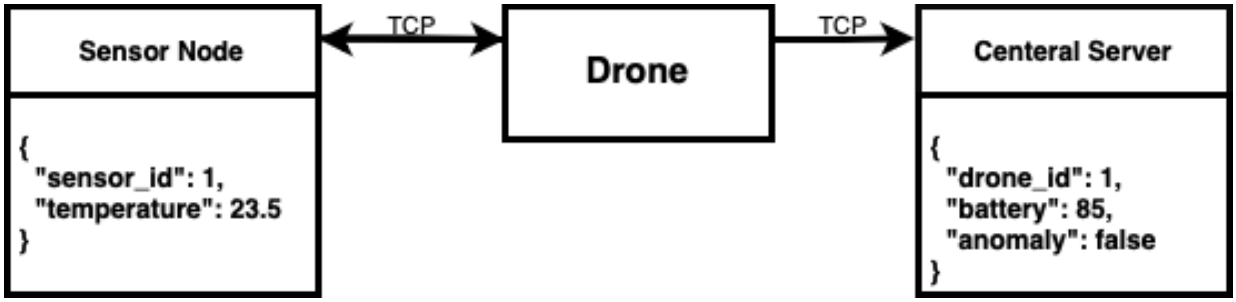
CS408 Project Phase 1

Project Title: Drone-Enabled Environmental Monitoring System
Group 14: Enes Coskun, Oğuz Temelli, Melihtan Özkut
Date: 12 April 2025

1. System Overview

In this project, we are trying to simulate a simple environmental monitoring system comprising sensor nodes, a drone acting as an edge node, and a central server. Sensor nodes generate and send periodic environmental data (temperature and humidity) via TCP to the drone. The drone processes this data, detects anomalies, keeps track of battery level, and forwards simplified results to the central server. We tried to keep the architecture simple and modular so that it can work better.

2. Architecture Diagram



Explanation:

- Sensor nodes work as TCP clients and send environmental data (in JSON format) to the drone at regular intervals.
- The drone listens as a TCP server, receives this data from sensors, does some processing like averaging and checking for anomalies, and also keeps track of its battery. After that, it acts as a TCP client and sends a summary of the data to the central server.
- The central server, which is a TCP server, gets this processed info from the drone and shows it on a GUI.

3. Module Descriptions

Module	Purpose	Inputs	Outputs	Core Logic
Sensor Node	Generate/send environmental data	Config (IP, port, interval)	JSON-formatted data via TCP	Data generation, reconnect logic
Drone	Process sensor data, forward summaries	Sensor data, battery parameters	JSON summaries, GUI status	Avg calculation, anomaly detection, battery management
Central Server	Visualize processed data/anomalies	Drone summaries	GUI visuals, anomaly/event logs	Data visualization, logging

4. Data Format & Protocol

For communication between components, we use TCP to make sure the data is reliably delivered without any loss.

There are two main types of data being sent in the system:

- **Sensor message**: includes values like sensor_id, temperature, humidity, and a timestamp to know when the reading was taken.
- **Drone summary message**: contains things like average_temperature, average_humidity, a list of any detected anomalies[], and again a timestamp.

5. Design Rationale

- **TCP vs UDP:** we decided to use TCP instead of UDP because it's more reliable ,we didn't want to risk losing any data during transmission.
- **Format:** For the data format, we went with JSON since it's simple, easy to read and works well with many programming languages.
- **Edge Processing:** We also added some edge processing on the drone side to reduce the server's workload. This way, the drone can detect anomalies before sending the data forward.
- **Modular Architecture:** we kept the system modular, so it's easier to develop and test each part separately, and it will be more flexible in next phases.

6. Potential Design Issues

Issue	Description	Planned Handling
Data Loss	Incomplete or corrupted transmissions	Logging and retransmission mechanisms
Sensor disconnection	Lost connections from sensor nodes	Auto-reconnect and detailed logging
Concurrent sensor data	Multiple sensors simultaneously connecting	Multithreading or asynchronous processing
Low battery mode	Drone energy-saving scenarios	Data queuing or selective disconnections
Anomaly definition	Detection of abnormal sensor values	Configurable threshold methods
GUI responsiveness	Potential GUI freezes during processing	Separate GUI threads or asynchronous I/O

7. Test Scenarios

Normal Operation: Sensors, the drone, and the server exchange data as expected, and the GUI shows the values correctly.

Sensor Disconnection: We simulate a sensor going offline. The drone should notice it, log the event, and the GUI should reflect that change.

Battery Drain: To test power handling, we lower the drone's battery level and expect it to either limit some functions or give a warning through the GUI.

Anomaly Injection: We send unusual or extreme sensor data to see if the drone and server can spot the problem and flag it properly.

8. Conclusion

In conclusion, this Phase 1 report explains how our drone-based monitoring system is designed to work. We focused on keeping the system modular and reliable so that it's easier to manage and expand later. The design also supports real time feedback, and Project is ready to move on to the implementation part in the next phase.