Plant Up! – A Hybrid IoT Architecture for Smart Urban Gardening

Richard Onuoha

December 2025

Contents

1 Introduction **1**

1.1 Motivation for Smart Gardening and the Social Internet of Things 1

1.2 Why Plant Up! Is a Strong Case Study 1

1.3 Contributions 1

1.4 Chapter Overview 2

2 Theoretical Foundations **3**

2.1 Event-Driven Architecture in IoT 3

2.2 Supabase and PostgREST as Micro-service Enablers 3

2.3 Energy Modeling of ESP32-S3 Deep-Sleep Modes 3

2.4 Related Work on Consumer-grade IoT Constraints 4

2.5 MQTT versus REST – A Deeper Theoretical Comparison 4

3 System Architecture **5**

3.1 Overview and Vision 5

3.2 Firmware Architecture 5

3.3 Sensor-Specific Corrections 6

3.4 Hybrid Communication Architecture 6

3.5 Data Consistency Handling 6

4 Methodology **7**

4.1 Research Design 7

4.2 Hardware and Software Configuration 7

4.3 Measurement Instruments 8

4.4 Repetition Strategy and Test Conditions 8

4.5 Validity and Reproducibility Measures 8

5 Experimental Evaluation **9**

5.1 Setup Recap 9

5.2 Results 10

5.3 Interpretation 10

5.4 Anomalies and Outliers 11

5.5 Network Conditions and Variance 11

6 Discussion and Limitations **12**

6.1 Implications for Real-World Deployment 12

6.2 Limitations 12

6.3 Comparison with Related Work 13

7 Conclusion and Future Work **14**

7.1 Answer to the Research Question 14

7.2 Architectural Reasoning Summary 14

7.3 Future Improvements 14

Chapter 1

Introduction

1.1 Motivation for Smart Gardening and the Social Internet of Things

In recent years, the convergence of the Internet of Things (IoT) with everyday activities has given rise to the concept of the Social Internet of Things (SIoT) [1]. Within the agricultural domain this convergence is often termed "Agriculture4.0" – a data-driven transformation that replaces intuition-based practices with precise, sensor-enabled decision making [2]. While large-scale farms have already adopted industrial IoT solutions, a comparable shift is occurring at the consumer level: smart urban gardening. Home growers increasingly rely on low-cost sensor nodes to monitor soil moisture, light, temperature, and nutrient status, thereby reducing water waste and improving plant health [3].

1.2 Why Plant Up! Is a Strong Case Study

The *Plant Up!* project embodies the SIoT paradigm by treating each plant as a digital entity that can "speak" to its caretaker. The system integrates distributed edge hardware, a scalable cloud micro-service backend, and a mobile application that presents the data in a gamified, socially connected interface. This hybrid architecture – MQTT for low-power sensor uplink and REST (via Supabase/PostgREST) for the user-facing app – provides a rich test-bed for evaluating the trade-offs between latency, energy consumption and data consistency in a realistic residential environment.

1.3 Contributions

The present thesis makes the following contributions:

1. A detailed analysis of the energy-impact of MQTT versus REST when operating on

battery-powered ESP32-S3 nodes.

2. A complete firmware description that clarifies the wake-measure-serialize- transmit-sleep cycle and the rationale for using JSON payloads.

3. An empirical evaluation that quantifies latency, payload size and reliability under controlled home-Wi-Fi conditions.

4. Architectural guidelines for hybrid IoT deployments that balance real-time responsiveness with long-term battery sustainability.

1.4 Chapter Overview

Chapter 2 introduces the theoretical foundations required to understand event-driven IoT, micro-service enablement with Supabase, and the energy-modeling of deep-sleep modes on the ESP32-S3. Chapter 3 describes the complete system architecture, including a low-level firmware analysis. Chapter 4 details the experimental methodology. Chapter 5 presents the results, while Chapter 6 interprets them and discusses limitations. Finally, Chapter 7 concludes the work and outlines future research directions.

Chapter 2

Theoretical Foundations

2.1 Event-Driven Architecture in IoT

Event-driven architectures decouple data producers from consumers by emitting messages that trigger processing pipelines only when a relevant change occurs. In the context of battery-constrained devices, this model reduces the need for periodic polling and therefore minimizes radio-on time [4]. The ESP32-S3 can be programmed to generate an event after each deep-sleep wake-up, which then initiates sensor acquisition and payload transmission.

