Trident System Requisites

Trident is the transport library shared by Kraken and standalone tooling.

Architecture Overview

Trident exposes a `Conduit` abstraction that wraps transport-layer streams and datagrams (TCP, TLS, UDP, DTLS). Conduits can be stacked to form pipelines such as `tcp → tls`. Every conduit surfaces the same life-cycle (`Dial`, `Close`), metadata reporting, pooled buffers and context-aware send/receive helpers.

---

1. High-Level Requirements

1.1 Core Functionality

- **HL-F1** — Provide a single `Conduit` interface that covers transport-layer stream and datagram interactions.
- **HL-F2** — Allow conduits to be stacked so developers can build custom protocol pipelines (e.g., TLS over TCP, DTLS over UDP).
- **HL-F3** — Ship production-ready conduits for TCP, UDP, TLS, and DTLS so Kraken modules can be wired without additional plumbing.
- **HL-F4** — Expose hooks for custom conduits so modules can add domain-specific layers (logging, fuzzing, replay, etc.).
- **HL-F5** — Capture timing/metadata for every send/receive and reuse pooled buffers to keep allocations predictable.

1.2 Developer Experience

- **HL-U1** — Publish Trident as an idiomatic Go module with a stable, versioned API surface.
- **HL-U2** — Make all blocking calls context-aware (`Dial`, `Recv`, `Send`, etc.).

1.3 System Qualities

- **HL-Q1** — Keep the library light: only rely on the Go standard library and a handful of well-known packages.
- **HL-Q2** — Ensure conduits can be used safely from multiple goroutines by guarding shared state.
- **HL-Q3** — Prefer secure defaults for TLS/DTLS (system root pool, hostname verification) while still allowing callers to override settings explicitly.

---

2. Low-Level Requirements

2.1 Core Conduit API (TRD-API)

Architectural

- **TRD-API-A1** — The `Conduit[V]` interface shall expose `Dial`, `Close`, `Kind`, `Stack`, and `Underlying`.

- **TRD-API-A2** — `Dial` shall be idempotent; calling it multiple times does not re-open the transport.
- **TRD-API-A3** — `Close` shall be safe to call multiple times and release underlying resources.
- **TRD-API-A4** — `Stack()` shall list the active protocol layers.

Functional
- **TRD-API-F1** — `Underlying()` shall return the interface (Stream, Datagram).
- **TRD-API-F2** — All send/receive methods shall accept `context.Context` plus per-call options for deadlines or buffer sizing.
- **TRD-API-F3** — Metadata objects shall be returned for every operation with timestamps, protocol hints, and extensible fields.

2.2 Layer Interfaces (TRD-LAYER)
- **TRD-LAYER-F1** — Stream conduits shall provide `Recv`, `Send`, `Close`, `CloseWrite`, `SetDeadline`, `LocalAddr`, `RemoteAddr`.
- **TRD-LAYER-F2** — Datagram conduits shall expose `Recv`, `Send`, and address accessors using `netip.AddrPort`.

2.3 Built-in Conduits (TRD-IMPL)
- **TRD-IMPL-F1** — Provide TCP stream conduits with options for keep-alive, graceful close, and immediate tear-down.
- **TRD-IMPL-F2** — Provide UDP datagram conduits with send/receive helpers and address metadata.
- **TRD-IMPL-F3** — Provide TLS/DTLS decorators that wrap another stream or datagram conduit and extend the stack with `tls` / `dtls`.
- **TRD-IMPL-F4** — Provide optional logging adapters that wrap any conduit and emit structured traces for debugging.
- **TRD-IMPL-N1** — Built-in conduits shall surface actionable errors when the OS denies privileges (e.g., sockets without sufficient permissions).
- **TRD-IMPL-N2** — Reuse pooled buffers for hot paths to avoid repeated allocations under load.

2.4 Security & TLS (TRD-SEC)
- **TRD-SEC-F1** — TLS/DTLS conduits shall accept a `tls.Config` / DTLS config so callers can enable mTLS, PSKs, custom roots, or insecure modes when testing.
- **TRD-SEC-F2** — Closing a TLS conduit shall also close the wrapped transport.

2.5 Tooling & Documentation (TRD-DOC)
- **TRD-DOC-F1** — Every public type or option shall include GoDoc comments.
- **TRD-DOC-F2** — Provide runnable examples (in `docs` or `_test.go`) showing how to stack conduits and how to stream data.
- **TRD-DOC-F3** — Offer mock/null conduits so downstream projects can run tests without touching the network.