ÉCOLE NATIONALE SUPÉRIEURE D'INGÉNIEURS DES SYSTÈMES AVANCÉS ET RÉSEAU
ESISAR - Valence

Filière Informatique, Réseaux et Cybersécurité

CS534 - Multi-agent systems

MQTT Lab

Realized by :
BOUKHRISS Youssef
ICHO Ibrahim

Academic Year 2025 - 2026

1 Technology choices

1.1 Programming Language : Python

Python was selected for this implementation for several compelling reasons :

— **Rapid prototyping :** Python's concise syntax enables quick iteration during development

— **Rich MQTT ecosystem :** The paho-mqtt library provides robust, well-documented MQTT client functionality

— **Built-in concurrency :** Python's threading and asyncio modules facilitate multi-agent simulation

1.2 MQTT Library : Eclipse Paho MQTT

The Paho-MQTT library was chosen as it is :

— Industry-standard with extensive community support

— Well-maintained by the Eclipse foundation

— Provides both synchronous and asynchronous APIs

2 Implementation details

2.1 First Client

This exercise consists of building a simple MQTT client that connects to a broker, subscribes to a topic, publishes 5 messages with 2-second intervals, and prints all received messages to console. The implementation uses Paho's callback system for asynchronous event handling. This non-blocking approach allows the client to maintain the connection while performing other operations.

2.2 Ping Pong

In this exercise we implement two agents that exchange "ping" and "pong" messages, demonstrating bidirectional communication and basic multi-agent interaction. Our Python program serves both roles, configured at launch via command-line argument **–mode**. This design enables code reuse while maintaining distinct agent behaviors.

2.3 Dynamic Sensor Network

The sensor network simulation comprises three agent types :

1. **Sensor agents :** Emit measurements at regular intervals
2. **Averaging agents :** Aggregate sensor data and compute statistics
3. **Interface agent :** Display system state to users

Sensors publish to composite topics encoding their location and identity :
/{zone}/{measurement_type}/{sensor_id} This hierarchical structure enables :

— **Selective subscription :** Averaging agents can subscribe to /living_room/ for all living room sensors

— **Scalability :** New zones add topics without code changes

— **Clarity :** Topic structure self-documents data origin

We chose to simulate realistic measurements using sinusoidal functions with random offsets. This approach creates realistic time-varying data with individual sensor characteristics while remaining computationally lightweight.
For the averaging agent, we chose to compute averages over sliding time windows rather than tumbling (fixed-interval) time windows because :

— More responsive to recent changes

— Smoother statistical outputs

— Better representation of system state

2.4 Anomaly Detection System

The anomaly detection exercise extends the sensor network with two new agent types :

— **Detection agent :** Identifies anomalous readings using statistical methods

— **Identification agent :** Coordinates recovery of faulty sensors

The statistical detection algorithm is designed such that :

— **Two-standard deviation threshold :** Balances sensitivity (catching real anomalies) with specificity (avoiding false positives)

— **Pooled statistics :** Uses all sensor readings for robust mean estimation rather than per-sensor statistics which may be unstable with limited data

— **Minimum data requirement :** Requires 10+ readings before detection to ensure statistical validity

Anomaly alerts include full diagnostic information. We chose /alerts/anomaly as topic to provide a dedicated channel for anomaly alerts, allowing multiple downstream agents (logging, notification, identification) to react independently.

Challenges encoutered

Challenge 1 : False positives during system startup
During initial startup with insufficient data, normal readings were flagged as anomalies due to unstable statistical estimates. That's why we implemented minimum data threshold (10 readings) before enabling anomaly detection.
Challenge 2 : Topic subscription timing
Detection agent needed to subscribe to all sensor topics, but sensors join dynamically. To resolve that, we use wildcard subscription (/) to capture all topics, then filter by topic pattern matching.

2.5 Contract Net Protocol

The contract net protocol is a task allocation mechanism where :

— **Supervisor** broadcasts task requests (Call for Proposals)

— **Machines** evaluate their capability and submit bids

— **Supervisor** selects the best bid and awards the task

— **Selected machine** executes the task

Call for Proposals use broadcast topics for simplicity and dynamic machine discovery. Bids and awards use machine-specific topics to enable easy source identification and targeted delivery, leveraging MQTT's native routing rather than requiring all agents to parse all messages.
The supervisor implements a 2-second deadline window for bid collection, balancing fairness (all machines can respond) with throughput (30 jobs/minute possible). A thread-safe queue.Queue prevents race conditions between MQTT callback threads and the main processing thread.
Greedy algorithm selects the machine offering shortest completion time : min(bids, key=lambda b : b['time']). While this doesn't optimize global makespan, it provides O(n) complexity with minimal latency.
Hierarchical topics (/zone/type/id) were chosen over flat topics with metadata in payloads because :

— MQTT's native wildcard subscription (/living_room/) enables efficient filtering

— Self-documenting message origins without payload parsing

— Scalable to large agent populations without central coordination

— Reduces processing overhead for irrelevant messages

3 Conclusion

This lab teaches us a lot about MQTT by implementing five progressively complex multi-agent systems demonstrating core principles : decentralized coordination, autonomous decisionmaking, dynamic agent populations, and emergent system behavior. The implementations showcase effective use of MQTT's publish-subscribe model, with thoughtful topic design enabling efficient message routing and agent coordination.