



Innovative wireless platforms for the internet of things

Report of Lab 5

Integration and Performance of MQTT in IoT Applications

Hossein Zahedi Nezhad s309247

Professor: Daniele Trincherò

June 2024

Overview:

This report details the functionality and outcomes of Lab 5, focusing on the implementation of the MQTT protocol within an Internet of Things (IoT) environment. It outlines three primary tasks involving the creation of MQTT message receivers and publishers, the simulation of a temperature sensor using Python and MQTT, and the management of data transmission using the Cayenne Low Power Payload (CayenneLPP) format. Each task demonstrates the successful setup and execution of MQTT-based communication between a publisher and subscriber, utilizing specific topics to manage data flow and ensure system accuracy and efficiency in real-world simulation scenarios.

Task 1

In the first task we tried to create a program capable of acting as an MQTT message receiver. The program needed to subscribe to a specific topic and handle incoming messages formatted in a specific JSON schema. The messages should be printed on the terminal, indicating both the sender's ID and the content of the message.

Code Implementation

1) Subscriber:

```
C: > Users > hosse > Desktop > Lab5.py > on_connect
1  import paho.mqtt.client as mqtt
2  import json
3
4  def on_connect(client, userdata, flags, rc):
5      if rc == 0:
6          print("Connected successfully.")
7          client.subscribe("IP4IoT/lab5/309247")
8      else:
9          print(f"Connection failed with code {rc}")
10
11 def on_message(client, userdata, msg):
12     message_data = json.loads(msg.payload.decode('utf-8'))
13     print(f"Message from {message_data['sender']}: {message_data['msg']}")
14
15
16 client = mqtt.Client()
17 client.on_connect = on_connect
18 client.on_message = on_message
19
20 try:
21     client.connect("localhost", 1883, 60)
22 except Exception as e:
23     print(f"Failed to connect to the broker: {e}")
24
25 client.loop_forever()
```

Figure 1. The Implemented Code for the Subscriber

The Implemented code is consisted of two main parts including:

1.1. Setup:

In order to create an MQTT client that can connect and subscribe to topics, we use a python library called `paho.mqtt.client` and through that we handle messages from an MQTT broker. Moreover, the `json` library is used to decode the JSON formatted messages received from the topic.

1.2. Client Configuration:

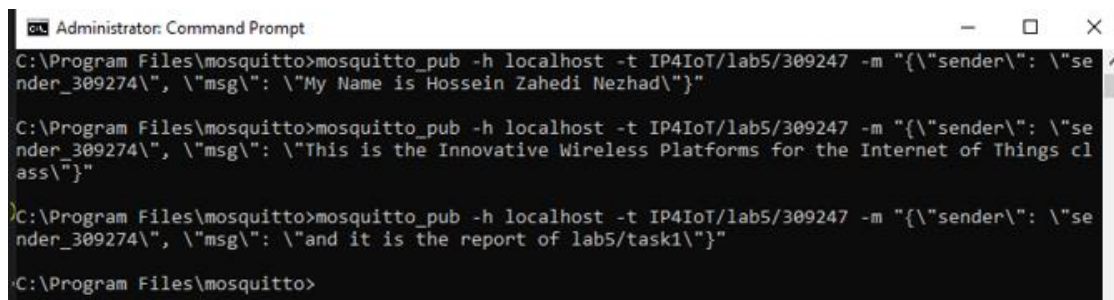
The MQTT client was configured to connect to an MQTT broker running on `localhost` at port 1883, which is the default port for MQTT. When the connection is established successfully, the client subscribes to the topic `IP4IoT/lab5/309247`, which is my student ID, allowing it to receive messages.

