Drone-Enabled Mobile Edge Computing for Environmental Monitoring Revised

1. Introduction and Motivation

Our company wants an innovative yet cost-effective approach to environmental monitoring in remote or expansive areas. Your task is to simulate a system where:
- Multiple Sensor Nodes collect environmental data and send it to a Drone.
- The Drone does edge processing averaging, anomaly detection before forwarding summarized data to a Central Server.
- The Central Server displays real-time visualizations for end-users.

Key Goals:
1. Demonstrate TCP communication client–server among all components.
2. Show real-time data processing and anomaly detection.
3. Produce interactive, user-friendly GUIs except for headless sensors.
4. Ensure thorough logging for easy grading.

2. System Components

2.1 Sensor Nodes Clients

Purpose: Simulate environmental sensors temperature, humidity, etc..

Operation and Requirements:
1. Configuration:
   - Accept connection parameters Drone’s IP and port via command-line arguments or a small config file.
   - Example: python sensor.py --drone_ip 127.0.0.1 --drone_port 5000 --interval 2
2. TCP Connection:
   - Automatically initiate a connection to the Drone server.
   - Retry on failure with a logged message indicating the attempt and result.
3. Data Transmission:
   - Periodically send sensor readings in a standardized JSON or CSV format.
      - Example: {"sensor_id": "sensor1", "temperature": 22.5, "humidity": 55, "timestamp": "2025-02-10T10:00:00Z"}
   - The transmission interval is configurable for example, every 2 seconds.
4. Error Handling and Logging:
   - Log every event: connection attempts, disconnections, data send events.
   - In case of disconnection, attempt reconnection periodically for example, every 5 seconds.

Note: Sensor nodes are headless; they do not need a GUI. They can be launched via a simple script.

2.2 Drone Mobile Edge

Purpose: Acts as both a TCP server for sensor nodes and a TCP client to the Central Server.

Key Functionalities:

1. TCP Server:
   - Listen on a configurable port for connections from multiple sensor nodes.
   - Manage connections concurrently threads or async I/O.
2. Edge Processing:
   - Aggregate incoming sensor data.
     - For instance, compute average temperature/humidity over the last N readings or T seconds.
   - Detect anomalies for example, out-of-range values and generate a log or alert.
3. Return to Base Simulation Battery or Flight-Time:
   - Have a configurable battery level or flight time parameter.
   - Once the battery level falls below a certain threshold:
     1. Log a low battery event.
     2. Update the Drone's status to Returning to base.
     3. Decide how to handle incoming data:
        - Option A: Temporarily disconnect from sensor nodes.
        - Option B: Queue the data internally but do not forward to the Central Server until battery is restored.
        - Clearly document the chosen approach in your code and system documentation.
4. TCP Client Forwarding to Central Server:
   - After processing or aggregation, forward data to the Central Server on a configurable IP or port.
   - This can be done periodically for example, every 5 seconds or after receiving a batch of data.

Drone GUI Requirements:

1. Real-Time Data View:
   - Display incoming sensor data in a table or chart for example, a rolling graph of temperature or humidity.
2. Aggregated Results and Anomalies:
   - Show computed averages, highlight or list anomalies timestamp, value, sensor ID.
3. Battery and Status Indicators:
   - Show current battery level.
   - Display an alert or banner stating Returning to base when battery is below threshold.
4. Logging Panel:
   - Real-time log of events connections, disconnections, anomaly detections, battery events.

2.3 Central Server

Purpose: Receive processed data from the Drone and provide a final visualization or log.

Key Functionalities:

1. TCP Server:
   - Listen on a configurable port for a single Drone connection though you may allow multiple.
2. Data Handling:
   - Receive the aggregated or processed data from the Drone.
   - Store or buffer the data for display in memory is fine for a simulation.
3. Central Server GUI:
   - Display the aggregated data for example, table, chart, or other appropriate visuals.
   - Maintain a log of all messages received with timestamps.
   - Indicate any anomalies flagged by the Drone.

3. Detailed Requirements

1. Configuration
   - Each component must support user-defined IP or port parameters and other relevant settings for example, Drone's battery threshold, sensor data intervals.
   - Sensible defaults can be provided, but the grader should be able to override them easily.

2. Data Format
   - Suggested: JSON objects with the following fields:
     - sensor_id string
     - temperature float
     - humidity float
     - timestamp string, ISO 8601 recommended
   - The Drone should parse these fields, compute rolling averages, detect anomalies, etc.

