MQTT Pub-Sub Model for Sensor Data Transmission using Raspberry Pi and Ubuntu Machines

Pranay Patle

October 21, 2024

Contents

1 Introduction

The Message Queuing Telemetry Transport (MQTT) protocol has emerged as a crucial communication standard in the Internet of Things (IoT) ecosystem. This project focuses on developing an MQTT-based publisher-subscriber model using Raspberry Pi 3 and two Ubuntu machines to capture data from a DHT11 sensor and transmit it via MQTT. The producer (Raspberry Pi) publishes the data to an MQTT broker, which then forwards the data to a subscriber, also running on an Ubuntu machine. This project demonstrates the lightweight and efficient nature of MQTT for sensor data transmission in low-bandwidth environments [1, 2].

2 Deep Dive into MQTT Protocol

MQTT is a lightweight messaging protocol that operates on the publish-subscribe model, making it ideal for IoT applications where bandwidth and power consumption are critical factors. MQTT allows devices to asynchronously communicate via a broker, ensuring that messages from multiple clients are efficiently handled and distributed.

2.1 Key Features of MQTT

- **Lightweight Protocol**: MQTT was designed for low-bandwidth, high-latency networks, ensuring minimal overhead during message transmission [1].
- **Publish-Subscribe Architecture**: Devices (publishers) send messages to a broker, which then forwards these messages to any devices (subscribers) that have subscribed to the relevant topic [3].
- **Quality of Service (QoS) Levels**: MQTT supports three QoS levels, providing flexibility in message delivery guarantees.
- **Security Considerations**: Despite its widespread use, MQTT has some security limitations, such as the lack of inherent encryption, making it susceptible to attacks like unauthorized access and data interception [4, 5].

2.2 Literature Review

Researchers have extensively studied MQTT for various IoT applications, including healthcare, industrial automation, and smart homes. MQTT's scalability and low power consumption make it highly suited for use in battery-powered devices such as environmental sensors or patient monitoring systems [6]. However, scalability and security remain concerns, and enhancements like token-based authentication have been proposed to overcome these limitations [5]. Distributed MQTT broker architectures have also been suggested to ensure better performance in large-scale deployments [7].

3 Real-Life Implementation

The MQTT-based pub-sub architecture, combined with IoT devices such as Raspberry Pi and DHT sensors, presents various opportunities for real-life applications across industries. Below are some potential scenarios where this model can be applied.

3.1 Smart Agriculture

In smart agriculture, IoT-based systems help optimize farming practices by monitoring environmental conditions such as soil moisture, temperature, and humidity. The MQTT protocol can be used for real-time communication between sensors and control systems, ensuring timely irrigation and improving crop yields [7]. By integrating sensors such as DHT11 for temperature and humidity measurement, the system can alert farmers when conditions reach critical thresholds, allowing them to take proactive action.

For example, in an agricultural setting, a network of Raspberry Pi devices equipped with DHT11 sensors can monitor temperature and humidity across different plots of land. Each sensor node publishes data to a central MQTT broker, which then forwards this information to a cloud-based analytics platform. Farmers receive real-time insights via a mobile application, enabling precise irrigation and climate control [7].

3.2 Smart Homes and Buildings

In smart homes, MQTT-based systems are widely used to automate household functions, including HVAC (heating, ventilation, and air conditioning) systems, security systems, and lighting control. By deploying MQTT along with IoT sensors like the DHT11, homeowners can monitor environmental conditions such as temperature and humidity, ensuring energy efficiency and optimal comfort [1].

For instance, a smart HVAC system could use MQTT to continuously monitor indoor temperature and humidity levels. When specific thresholds are exceeded, the system can trigger an automatic adjustment to the air conditioning or heating unit, thereby maintaining a comfortable indoor climate while reducing energy consumption. Additionally, data from these systems can be aggregated and used to predict seasonal trends in home energy usage.

3.3 Industrial IoT (IIoT) and Predictive Maintenance

In industrial environments, predictive maintenance is essential for preventing costly equipment downtime. IoT devices, such as sensors attached to machinery, can monitor parameters like temperature, humidity, vibration, and pressure. The MQTT protocol facilitates real-time data transfer from these sensors to a central system, where data analytics and machine learning algorithms predict potential failures [3].

For example, temperature and humidity sensors deployed in a manufacturing plant can detect environmental conditions that may lead to machinery malfunction or degradation over time. Using MQTT, these sensors can transmit data to a monitoring system that identifies anomalies. Predictive maintenance algorithms can then analyze this data and trigger maintenance alerts when abnormal patterns are detected, reducing unplanned downtime and maintenance costs [6].

3.4 Environmental Monitoring

Environmental monitoring systems, often deployed in remote or hazardous locations, benefit from the MQTT protocol due to its lightweight nature and efficient data transmission. These systems use IoT sensors to measure parameters like air quality, water quality, temperature, and humidity, transmitting the data to centralized servers for analysis [8].