2.2 Supabase and PostgREST as Micro-service Enablers

Supabase provides a managed PostgreSQL instance together with an auto-generated RESTful API (PostgREST) and a real-time subscription layer. By exposing each schema (e.g., `user_schema`, `microcontroller_schema`) as a bounded context, developers can enforce data-ownership and apply Row-Level Security (RLS) policies that guarantee only authorised services may read or write specific tables **supabaseDocs**. This approach eliminates the need for a bespoke API gateway while preserving the micro-service principle of loose coupling.

2.3 Energy Modeling of ESP32-S3 Deep-Sleep Modes

The ESP32-S3 offers several power states. In deep-sleep the main CPU, Wi-Fi radio and RAM are powered down; only the Ultra-Low-Power (ULP) coprocessor and real-time clock (RTC) memory remain active. The ULP can keep the RTC alive and perform minimal sensor checks, but the bulk of processing occurs after the device wakes. Typical current draws are **10–150µA** in deep-sleep versus **160–260 mA** in active mode [5]. The "wake-up tax" – the energy required to re-initialise Wi-Fi and obtain an IP address – dominates the per-cycle budget and is therefore a key factor in the choice of communication protocol.

2.4 Related Work on Consumer-grade IoT Constraints

Several studies have highlighted the challenges of deploying IoT at the consumer level, where devices must operate unattended for months on a single battery and cope with noisy home Wi-Fi environments [6]. Approaches such as duty-cycling, adaptive transmission power and lightweight protocols (e.g., CoAP, MQTT) have been proposed, yet few works provide a head-to-head quantitative comparison of MQTT versus REST under identical hardware constraints.

2.5 MQTT versus REST – A Deeper Theoretical Comparison

MQTT follows a publish-subscribe paradigm, decoupling senders (publishers) from receivers (subscribers) via a broker. This reduces per-message overhead because only a small fixed header (typically 2bytes) is transmitted, and the payload remains unchanged across multiple subscribers. REST, by contrast, is a request-response model that requires a full TCP three-way handshake and, when secured with TLS, an additional handshake phase. The resulting latency difference can be substantial, especially when the payload is small (tens of bytes) and the network is congested. However, MQTT's performance gains are sensitive to QoS level, TLS configuration and the underlying Wi-Fi quality – higher QoS or encrypted channels increase packet size and processing time.

Chapter 3

System Architecture

3.1 Overview and Vision

The *Plant Up!* platform unifies three layers: the edge node (sensor hardware), the cloud layer (Supabase micro-services) and the application layer (mobile app). The primary goal is to collect high-precision environmental data from battery-powered ESP32-S3 nodes, synchronise it through a cloud ingestion pipeline, and present it to users in a gamified interface that encourages regular plant care.

3.2 Firmware Architecture

The firmware follows a deterministic wake-measure-serialize-transmit-sleep cycle:

1. **Wake-up**: The ESP32-S3 exits deep-sleep, the ULP coprocessor hands control to the main CPU, and the RTC provides the current timestamp.
2. **Sensor Acquisition**: The node powers the attached sensors – LM393 soil-moisture sensor (measures conductivity), BH1750 light sensor (lux range up to 65kLux), and DFRobot Gravity V2 EC sensor (measures ionic conductivity) – and reads their raw values.
3. **Serialization**: Raw measurements are mapped to a JSON object that includes a UTC timestamp, device identifier and the calibrated sensor values. JSON is chosen for its human-readability, language-agnostic support and seamless integration with Supabase's REST endpoints.
4. **Transmission**: Depending on the configured mode the payload is sent either via MQTT (publish to topic `plantup/sensor/<id>`) or via a HTTP POST request to the Supabase PostgREST endpoint. In both cases the payload remains JSON; MQTT does not use a binary format.

5. **Sleep**: After confirming receipt (MQTT PUBACK or HTTP 200), the node immediately returns to deep-sleep, preserving only RTC memory and the ULP coprocessor.

3.3 Sensor-Specific Corrections

- **LM393 Soil-Moisture Sensor**: Contrary to earlier statements, the LM393 is a resistive sensor that measures soil conductivity; it does not directly sense dielectric permittivity.
- **EC Sensor (DFRobot Gravity V2)**: The sensor reports ionic conductivity, which correlates with nutrient concentration, rather than generic "salt content".
- **BH1750 Light Sensor**: Provides lux measurements with a practical range up to approximately 65kLux and a limited accuracy due to its digital I^2C interface.
- **MLX90614 Infrared Sensor**: Measures surface temperature via infrared radiation; it does not report ambient air temperature.