2) Publisher:

The publisher sends messages to a specific topic on an MQTT broker. It's responsible for creating and sending data that other clients (subscribers) can listen to. To do that, we use the `mosquitto_pub` command line tool to send messages to the subscribed topic. These messages were structured according to the specified JSON schema.

```
{sender: <sender_student_id>, msg: <string>}
```

In addition, we use a topic for routing and receiving messages by the specific subscriber which is here: `IP4IoT/lab5/309247`



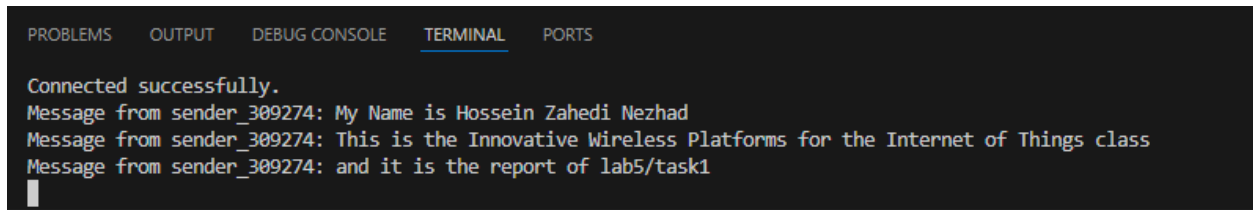
```
Administrator: Command Prompt
C:\Program Files\mosquitto>mosquitto_pub -h localhost -t IP4IoT/lab5/309247 -m '{"sender\": \"se
nder_309274\", \"msg\": \"My Name is Hossein Zahedi Nezhad\"}'
C:\Program Files\mosquitto>mosquitto_pub -h localhost -t IP4IoT/lab5/309247 -m '{"sender\": \"se
nder_309274\", \"msg\": \"This is the Innovative Wireless Platforms for the Internet of Things cl
ass\"}'
C:\Program Files\mosquitto>mosquitto_pub -h localhost -t IP4IoT/lab5/309247 -m '{"sender\": \"se
nder_309274\", \"msg\": \"and it is the report of lab5/task1\"}'
C:\Program Files\mosquitto>
```

Figure 2. Using Command Line to Publish Messages to MQTT Broker

Results

From the publisher side via the command line, we transmitted three distinct messages: **"My Name is Hossein Zahedi Nezhad,"** **"This is the Innovative Wireless Platforms for the Internet of Things class,"** and **"and it is the report of lab5/task1."** As depicted in Figure 3, these messages are received and displayed by the subscriber's terminal.

Additionally, the “Connected successfully” message confirms that the program has successfully established a connection with the MQTT broker.

