# Contents

# ZTLP: Zero Trust Layer Protocol

## Identity-First Networking for the Post-Perimeter Internet

**Version 0.9.13 — March 2026**

**Steven Price** Tech Rockstar Academy / ZTLP.org

---

## Abstract

The modern Internet was built on a foundational assumption: any device can send packets to any other device. This open connectivity model, while instrumental in the Internet's growth, has become its greatest security liability. Distributed denial-of-service attacks, credential stuffing, port scanning, and unauthorized service discovery are all consequences of a network architecture where reachability precedes identity.

The Zero Trust Layer Protocol (ZTLP) inverts this model. ZTLP is a transport-layer overlay protocol in which cryptographic identity verification is a precondition for network connectivity — not a consequence of it. Services protected by ZTLP have no public ports, no discoverable attack surface, and no exposure to unauthenticated traffic. To an observer without valid credentials, a ZTLP-protected service simply does not exist on the network.

This whitepaper presents the architecture, design rationale, security properties, and measured performance of the ZTLP reference implementation. The protocol provides mutual authentication via the Noise_XX handshake framework, a three-layer DDoS-resistant admission pipeline, a distributed identity namespace (ZTLP-NS) with federated replication and a person-centric identity model, relay mesh routing with consistent-hash load distribution, a gateway model that enables incremental deployment in front of existing infrastructure without application modification, and a client agent daemon that makes ZTLP connectivity transparent to applications.

A working reference implementation comprising over 30,000 lines of Rust, Elixir/OTP, C, and Ruby — with 2,278 tests and zero failures — demonstrates that ZTLP is not merely theoretical. The implementation has been validated in a real-world production deployment protecting a web application behind a ZTLP gateway, with end-to-end encrypted tunnels from a macOS client through ZTLP infrastructure to an internal service with zero exposed ports. Benchmarks show Layer 1 packet rejection in 19 nanoseconds, full Noise_XX handshake completion in under 300 microseconds, and tunnel throughput exceeding 400 MB/s on commodity hardware, confirming the protocol's viability for production deployment.

---

## Table of Contents



---

## 1. The Problem: Anonymous Connectivity as a Security Primitive

Every service on the public Internet today — from a bank's login portal to a hospital's patient records system — must accept packets from any source address on Earth before it can determine whether the sender is authorized. Security is applied after connectivity is established: firewalls filter by IP, WAFs inspect HTTP, rate limiters count requests, and DDoS mitigation services absorb volumetric attacks. All of these defenses share a structural weakness: they operate on traffic that has already reached the service.

The consequences are measurable:

- **DDoS attacks** exceeded 5.6 Tbps in early 2025 (Cloudflare), with the average enterprise spending $6,000–$12,000 per minute of downtime during an attack.
- **Credential stuffing** accounts for over 60% of login traffic on major web properties (Akamai State of the Internet report).
- **Port scanning and reconnaissance** tools like Shodan index over 3 billion exposed services, providing attackers with a free target map of the Internet.
- **Ransomware** exploits exposed management ports (RDP, SSH, SMB) — the initial access vector in the majority of ransomware incidents.

These are not implementation failures. They are architectural consequences of a network model where reachability is the default and identity is optional.

ZTLP proposes a different foundation: **identity before connectivity**. A service protected by ZTLP does not listen on a public port. It does not respond to unauthenticated probes. It generates no observable surface for scanning or exploitation. To reach it, a client must first prove — cryptographically — who it is. Only then does a session exist. Only then can packets flow.

This is not a VPN. VPNs still expose their own endpoints to the Internet. This is not a firewall rule. Firewall rules still require the packet to arrive before it can be evaluated. ZTLP operates at the transport layer, below application logic but above IP routing, making identity verification a structural property of the network path rather than a policy applied after the fact.

---

## 2. Design Principles

ZTLP was designed under five governing principles:

### 2.1 Identity Is the Network Primitive

In ZTLP, a 128-bit NodeID — cryptographically bound to a public key pair (X25519 for key exchange, Ed25519 for signatures) — is the fundamental unit of network participation. NodeIDs are not derived from IP addresses, MAC addresses, or any location-dependent identifier. A node's identity persists across network changes, device migrations, and IP reassignment.

IP addresses serve only as transport hints — the means by which packets reach relays and gateways. They carry no trust, no authorization, and no identity semantics.

### 2.2 No Open Ports

A ZTLP-protected service has zero publicly reachable sockets. Traffic arrives exclusively through authenticated ZTLP sessions terminated at a gateway. The service is invisible to network scanners, unreachable by bots, and unexposed to volumetric floods. This eliminates entire categories of attack — not by mitigating them, but by removing the surface they require.

### 2.3 Cheap Rejection, Expensive Admission

ZTLP's admission pipeline is designed so that rejecting unauthorized traffic is orders of magnitude cheaper than admitting legitimate traffic. Invalid packets are discarded in nanoseconds. Valid sessions require a full Noise_XX handshake — three message round-trips, six Diffie-Hellman operations, and policy evaluation. This asymmetry ensures that attackers bear the computational cost, not defenders.

### 2.4 Overlay Deployment

ZTLP operates over standard UDP on existing IPv4/IPv6 infrastructure. No router changes, no ISP support, no kernel modifications are required for initial deployment.

This follows the proven deployment path of QUIC, WireGuard, and Tailscale — protocols that achieved adoption precisely because they could deploy immediately on today's Internet.

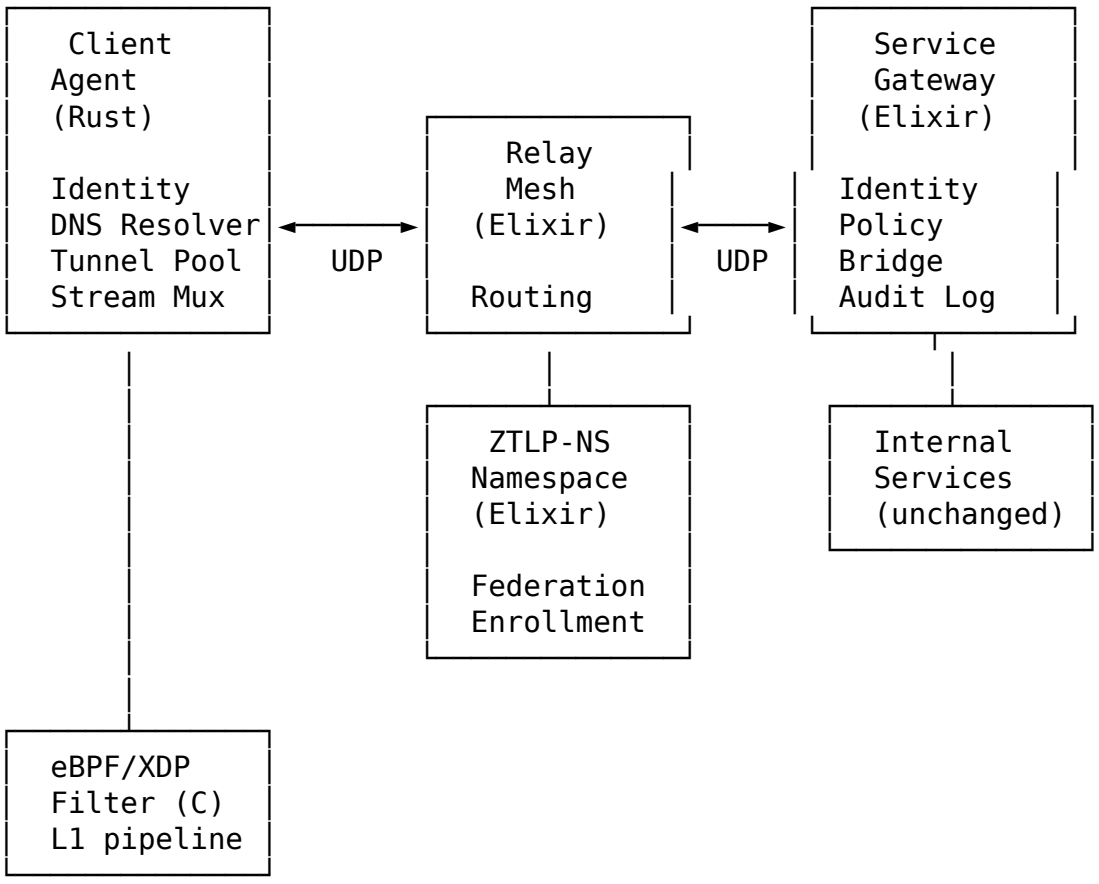
### 2.5 Gateway-First Adoption

Organizations should not need to rewrite applications to adopt ZTLP. The protocol is designed to sit in front of existing services via an edge gateway, leaving internal infrastructure (HTTP, gRPC, databases, microservices) unchanged. A ZTLP gateway is analogous to a reverse proxy or load balancer — but one that enforces cryptographic identity before any traffic reaches the application.