3.4 Hybrid Communication Architecture

Sensor nodes publish telemetry to the MQTT broker, which forwards the data to a Supabase ingestion service. The mobile application, built with Flutter, consumes the same data via Supabase's PostgREST API (REST). This hybrid approach leverages MQTT's low-overhead uplink while retaining the rich query capabilities of REST for user-facing features. Supabase Row-Level Security (RLS) policies ensure that only authorised services may insert into `microcontroller_schema` and that users can only read their own records.

3.5 Data Consistency Handling

Because the system spans multiple micro-services, eventual consistency is adopted for non-critical telemetry. The MQTT broker buffers messages during network outages; once connectivity is restored the ingestion service writes the records to the PostgreSQL backend, where Supabase's real-time listeners update subscribed clients.

Chapter 4

Methodology

The experimental evaluation follows a controlled laboratory design that isolates the impact of the communication protocol on latency, payload size and energy consumption.

4.1 Research Design

- **Independent Variable**: Communication protocol (MQTT vs. REST).
- **Dependent Variables**: End-to-end latency (boot $\rightarrow$ server acknowledgment), effective payload size (bytes on the wire) and packet loss rate.
- **Hypotheses**:

  H1 *H1*: MQTT will exhibit lower latency than REST under identical Wi-Fi conditions.

  H2 *H2*: MQTT will transmit fewer bytes per measurement due to its minimal header overhead.

  H3 *H3*: The energy cost per transmission will be lower for MQTT because the device spends less time in the active Wi-Fi state.

4.2 Hardware and Software Configuration

- **Device**: ESP32-S3 development board equipped with LM393, BH1750, DFRobot Gravity V2 and MLX90614 sensors.
- **Firmware**: Custom FreeRTOS application compiled with ESP-IDF 5.2, implementing the wake-measure-serialize-transmit-sleep cycle.
- **Network**: Standard 2.4GHz home Wi-Fi router, RSSI between –65dBm and –75dBm.

- **Backend**: Supabase Production instance (PostgreSQL 13) with MQTT broker (Mosquitto) and PostgREST endpoint.

4.3 Measurement Instruments

1. **High-Precision Timers**: The ESP32-S3's hardware timer records timestamps with microsecond resolution for each stage of the cycle.
2. **SQL Verification**: After each transmission a SQL query confirms the presence of the record in the `Controllers` table.
3. **Packet Capture**: Wireshark captures the raw frames on the router interface, allowing calculation of the actual payload size.

4.4 Repetition Strategy and Test Conditions

Each protocol was exercised for 200 independent transmission cycles. The payload was a static JSON object (identical for both protocols) to eliminate variability due to sensor reading time. The device was reset between runs to ensure a fresh deep-sleep entry.

4.5 Validity and Reproducibility Measures

The experiment was performed in a single residential environment; to improve external validity, the Wi-Fi signal strength was varied within the stated range. All source code, configuration files and raw logs are version-controlled in the project repository, enabling exact replication.

Chapter 5

Experimental Evaluation

5.1 Setup Recap

The device firmware was compiled with the “Protocol Time” instrumentation enabled, allowing us to isolate the time spent in wake-up, measurement, serialization, transmission and sleep phases. The same static JSON payload (150bytes) was used for both MQTT and REST trials.

5.2 Results

`images/markdown-table.png`

Figure 5.1: Comparative performance metrics of MQTT vs. REST under identical test conditions

The average end-to-end latency was **185ms** for MQTT and **612ms** for REST – a reduction of roughly 70% under the tested QoS0 configuration. Effective payload size measured by Wireshark was **312bytes** per MQTT message versus **982bytes** for the REST POST request, confirming the header overhead difference.

5.3 Interpretation

The latency advantage stems primarily from MQTT's avoidance of a full TCP/TLS handshake on each transmission; the broker maintains a persistent connection once the device wakes. The smaller payload further reduces radio-on time, contributing to lower energy consumption per cycle.

5.4 Anomalies and Outliers

A small number of REST transmissions exhibited latency spikes ($>$1s) caused by temporary Wi-Fi retransmissions. MQTT's asynchronous nature allowed it to queue messages during brief connectivity lapses, resulting in fewer outliers.

5.5 Network Conditions and Variance

All measurements were taken with RSSI between –65dBm and –75dBm. Under poorer signal strength the absolute latency increased for both protocols, but the relative advantage of MQTT remained consistent.

