Project Proposal: Emergent-Time Multi-Hop IoT Mesh

Using CoAP with Statistical Synchronization and Central Logging

Student: Arjan Werren

MSc Data Science, ZHAW IDP

Course: IOT1

Date: November 20, 2025

Estimated Effort: 20 hours

1. Introduction & Motivation

Industrial IoT deployments often lack global time sources (NTP, GPS, PTP), yet synchronized time is critical for correlating distributed sensor readings and interpreting system states.
This project builds a minimal four-node IoT mesh where time synchronization emerges statistically from peer-to-peer delay measurements exchanged over CoAP. No device acts as clock master; instead, a shared timescale arises from local interactions.

Core demonstration:

- Four LEDs blinking in perfect phase (visual proof of sync)
- Multi-hop sensor data with consistent timestamps
- Disturbance injection and recovery
- Central logging and web visualization

The system is small enough for lab deployment but demonstrates real distributed systems concepts: statistical consensus, multi-hop routing, and timestamp coherence without external references.

2. System Context (Fixed)

2.1 Hardware Setup

Four Raspberry Pis: A, B, C, D

Each Pi has:

- 1 Grove LED
- 1 Grove sensor (temperature, light, or potentiometer)
- Static IP address on same Ethernet subnet
- Node D additionally has: 1 button for disturbance injection

Physical topology (ring neighbors):

```
A ⟷ B
↑     ↓
↓     ↑
D ⟷ C
```

Sync beacons flow between all adjacent pairs: A–B, B–C, C–D, D–A.

Logical routing tree (rooted at C):

```
         C (logger, root)
        / \
       B   D
      /
     A
```

Data flows: A → B → C, D → C

2.2 Network Configuration

- **Transport:** Ethernet with dedicated switch (static IPs)
- **Protocol:** CoAP over UDP
- **Implementation:** Python with aiocoap library
- **Link-layer agnostic design:** Will work identically over Wi-Fi if needed

3. MVP Scope – Core Deliverables

3.1 Time Synchronization Layer

4-timestamp exchange between each neighbor pair:

```
Node i                          Node j
------                          ------
t1 = send time (i)       ─────>
                           t2 = recv time (j)
                           t3 = send time (j)
t4 = recv time (i)       <─────
```

Per-link metrics computed on each node:

Round-trip delay:

```
δ_ij = (t4 - t1) - (t3 - t2)
```

Clock offset (i relative to j):

```
θ_ij = [(t2 - t1) + (t3 - t4)] / 2
```

Robust jitter estimation:
For each link, maintain rolling window of δ_{ij} samples (30-40 values). Compute jitter using interquartile range:

$$\sigma_{ij} \approx 0.7413 \times \mathrm{IQR}(\delta_{ij})$$

This filters outliers and asymmetric delays without parametric assumptions.

Inverse-variance fusion:
Each node fuses offsets from all neighbors:

$$\hat{\theta}(i) = \Sigma_j \, (\theta_{ij} / \sigma_{ij}^2) \, / \, \Sigma_j \, (1 / \sigma_{ij}^2)$$

Stable links (low σ) receive higher weight. Noisy links have minimal influence.

Slew-limited adjustment:
Offset corrections applied gradually to prevent oscillation:

```
max adjustment rate = 5 ms/s
```

Mesh time definition:

$$T_{mesh}(i) = T_{monotonic}(i) + \hat{\theta}(i)$$

Where T_monotonic is clock_gettime(CLOCK_MONOTONIC).

Key property: Because each node adjusts its local clock with $\hat{\theta}(i)$, and these offsets converge through shared peer constraints, all T_mesh(i) align onto a common timescale. Events timestamped with T_mesh are directly comparable across nodes.

Beacon timing:
To avoid collisions, beacons sent at:

```
interval = 1.0s ± uniform(±0.1s)
```

3.2 LED Synchronization Demo

Each node blinks its LED when the mesh time crosses multiples of 500 ms (e.g., T_mesh(i) % 0.5 < ε for a small ε).

Expected behavior:
- Cold start: LEDs blink randomly (unsynchronized)
- After ~20-30s: All four LEDs blink in phase (±10ms)
- Visual proof of time synchronization quality

3.3 Disturbance Mechanism

Button on Node D:

When pressed:

- Sends CoAP multicast POST /sync/disturb with random offset (±200ms)
- All nodes receive and corrupt their $\hat{\theta}$: $\hat{\theta}$ += disturbance
- LEDs immediately desynchronize
- Statistical fusion gradually restores consensus
- LEDs re-align within ~20 seconds

Purpose: Demonstrates robustness and convergence behavior.

3.4 Routing & Multi-Hop Data

Static routing configuration (routing.json):

```
{
  "A": {"parent": "B"},
  "B": {"parent": "C"},
  "C": {"parent": null},
  "D": {"parent": "C"}
}
```

CoAP endpoints:

Endpoint	Method	Purpose
/sync/beacon	POST	4-timestamp exchange
/relay/ingest/sensor	POST	Multi-hop sensor forwarding
/sync/disturb	POST	Chaos injection
/status	GET	Sync quality metrics

Forwarding logic:
Node A samples sensor → sends to parent B → B forwards to C → C logs to database
Node D samples sensor → sends directly to C (one hop)

3.5 Sensor Sampling with Mesh Timestamps

Each Pi samples its Grove sensor periodically (1 Hz).

Message payload:

```
{
  "node_id": "A",
  "mesh_timestamp": 1234567.890,
  "local_monotonic": 9876.543,
  "sensor_type": "temperature",
  "value": 23.4,
  "hop_count": 2,
  "path": ["A", "B", "C"]
}
```

Each forwarding hop appends its ID to path and increments hop_count.