---

## 3. Protocol Architecture

ZTLP defines six components that together form a complete identity-first network:




### 3.1 Client Agent

A Rust daemon that provides transparent ZTLP connectivity. The agent includes a DNS resolver that intercepts queries for ZTLP-mapped domains, a VIP pool that assigns local

loopback addresses to remote services, a tunnel pool with automatic reconnection, and stream multiplexing supporting up to 256 concurrent streams per tunnel. Applications connect to local VIP addresses using standard protocols — they have no awareness of ZTLP. The agent also provides SSH ProxyCommand integration, credential renewal, and system DNS configuration.

### 3.2 Relay Nodes

Elixir/OTP processes that form a distributed mesh for session routing. Relays perform admission control (handshake validation, identity verification, policy enforcement) for new sessions and high-speed SessionID-based forwarding for established sessions. The BEAM VM provides per-session fault isolation via supervised GenServer processes, with ETS tables delivering O(1) session lookups at millions of operations per second.

### 3.3 ZTLP-NS Namespace

A distributed, hierarchical identity namespace with Ed25519-signed records, backed by Mnesia for persistent storage. ZTLP-NS provides identity-to-key bindings, service discovery, relay advertisements, zone delegation, credential revocation, and the identity model (DEVICE, USER, GROUP records). NS nodes support federated replication with Merkle-tree anti-entropy synchronization, enabling multi-node clusters with automatic conflict resolution.

### 3.4 Gateway Nodes

Edge terminators that bridge ZTLP sessions to internal service protocols. A gateway performs the full Noise_XX handshake as responder, evaluates access policy against the client's identity — including group membership resolution — and forwards authenticated traffic to internal services over conventional protocols. The gateway supports multiple named backend services, zone-based wildcard policies, and comprehensive audit logging.

### 3.5 eBPF/XDP Filter

A C program loaded into the Linux kernel's XDP hook that performs Layer 1 packet validation at the NIC driver level. The filter checks the ZTLP magic byte and optionally validates SessionIDs against a kernel-space allowlist, rejecting invalid traffic before it reaches userspace. The filter supports dual-port operation (client and mesh traffic on separate ports), peer allowlists, FORWARD packet TTL validation, and RAT-aware HELLO processing.

### 3.6 Bootstrap Management Server

A Ruby on Rails web application that provides fleet management for ZTLP deployments. Bootstrap handles machine provisioning via SSH, Docker container orchestration for

ZTLP components (relay, NS, gateway), user and device CRUD operations, group management, enrollment token generation with QR codes, audit logging, and real-time connectivity monitoring with health indicators.

### 3.7 Packet Format

ZTLP uses two wire formats optimized for their respective roles:

**Handshake Header (96 bytes, fixed):** Carries the full identity and crypto fields needed for session establishment — Magic, Version, Flags, HdrLen, MessageType, CryptoSuite, KeyID, SessionID, PacketSequence, Timestamp, SrcNodeID, DstSvcID, PolicyTag, and HeaderAuthTag.

**Compact Data Header (42 bytes):** Used for all post-handshake traffic. Carries only Magic, Version, Flags, SessionID, PacketSequence, and HeaderAuthTag. NodeIDs are deliberately absent from the data path — they were verified during the handshake and are not needed for forwarding. This keeps the per-packet overhead minimal and prevents passive observers from correlating traffic to specific identities.

Packet type discrimination uses the HdrLen field: 24 words = handshake, 11 words = data. This allows the admission pipeline to classify packets with a single field comparison.

---

## 4. The Three-Layer Admission Pipeline

The core DDoS resilience property of ZTLP is a three-layer admission pipeline where each layer is progressively more expensive — and progressively fewer packets survive to reach the next layer:

| Layer | Check | Cost (Rust) | Cost (Elixir) | What It Rejects |
|---|---|---|---|---|
| **L1: Magic** | 1-byte comparison | **19 ns** | **89 ns** | All non-ZTLP traffic (random floods, port scans, protocol confusion) |
| **L2: SessionID** | Hash table lookup | **31 ns** | **1.05 µs** | Traffic with unknown session identifiers (forged or expired) |
| **L3: HeaderAuthTag** | ChaCha20-Poly1305 AEAD | **840 ns** | **5.8 µs** | Packets targeting valid sessions but with forged authentication |

**The key insight:** In a volumetric DDoS attack, the vast majority of flood traffic is random garbage. It fails L1 in 19 nanoseconds — a single branch instruction — consuming effectively zero resources. Traffic crafted with the correct magic byte but a random SessionID fails L2 in 31 nanoseconds via a hash table miss. Only traffic that

guesses both the correct magic byte AND a valid 96-bit SessionID (probability: $2^{-96}$ against a sparse session table) reaches the cryptographic layer.

**Layer 1 — eBPF Fast Path:** The first layer runs as an XDP program in the Linux kernel, checking the magic byte before the packet reaches userspace. On systems without eBPF support, the check runs in userspace with slightly higher latency (~89 ns in Elixir) but identical filtering effectiveness.

**Measured impact:** The Rust client rejects invalid traffic 42× cheaper than it admits valid traffic. In practical terms, a single CPU core can reject 54 million garbage packets per second at L1 while simultaneously processing 1.1 million legitimate packets through the full pipeline.

This is the structural DDoS advantage: defenders pay nanoseconds to reject what attackers spend bandwidth to send.

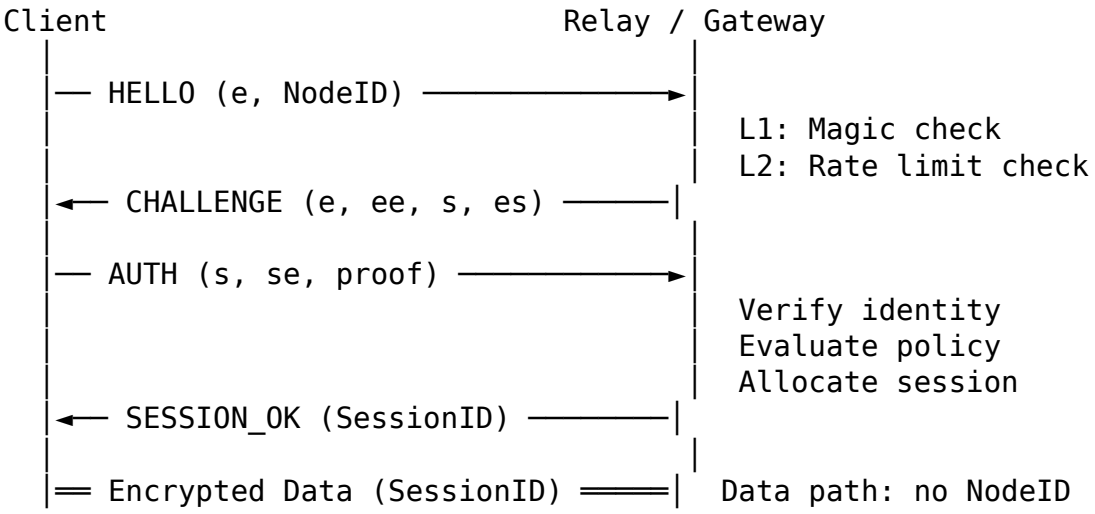
---

## 5. Session Establishment: Noise_XX Handshake

ZTLP establishes sessions using the Noise_XX handshake pattern from the Noise Protocol Framework (Trevor Perrin, 2018). Noise_XX was selected for three properties critical to ZTLP's design:

1. **Mutual authentication** — Both initiator and responder prove possession of their static keys. There are no anonymous or unauthenticated sessions in ZTLP.
2. **Forward secrecy** — Ephemeral X25519 keys are generated per-session and discarded after key derivation. Compromise of long-term keys does not reveal past session data.
3. **No PKI dependency** — Unlike TLS, Noise_XX does not require certificate authorities. Identity binding is handled by ZTLP-NS, not by external trust infrastructure.

