CS408 PROJECT

DRONE-EDGE-MONITORING

Ertem Ata Kavaz – 32790

Ege Çetinkaya - 32661

Karya Üstünçelik - 30903

1. Abstract:

This project implements a **three-tier** environmental monitoring simulation in Python:

1. **Sensor Nodes –** Data Generation
2. **Drone Gateway –** Edge Processing
3. **Central Server** – Data Visualization

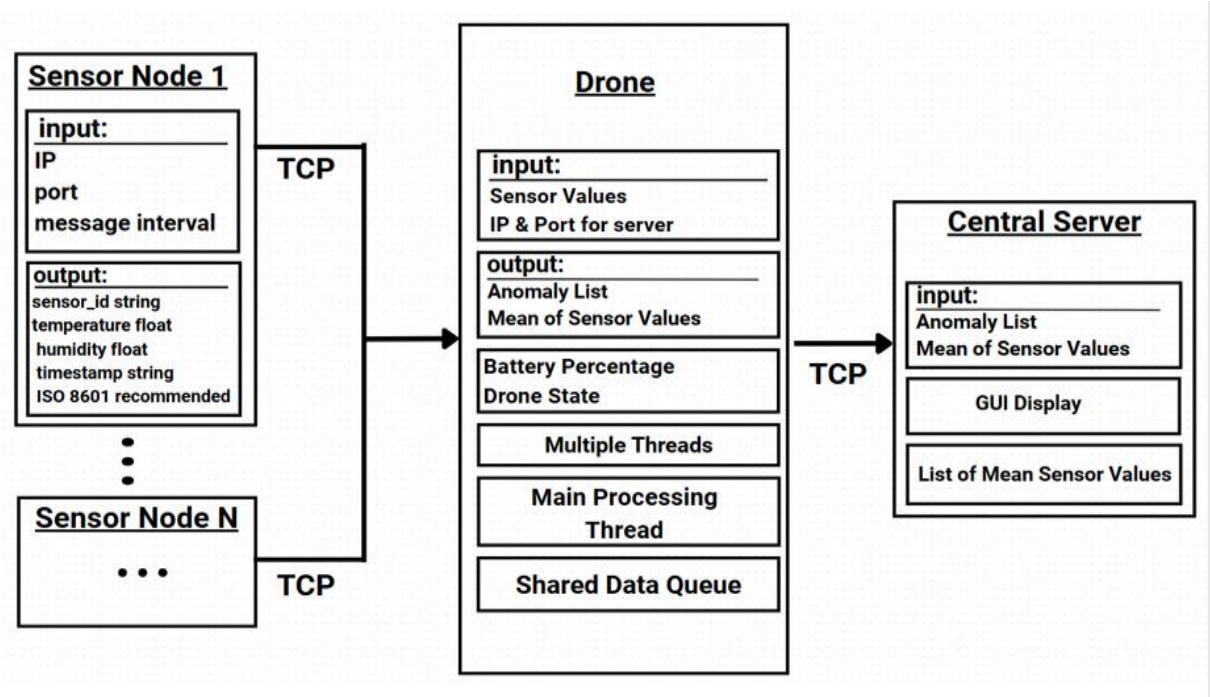
Sensor Nodes generate periodic temperature and humidity readings and transmit them via TCP. The Drone Gateway aggregates readings in rolling windows, detects out-of-range anomalies, simulates a battery that triggers return-to-base behavior, and queues data when offline or charging. Summarized statistics are forwarded to the Central Server, which displays live dashboards of all sensors. The system demonstrates real-time networking, multithreading, stateful edge processing, GUI design with Tkinter, and fault-tolerant logging.

2. Introduction & Motivation:

Modern environmental monitoring often relies on distributed IoT deployments that must handle *"intermittent connectivity"*, *"localized data processing"*, and *"efficient bandwidth usage"*. In remote or resource-constrained settings (e.g., forests, agricultural fields), sending every raw sensor reading to a central cloud server is impractical.