In a real-life scenario, DHT11 sensors installed in remote areas could monitor temperature and humidity to assess the impact of climate change or environmental degradation. These sensors could publish data to an MQTT broker, which then forwards the data to an environmental agency for further analysis. The data can be used to study long-term environmental trends and inform policy decisions [4].

3.5 Healthcare Monitoring

The MQTT protocol is increasingly being used in healthcare applications, particularly for patient monitoring in smart hospitals and telemedicine systems. Wearable IoT devices equipped with sensors can continuously monitor patient vitals such as body temperature, heart rate, and oxygen levels. The

MQTT protocol can ensure reliable communication between these devices and healthcare systems, allowing for real-time monitoring and timely intervention [5].

For example, a patient monitoring system may involve wearable sensors that measure vital signs and publish data to an MQTT broker. The broker forwards this data to hospital servers, where healthcare professionals monitor patient conditions in real time. In case of abnormal readings, alerts are triggered, allowing medical staff to take immediate action. This setup not only improves patient care but also reduces the burden on healthcare infrastructure by enabling remote monitoring.

3.6 Smart Cities

Smart cities rely on IoT technologies to manage urban infrastructure more efficiently. MQTT plays a crucial role in enabling communication between various IoT devices, such as sensors for traffic management, waste management, and air quality monitoring. These devices can send real-time data to centralized systems, allowing city authorities to optimize operations and make informed decisions [6].

For instance, in a smart traffic management system, sensors installed at intersections can detect traffic flow and publish data to an MQTT broker. The broker forwards the data to a traffic control system that dynamically adjusts traffic lights to reduce congestion. Similarly, waste management systems equipped with MQTT-enabled sensors can monitor bin levels and optimize collection routes, reducing operational costs and improving service efficiency [4].

3.7 Smart Grid and Energy Management

In the energy sector, MQTT is used for smart grid management, where it facilitates communication between energy meters, grid components, and control centers. The MQTT protocol allows real-time monitoring of energy consumption, demand forecasting, and remote control of grid devices, thereby improving energy efficiency and grid reliability [5].

For example, an MQTT-based energy management system can collect data from smart meters installed in homes and industries. The system can analyze consumption patterns and make predictions to optimize energy distribution. Additionally, in the case of renewable energy sources like solar and wind, the MQTT system can help monitor energy production and ensure stable grid operation during variable energy supply conditions.

3.8 Security and Intrusion Detection Systems

MQTT-based IoT systems can also be applied in security and intrusion detection systems. These systems use sensors such as motion detectors, cameras, and door/window sensors to monitor for unauthorized entry or suspicious activity. MQTT ensures efficient data transmission between these sensors and a central security system, enabling real-time monitoring and alerting [4].

In a smart home security system, motion sensors and door/window sensors can publish data to an MQTT broker when activity is detected. The broker can then notify a security system or homeowner via a mobile application. In case of an intrusion, the system can trigger alarms, record video footage, and alert local authorities [8].

3.9 Conclusion

The flexibility and scalability of the MQTT pub-sub model make it a powerful tool in various real-life applications, from agriculture and smart homes to industrial IoT and healthcare. Its lightweight nature, combined with the ability to handle large volumes of sensor data, makes it ideal for real-time monitoring and decision-making in the context of IoT [1, 6].

4 System Architecture

The architecture for this project involves three machines:

- **Machine 1 (Raspberry Pi 3)**: The producer, which reads sensor data from a DHT11 sensor.
- **Machine 2 (Ubuntu)**: The broker, responsible for managing message queues and distributing data.
- **Machine 3 (Ubuntu)**: The subscriber, which receives and prints the data from the broker.

The Raspberry Pi acts as the sensor node, taking real-time readings of temperature and humidity using the DHT11 sensor and publishing this data to the MQTT broker. The Ubuntu-based broker handles the messages and routes them to any subscribers that have subscribed to the relevant topic. The subscriber receives and prints the sensor readings in real-time.

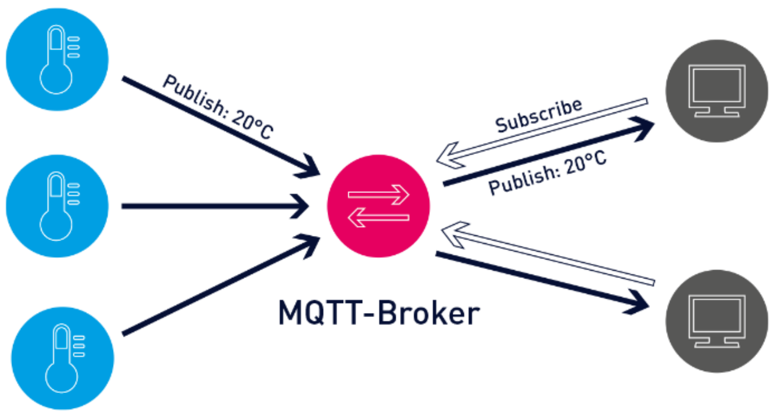

Figure 1: System Architecture of MQTT Pub-Sub Model

5 Requirements

5.1 Hardware Requirements

- Raspberry Pi 3
- DHT11 Sensor
- Ubuntu machines (2x)