### 5.1 Handshake Flow



The three-message exchange performs six X25519 Diffie-Hellman operations and derives symmetric session keys via BLAKE2s-based HKDF. The entire handshake completes in **293 µs (Rust)** or **471 µs (Elixir)** — enabling a single gateway core to establish over 2,100 new sessions per second.

### 5.2 Handshake Reliability

ZTLP implements handshake retransmission with exponential backoff (500ms to 5 seconds) to handle packet loss during session establishment. A half-open cache (64 entries, 15-second TTL) prevents resource exhaustion from retransmitted HELLOs, and an amplification limit of 3 retransmits per session prevents the handshake mechanism from being abused as a reflector.

### 5.3 Post-Handshake Data Path

After `SESSION_OK`, all traffic carries only the 96-bit SessionID — no NodeIDs, no identity fields, no policy metadata. This design serves three purposes:

- **Privacy:** Passive observers see only random-looking session identifiers that change per session. Traffic cannot be correlated to specific identities by network observers.
- **Performance:** Relays perform SessionID-based label switching — a constant-time ETS/HashMap lookup — without touching any cryptographic state on the forwarding path.
- **Scalability:** The separation of identity verification (admission plane) from packet forwarding (forwarding plane) allows relay infrastructure to scale horizontally.

### 5.4 Session Lifecycle

Sessions have a maximum lifetime of 24 hours with mandatory rekeying every hour. Rekeying is transparent to applications — a new SessionID is issued, relay forwarding tables are updated atomically, and the old SessionID remains valid for a 5-second overlap window to drain in-flight packets. Sessions may also terminate via explicit CLOSE, lifetime expiry, or inactivity timeout (default: 5 minutes for interactive sessions).

---

## 6. Transport Reliability

ZTLP operates over UDP, requiring the protocol to implement its own reliability mechanisms. The transport layer provides TCP-competitive reliability with optimizations for the encrypted tunnel use case.

### 6.1 Congestion Control

ZTLP implements a modern congestion control stack:

- **SACK-based selective retransmission** — Receivers report gaps in received sequences, enabling the sender to retransmit only lost packets rather than retransmitting everything after a gap.
- **Proportional Rate Reduction (PRR)** — On loss detection, the sending rate is reduced proportionally rather than halved, maintaining higher throughput during recovery.
- **Token bucket pacing** — Outgoing packets are paced using a token bucket to prevent burst-induced congestion, with configurable inter-batch delays.
- **Eifel spurious retransmission detection** — Prevents unnecessary rate reduction when retransmissions are triggered by reordering rather than actual loss.
- **Jacobson/Karels RTT estimation** — Adaptive RTO calculation with smoothed RTT and RTT variance tracking, bounded at 4 seconds maximum.
- **Fast retransmit** — Immediate retransmission after 3 duplicate ACKs, without waiting for the retransmission timeout.

### 6.2 NAT Traversal

ZTLP supports Nebula-style NAT traversal:

- **UDP hole punching** — Coordinated through relay nodes to establish direct paths between clients behind NAT.
- **NAT timeout auto-detection** — Periodic keepalive probes detect the NAT mapping lifetime and adjust keepalive intervals to maintain traversal.
- **Roaming support** — Sessions survive IP address changes (Wi-Fi to cellular, DHCP renewal) by rebinding to the new address transparently.
- **Tunnel health monitoring** — Continuous RTT and loss measurement with automatic failover to relay-mediated paths when direct connectivity degrades.

### 6.3 Flow Control

The transport provides windowed flow control with a configurable send window (default: 2,048 outstanding packets). Each ZTLP data packet carries up to 16,375 bytes of plaintext (approximately 16KB), encrypted with ChaCha20-Poly1305. Receivers send acknowledgments every 16 packets or every 5 milliseconds (whichever comes first), with a cumulative ACK plus selective ACK bitmap.

### 6.4 Measured Tunnel Throughput

End-to-end tunnel benchmarks on a 4-vCPU system (localhost loopback, including Noise_XX handshake, ChaCha20-Poly1305 encryption, flow control, and reassembly):

| Transfer Size | Throughput | Notes |
|---|---|---|
| 64 KB | 101 MB/s | Small transfer, handshake dominates |

| Transfer Size | Throughput | Notes |
|---|---|---|
| 1 MB | 334–415 MB/s | Peak throughput (congestion window covers full transfer) |
| 10 MB | 235–275 MB/s | Steady-state with pacing |
| SCP 10 MB | 31 MB/s | Real-world SCP through ZTLP tunnel (~1.6× overhead vs. direct SSH) |

Cryptographic overhead is negligible: ChaCha20-Poly1305 encrypt/decrypt takes approximately 10 µs per 16KB packet, contributing only 6% of total transfer time for multi-megabyte transfers.

---

## 7. ZTLP-NS: Distributed Identity Namespace

ZTLP-NS is a protocol-native namespace that provides the trust infrastructure ZTLP requires without depending on external certificate authorities or DNS.

### 7.1 Record Types

| Record Type | Wire Code | Purpose |
|---|---|---|
| `ZTLP_KEY` | `0x01` | Binds a NodeID to its Ed25519 public key. Signed by an Enrollment Authority. |
| `ZTLP_SVC` | `0x02` | Publishes a service identity, gateway endpoint, and policy requirements. |
| `ZTLP_RELAY` | `0x03` | Advertises a relay node’s availability, capacity, and geographic region. |
| `ZTLP_POLICY` | `0x04` | Defines which NodeIDs or identity classes may access a service. |

| Record Type | Wire Code | Purpose |
|---|---|---|
| ZTLP_REVOKE | 0x06 | Invalidates a previously issued identity binding or credential. |
| ZTLP_ZONE | 0x07 | Delegates authority over a sub-namespace to a subordinate zone. |
| ZTLP_DEVICE | 0x10 | Hardware-bound identity record linking a device to a user (see Section 8). |
| ZTLP_USER | 0x11 | Person-bound identity record with role and device associations (see Section 8). |
| ZTLP_GROUP | 0x12 | Named group record with flat membership for policy evaluation (see Section 8). |

All records are Ed25519-signed. Record validity is enforced via explicit TTLs and expiration timestamps.

### 7.2 Hierarchical Delegation

ZTLP-NS uses hierarchical zone delegation analogous to DNS, but with cryptographic integrity at every level:

```
Root Trust Anchor
  └── org.ztlp (delegated to Org enrollment authority)
       ├── engineering.org.ztlp
       │    ├── ZTLP_KEY: node-a (NodeID → pubkey binding)
       │    └── ZTLP_SVC: api.engineering.org.ztlp
       └── partners.org.ztlp
            └── ZTLP_POLICY: partner-access rules
```

Each delegation is signed by the parent zone authority. Trust chain verification traverses from the queried record up through delegation records to a configured trust anchor. The reference implementation verifies a 2-level trust chain in approximately 250 µs.