By moving aggregation and anomaly detection to an intermediate "drone" node, we can reduce network load, react more quickly to critical events, and emulate realistic return-to-base constraints imposed by battery life. This project explores how to architect and implement such a system end-to-end in Python, with emphasis on modularity, robustness, and user-friendly visualization.

3. System Architecture:



1. Sensor Nodes: Headless scripts that simulate DHT-style sensors.

2. Drone Gateway:

- **Listener Thread** accepts multiple sensor connections concurrently.
- **Processing Thread** maintains a 10-sample rolling window, computes averages and anomaly counts.
- **Battery Thread** drains at 1 %/sec, suspends forwarding below 15 %, recharges to 100 %, then flushes buffered data.

3. Central Server: Aggregates incoming summaries in a 100-record buffer and displays live tables, logs, and per-sensor time-series charts.

4. Components & Modules:

4.1) Sensor Nodes (sensor.py)

- **CLI flags**: **--drone_ip, --drone_port, --interval, --sensor_id, --count**
- **Behavior**: On startup, each thread connects to the Drone, generates JSON readings every interval seconds, logs send events, and retries on failure.

4.2) Drone Gateway (drone.py)

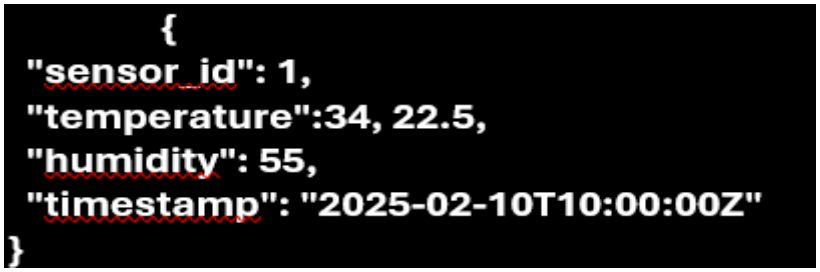
- **CLI flags**: **--serverip, --serverport, --listenip, --listenport**
- **Listener Thread**: Accepts and parses incoming JSON, updates a Stats object per sensor_id.
- **Stats Module**: Maintains deques of the last 10 temperature/humidity readings, computes rolling means, and increments an anomaly counter when values fall outside configured thresholds.
- **Battery Thread**: Toggles between "RUNNING" and "CHARGING" modes based on simulated battery level, pausing or resuming the listener and buffering or flushing summaries accordingly.
- **Sender Factory**: Provides a thread-safe method to forward summaries reliably, queueing messages on connection failures.
- **GUI (Tkinter)**:
    - Table view of per-sensor averages, last timestamp, and anomaly count.
    - Progress bar with color-coded battery status.
    - Live log panel

4.3 Central Server (central.py)

- **CLI flags**: **--ip, --port**
- **TCP Server**: Spawns a processDrone thread for each connecting Drone, reads newline-delimited JSON.
- **State Management**: Stores up to 100 summary packets in memory, maintains an event log capped at 500 entries.
- **GUI (Tkinter)**:
    - "Latest Aggregated Readings" text box.
    - Scrollable event log with anomaly highlighting.
    - Dynamically generated per-sensor graph panels supporting zoom and pan.

5. Data Flow & Message Formats:

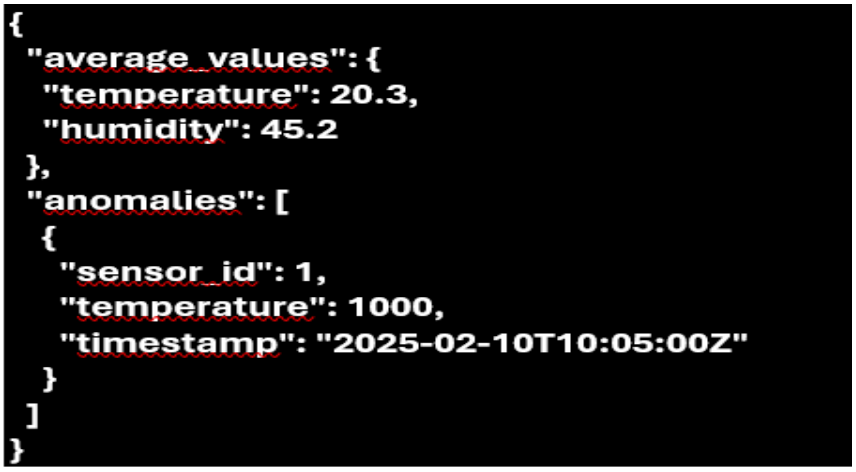
5.1) Sensor → Drone (in JSON):