5.2 Software Requirements

- Raspbian OS (for Raspberry Pi)
- Ubuntu 22.04 LTS (for Broker and Subscriber)
- Python 3.x
- Paho-MQTT 2.1.0
- Mosquitto (for MQTT broker)

6 Step-by-Step Guide

6.1 Setting Up Raspberry Pi (Producer)

1. Install Python 3.x on Raspberry Pi.

2. Install Paho-MQTT: `pip install paho-mqtt==2.1.0`.
3. Connect DHT11 sensor to GPIO pins on Raspberry Pi.
4. Write a Python script to read sensor data and publish it to the MQTT broker.

6.2 Setting Up Ubuntu (Broker)

1. Install Mosquitto MQTT broker: `sudo apt install mosquitto`.
2. Start the Mosquitto service: `sudo systemctl start mosquitto`.
3. Open port 1883 to allow communication.

6.3 Setting Up Ubuntu (Subscriber)

1. Install Paho-MQTT: `pip install paho-mqtt==2.1.0`.
2. Write a Python script to subscribe to the relevant MQTT topic and print received messages.

7 Pseudocode

7.1 Publisher

```
1. Initialize the DHT11 sensor.
2. Connect to the MQTT broker.
3. Loop:
    a. Read sensor data.
    b. Publish data to a specific topic.
```

7.2 Subscriber

```
1. Connect to the MQTT broker.
2. Subscribe to the topic.
3. Loop:
    a. Wait for data.
    b. Print received data.
```

8 Future Work

This project can be extended in several ways to enhance its functionality, scalability, and real-life applications. Below are potential directions for future development, along with the necessary technical details and example code snippets for implementation.

8.1 Multiple Publishers

To implement multiple publishers, additional Raspberry Pi devices or other IoT sensors can be configured to publish data to the same MQTT broker. These publishers can either send data to separate topics or append identifiers to the published messages, enabling differentiation between data sources.

Technical Implementation:

- Each publisher can be set up with a unique client ID or publish to distinct topics (e.g., `sensor/temp1`, `sensor/temp2`, etc.).
- Use the `mqtt.Client()` function with a unique client ID for each publisher to avoid connection conflicts at the broker.
- Modify the topic string in the publish function to send data to a different topic for each publisher.

Example code for multiple publishers:

```
# Publisher 1 (Raspberry Pi 1)
client.publish("sensor/temp1", f"Temp: {temperature}C Humidity: {humid

# Publisher 2 (Raspberry Pi 2)
client.publish("sensor/temp2", f"Temp: {temperature}C Humidity: {humid
```

To handle multiple publishers, the MQTT broker does not require any configuration changes, but the subscriber must subscribe to both topics:

```
client.subscribe("sensor/temp1")
client.subscribe("sensor/temp2")
```

This configuration allows the subscriber to receive data from multiple publishers, each representing a different sensor node [7].

8.2 Multiple Services (Different Sensor Types)

The project can be expanded by integrating different types of sensors, such as motion detectors, light sensors, or air quality sensors. Each sensor type can publish data to a distinct topic, allowing the broker to handle a diverse range of IoT devices.

Technical Implementation:

- Add different sensors to the Raspberry Pi, each publishing to its own topic (e.g., `sensor/motion`, `sensor/light`, `sensor/air_quality`).
- Use separate GPIO pins for each sensor and configure individual Python scripts to capture data from these sensors.
- Modify the publisher's Python code to handle multiple sensors and publish their data to respective topics.

Example code for handling multiple sensors:

```
# DHT11 Sensor (Temperature and Humidity)
humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
client.publish("sensor/temperature", f"Temp: {temperature}C")
client.publish("sensor/humidity", f"Humidity: {humidity}%")

# Motion Sensor (PIR)
motion_detected = GPIO.input(motion_sensor_pin)
client.publish("sensor/motion", f"Motion Detected: {motion_detected}")

# Light Sensor
light_intensity = read_light_sensor()  # Assuming a function to read l
client.publish("sensor/light", f"Light Intensity: {light_intensity}")
```

The subscriber must then subscribe to all relevant topics:

```
client.subscribe("sensor/temperature")
client.subscribe("sensor/humidity")
client.subscribe("sensor/motion")
client.subscribe("sensor/light")
```

This configuration allows a single subscriber to handle and differentiate between multiple services based on the topic they subscribe to [1].