### 7.3 Multi-Root Trust

ZTLP does not require a single global root authority. Deployments may configure multiple independent trust roots — enterprise roots, partner roots, public roots — and define per-interaction trust policies. This federated model avoids the single-point-of-failure inherent in centralized PKI while maintaining auditable trust chains.

### 7.4 NS Federation

ZTLP-NS supports multi-node federation for high availability and geographic distribution:

- **Eager replication** — Write operations are propagated to all cluster peers immediately upon acceptance.
- **Merkle-tree anti-entropy** — Periodic synchronization using Merkle trees detects and repairs divergence between NS nodes, ensuring eventual consistency even after network partitions.
- **Conflict resolution** — When records conflict (due to concurrent writes during a partition), resolution follows a deterministic priority order: revocation records always win, then highest serial number, then signature comparison, then shortest TTL. This ensures convergence without coordination.
- **Cluster management** — NS nodes discover peers via configuration or DNS-based discovery and maintain health status through periodic heartbeats.

### 7.5 Security Hardening

Production NS deployments include:

- **Registration authentication** — Ed25519 signature verification on all registration requests, with zone-scoped authorization.
- **Rate limiting** — Per-source query and registration rate limits to prevent abuse.
- **Amplification prevention** — Response size limited to 8× the request size, with truncation when exceeded.
- **Packet and record size limits** — 8KB maximum packet size, 4KB maximum record size, DNS-compatible name validation.
- **Pubkey reverse index** — Mnesia-backed O(1) lookups from public key to registered name, enabling identity resolution on the gateway data path.
- **Worker pool** — Task.Supervisor with configurable concurrency limits (default: 100 workers) to prevent resource exhaustion.
- **Audit logging** — Structured logging of all registration, query, and administrative operations.

### 7.6 Revocation

Credential revocation is a first-class operation. `ZTLP_REVOKE` records propagate through the namespace and via federation to all cluster peers. Revoked NodeIDs are rejected at the admission stage — before any session resources are allocated. Revocation is fast (sub-second propagation to connected relays) and does not require

cooperation from the revoked party. Revoking a USER record automatically cascades to revoke all linked DEVICE records.

---

## 8. Identity Model: Devices, Users, and Groups

ZTLP v0.9 introduced a person-centric identity model that maps real-world organizational relationships — people, their devices, and their roles — into the cryptographic namespace.

### 8.1 Design Rationale

The original ZTLP identity model used only NodeIDs and KEY records — sufficient for device-to-device authentication, but inadequate for answering the question "who is connecting?" In managed IT environments, access policy is defined in terms of people and roles: "technicians can access monitoring dashboards" or "administrators can reach the database." ZTLP's identity model bridges the gap between cryptographic device identity and organizational human identity.

### 8.2 Record Types

**DEVICE (0x10)** — Represents a physical or virtual machine. Each device has a unique NodeID, X25519 key pair (for Noise_XX handshakes), Ed25519 key pair (for signing), and an optional owner link to a USER record. Devices can exist without owners (kiosks, servers, infrastructure nodes).

**USER (0x11)** — Represents a person. Users have a role (admin, tech, or user), an optional contact email, an Ed25519 signing key for administrative operations, and a list of associated devices. Users do not directly participate in Noise_XX handshakes — their devices do.

**GROUP (0x12)** — Represents a named collection of users. Groups have flat membership (no nested groups) and are the primary unit of policy evaluation. When a device connects to a gateway, the gateway resolves the device's owner, enumerates the owner's group memberships, and evaluates the access policy against those groups.

### 8.3 Policy Evaluation Flow

```
Device connects → Gateway extracts X25519 pubkey from handshake
  → NS reverse lookup: pubkey → DEVICE record
  → DEVICE.owner → USER record
  → USER → enumerate GROUP memberships
  → Policy engine: does any group match the service's access policy?
  → ALLOW or DENY
```

Group membership is cached at the gateway with a configurable TTL to avoid per-connection NS queries. The entire resolution chain — from handshake completion to policy decision — adds sub-millisecond overhead.

### 8.4 Administration

Identity management is performed via the `ztlp admin` CLI:

```
# Create a user
ztlp admin create-user alice --role tech --zone clients.example.ztlp

# Create a group
ztlp admin create-group technicians --zone clients.example.ztlp

# Add user to group
ztlp admin group add technicians alice

# List identities
ztlp admin ls --zone clients.example.ztlp

# Revoke a user (cascades to all linked devices)
ztlp admin revoke alice

# Audit log
ztlp admin audit --since 24h
```

All admin commands support `--json` output for programmatic integration. The Bootstrap web UI provides the same capabilities through a graphical interface with QR code generation for device enrollment.

### 8.5 Key Overwrite Protection

ZTLP-NS rejects registration attempts for names that already exist with a different public key, preventing identity hijacking. Administrators can force-overwrite with an explicit flag when legitimate key rotation is needed.

---

## 9. Device Enrollment

ZTLP provides a streamlined enrollment system for onboarding new devices into the network, designed for managed service providers who need to enroll dozens or hundreds of client devices.

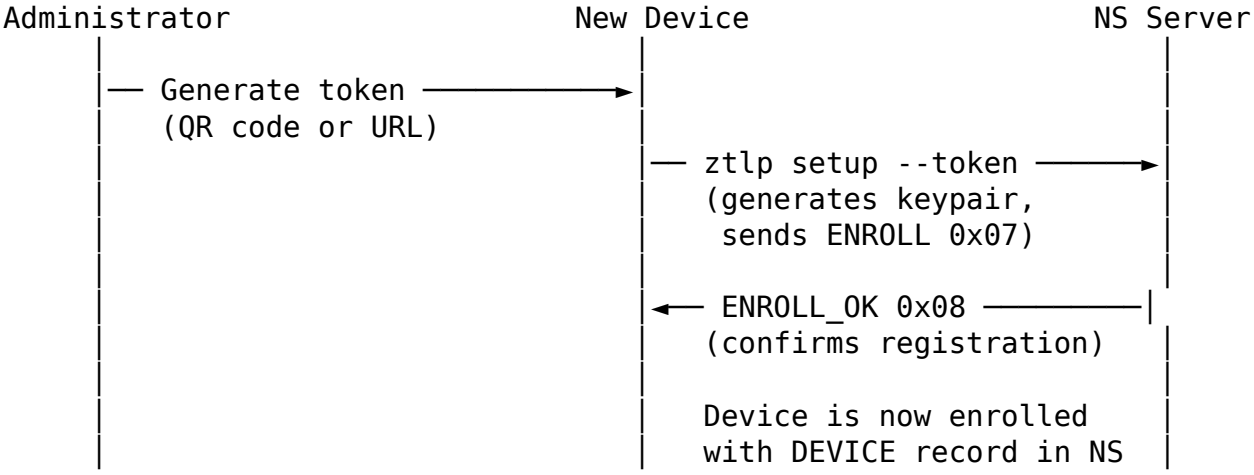
### 9.1 Enrollment Tokens

An Enrollment Token is a compact, authenticated credential that authorizes a new device to register with a specific zone. Tokens are HMAC-BLAKE2s authenticated, carry an expiration timestamp and maximum usage count, and encode to a `ztlp://enroll/` URI suitable for QR code generation.

Token wire format (compact binary, base64url-encoded for URIs):

| Field | Size | Description |
|---|---|---|
| Version | 1 byte | Token format version |
| Zone | variable | Target zone name |
| Expiry | 8 bytes | Unix timestamp (seconds) |
| Max uses | 4 bytes | Maximum enrollment count |
| MAC | 32 bytes | HMAC-BLAKE2s authentication tag |

### 9.2 Enrollment Flow




The `ztlp setup` command provides an interactive wizard that generates a new identity, sends an enrollment request to the NS server, and configures the local agent — all in a single step. For unattended enrollment, `ztlp setup --token TOKEN` runs non-interactively.

