Project Report: MQTT Smart Home with Raspberry Pi 5 and ESP32

Author: Group 2

December 22, 2025

1 Setup

For this project, we utilized a Raspberry Pi 5 to run the Mosquitto MQTT broker. We installed the Mosquitto broker and client tools using the standard package manager:

```
sudo apt update
sudo apt install mosquitto mosquitto-clients
```

We verified that the service is active and running on the default port 1883:

2 ESP32: Publisher / Subscriber Node

Our system models a home with two rooms: a living room and a bedroom. The ESP32 acts as a publisher and publishes temperature data and a subscriber for lighting control commands. The table 1 below lists each implemented topic, along with the direction, QoS and a description.

Noticeably, we chose QoS 1 (at least once) for all topics. This is due to the fact that in a home automation context, reliability is preferred over speed. A QoS level 1 ensures that even if the network is unstable, temperature readings and light commands are queued and eventually delivered. This prevents any "lost" actions like a light failing to turn off.

3 Wildcards

For this project, we used both single-level and multi-level wildcards for topic subscriptions.

3.1 Single-level Wildcard +

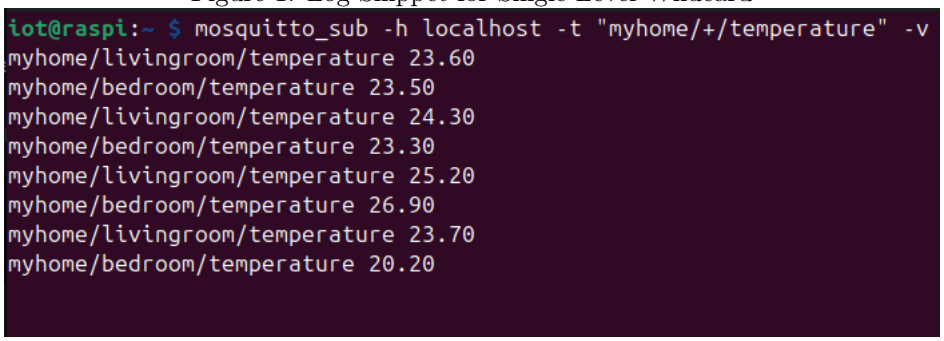
A single-level wildcard was used to subscribe to the topic `myhome/+/temperature`. Because of this wildcard, this subscription receives messages from both `myhome/livingroom/temperature` and `myhome/bedroom/temperature`. Important to note is that since lighting topics are located deeper in the hierarchy, (e.g. `.../light/bedside`, it does not match the pattern `myhome/+/temperature` and therefore, any light topics will not be received.

Figure 1 shows a log snippet indicating that only temperature data is being received.

Table 1: Table with implemented topics

Topic	Direction	QoS	Description
myhome/livingroom/temperature	Publisher	1	Simulates living room temperature (published every 10s).
myhome/bedroom/temperature	Publisher	1	Simulates bedroom temperature (published every 10s).
myhome/livingroom/light/colorLED1	Subscriber	1	Controls the color LED in the living room.
myhome/livingroom/light/whiteLED1	Subscriber	1	Controls the white LED in the living room.
myhome/bedroom/light/bedside	Subscriber	1	Controls the bedside light in the bedroom.

Figure 1: Log Snippet for Single-Level Wildcard



3.2 Multi-level Wildcard

A multi-level wildcard was used to subscribe to the topic `myhome/bedroom/#`. In this case, the # wildcard matches the current level and all subsequent sub-levels recursively, which means that all data related to the bedroom will be received. More specifically, we will receive all traffic related to `myhome/bedroom/temperature` and `myhome/bedroom/light/bedside`.