5.2) Drone Processing:

- Accumulates **10 readings** in memory.
- Computes **`avg_temp`**, **`avg_hum`**, and counts **anomalies** (`temperature` or `humidity` outside thresholds).

5.3) Drone → Central (in JSON):



6. Design Rationale:

- **Edge Processing**: Reduces network traffic by summarizing data locally.
- **Thread-per-Connection Model**: Simplifies concurrency for sensor streams while preventing blocking.
- **Lock & Queue**: Ensures thread-safe state updates and preserves data during offline/charging phases.
- **Tkinter GUI**: Built-in, lightweight toolkit for rapid dashboard development without external dependencies.
- **JSON over TCP**: Human-readable, language-agnostic format that simplifies debugging and extensibility.

7. GUI Walkthrough:

7.1) Drone Dashboard:

- **Table**: Live rolling averages & anomaly counts per sensor.
- **Battery Bar**: Color transitions (green → yellow → red) with "RUNNING/CHARGING" label.
- **Log Panel**: Timestamped events (connections, disconnections, anomalies, battery state changes).

7.2) Central Server Dashboard:

- **Latest Readings**: Scrollable text area showing the **last 10 summaries.**
- **Event Log**: Colored tags for anomaly events.
- **Sensor Graphs**: Auto-generated canvas panels per sensor with zoom & pan controls to inspect time-series data.

8. Error Handling & Logging:

8.1) Sensor: Logs connection attempts, failures, and each data transmission.

8.2) Drone:

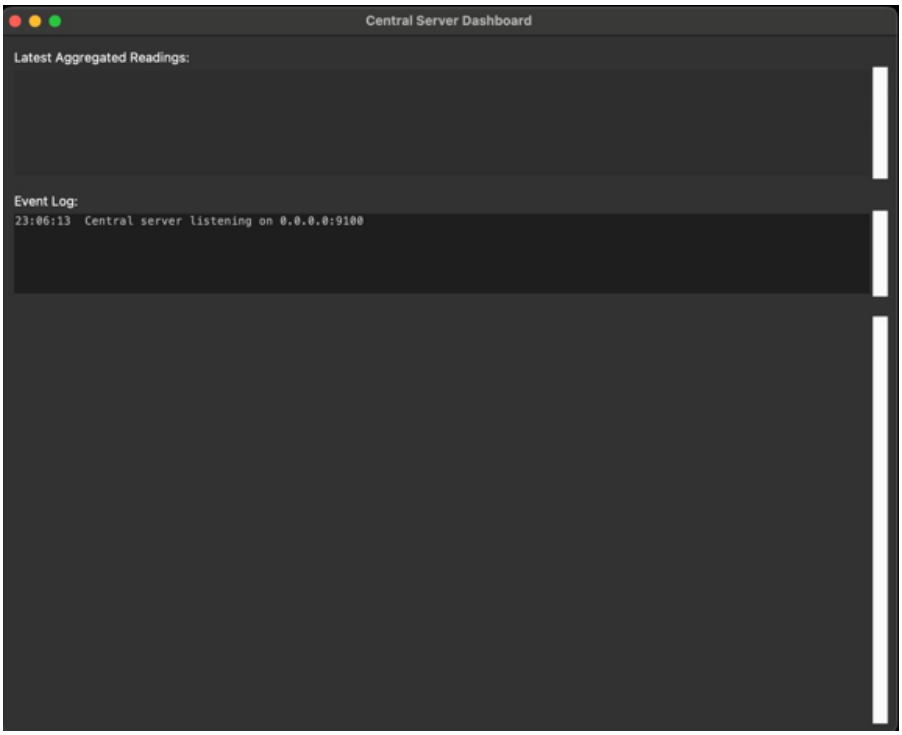
- Connection lifecycle (listener start/stop, client connects/disconnects).
- Anomaly detection (appends "⚠" to log entries).
- Central offline/timeout events with retry and queueing logs.
- Battery transitions ("Battery low – RETURNING", "Battery full – RUNNING").