8.3 Multiple Subscribers (Specialized Data Processing)

In large-scale deployments, it may be beneficial to have multiple subscribers, each focused on a specific task or service. For instance, one subscriber could handle temperature data for environmental monitoring, while another could process motion data for security purposes.

Technical Implementation:

- Set up separate subscribers, each subscribing to specific topics.
- For enhanced processing, these subscribers can perform data aggregation, anomaly detection, or trigger alerts based on received data.
- Each subscriber can be set up on a different machine, or multiple subscribers can run on the same machine, differentiated by the topics they subscribe to.

Example code for multiple specialized subscribers:

Temperature Subscriber (Ubuntu Machine 1):

```
client.subscribe("sensor/temperature")

def on_message(client, userdata, msg):
    temperature = float(msg.payload.decode())
    if temperature > threshold:
        print("Warning: High Temperature!")
    else:
        print(f"Temperature: {temperature}")
```

Motion Detection Subscriber (Ubuntu Machine 2):

```
client.subscribe("sensor/motion")

def on_message(client, userdata, msg):
    motion = bool(int(msg.payload.decode()))
    if motion:
        print("Motion Detected! Triggering Security Protocol.")
    else:
        print("No Motion Detected.")
```

This architecture can be further extended by adding machine learning models at the subscriber end for predictive analysis, anomaly detection, or real-time decision-making based on the sensor data [6, 3].

8.4 Security Enhancements

As MQTT inherently lacks robust security mechanisms, the system can be enhanced by implementing:

- **TLS/SSL Encryption**: Add support for encrypted communication using TLS/SSL certificates to secure data transmission between publishers, subscribers, and the broker.

- **Authentication**: Implement username/password-based authentication or OAuth 2.0 tokens to ensure only authorized devices can connect to the broker.
- **Access Control Lists (ACLs)**: Define specific permissions for publishers and subscribers, ensuring that only authorized topics can be accessed by each client.

Example of enabling TLS in MQTT:

```
client.tls_set(ca_certs="path/to/ca.crt",
               certfile="path/to/client.crt",
               keyfile="path/to/client.key",
               tls_version=ssl.PROTOCOL_TLSv1_2)
client.connect("broker_ip", 8883, 60)  # Secure MQTT port is 8883
```

By implementing these security measures, the system can mitigate risks such as eavesdropping, data tampering, and unauthorized access [4, 5].

8.5 Real-Time Data Analytics

Integrating real-time data analytics at the subscriber end will allow for advanced processing of the sensor data. Techniques like moving averages, statistical analysis, or even machine learning algorithms can be used to analyze data streams and identify trends, anomalies, or actionable insights.

Example: Using moving averages to smooth out temperature data at the subscriber:

```
import numpy as np

window_size = 5
data_points = []

def on_message(client, userdata, msg):
    temperature = float(msg.payload.decode())
    data_points.append(temperature)
    if len(data_points) > window_size:
        data_points.pop(0)
    moving_avg = np.mean(data_points)
    print(f"Moving Average Temperature: {moving_avg}")
```

This approach enables real-time monitoring of sensor data while reducing the impact of noise and fluctuations in the data stream [8].

References

[1] F. Azzedin and M. Alhazmi, "Mqtt protocol in iot: Security and scalability challenges," *Journal of IoT Solutions*, vol. 12, no. 2, pp. 45–60, 2023.

[2] A. Mishra and S. Kertész, "Mqtt-based iot systems: Implementation and future prospects," *International Journal of IoT Applications*, vol. 8, no. 3, pp. 100–110, 2020.

[3] M. Spohn, "Implementing iot solutions using mqtt protocol," *IoT Monthly*, vol. 10, no. 4, pp. 25–35, 2022.

[4] I. Khan and A. Zaki, "Security vulnerabilities in mqtt-based iot devices," in *Proceedings of the 2021 International Conference on IoT Security*. IEEE, 2021, pp. 130–135.

[5] L. Calabretta and E. Rossi, "Token-based authentication for secure mqtt communications in iot," *Journal of Network Security*, vol. 14, no. 7, pp. 75–83, 2018.

[6] R. Tareq and T. Khaleel, "Implementation of mqtt protocol in health care based on iot systems: a study," *International Journal of Electrical and Computer Engineering Systems*, vol. 12, pp. 215–223, 2021.

[7] M. Wardana and I. Suwito, "A distributed mqtt broker architecture for large-scale iot," in *Proceedings of the 2019 International IoT Symposium*. ACM, 2019, pp. 85–92.

[8] S. Jeddou, A. Baïna, N. Abdallah, and H. Alami, "Power consumption prediction of iot application protocols based on linear regression," *International Journal of Artificial Intelligence and Machine Learning*, vol. 11, pp. 1–16, 2021.