### 9.3 Wire Protocol

Enrollment uses two dedicated message types:

- **ENROLL (0x07)** — Sent by the new device. Contains the enrollment token, the device's newly generated public key, requested name, and zone.
- **ENROLL_OK (0x08)** — Sent by the NS server on success. Contains the assigned NodeID and confirmation of registration.

Error response codes: `0x01` (token expired), `0x02` (usage exhausted), `0x03` (invalid MAC), `0x04` (zone mismatch), `0x05` (name already taken), `0x06` (validation error).

---

## 10. Relay Mesh Architecture

ZTLP relays form a distributed mesh that provides session routing, admission control, NAT traversal, and path optimization.

### 10.1 Consistent-Hash Routing

Relays are organized into a hash ring using BLAKE2s with 128 virtual nodes per physical relay. Session admission is deterministically assigned to a bounded set of ingress relays based on consistent hashing of the client's NodeID or target ServiceID. This distribution prevents any single relay from being overwhelmed by targeted admission floods and enables predictable load distribution across the mesh.

### 10.2 PathScore-Based Selection

Relay selection uses a composite scoring function:

```
PathScore = RTT × (1 + loss × 10) × (1 + load × 2) × (1 + jitter / 100)
```

Clients maintain real-time PathScore measurements via sequence-numbered PING/PONG probes with a 20-probe sliding window. Jitter is tracked as the standard deviation of RTT samples. Relay health states (healthy / degraded / unreachable) use hysteresis to prevent flapping. Clients maintain at least two active relay paths for failover.

### 10.3 Multi-Hop Forwarding

When no single relay provides adequate connectivity, ZTLP supports multi-hop forwarding through up to 4 relay hops. Each forwarded packet carries a TTL byte and traversed-path list for loop detection. Route planning selects ingress → transit → service paths using measured PathScores.

### 10.4 Relay Admission Tokens

Transit relays authenticate forwarded sessions using Relay Admission Tokens (RATs) — 93-byte HMAC-BLAKE2s authenticated tokens that bind a session to a specific relay path. RAT issuance runs at 275,000 tokens/sec; verification at 393,000/sec. Key rotation is supported for zero-downtime relay credential updates.

### 10.5 NS-Based Relay Discovery

Relays register themselves with ZTLP-NS on startup and refresh their registration periodically (default: 60 seconds). Clients discover available relays by querying NS for RELAY records in their target zone. When NS is not configured, relays fall back to a static bootstrap list.

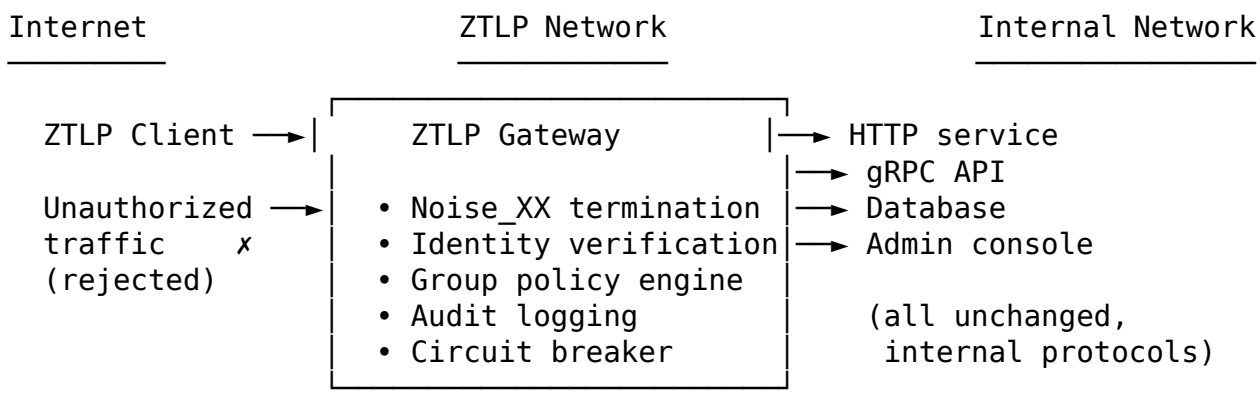
### 10.6 Admission Plane / Forwarding Plane Separation

ZTLP explicitly separates expensive admission operations (identity verification, policy evaluation, handshake processing) from cheap forwarding operations (SessionID lookup, packet relay). Admission is confined to ingress relays; transit relays perform only label-switching. Under DDoS conditions, handshake floods are contained to the admission plane while established sessions continue uninterrupted on the forwarding plane.

## 11. Gateway Deployment Model

The ZTLP gateway is the critical adoption enabler. It allows organizations to protect existing services without modifying them.

### 11.1 Architecture




The gateway performs the full Noise_XX handshake as responder, resolves the client's identity via ZTLP-NS (including reverse pubkey lookup for human-readable identity resolution), evaluates access policy including group membership, and bridges authenticated ZTLP sessions to internal TCP services. Internal services see standard protocol traffic — they have no awareness of ZTLP.

### 11.2 Named Backend Services

A single gateway can front multiple internal services, each identified by a service name:

```
ZTLP_GATEWAY_BACKENDS=web:127.0.0.1:80,api:127.0.0.1:8080,db:127.0.0.1:5432
ZTLP_GATEWAY_POLICIES=admins@:*,techs@:web,techs@:api
```

The client specifies the target service during the handshake. The gateway routes to the appropriate backend based on the service name and evaluates service-specific access policy.

### 11.3 Policy Engine

The gateway policy engine supports:

- **Exact NodeID matching** — Allow specific devices by cryptographic identity.
- **Zone-based wildcards** — `*.engineering.org.ztlp` matches all identities in a zone.
- **Group-based access** — `admins@` grants access to all members of the "admins" group, resolved dynamically via NS.
- **Service-specific rules** — Different policies per backend service.

- **Assurance level requirements** — Distinguish between software keys, hardware tokens, and attested devices.
- **Audit logging** — Every admission decision is logged with timestamp, identity, service, and outcome.

Policy evaluation adds sub-microsecond overhead — even with 10 pattern rules, the engine completes in 371 ns.

### 11.4 Operational Resilience

The gateway includes production-grade operational features:

- **Circuit breaker** — Prevents cascade failures when backend services are unresponsive, with configurable thresholds and recovery intervals.
- **Backpressure** — Flow control between the ZTLP session and the backend TCP connection prevents memory exhaustion under load.
- **Rate limiting** — Per-source admission rate limits prevent handshake floods from consuming gateway resources.

### 11.5 Phased Adoption

Organizations adopt ZTLP incrementally:

1. **Phase 1:** Protect administrative interfaces and privileged access (SSH, management consoles)
2. **Phase 2:** Protect authentication endpoints (login portals, MFA flows)
3. **Phase 3:** Protect APIs and partner integrations (B2B, financial endpoints)
4. **Phase 4:** Extend to internal east-west service traffic

Each phase is independently valuable. Phase 1 alone eliminates exposed management ports — the initial access vector in the majority of ransomware incidents.