3. Error Handling and Logging
   - All components must produce logs sufficient for graders to trace events:
     - Connection attempts, successes, failures.
     - Data sent or received.
     - Anomalies detected by the Drone.
     - Battery threshold triggers and return to base events Drone.
   - The logs can be console-based or written to files. If GUI-based, ensure a scrolled text panel or similar is available.

4. Interactivity and Demonstration
   - Drone GUI:
     - Must allow graders to see real-time data updates, battery levels, anomaly flags.
     - Must allow some user interaction for example, a button to simulate battery consumption or a slider to set the threshold.
   - Central Server GUI:
     - Must display the summarized data it receives tables, charts, or textual readouts.
   - Sensor Nodes:

- Headless. They simply run and send data. However, they should output minimal logs in the console so we can see that they are transmitting.

5. Testing and Scenarios

1. Normal Operation
   - At least two sensor nodes start sending data.
   - The Drone receives data, aggregates it, forwards it to the Central Server.
   - The Drone and Central Server GUIs show the correct incoming or aggregated data.

2. Sensor Disconnection
   - One sensor node stops or crashes mid-run.
   - Drone logs the disconnection.
   - The sensor node tries to reconnect if implemented.
   - Drone's GUI should reflect the drop in data feed for that sensor.

3. Low Battery Return to Base
   - Simulate the drone battery dropping below a set threshold.
   - Drone logs the event and transitions to Returning to base.
   - Drone either disconnects from or queues incoming data depending on design choice.
   - Drone's GUI shows the new status.
   - Once battery is restored, the Drone resumes normal operation.

4. Anomaly Detection
   - Force a sensor to send out-of-range values (e.g., temperature = 1000°C).
   - Drone's anomaly detection logs it, and the GUI flags it.
   - The Central Server should also receive and display this anomaly record if forwarded.

5. Documentation & Deliverables
   1. Working Prototype (Code)
   - Submit well-documented Python code with clear separation of modules/classes for:
     - Sensor Nodes
     - Drone (Edge)
     - Central Server
   - Ensure each main component can be run independently with command-line or minimal input.
   2. System Documentation (2–3 pages)
   - Architecture Diagram: Show the full TCP connectivity: multiple sensor nodes → Drone → Central Server.
   - Module Descriptions: Outline the classes/functions in each component, focusing on:
     - Inputs/outputs
     - Connection handling
     - Edge processing logic (anomaly thresholds, data aggregation)
     - "Return to base" logic

- Design Rationale: Summarize why you chose certain data structures or communication patterns.
- Test Cases: Detail at least four test scenarios (as mentioned above) with expected results.

3. Demonstration
    - A short live demo illustrating normal data flow, a battery return event, and anomaly detection.
    - Graders must be able to replicate these demos quickly.

6. Project Phases & Timeline
    1. Phase 1 (Weeks 8–10): System Architecture & Design
        - Tasks:
            - Finalize architecture, data formats, and connection parameters.
            - Document potential issues (e.g., concurrency, queue management).
        - Deliverable:
            - A design document (2–3 pages) with diagrams and detailed module specs.
    2. Phase 2 (Weeks 10–12): Partial Implementation
        - Tasks:
            - Implement basic TCP connections, preliminary GUI elements, logging framework.
            - Show sensors sending data to the Drone, Drone displaying it, Drone forwarding some data to the Central Server.
        - Deliverable:
            - A partial prototype, plus initial test case results (e.g., Wireshark screenshots are optional but can be used to illustrate traffic).
    3. Phase 3 (Weeks 12–14): Full Implementation & Testing
        - Tasks:
            - Complete all features: anomaly detection, battery simulation, robust GUIs, error handling.
            - Conduct final tests (sensor disconnection, battery threshold, anomaly injection).
        - Deliverable:
        - A fully working system with updated documentation and a demonstration.

7. Grading Criteria
    1. Functionality & Correctness (40%)
        - Proper TCP communication, data aggregation, anomaly detection, battery simulation.
    2. GUI & Interactivity (30%)
        - Clarity of the Drone and Central Server GUIs, ease of demonstration.
    3. Logging & Documentation (20%)
        - Quality of logs, clarity of the 2–3 page documentation, and the presence of well-defined test scenarios.

4. Code Quality (10%)
    - Code organization, comments, readability, and maintainability.