Braid Tendril v0.8.2-alpha (Sovereign WAN Mesh Node Prototype)

Braid Tendril v0.8.2-alpha is a WAN-kernel alpha with signed identities, verifiable 384d vector envelope fixtures, persistent trust policy, and local runtime vector-transfer demo. It operates completely cloud-free, allowing local enclaves and edge devices to establish policy-gated neural coordinate synchronizations over physical subnets and mapped WAN tunnels.

Note: This is an experimental pre-1.0 developer release. It is designed as a secure, WAN-routing prototype, not a production-complete WAN solution.

This release represents **Braid Tendril v0.8.2-alpha** (Package `0.8.2-alpha` / Wire Protocol version `8`).

Architectural Specifications

1. **NodeId-First WAN Routing**: Dynamic route resolution via `send_to_node(NodeId, envelope)`. Transport and session layers communicate exclusively using 32-byte hash NodeIds rather than raw physical socket addresses.
2. **Untrusted Signed-Presence Rendezvous**: Sockets and route hints are published as signed `BraidPresenceRecord` announcements containing raw public key materials and absolute expirations (`expires_at_ms`). Edge nodes verify presence authenticity directly against these keys before updating rendezvous stores.
3. **Cryptographic Handshake State Machine**: Handles multi-message handshake sequences over direct routes (`MSG_SESSION_HELLO`, `MSG_SESSION_CHALLENGE`, `MSG_SESSION_RESPONSE`, and `MSG_SESSION_READY`) to authenticate peer NodeId identity before state transitions to `Ready`.
4. **Transit Relay Forwarding**: Supports transit packet forwarding via signed `BraidRelayFrame` envelopes supporting target NodeId routing, sender tracking, and absolute TTL expirations to prevent loop leaks. Supports final-recipient inline packet ingestion.
5. **Message-Type-Keyed Replay Ledger**: Protects state against cross-protocol replay threats by tracking sequence numbers independently per message type.
6. **Quantized INT8 Decompression**: Active decompression pipeline for 384-dimension quantized vector states.

Verification & Clean Builds

Ensure that the project passes all pre-release tests before commits:

```bash
# 1. Run the local conformance verification suite
python3 conformance_verify.py

# 2. Compile and test Rust backend
```

```
# (Run from src-tauri folder)
cargo check
cargo test
```

Quickstart

```bash
# 1. Install dependencies
npm install

# 2. Compile and run frontend
npm run dev

# 3. Launch Braid WAN Daemon (Rust backend)
npm run tauri dev -- --port 12000
```