---

## 12. Agent Daemon

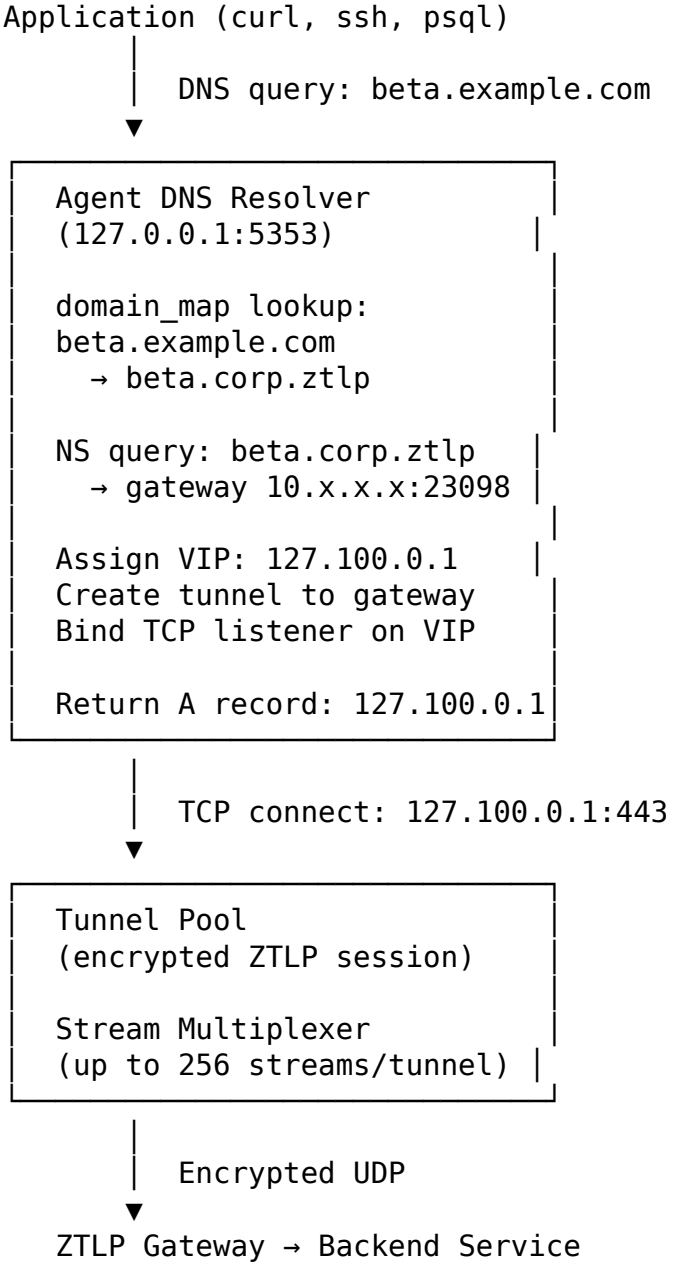
The ZTLP agent is a background daemon that makes ZTLP connectivity transparent to applications. Instead of manually establishing tunnels, users interact with ZTLP-protected services as if they were regular network hosts.

### 12.1 Design Goals

- **Zero-config after enrollment** — `ztlp setup` + `ztlp agent start` and all services are reachable.
- **Works with any TCP application** — SSH, HTTP, databases, file transfers.
- **Custom domain support** — Organizations use their own domain names; ZTLP handles identity underneath.
- **Auto-reconnect** — Tunnels survive network changes (Wi-Fi to cellular, IP reassignment).
- **Credential lifecycle** — Automatic renewal of certificates and NS record refresh.

- **Minimal privileges** — Runs as an unprivileged user (DNS on loopback, no TUN device required).

### 12.2 Architecture




### 12.3 VIP Pool

The agent allocates virtual IP addresses from a configurable loopback range (default: `127.100.0.0/16`, providing 65,534 addresses). Each ZTLP service is assigned a unique VIP. Applications connect to these VIPs using standard TCP — the agent intercepts the connection and routes it through the appropriate ZTLP tunnel.

### 12.4 Stream Multiplexing

Multiple application connections to the same service share a single ZTLP tunnel via stream multiplexing. The agent supports up to 256 concurrent streams per tunnel using three frame types:

- **STREAM_OPEN (0x05)** — Establishes a new stream within an existing tunnel.
- **STREAM_DATA (0x06)** — Carries application data for a specific stream.
- **STREAM_CLOSE (0x07)** — Terminates a stream, releasing its resources.

This avoids the overhead of establishing a new Noise_XX handshake for every application connection while maintaining stream-level isolation.

### 12.5 Tunnel Pool and Auto-Reconnect

The agent maintains a pool of tunnels with automatic lifecycle management:

- **Auto-reconnect** — When a tunnel fails, the agent reconnects with exponential backoff (1 second to 60 seconds).
- **Keepalive** — Periodic probes (default: every 30 seconds) detect tunnel failures before the application timeout.
- **Idle timeout** — Tunnels with no active streams are closed after a configurable period (default: 5 minutes).
- **Connection pooling** — Multiple services on the same gateway share a tunnel when possible.

### 12.6 Credential Renewal

The agent automatically manages the credential lifecycle:

- **Certificate renewal** — Initiated at 67% of the certificate's lifetime.
- **NS record refresh** — Re-registered at 75% of the record's TTL.
- **Jitter** — ±10% randomization prevents thundering herd when many devices renew simultaneously.
- **Failure backoff** — Exponential backoff on renewal failures with a credential tracker that monitors all credential expiry times.

### 12.7 SSH Integration

The agent provides SSH ProxyCommand integration for transparent SSH tunneling:

```
# Direct use
ssh -o ProxyCommand="ztlp proxy %h" user@server.corp.ztlp

# SSH config integration
Host *.corp.ztlp
    ProxyCommand ztlp proxy %h
```

### 12.8 System Integration

- **DNS setup** — `ztlp agent dns-setup` configures the system DNS to route ZTLP domains through the agent (supports systemd-resolved, resolv.conf, and macOS /etc/resolver).
- **Service installer** — `ztlp agent install` generates a systemd unit file or macOS LaunchAgent plist for automatic startup.
- **DNS TXT discovery** — Queries `_ztlp` TXT records for automatic NS server discovery, enabling zero-configuration connectivity when the organization publishes its ZTLP infrastructure in DNS.

---

## 13. Threat Model and Security Properties

### 13.1 Attacker Classes

ZTLP's security analysis considers five attacker positions with increasing capability:

| Attacker Position | Payload Visibility | Can Impersonate | Can Disrupt Service | Can Discover Services |
|---|---|---|---|---|
| **External (no identity)** | None | No | Bandwidth saturation only | No |
| **Network observer** | None | No | No | No |
| **Compromised relay** | None | No | Selective packet drop | Limited metadata |
| **Authorized insider** | Own sessions only | No (others) | Application-layer only | Authorized services only |
| **Endpoint compromise** | That node's traffic | That node only | That node only | That node's access only |

### 13.2 Cryptographic Properties

| Property | Mechanism |
|---|---|
| **Forward secrecy** | Ephemeral X25519 DH per session; keys destroyed after derivation |
| **Mutual authentication** | Noise_XX: both parties prove static key possession |
| **Replay protection** | 64-bit sequence numbers with sliding anti-replay window |
| **Endpoint confidentiality** | ChaCha20-Poly1305 AEAD; relays forward opaque ciphertext |
| **Key freshness** | Mandatory rekeying every hour; 24-hour max session lifetime |

| Property | Mechanism |
|---|---|
| **Identity integrity** | Ed25519-signed NS records; registration authentication |

### 13.3 Trust Assumptions

When deploying ZTLP, the operator trusts:

1. **The cryptographic primitives** — Ed25519, X25519, ChaCha20-Poly1305, BLAKE2s. Well-studied, widely deployed, but not post-quantum resistant (see Section 17).
2. **The enrollment authority** — Issues NodeIDs. A compromised EA can mint valid identities. Mitigation: hardware-backed EA keys, audit logging, multi-root trust.
3. **The trust root** — Anchors the namespace. Compromise means arbitrary zone delegation. Mitigation: offline keys, multi-party signing.
4. **Endpoint integrity** — ZTLP secures the network layer, not the endpoint. Pair with endpoint hardening.
5. **Relay availability (not confidentiality)** — Relays can drop traffic but cannot read it. A malicious relay can deny service but cannot compromise data.

### 13.4 Explicit Non-Goals

ZTLP is transparent about what it does not defend against:

- **Traffic analysis** — ZTLP is not an anonymity network. Relay IP addresses are visible; traffic patterns are observable. Multi-hop relaying provides some topology hiding, but a global passive adversary can still correlate flows.
- **Endpoint compromise with key extraction** — If an attacker holds the private key, they ARE the node. Hardware key storage (TPM, Secure Enclave, YubiKey) mitigates this.
- **Application-layer vulnerabilities** — ZTLP protects the transport path, not the application logic.
- **Bandwidth saturation** — While ZTLP makes application-layer DDoS structurally ineffective, filling the network pipe before packets reach the eBPF filter is outside ZTLP's scope.
- **Social engineering** — ZTLP cannot distinguish between legitimate and coerced use of valid credentials.