Chapter 6

Discussion and Limitations

The experimental results confirm the hypothesis that MQTT provides a substantial latency and bandwidth advantage over REST for low-payload, battery- constrained IoT devices. In the context of *Plant Up!*, this translates into a lower "wake-up tax" and consequently longer battery life – a critical factor for devices intended to operate unattended for months.

6.1 Implications for Real-World Deployment

Adopting MQTT for sensor uplink enables the system to react to events (e.g., soil moisture dropping below a threshold) within a few hundred milliseconds, which is perceptible to the end-user in the mobile app. The hybrid approach also preserves the rich query capabilities of REST for user-facing features such as profile management and social interactions.

6.2 Limitations

- **TLS Not Tested**: All MQTT transmissions were performed without TLS encryption; enabling TLS would increase payload size and processing time, potentially narrowing the performance gap.
- **QoS Level**: Experiments used QoS0 (fire-and-forget). Higher QoS levels provide delivery guarantees but incur additional handshake steps.
- **Home Wi-Fi Variability**: The test environment reflects a typical residential network; enterprise or outdoor deployments may exhibit different latency characteristics.
- **No Direct Power Measurement**: Energy consumption was inferred from radio-on time and duty-cycle; a dedicated power-monitoring IC (e.g., INA219) would provide more accurate measurements.

6.3 Comparison with Related Work

Prior studies have demonstrated MQTT's suitability for low-power IoT [7], yet few have quantified the latency improvement in a real-world home setting. Our findings align with those works but extend them by providing a side-by-side REST baseline under identical hardware and network conditions.

Chapter 7

Conclusion and Future Work

7.1 Answer to the Research Question

The comparative study shows that, for battery-powered ESP32-S3 nodes, MQTT outperforms REST in terms of latency, payload efficiency and, by inference, energy consumption. The hybrid architecture – MQTT for sensor uplink and REST for the mobile application – therefore represents the optimal design for the *Plant Up!* platform.

7.2 Architectural Reasoning Summary

By decoupling the low-power telemetry path (MQTT) from the user-centric API (REST), the system satisfies both real-time responsiveness and rich data access. Supabase's micro-service model and RLS policies provide a secure, scalable backend without the overhead of a custom API layer.

7.3 Future Improvements

1. **MQTT-SN**: Investigate the MQTT-for-Sensor-Networks (MQTT-SN) protocol, which further reduces header size for constrained devices.
2. **Batching and Compression**: Aggregate multiple measurements before transmission to amortise the wake-up tax.
3. **TLS and QoS 1/2**: Evaluate the impact of encrypted channels and higher QoS levels on latency and energy.
4. **Alternative Radio Technologies**: Explore LoRaWAN for ultra-low-power long-range communication.

5. **Machine-Learning Anomaly Detection**: Deploy on-device inference to filter out spurious sensor readings before transmission.

6. **Direct Power Measurement**: Integrate an INA219 or similar IC to obtain precise current consumption profiles.

Bibliography

[1] J. Jung and I. Weon, "The social side of internet of things: Introducing trust-augmented social strengths for iot service composition," *Sensors*, vol. 25, no. 15, p. 4794, 2025, Accessed: 2025-12-09. [Online]. Available: `https://mdpi.com/1424-8220/25/15/4794`

[2] Leher, *Exploring agriculture 4.0: Technology's role in future farming*, `https://leher.ag/blog/technology-role-agriculture-4-0-future-farming`, Accessed: 2025-12-09, 2024.

[3] StreetSolver, *The smart urban garden: Top tech & ai helping you grow more food at home in 2026*, `https://streetsolver.com/blogs/the-smart-urban-garden-top-tech-ai-helping-you-grow-more-food-at-home-in-2026`, Accessed: 2025-12-09, 2024.

[4] AWS, *Microservices architecture - key concepts*, `https://docs.aws.amazon.com/whitepapers/latest/monolith-to-microservices/microservices-architecture.html`, Accessed: 2025-12-09, 2019.

[5] Espressif Systems, *Esp32-s3 series datasheet*, Accessed: 2025-12-09, 2024. [Online]. Available: `https://espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf`

[6] KaaIoT, *Iot device challenges – power & connectivity constraints*, `https://kaaiot.com/iot-knowledge-base/how-does-wi-fi-technology-link-to-the-iot-industry`, Accessed: 2025-12-09, 2024.

[7] PsiBorg, *Advantages of using mqtt for iot devices*, `https://psiborg.in/advantages-of-using-mqtt-for-iot-devices/`, Accessed: 2025-12-09, 2024.