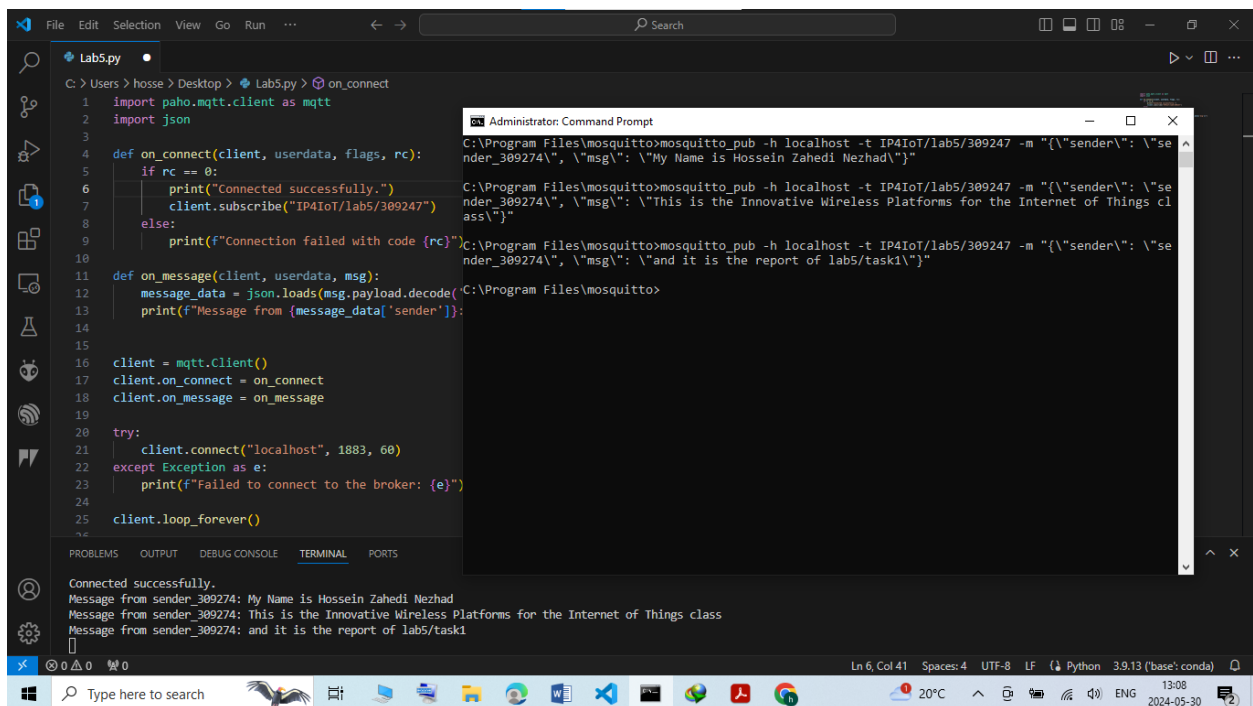


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Connected successfully.
Message from sender_309274: My Name is Hossein Zahedi Nezhad
Message from sender_309274: This is the Innovative Wireless Platforms for the Internet of Things class
Message from sender_309274: and it is the report of lab5/task1
```

Figure 3. MQTT Subscriber Terminal Displaying Received Messages

It is worth mentioning that, the number "309274" is the identifier of the sender specified in the payload of MQTT messages. It distinguishes who sent each message to the MQTT topic "IP4IoT/lab5/<309247>"



```
File Edit Selection View Go Run ... Search
Lab5.py
C:\Users>hosse> Desktop > Lab5.py > on_connect
1 import paho.mqtt.client as mqtt
2 import json
3
4 def on_connect(client, userdata, flags, rc):
5     if rc == 0:
6         print("Connected successfully.")
7         client.subscribe("IP4IoT/lab5/309247")
8     else:
9         print(f"Connection failed with code {rc}")
10
11 def on_message(client, userdata, msg):
12     message_data = json.loads(msg.payload.decode())
13     print(f"Message from {message_data['sender']}")
14
15
16 client = mqtt.Client()
17 client.on_connect = on_connect
18 client.on_message = on_message
19
20 try:
21     client.connect("localhost", 1883, 60)
22 except Exception as e:
23     print(f"Failed to connect to the broker: {e}")
24
25 client.loop_forever()
```