8.3) Central:

- Connection errors and JSON parse failures.
- Maintains a timestamped log of up to 500 entries in memory.

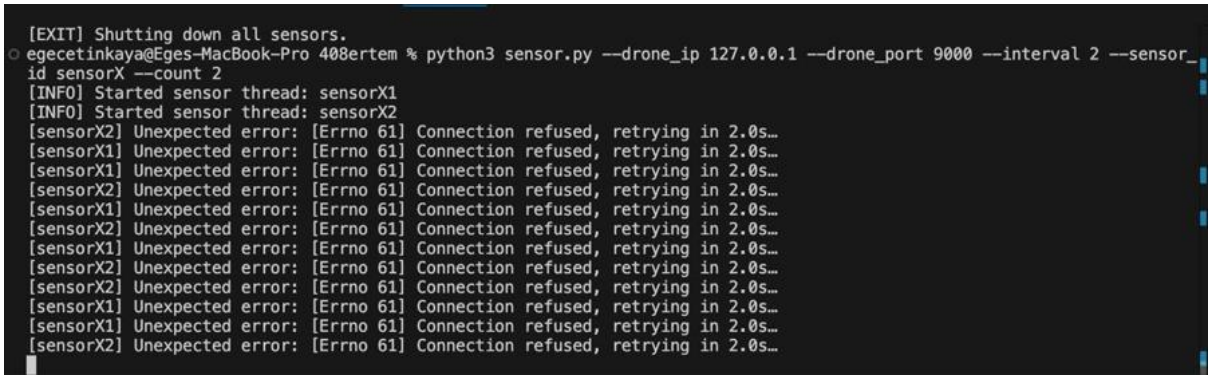
9. Live Demo Plan:

9.1) This is the initial Central Server Dashboard:

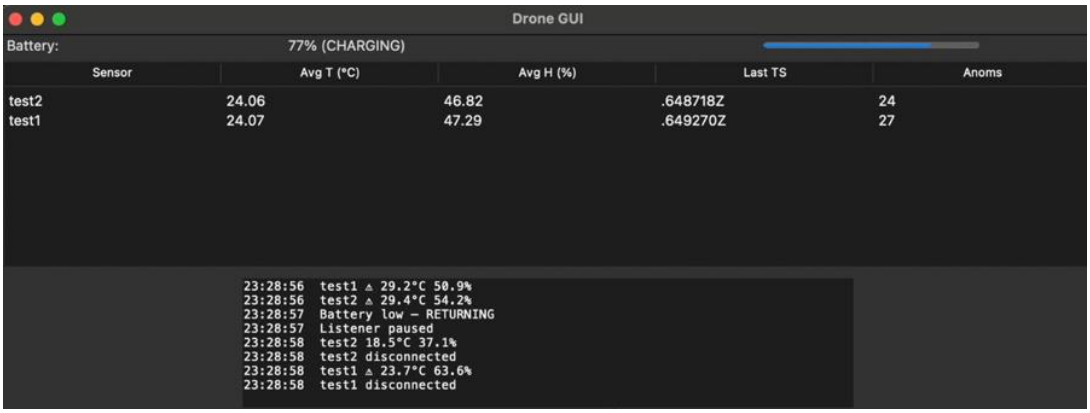


9.2) This is the initial Drone Graphical User Interface:

