3 Performance

3.1 Benchmarks

Theorem 3 (Throughput Bound)**.** *ZAP's maximum throughput is bounded by:*

$$T_{\max} = \min\left(\frac{B_{NIC}}{|frame|}, \frac{1}{\tau_{verify}}\right)$$

where B_{NIC} is the NIC bandwidth and τ_{verify} is the Ed25519 verification time. On 25Gbps NICs with 64-byte payloads, $T_{\max} \approx 24M$ frames/sec, consistent with the 20.26M measured.

3.2 Batch Mode

Configuration	Throughput	Latency	Allocs/op
Single connection	114K TPS	8.7μs	0
20 parallel connections	376K TPS	2.7μs	0
50 conns + batch 1000	20.26M TPS	0.05μs	0
gRPC (comparison)	45K TPS	22μs	12
JSON-RPC (comparison)	18K TPS	55μs	28
NATS (comparison)	85K TPS	11μs	4

Algorithm 2 ZAP Batch Send

Require: Messages $m_1, \ldots, m_k$, connection conn
Ensure: All messages sent as a single syscall
1: iov $\leftarrow$ gather vector of all frames
2: **for** $i = 1$ to k **do**
3: iov$[i] \leftarrow$ EncodeFrame(m_i) {into pre-allocated ring buffer}
4: **end for**
5: writev(conn.fd, iov) {single syscall via scatter-gather I/O}

The `writev` syscall sends all frames in a single kernel transition, amortizing the syscall overhead across k messages. With $k = 1000$, per-message syscall cost drops from $\sim$1μs to $\sim$1ns.

4 Transport Security

4.1 PQ TLS 1.3

ZAP connections are wrapped in TLS 1.3 with post-quantum hybrid key exchange:

1. **X25519MLKEM768**: Hybrid classical + post-quantum (Go 1.24+ default)
2. **SecP256r1MLKEM768**: Alternative hybrid for FIPS environments
3. **SecP384r1MLKEM1024**: Maximum security variant

Theorem 4 (Dual-Layer Authentication)**.** *ZAP provides two independent authentication layers:*

1. *TLS certificate authentication (Ed25519 self-signed certs)*
2. *Per-message Ed25519 signature verification*

An attacker must compromise both layers to inject a forged message. TLS prevents passive eavesdropping; per-message signatures prevent active injection even if TLS is broken.

4.2 Deduplication

Definition 2 (Message Deduplication)**.** *ZAP maintains a TTL-based deduplication map: seen* : *Hash*$(sender \| payload) \mapsto$ *timestamp. Messages with hashes in seen within the TTL window (default 10 minutes) are dropped. Cleanup runs every 2 minutes.*

Theorem 5 (Replay Protection)**.** *Under the deduplication scheme, a replayed message m is accepted if and only if it arrives more than TTL seconds after the original. For consensus protocols with round durations $\ll$ TTL, replay is impossible within the same consensus round.*

5 Consensus Safety Preservation

Theorem 6 (Message Ordering and Safety)**.** *Let* Π *be a BFT consensus protocol safe under partial synchrony with message delay bound* Δ*. ZAP's TCP-ordered delivery preserves* Π*'s safety if:*

1. *ZAP delivers messages in FIFO order per connection (TCP guarantees this)*
2. *ZAP's authentication rejects messages from non-validators*
3. *ZAP's deduplication prevents double-counting votes*

Proof. Safety in BFT protocols requires that no two conflicting blocks are finalized. This depends on quorum intersection: $|Q_1 \cap Q_2| \geq 1$ honest node for any two quorums Q_1, Q_2. ZAP's authentication ensures only validator messages contribute to quorums (condition 2). FIFO ordering ensures a validator's messages arrive in the order sent (condition 1), preventing reordering that could create phantom votes. Deduplication ensures each vote is counted exactly once (condition 3). Since ZAP preserves the message delivery model assumed by Π, Π's safety proof applies. □

6 Deployment

ZAP is deployed across three production systems:

- **MPC cluster**: 3-node consensus, threshold signing (types 0x3C–0x43)
- **DEX matching engine**: Order broadcast, fill notification (types 0x50–0x52)
- **Quasar consensus**: Block proposal, voting, commit certificates

The ZapDB embedded database (fork of BadgerDB, since 2017) uses the same binary encoding principles for on-disk storage, providing consistent serialization across network and storage layers.

6.1 Teleport Bridge Watcher Consensus

ZAP serves as the inter-watcher communication layer for the Teleport cross-chain bridge. Each watcher node monitors deposit events on external chains (Bitcoin, Ethereum, Solana, TON) and broadcasts observations to the watcher quorum over ZAP.

1. **Deposit event broadcast**: Watchers emit `MPCBroadcast` (0x3C) messages containing deposit proofs. Per-message latency is 8.7μs with zero allocations, ensuring deposit observations propagate before the next CGGMP21 signing round begins.
2. **CGGMP21 threshold signing**: Signing rounds use `MPCSign` (0x41) for point-to-point shares and `MPCBroadcast` (0x3C) for commitment distribution. ZAP's FIFO ordering guarantees that round messages are processed in protocol order, a requirement for CGGMP21 correctness.
3. **W3C DID identity**: Each watcher authenticates via a W3C Decentralized Identifier bound to an ML-DSA-65 (Dilithium) post-quantum key pair, providing quantum-resistant identity independent of the Ed25519 transport signatures.
4. **Quasar finality**: Threshold signature results (`MPCResult`, 0x43) are submitted to the Quasar consensus layer for finality certification before relay to the destination chain.
5. **Spike handling**: ZAP's 20.26M TPS batch mode absorbs Bitcoin block deposit spikes (up to ∼4000 transactions per block) without backpressure on the signing pipeline.

7 Conclusion

ZAP achieves 20.26M TPS with zero allocations, post-quantum transport encryption, and provable consensus safety preservation. It eliminates the serialization overhead of gRPC (4.5× faster), the bandwidth waste of JSON-RPC (11× faster), and the broker dependency of NATS (5× faster). The protocol is production-deployed across MPC, DEX, and consensus systems.

References

[1] Jain, M. et al. *BadgerDB: Fast Key-Value Store in Go*. Dgraph Labs, 2017.

[2] Kelling, Z. *Quasar Consensus: Dual-Certificate Quantum-Secure Finality*. Lux Industries, 2025.

[3] Kelling, Z. and Seesahai, V. *Universal Threshold Signatures*. Lux Partners Limited Limited, 2021.