

MQTT-Based Multi-Agent Systems

Fradj DORBEZ and Hamza OUNI – ESISAR 5A IR&C

December 16, 2025

Abstract

This report presents a modular implementation of a multi-agent system based on MQTT. Three exercises are addressed: a first communication client, a smart-home sensor network, and a Contract Net protocol for task allocation. The focus is placed on architectural choices, communication design, execution traces, and practical difficulties encountered during implementation.

1 Technical Choices

1.1 Programming Language

Python 3 was chosen for this project due to its rapid prototyping capabilities, readability, and extensive ecosystem. These characteristics are well-suited for multi-agent simulations where clarity and modularity are more important than raw performance.

1.2 MQTT Library

The `paho-mqtt` library was used as the MQTT client implementation. It is lightweight, stable, widely adopted, and provides a clear callback-based API. Its background network loop integrates naturally with threaded agent architectures.

1.3 Concurrency Model

A multi-threaded approach was selected instead of an asynchronous (`asyncio`) design.

The main reasons are:

- Sensor agents perform periodic I/O-bound operations (sleep + publish).
- `paho-mqtt` already runs its own network loop in a background thread.
- Threading simplifies the design and improves readability for the lab scope.

Each sensor or worker agent runs in its own lightweight thread, while MQTT communication is handled by a shared background loop.

1.4 Topic Structure and Identifiers

A hierarchical topic structure was adopted:

`home/<room>/<measurement>/<sensor_id>`

This choice allows:

- Wildcard subscriptions (e.g., `home/+/temperature/#`)
- Clear separation between rooms and measurements
- Stateless messages (no implicit context required)

Sensor identifiers are included in the topic path rather than only in the payload, which simplifies routing and filtering at the broker level.

1.5 Serialization Format

All messages are serialized using JSON. JSON is human-readable, well-supported in Python, and sufficient for the small payload sizes used in this lab.

2 Sensor Network Architecture

2.1 Global Organization

The smart-home simulation follows a hierarchical multi-agent architecture.

Each physical room is modeled by a **RoomAgent**, which acts as a local coordinator. Sensors, aggregators, and detectors remain independent agents communicating exclusively through MQTT.

2.2 RoomAgent

The RoomAgent is responsible for:

- Creating and managing sensors via a Factory pattern
- Maintaining actuator states (heating, window)
- Applying environmental effects to sensor baselines
- Processing control commands received via MQTT

Control commands are received on:

```
home/<room>/control/#
```

This design avoids direct coupling between external controllers and internal sensor logic.

2.3 Sensor Agents

Sensor agents periodically publish simulated values for: temperature, humidity, luminosity, and presence.

Each sensor publishes on:

```
home/<room>/<measurement>/<sensor_id>
```

Sensor behavior is configurable (period, baseline, amplitude, noise) and instantiated using a **SensorFactory**, improving extensibility and code reuse.

2.4 Aggregation and Detection

Averaging agents subscribe to sensor topics using MQTT wildcards and compute rolling averages over a sliding window. Results are published to:

```
home/<room>/<measurement>/average
```

Detection agents monitor raw sensor streams and trigger alerts when a value deviates by more than two standard deviations from the recent mean. Alerts are published to:

```
home/alerts/<room>
```

2.5 Interface Agent

An interface agent subscribes to averages and alerts and logs human-readable information to the console, acting as a lightweight monitoring interface.

3 Contract Net Protocol

The Contract Net protocol is implemented using MQTT topics.

The supervisor publishes a Call for Proposals (CfP):

```
contractnet/cfp
```

Machine agents reply with proposals on:

```
contractnet/proposal
```

After a fixed deadline, the supervisor selects the best proposal (minimum execution time) and sends:

```
contractnet/accept/<machine_id>
```

```
contractnet/reject/<machine_id>
```

Accepted machines simulate job execution and publish completion events on:

```
contractnet/done
```

This implementation follows the classical Contract Net interaction pattern while remaining fully decentralized and loosely coupled.

4 Execution Traces

4.1 Sensor Publication

Topic: home/bedroom1/temperature/bedroom1_temp_01

Payload: {"timestamp": 1700000123, "sensor_id": "bedroom1_temp_01", "value": 21.4}

4.2 Control Command

Topic: home/bedroom1/control/command

Payload: {"command": "heating", "value": true}

4.3 Dynamic Sensor Addition

Topic: home/bedroom1/control/command

Payload: {"command": "add_sensor",
 "measurement": "temperature",
 "sensor_id": "bedroom1_temp_03",
 "baseline": 22.0}

4.4 Anomaly Detection

Topic: home/alerts/bedroom1

Payload: {"sensor_id": "bedroom1_temp_bad",
 "value": 80.0,
 "mean": 21.2,
 "stddev": 0.8}

4.5 Contract Net Proposal

Topic: contractnet/proposal

Payload: {"machine_id": "machine_1", "job": {"name": "A"}, "time": 2}

5 Difficulties and Solutions

5.1 Dynamic Agent Management

Sensors may appear or disappear during execution. This was addressed by wildcard MQTT subscriptions and per-sensor buffers in aggregation agents, allowing seamless adaptation without re-configuration.

5.2 Actuator Consistency

Introducing heating and window actuators raised consistency issues between sensor values and environmental changes. The solution was to centralize actuator effects in the RoomAgent by adjusting sensor baselines rather than modifying raw published values.

5.3 Loose Coupling

Ensuring low coupling between agents was critical. All interactions (data, control, coordination) are performed via MQTT topics, making the system robust to agent failures and easy to extend.

Perspectives

Several extensions could be considered to further improve the proposed system.

First, deploying agents as independent processes or containers would better reflect real-world distributed environments.

Second, introducing persistent storage (e.g., a database) would allow long-term analysis of sensor data.

Finally, more advanced decision mechanisms could be implemented, such as adaptive thresholds for anomaly detection or learning-based strategies for task allocation in the Contract Net protocol.

Conclusion

This project demonstrates how MQTT can be used as a coordination middleware for multi-agent systems. The proposed architecture is modular, extensible, and realistic for IoT and smart-home scenarios. The separation between room-level orchestration, sensor autonomy, and coordination protocols enables dynamic reconfiguration and clear reasoning about agent responsibilities.