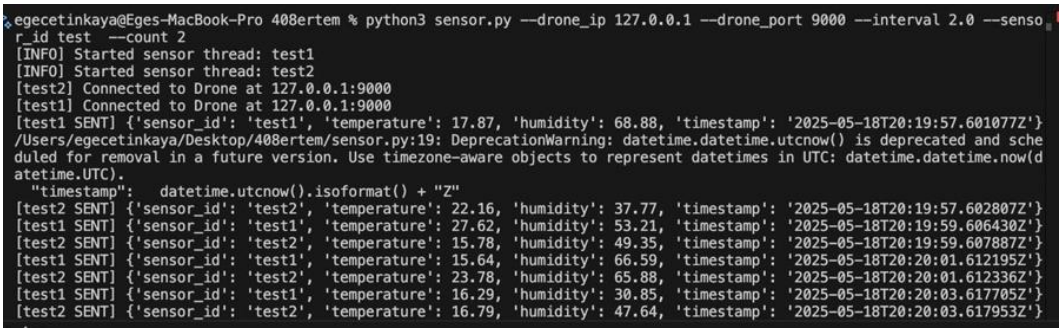
9.3) Sensors when there is no drone connected:



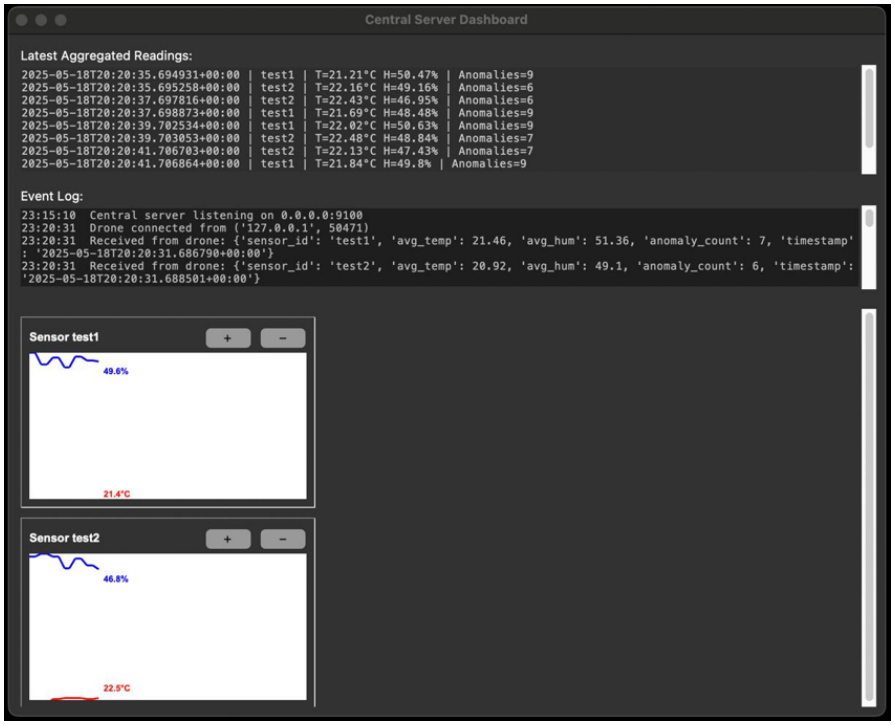
9.4) Drone GUI while the drone is charging:



9.5) Sensor's initial connection with the drone:



9.6) Central Server Dashboard after connection:



10. Conclusions & Future Work:

This system showcases a practical edge-computing architecture for environmental monitoring, combining real-time processing, reliable networking, and intuitive dashboards without external dependencies. **Future enhancements** could include:

- Switching to asynchronous I/O (e.g., asyncio) for scalability.
- Persisting data in a lightweight database (SQLite) for historical analysis.
- Adding secure channels (TLS) and authentication.
- Extending to more complex anomaly algorithms (machine learning-based).