MI Diagnostics – System Design Document

Author: James Covert

Date: October 2025

This document describes the architecture, design philosophy, and implementation details of the MI Diagnostics system — a distributed telemetry and diagnostics prototype that demonstrates reliable data collection, anomaly detection, fault tolerance, and runtime configuration across multiple embedded nodes.

Goals

The primary goal of this system is to demonstrate a robust and maintainable distributed telemetry framework capable of reliable data acquisition, anomaly detection, and diagnostics reporting across multiple embedded nodes. The system ensures continuous operation even under intermittent network outages through replay mechanisms, while supporting dynamic configuration updates and bounded resource usage. The design prioritizes simplicity, testability, and observability for easy extension to production environments.

System Architecture

The MI Diagnostics system uses a two-tier architecture: multiple edge nodes performing telemetry collection and diagnostics, and a host controller that aggregates, verifies, and logs this data. Communication occurs over a lightweight, line-delimited JSON protocol using TCP connections. The design allows the host to handle multiple nodes simultaneously and supports replay of unacknowledged data after network interruptions.

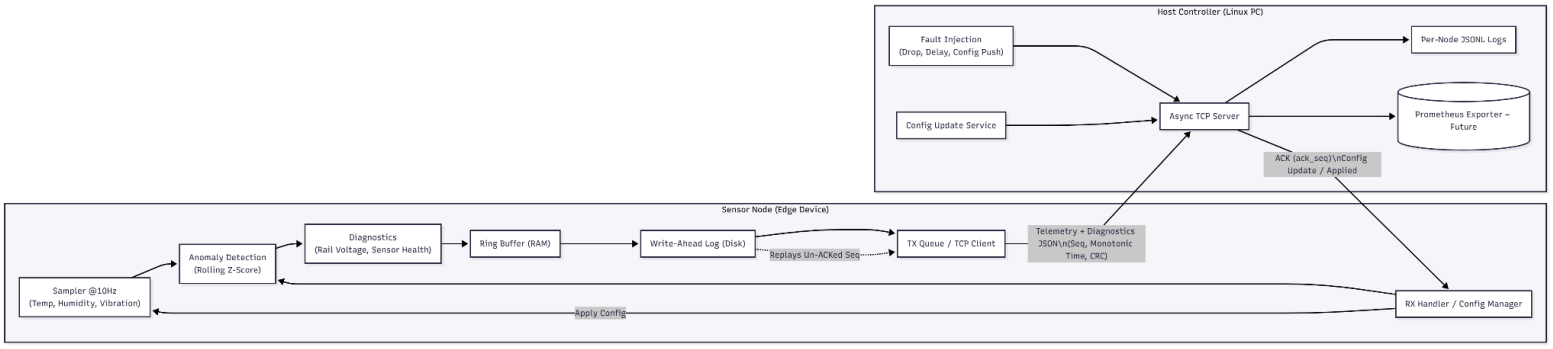

Figure 1: System Architecture Diagram – showing data flow from sensor nodes to the host controller. Each node samples data, performs anomaly and diagnostic processing, and transmits telemetry messages with unique sequence numbers. The host acknowledges receipt, logs the data, and can send configuration updates back to the node.

Dataflow Overview

1. Sensors produce raw readings (temperature, humidity, vibration).
2. The node performs anomaly detection using rolling z-scores and executes basic diagnostics checks.
3. Results are serialized into JSON messages containing sequence numbers and timestamps.
4. Data is stored in a bounded ring buffer and persisted to a write-ahead log (WAL).
5. The node transmits telemetry via TCP to the host.
6. The host logs the data and returns acknowledgments (ACKs).
7. Upon reconnection after outage, nodes replay unacknowledged records.
8. Configuration updates flow from host to nodes for runtime adjustment.

Subsystem Descriptions

Host

The host subsystem manages multiple node connections concurrently. It acts as the central data sink, ensuring data integrity, timing verification, and durable logging. It also serves as the configuration authority for connected nodes. The host includes fault-injection capabilities to simulate network failures and verify system resilience.

Node

Each node is responsible for periodic data sampling at 10 Hz, computing rolling z-score anomalies, performing diagnostic health checks, and safely transmitting data to the host. Nodes maintain a bounded in-memory ring buffer and a persistent WAL for reliability. The node's transmit/receive loop handles acknowledgments and configuration updates, ensuring deterministic and fault-tolerant operation.

Key Design Decisions and Trade-offs

1. **Protocol:** JSON lines were chosen for clarity and ease of debugging during the prototype phase. In production, a binary protocol or Protobuf framing could reduce overhead.
2. **Durability:** WAL guarantees recovery from power loss or connection drops. Compacting acknowledged entries limits disk usage.
3. **Ordering & Integrity**: Each telemetry message has a unique sequence ID, ensuring ordered replay and validation.
4. **Anomaly Detection:** The lightweight O(1) z-score algorithm adapts to real-world signal drift while maintaining responsiveness.
5. **Runtime Config:** Host-sent configuration messages allow threshold tuning without redeployment.
6. **Memory Management:** Nodes track buffer fill percentage and mark themselves degraded when nearing capacity.

Reliability, Timing, and Data Integrity

The system guarantees unique and ordered telemetry through per-node sequence counters. Monotonic timestamps support temporal ordering even if system clocks are adjusted. Data integrity is verified by ACK-based delivery; only acknowledged frames are compacted from persistent storage.

Outage Tolerance and Replay

If connectivity between a node and the host is lost, the node continues sampling uninterrupted. Each telemetry record is appended to the WAL and transmitted once connectivity is restored. The host acknowledges the highest contiguous sequence number received, allowing the node to safely compact its log and maintain exactly-once semantics.

Potential Security and Firmware Update Design

1. **A/B Partition Updates:** Firmware is downloaded to an inactive slot and activated only after validation.
2. **Signed Images & Secure Boot:** Ensures only verified firmware runs.
3. **Anti-Rollback Protection:** Prevents downgrading to older, potentially insecure versions.
4. **Health Check and Rollback:** Automatic rollback to a stable version if post-update validation fails.
5. **Transport Security:** Encrypted communication with mTLS and certificate pinning.

Potential Future Extensions

1. **Binary Framing + TLS:** Improve efficiency and security.
2. **Prometheus/Grafana Integration:** Real-time monitoring of anomaly rates and replay backlog.
3. **Local Health Dashboard:** On-device LED or web interface for field diagnostics.
4. **Automated Testing and CI:** Continuous validation of core mechanisms such as WAL compaction and replay logic.

Conclusion

The diagnostics system demonstrates an architecture that balances simplicity with reliability, providing a foundation for real-world distributed telemetry deployments. Its modular design allows incremental enhancement toward production-grade embedded software while maintaining strong fault tolerance and configurability.