Lateralus Energy Pipeline Protocol

A typed pipeline model for monitoring and controlling distributed energy systems

Lateralus Language

bad-antics · April 2026 · Lateralus Language Research

ABSTRACT This paper defines the Lateralus Energy Pipeline Protocol (LEPP), a typed data exchange protocol for distributed off-grid energy systems. LEPP uses the Lateralus pipeline model to express energy flows between generators, storage units, and loads. Each component exposes a typed pipeline interface; the system controller composes these interfaces into a whole-system pipeline. LEPP provides: real-time telemetry, deterministic fault response, and formal energy balance verification at the type level.

1. Motivation

Off-grid energy systems consist of heterogeneous components from multiple vendors: solar charge controllers, battery management systems, inverters, generator controllers, and load management units. These components communicate via incompatible protocols (Modbus RTU, CAN bus, proprietary UART, or none at all).

LEPP provides a unified typed interface layer. Each component exposes a LEPP adapter; the system controller communicates with all components through a single typed pipeline. Adapters translate between LEPP messages and the component's native protocol.

2. Core Types

LEPP defines four core measurement types:

```
// Power and energy
record PowerSample  { watts: f32, timestamp: Instant }
record EnergySample { watt_hours: f32, period_ms: u32 }

// State of charge and voltage
record BatteryState { soc_pct: f32, voltage_v: f32,
                      current_a: f32, temp_c: f32 }

// Component health
record HealthStatus { component_id: ComponentId,
                      fault_code: Option<FaultCode>,
                      uptime_s: u64 }
```

All measurements are timestamped with monotonic clock values. The protocol uses 32-bit floats for measurements (sufficient for energy system precision) to reduce bandwidth on low-speed serial links.

3. Pipeline Interface Definition

Each LEPP component exposes a typed pipeline interface with three channels:

```
// LEPP component interface
interface EnergyComponent {
    // Telemetry: component pushes measurements
    fn telemetry_stream() -> Pipeline<(), PowerSample>

    // Control: controller pushes commands
    fn control_sink() -> Pipeline<ControlCommand, Result<(), FaultCode>>

    // Health: periodic health check
    fn health_check() -> Pipeline<(), HealthStatus>
}
```

The controller composes component pipelines into a system pipeline. Type checking at composition time verifies that control commands sent to a component match the commands it accepts — a command type mismatch is a compile error.

4. Energy Balance Verification

LEPP uses phantom type parameters to track energy balance at the type level. The controller's power dispatch function is typed to reject configurations where generation is less than load:

```
// Phantom types for energy accounting
type Watts<N: u32>;   // N watts available

fn dispatch<G: u32, L: u32>(
    generation: PowerSource<Watts<G>>,
    load:       LoadSink<Watts<L>>,
) -> DispatchPlan
where
    G >= L  // compile error if generation < load
{  ...  }
```

In practice, generation and load are dynamic values, so the phantom type check operates at the configuration level (static capacity planning) rather than the instantaneous dispatch level. Runtime balance is enforced by the fault propagation pipeline.

5. Fault Propagation

LEPP uses the |?> operator to propagate faults through the energy dispatch pipeline. A fault in any stage triggers the fault handler, which activates the safe shutdown sequence:

```
fn energy_dispatch_loop(system: &SystemState) -> Result<(), Fault> {
    system
        |>  read_all_telemetry
        |?> check_battery_limits       // fault if SOC < 10%
        |?> check_generator_health     // fault if temp > 85°C
        |>  compute_dispatch_plan
        |?> send_control_commands      // fault on comms error
        |>  log_dispatch_record
}
```

When a fault is raised, the pipeline aborts and the fault is passed to the safe-shutdown handler. The handler follows a deterministic sequence: disconnect loads in priority order, then disconnect sources, then notify the operator.

6. Transport Layer

LEPP messages are serialized using CBOR (Concise Binary Object Representation, RFC 8949) for compact encoding on serial links. The transport options are:

```
Transport         Baud / bandwidth    Typical use
-----------------------------------------------
RS-485 (Modbus)   9600-115200 baud    Legacy components
CAN 2.0B          1 Mbit/s            Modern components
Ethernet/TCP      100 Mbit/s          Networked inverters
Meshtastic/LoRa   250 kbaud            Remote monitoring
```

Each transport has a LEPP adapter that handles framing, CRC, and retry logic. The pipeline interface above the adapter is transport-agnostic; swapping a component's transport requires only replacing its

adapter.

7. Reference Implementation

The LEPP reference implementation is part of the Lateralus standard library under lateralus-energy. It provides:

- Adapter implementations for Modbus RTU, CAN bus, and TCP.
- Simulator adapters for testing without physical hardware.
- A system dashboard that renders the energy pipeline in the terminal.
- A logging backend that records all telemetry to a SQLite database.

```
# Run the system dashboard against a QEMU-simulated system
ltl run lateralus-energy::dashboard \
    --config examples/5kw-solar-hho.toml \
    --transport simulator
```

8. Future Work

Planned extensions to LEPP:

- **Formal energy balance proofs**: use the Lateralus formal verification framework to prove that a given system configuration maintains energy balance under all possible fault scenarios.
- **Grid-tie mode**: extend LEPP to model bidirectional grid connection (net metering), where the grid acts as an infinite storage/source component.
- **Multi-site federation**: federate multiple off-grid sites via Meshtastic mesh networking, allowing peer-to-peer energy trading between proximate sites.