---

## 14. Performance

All benchmarks measured on a 4-vCPU AMD EPYC 4564P system with 7.8 GiB RAM running Linux 5.15.

### 14.1 Admission Pipeline

| Operation | Rust | Elixir |
|---|---|---|
| L1 reject (bad magic) | **19 ns** — 54M pkt/sec | **89 ns** — 11.3M pkt/sec |
| L2 reject (unknown session) | **31 ns** — 32M pkt/sec | **1.05 µs** — 950K pkt/sec |
| L3 AEAD verify | **840 ns** | **5.8 µs** |
| Full pipeline (valid packet) | **904 ns** — 1.1M pkt/sec | **7.5 µs** — 133K pkt/sec |

### 14.2 Handshake

| Implementation | Latency | Throughput |
|---|---|---|
| Rust (client) | **293 µs** | 3,413 handshakes/sec |
| Elixir (gateway) | **471 µs** | 2,125 handshakes/sec |

With 4 BEAM schedulers: **~8,500 new sessions/sec** per gateway node.

### 14.3 Cryptographic Operations

| Operation | Rust Latency |
|---|---|
| ChaCha20-Poly1305 (64 B) | 1.15 µs |
| ChaCha20-Poly1305 (1 KB) | 1.60 µs |
| ChaCha20-Poly1305 (8 KB) | 5.12 µs |
| Ed25519 sign | ~50 µs |
| Ed25519 verify | ~80 µs |

### 14.4 Tunnel Throughput

| Transfer Size | Throughput |
|---|---|
| 1 MB | **334–415 MB/s** |
| 10 MB | **235–275 MB/s** |
| SCP 10 MB (real-world) | **31 MB/s** (1.6× overhead vs. direct SSH) |

### 14.5 Namespace

| Operation | Latency |
|---|---|
| ETS lookup (cache hit) | **432 ns** — 2.3M lookups/sec |
| Verified query (lookup + Ed25519 verify) | **79.7 µs** — 12.5K queries/sec |
| Trust chain verification (2-level) | **250 µs** |

### 14.6 Relay Mesh

| Operation | Throughput |
|---|---|
| Hash ring lookup | 36K ops/sec |
| Packet forwarding | 233K pkt/sec |
| Mesh overhead | 3.2% (vs. non-mesh) |
| RAT issuance | 275K tokens/sec |
| RAT verification | 393K tokens/sec |

### 14.7 Gateway Data Path

| Operation | Throughput |
|---|---|
| Full data path (decrypt + identity + policy) | **669K ops/sec** per core |
| Policy evaluation (10 rules) | **371 ns** |

**Throughput projections** (4-core gateway): - **2.7M packets/sec** steady-state data path - **~19 Gbps** theoretical at 1KB MTU (without per-packet auth) - **~12 Gbps** theoretical at 1KB MTU (with per-packet auth)

---

## 15. Production Readiness

The ZTLP reference implementation includes production-grade operational infrastructure.

### 15.1 Observability

- **Structured logging** — JSON-formatted log output across all components with consistent field schemas, enabling log aggregation and analysis.
- **Prometheus metrics** — Eight metric modules expose counters, histograms, and gauges for handshake rates, pipeline layer rejection counts, session counts, relay forwarding rates, NS query latencies, and federation sync status.
- **Grafana dashboards** — Pre-built dashboard with 9 panels covering admission pipeline, session lifecycle, relay mesh health, and NS federation status.

### 15.2 Resilience

- **Backpressure** — Flow control on the relay forwarding path prevents memory exhaustion under sustained load.
- **Circuit breakers** — Gateway circuit breakers trip when backend services become unresponsive, returning fast failures instead of accumulating timeouts.
- **Rate limiters** — Per-source rate limiting on NS queries, relay admission, and gateway handshakes.

- **Inter-component authentication** — Ed25519 challenge-response authentication between relay, NS, and gateway nodes prevents unauthorized components from joining the infrastructure.

### 15.3 Deployment

- **OTP releases** — All Elixir components ship as self-contained OTP releases with runtime configuration via `runtime.exs` and environment variables.
- **Hot upgrades** — Appup templates enable in-place upgrades without dropping active sessions.
- **Docker packaging** — Dockerfiles for all components with a `docker-compose.yml` for full-stack deployment.
- **Bootstrap web UI** — Rails-based fleet management with SSH-driven provisioning, enabling one-click deployment and monitoring of ZTLP infrastructure across multiple machines.

### 15.4 Operations Documentation

- **Ops runbook** (1,687 lines) — Procedures for common operational tasks: deployment, scaling, troubleshooting, incident response, backup, and recovery.
- **Key management guide** (1,367 lines) — Comprehensive key lifecycle documentation covering generation, storage (including HashiCorp Vault and cloud KMS integration), rotation procedures, and incident response for key compromise.

### 15.5 Testing Infrastructure

The reference implementation maintains comprehensive test coverage:

| Component | Tests | Failures |
|---|---|---|
| Rust (library + agent + CLI + interop) | 853 | 0 |
| Elixir relay | 557 | 0 |
| Elixir NS | 608 | 0 |
| Elixir gateway | 260 | 0 |
| **Total** | **2,278** | **0** |

Additional testing infrastructure: - **Fuzz testing** — 8 mutation strategies, 0 panics in 50,000 iterations. - **Stress testing** — Userspace impairment proxy simulating packet loss, delay, reordering, and corruption across 11 extreme network scenarios. - **Docker chaos lab** — 7 automated scenarios testing mesh formation, failover, partition recovery, and federation convergence. - **Network test scenarios** — 10 Docker-based end-to-end scenarios with 3 isolated networks and 7 containers, testing the full stack under realistic conditions.

---

## 16. Comparison with Existing Systems

| Capability | ZTLP | WireGuard | Tailscale | Cloudflare Access | Teleport |
|---|---|---|---|---|---|
| Identity-first admission | ✓ Trans-port layer | ✗ | Partial (control plane) | Partial (HTTP only) | Partial (app proxy) |
| No open ports | ✓ | ✗ (UDP port exposed) | ✗ (device IPs visible) | ✗ (proxy exposed) | ✗ (proxy exposed) |
| DDoS-resistant pipeline | ✓ Three-layer, 19ns reject | ✗ | ✗ | ✓ (L3/L4/L7 mitigation) | ✗ |
| Distributed relay mesh | ✓ Consistent hash | ✗ (point-to-point) | ✓ (DERP relays) | ✓ (Anycast CDN) | ✗ |
| Protocol-native namespace | ✓ ZTLP-NS (federated) | ✗ | ✗ | ✗ | ✗ |
| Person-centric identity | ✓ USER/DEVICE/GROUP | ✗ | ✗ | Partial (via IdP) | ✓ (RBAC) |
| Multi-root federated trust | ✓ | ✗ | ✗ (single control plane) | ✗ (single provider) | ✗ (single cluster) |
| Hardware identity support | ✓ (TPM/YubiKey/SE) | ✗ | ✗ | ✗ | ✓ (hardware keys) |
| Gateway for existing services | ✓ | ✗ | ✗ | ✓ | ✓ |
| Transparent agent daemon | ✓ (DNS + VIP + tunnel) | ✗ | ✓ (Tailscale client) | ✗ | ✗ |
| Device enrollment (QR/token) | ✓ | ✗ | ✓ (SSO) | ✓ (IdP) | ✓ (token) |
| Forward secrecy | ✓ (per-session) | ✓ | ✓ | ✓ (TLS) | ✓ (TLS) |

| Capability | ZTLP | WireGuard | Tailscale | Cloudflare Access | Teleport |
|---|---|---|---|---|---|
| Open protocol/spec | ✓ (full spec + impl) | ✓ | ✗ (proprietary) | ✗ (proprietary) | ✓ |

**Key differentiator:** ZTLP is a transport protocol — not a product, not a service, not an application proxy. It provides the identity and session layer that existing systems implement partially or proprietarily. WireGuard provides excellent encryption but no identity namespace, no relay mesh, and no gateway model. Tailscale adds coordination but depends on a single proprietary control plane. Cloudflare Access protects HTTP but not arbitrary protocols. Teleport provides identity-first access for specific infrastructure (SSH, databases, Kubernetes) but operates as an application-layer proxy rather than a general-purpose transport.

