Drone-Enabled Mobile Edge Computing for Environmental Monitoring

Phase-3 System Documentation

Tuğberk Abdulkadiroğlu 26492

Emre Gökay Kılınç 28086

Melike Ceyda Gür 31050

1 Introduction & Motivation

Conventional fixed-station monitoring is costly and blind to vast rural areas.
We simulate a **drone-centric edge–computing pipeline** that lets a single UAV collect, pre-process and uplink environmental readings in real time (Fig. 1).
The prototype emphasises four goals:

G-ID	Key goal	How we satisfy it
G1	Reliable end-to-end TCP between every pair	*asyncio.start_server / open_connection* + automatic reconnect loops
G2	On-device edge processing	rolling averages, anomaly filters, battery logic in the Drone
G3	Interactive GUIs	Tkinter dashboards on Drone & Central; live Matplotlib charts; headless sensors
G4	Thorough logging	colour-coded console + scrollable GUI panes; every event (connect, drop, anomaly, battery)

---

2 Architecture Overview

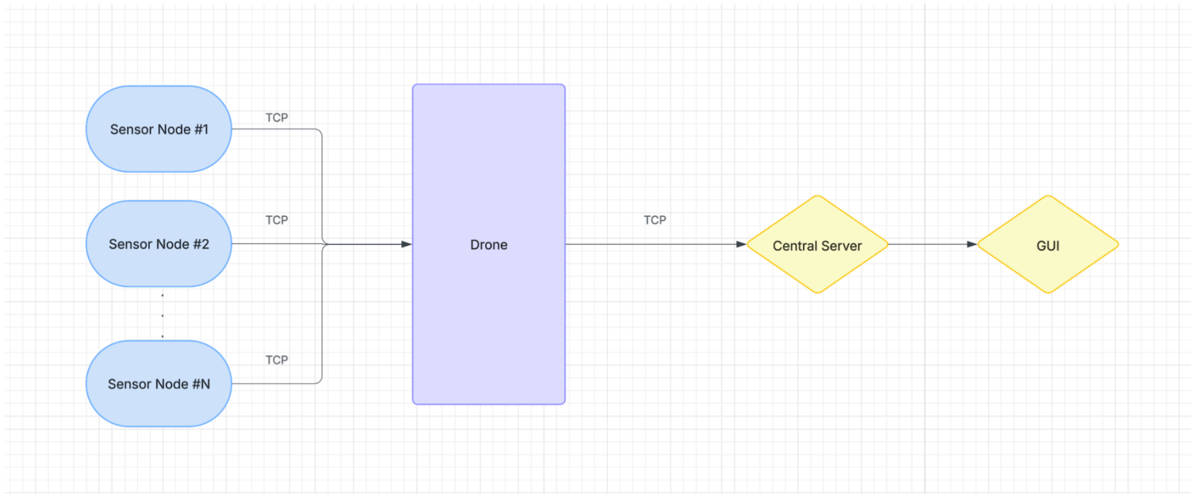

Fig. 1 — multi-sensor → Drone → Central data flow

2.1 Sensor Node (*sensor/sensor.py*)

- **Input (switched at runtime)** – *--drone-ip, --drone-port, --interval*

Output – one JSON per *interval*:
{"sensor_id":"s1","temperature":23.4,"humidity":51.0,"timestamp":"2025-05-18T17:15:06Z"}

- **Features**
    - Randomised but realistic data; optional one-shot *spike* flags (*--spike-temp, --spike-hum*) for anomaly tests.
    - Infinite reconnect loop with 5 s back-off; non-blocking (*asyncio*).

2.2 Drone Edge Node (drone/drone_server.py + drone/gui_app.py)

- **Server side (to sensors)** – async TCP on configurable port; each sensor handled in its own coroutine.
- **Client side (to Central)** – persistent writer guarded by asyncio.Lock; automatic re-dial.
- **Edge logic**
    - Rolling window = 10 latest readings per sensor.
    - mean() computes **average T / H** every 5 s.
    - **Anomalies** = value ∉ [0 – 60 °C] or [0 – 100 %] **or** drone disconnect **or** battery < 20 %.
- **Battery simulation**
    - Linear drain 2%/min; GUI buttons or Central commands can drain/recharge.

  - Below 20 % ⇒ returning mode → keep listening but stop forwarding averages (only heartbeat + battery).
- **GUI** (Tkinter + Matplotlib)
  - Top table: raw packets (last 100).
  - Progress-bar + label battery%.
  - Anomaly list and scrolling log.

2.3 Central Server (central/central_server.py)

- **TCP server** on port 6000; supports many drones.
- **GUI panels**
  - Rolling table of every summary (battery, averages).
  - Per-drone battery bars.
  - Global anomaly list (red).
  - Log pane (connect, drop, control msgs).
- **Control channel** – buttons can emit JSON {type:"battery",action:...} back down the existing socket.

---

3 Design Rationale

Requirement	Design choice	Rationale
Reliability	Plain **TCP**	ordered, lossless, easier than UDP + ACK.
Human-readable packets	**JSON lines**	trivial parsing, Wireshark-friendly, extensible.
Concurrency	asyncio coroutines	fewer threads, simple await + locks.
Edge compute	store → rolling mean	avoids back-haul flood; only 8 KiB memory for 10 × N sensors.
Battery logic	client-side flag returning	lets Central & GUI react immediately without new channels.

---

4 Test Scenarios & Expected Results

#	Scenario	Steps	Expected
T1	Normal	start ≥ 2 sensors → Drone → Central	live tables & charts fill; no anomalies.
T2	Sensor crash	Ctrl-C one sensor	Drone logs *Sensor X disconnected*; anomaly appears red on both GUIs.
T3	Low battery	click **Drain 5 %** until <20 %	Drone shows 🛬, averages blank; Central shows same; after **Recharge**flow resumes.
T4	Anomaly value	--spike-temp 1000	Drone flags *temperature=1000*; red line in lists; Central receives same JSON.
T5	Reconnect	restart crashed sensor	GUI tables continue with new timestamps; disconnect anomaly stops repeating.