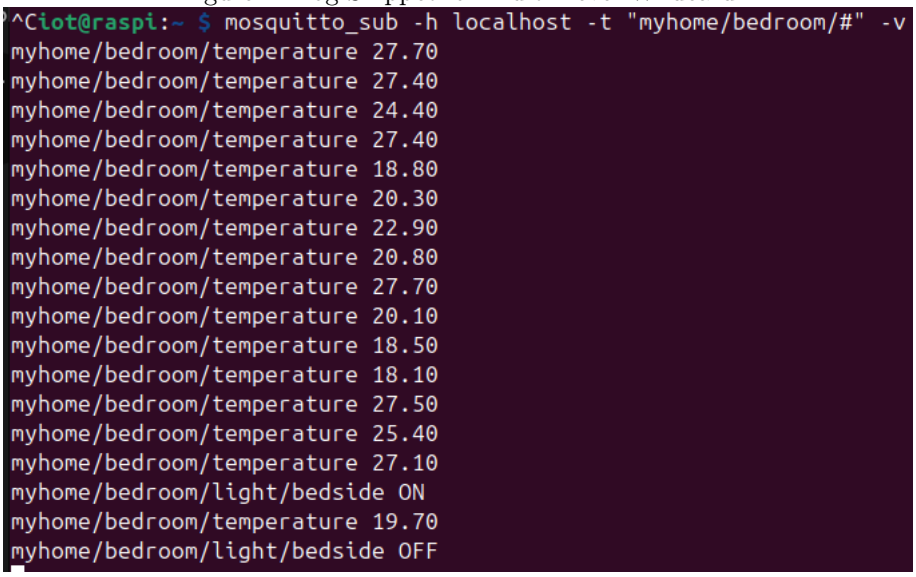
Figure 2 shows a log snippet received by `myhome/bedroom/#`.

4 Quality of Service (QoS)

We also conducted experiments to compare QoS levels:

1. QoS 0 (At most once): When we disconnected the subscriber while the ESP32 continued to publish, the messages that were generated during the downtime were permanently lost. This is due to this level of service having no acknowledgement mechanism. Since messages are sent at most once, there are no duplicates. With this level we have less overhead but also less reliability.
2. QoS 1 (At least once): In contrast, when we used QoS 1, the broker queued messages sent while the subscriber was offline. After the subscriber reconnected, the "missed" messages were received. This level of service provides higher reliability but also incurs a slightly higher network overhead due to acknowledgment packets. It should be noted that with QoS level 1, messages might be delivered more than once due to retransmissions. Subscriber applications need to be able to handle duplicates.

Figure 2: Log Snippet for Multi-Level Wildcard



5 Transient vs. Durable Subscriptions

5.1 Transient Subscription (Clean Session)

By default, the client starts with a clean session. In this case, if the client disconnects, the broker discards any subscription data. In our project, any message published to `myhome/livingroom/light/colorLED1` while the client was offline were not delivered upon reconnection.

5.2 Durable Subscription (Persistent Session)

A durable session was established by using the `-c` flag and a fixed Client ID `-i clientID`. In this case, when `clientID` disconnected, the broker stored the messages published to the subscribed topic. Upon reconnecting, all missed messages were immediately delivered.

6 Security

Security is crucial when it comes to Internet of Things. Using MQTT unencrypted and unauthenticated can expose the user to several risks:

- **Unauthorized Control:** An attacker could publish commands to topics, e.g., to toggle lights physically, which may cause disturbance or even safety risks. A possible mitigation for this problem includes Access Control Lists (ACLs)
- **Spoofing:** A malicious actor could inject fake temperature data into topics such as `myhome/bedroom/temperature`. This could potentially cause confusion (e.g. heating automation systems). A way to mitigate this is to user Username/Password Authentication.
- **Eavesdropping:** Without proper encryption, it is possible for an attacker on the same network to subscribe to the topics and read all sensor data. To mitigate this, one could use TLS encryption.
- **Denial of Service (DoS):** During a DoS attack, the broker may be flooded with a massive amount of messages. This can exhaust system resources and prevent proper communication between devices. To prevent this type of attack, one could introduce rate limiting (limiting messages per second) or connection limits in the broker configuration.

6.1 Implementation: Username/Password

In our project, we secured the Mosquitto broker by enforcing password authentication. This was done by creating a configuration file (along with a password file), which Mosquitto then loaded on the Pi:

```
allow_anonymous false
password_file /etc/mosquitto/passwd
```

The ESP32 client was then updated to authenticate against these credentials:

```
mqttClient.setUsernamePassword("myuser", "mypassword");
  if (!mqttClient.connect(broker, port)) {
    // Error handling
  }
```

This configuration prevents unauthorized users from publishing or subscribing from our topics.