ZTLP aims to be the layer beneath all of these — the transport primitive that makes identity-first connectivity a protocol property rather than a product feature.

---

## 17. Post-Quantum Migration Path

ZTLP's current cryptographic suite (X25519, Ed25519, ChaCha20-Poly1305, BLAKE2s) provides ~128-bit classical security. The symmetric primitives (ChaCha20, BLAKE2s) retain adequate security under quantum models (Grover's algorithm halves the effective key length, leaving 128-bit security). The asymmetric primitives (X25519, Ed25519) are vulnerable to Shor's algorithm on a fault-tolerant quantum computer.

### 17.1 Migration Strategy

ZTLP's migration to post-quantum cryptography follows a staged approach:

**Phase 1 — Hybrid Key Exchange:** Replace the Noise_XX ephemeral key exchange with a hybrid construction combining X25519 with ML-KEM (CRYSTALS-Kyber, NIST FIPS 203). This provides quantum resistance for session key derivation while preserving classical security as a fallback. The handshake payload increases by ~1,568 bytes (ML-KEM-768 public key + ciphertext) — acceptable for the three-message handshake.

**Phase 2 — Hybrid Signatures:** Replace Ed25519 identity signatures with a hybrid scheme combining Ed25519 with ML-DSA (CRYSTALS-Dilithium, NIST FIPS 204). This increases ZTLP-NS record sizes and handshake authentication payloads but preserves the existing trust model. ML-DSA-65 signatures are 3,309 bytes — larger than Ed25519's 64 bytes, but manageable for handshake and namespace operations that are not on the per-packet data path.

**Phase 3 — Pure Post-Quantum:** Once confidence in lattice-based constructions is established through widespread deployment and continued cryptanalysis, the hybrid fallback may be removed. This phase is not expected before 2030.

### 17.2 Forward Secrecy Under Quantum Threat

ZTLP's existing forward secrecy property provides a meaningful defense: an attacker who records encrypted sessions today cannot derive past session keys even after obtaining a quantum computer, because the ephemeral X25519 values were never stored. The hybrid ML-KEM extension strengthens this by making future sessions quantum-resistant as well.

### 17.3 Design Decisions

- **Symmetric primitives unchanged:** ChaCha20-Poly1305 and BLAKE2s are not quantum-vulnerable at their current security levels.
- **Per-packet overhead unchanged:** Post-quantum primitives affect only the handshake and namespace operations. The compact 42-byte data header is unaffected.
- **Algorithm agility deferred:** The current specification fixes the algorithm suite to prevent downgrade attacks. Post-quantum migration will be introduced via a versioned CryptoSuite field in the handshake header, with explicit negotiation rules.

---

## 18. Deployment Roadmap

### Stage 1 — Private Network Tool (Deployed)

ZTLP has been validated in a production deployment protecting a web application (Ruby on Rails) behind a ZTLP gateway on AWS infrastructure. The deployment consists of a ZTLP gateway with named backend services, an NS server for identity resolution, and a relay for session routing — all deployed as Docker containers via the Bootstrap management server. A macOS client enrolled with a cryptographic identity connects through an encrypted ZTLP tunnel to access the application with zero exposed HTTP ports. This deployment demonstrates the complete lifecycle: identity generation, device enrollment, tunnel establishment, gateway policy enforcement, and bidirectional application data flow.

### Stage 2 — Enterprise Adoption

Organizations deploy ZTLP at scale for compliance, access control, and remote workforce requirements. Enterprises operate their own trust roots, enrollment authorities, and relay infrastructure. The Bootstrap web UI enables fleet management with group-based access policies. Managed service providers deploy ZTLP across client networks using the MSP deployment guide, enrollment tokens, and QR-code-based device onboarding.

### Stage 3 — Service Provider Integration

Cloud providers, CDN operators, and security vendors integrate ZTLP gateway and relay functionality into their platforms. Organizations adopt ZTLP-secured access with-

out operating relay infrastructure themselves. External IdP integration (OIDC, SAML, LDAP) enables enterprise SSO workflows during enrollment.

### Stage 4 — Public Ecosystem

ZTLP becomes a generally available Internet security layer. Public services offer ZTLP-authenticated access alongside legacy Internet access. Independent relay operators establish commercial relay networks. Developer SDKs (Rust, Go) enable integration across application frameworks.

---

## 19. Conclusion

The Internet's anonymous connectivity model was designed for an era when the primary challenge was making communication possible. That era has passed. The primary challenge now is making communication trustworthy — and the current model, where security is applied after connectivity is already established, is structurally inadequate.

ZTLP proposes a different foundation. By making cryptographic identity a precondition for connectivity rather than an afterthought, the protocol eliminates entire categories of attack at the transport layer. Services become invisible to unauthorized parties. DDoS floods are rejected in nanoseconds. A person-centric identity model maps real-world organizational relationships — people, their devices, and their roles — into cryptographic policy enforcement. Relay infrastructure scales horizontally through consistent-hash distribution and label-switching forwarding. Existing applications gain identity-first protection without modification, deployed incrementally through edge gateways. A transparent agent daemon makes encrypted tunnels invisible to applications.

The reference implementation — over 30,000 lines of Rust, Elixir/OTP, C, and Ruby with 2,278 tests and zero failures — demonstrates that this architecture is not merely theoretical. A production deployment protecting a live web application confirms that ZTLP works end-to-end: from device enrollment through encrypted tunnel establishment to gateway policy enforcement and application delivery, with zero exposed ports. The benchmarks show that identity-first networking is practical at Internet scale: 19-nanosecond packet rejection, sub-300-microsecond handshake completion, 400+ MB/s tunnel throughput, and multi-gigabit steady-state gateway capacity on commodity hardware.

ZTLP is not a product. It is a protocol specification and open reference implementation, released under Apache 2.0, designed to be the transport-layer foundation for identity-first networking. The specification, implementation, benchmarks, and documentation are available at **ztlp.org**.

The Internet does not need a new application. It needs a new transport primitive — one where identity comes first.

---

## References

1. Noise Protocol Framework. Trevor Perrin, 2018.
2. RFC 7748 — Elliptic Curves for Security (X25519).
3. RFC 8032 — Edwards-Curve Digital Signature Algorithm (Ed25519).
4. RFC 8439 — ChaCha20 and Poly1305 for IETF Protocols.
5. RFC 7693 — The BLAKE2 Cryptographic Hash and Message Authentication Code.
6. RFC 9000 — QUIC: A UDP-Based Multiplexed and Secure Transport.
7. RFC 8446 — The Transport Layer Security (TLS) Protocol Version 1.3.
8. NIST FIPS 203 — Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM).
9. NIST FIPS 204 — Module-Lattice-Based Digital Signature Algorithm (ML-DSA).
10. WireGuard: Next Generation Kernel Network Tunnel. Donenfeld, J., NDSS 2017.
11. BeyondCorp: A New Approach to Enterprise Security. Ward & Beyer, Google.
12. SCION: A Secure Internet Architecture. Barrera et al., ETH Zurich.
13. RFC 6298 — Computing TCP's Retransmission Timer (Jacobson/Karels).
14. RFC 3517 — A Conservative SACK-based Loss Recovery Algorithm for TCP.
15. RFC 6937 — Proportional Rate Reduction for TCP.

---