3.6 Central Logging on Node C

SQLite database (mesh_data.db):

```
CREATE TABLE sensor_readings (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    node_id TEXT NOT NULL,
    mesh_timestamp REAL NOT NULL,
    local_monotonic REAL NOT NULL,
    sensor_type TEXT NOT NULL,
    value REAL NOT NULL,
    hop_count INTEGER NOT NULL,
    path TEXT NOT NULL,
    received_at REAL NOT NULL,
    UNIQUE(node_id, mesh_timestamp)
);
```

Simple INSERT-only writer. Uses SQLite WAL mode for concurrent access.

3.7 Minimal Web UI on Node C

Flask-based single-page interface showing:

- **Latest sensor values:** Current reading from each node (A, B, D), mesh timestamp of last reading, multi-hop path visualization
- **Basic time-series plot:** A simple time-series plot of recent readings (e.g., last 50–100 points) from at least two nodes, with mesh time on the x-axis
- **Sync status table:** Current $\hat{\theta}$ per node, aggregate σ per node, number of active neighbors

Access: http://<node_c_ip>:5000/

3.8 Basic Sync Metrics

Each node tracks:

```
current_offset = θ^(i)

aggregate_jitter = weighted_mean(σ_ij)

num_neighbors = len(active_peers)
```

Node C periodically queries /status from all nodes and logs sync data. Stored in memory and displayed on web UI.

Key observables:
- Offset stability over time
- Convergence after disturbance
- LED phase alignment (visual + quantitative)

4. Architecture

4.1 Layered Design

```
+-----------------------------------------------------------------+
|  Web UI (Flask) + SQLite Logger                                 |
+---------------------- Application Layer -----------------------+
|  Sensor Sampling      |  LED Blinking      |  Data Ingestion   |
+---------------------- Mesh-Time Layer -------------------------+
|  4-Timestamp Sync  |  IQR Jitter Est.  |  Inverse-Var Fusion |
+---------------------- Routing Layer ---------------------------+
|  Static Parent Tables          |  Hop-by-Hop Forwarding       |
+---------------------- Transport Layer -------------------------+
|  CoAP over UDP                 |  Ethernet                     |
+-----------------------------------------------------------------+
```

Key separation:

- **Sync layer:** Uses physical ring neighbors (A–B–C–D–A)
- **Routing layer:** Uses logical tree parent (configured statically)

5. Implementation Plan (20 Hours)

Week 1 (5h): Core Synchronization

- **Task 1.1 (2h):** CoAP server setup + basic endpoints
- **Task 1.2 (3h):** 4-timestamp exchange, compute θ_{ij} and δ_{ij} per link

Milestone: Each node can measure offset/delay to neighbors.

Week 2 (5h): Fusion + LED Demo

- **Task 2.1 (2h):** Jitter estimation + fusion
- **Task 2.2 (2h):** LED synchronization
- **Task 2.3 (1h):** Disturbance button

Milestone: Four LEDs blink in phase, desync on button press, resync within 20s.

Week 3 (5h): Data Pipeline

- **Task 3.1 (2h):** Static routing + forwarding
- **Task 3.2 (1h):** Sensor sampling
- **Task 3.3 (2h):** SQLite logging on C

Milestone: Sensor readings from A and D logged to database with mesh timestamps.

Week 4 (5h): UI + Polish

- **Task 4.1 (2h):** Minimal web interface
- **Task 4.2 (1h):** Sync metrics collection
- **Task 4.3 (2h):** Testing + documentation

Milestone: Complete working demo ready for presentation.

6. Performance Targets

Metric	Target	Measurement
Cold-start convergence	< 30s	Time to LED synchrony (±10ms)
Steady-state jitter	< 5ms	LED phase error over 5 min
Disturbance recovery	< 20s	Visual LED re-alignment
End-to-end latency	< 200ms	received_at - mesh_timestamp
Database throughput	1 sample/s/node	Sustained over 10 minutes

These targets are realistic for an Ethernet-based lab setup and serve as guideline performance goals for evaluating the system.

7. Configuration Parameters

Parameter	Value	Rationale
Beacon interval	1.0s ± 0.1s	Balance update rate vs. network load
Jitter window size	30 samples	~30s history for robust estimation
Slew limit	5 ms/s	Smooth adjustment without LED flicker
Sensor sample rate	1 Hz	Sufficient for demo, low overhead
LED blink period	500 ms	Easily visible synchrony
Disturbance magnitude	±200ms	Dramatic desync, recoverable

9. Expected Outcomes

Upon completion, this project demonstrates:

- **Distributed clock synchronization** without external time source
- **Statistical inference** for robust estimation under uncertainty (IQR-based σ, inverse-variance weighting)
- **Multi-hop routing** and data forwarding in IoT mesh
- **Embedded systems integration** (sensors, LEDs, GPIO)
- **Database management** and web visualization

Deliverables:

- Working 4-node mesh with synchronized LEDs
- SQLite database with multi-hop sensor readings
- Minimal web interface showing real-time data
- Video demonstration of disturbance/recovery
- Brief technical report (~5 pages) analyzing convergence

10. Conclusion

This project builds a complete IoT mesh demonstrating emergent time synchronization, multi-hop routing, and centralized data collection—all without external infrastructure.

The synchronization algorithm uses proven statistical techniques (robust estimation, inverse-variance fusion) adapted to a distributed setting. The visual LED demo provides immediate feedback on time quality, while the database and web UI satisfy course requirements for data storage and remote access.

The scope is deliberately focused on a 20-hour implementation timeline, with clear milestones and fallback strategies for common risks. The architecture remains extensible for future enhancements (dynamic routing, larger meshes, external time anchoring) while delivering a complete, working system within lab constraints.

Submitted by: Arjan Werren
Email: werrearj@students.zhaw.ch
Date: November 20, 2025