```
Administrator: Command Prompt
C:\Program Files\mosquitto>mosquitto_pub -h localhost -t IP4IoT/lab5/309247 -m "{\"sender\": \"sender_309274\", \"msg\": \"My Name is Hossein Zahedi Nezhad\"}"
C:\Program Files\mosquitto>mosquitto_pub -h localhost -t IP4IoT/lab5/309247 -m "{\"sender\": \"sender_309274\", \"msg\": \"This is the Innovative Wireless Platforms for the Internet of Things class\"}"
C:\Program Files\mosquitto>mosquitto_pub -h localhost -t IP4IoT/lab5/309247 -m "{\"sender\": \"sender_309274\", \"msg\": \"and it is the report of lab5/task1\"}"
C:\Program Files\mosquitto>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Connected successfully.
Message from sender_309274: My Name is Hossein Zahedi Nezhad
Message from sender_309274: This is the Innovative Wireless Platforms for the Internet of Things class
Message from sender_309274: and it is the report of lab5/task1
```

Ln 6, Col 41 Spaces: 4 UTF-8 LF Python 3.9.13 (base: conda) 13:08 2024-05-30

Figure 4. Complete Overview of the Connection

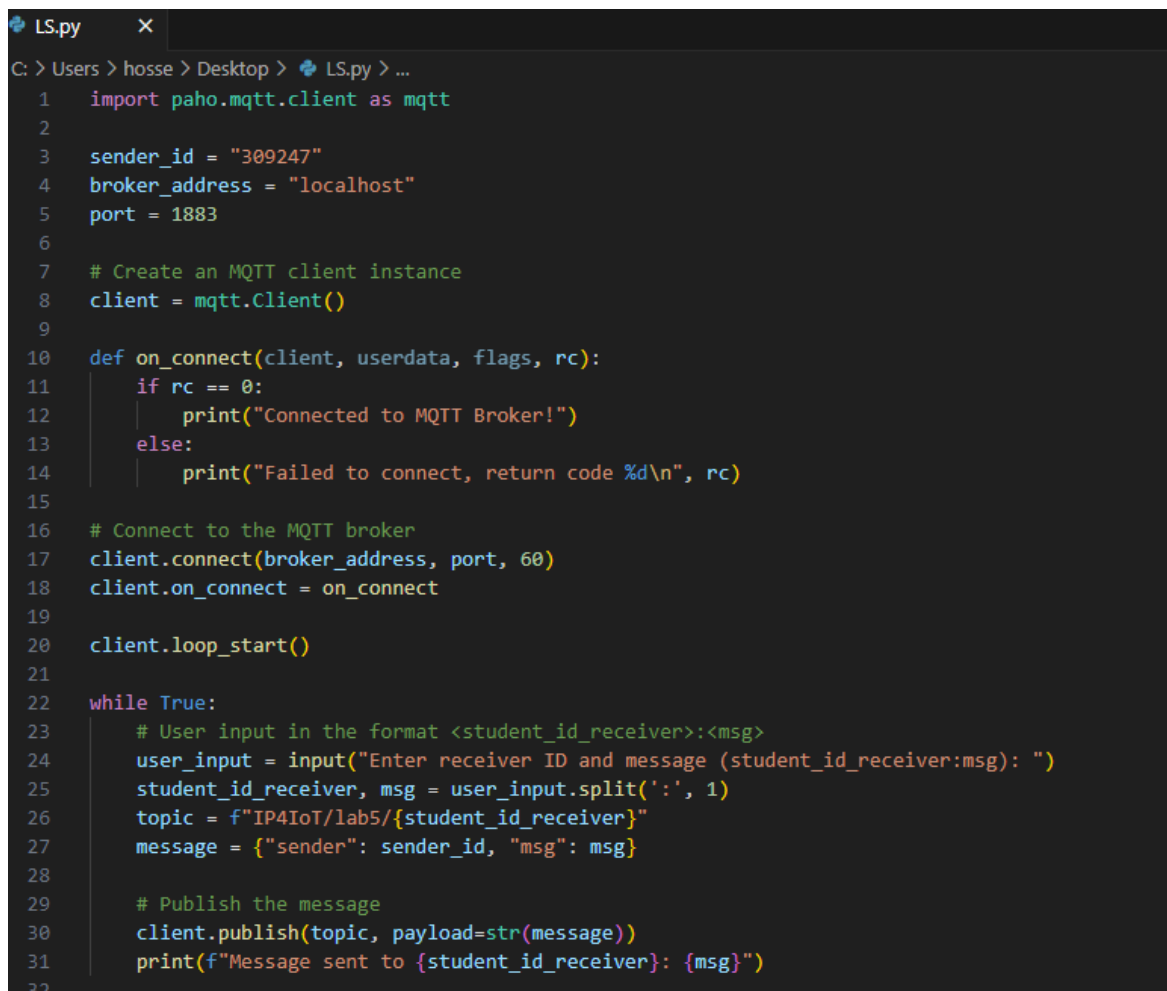
Task 2

The task involved developing an MQTT publisher program that allows a user to send messages formatted in a specific JSON schema. The program is intended to publish messages to a dynamically specified MQTT topic based on the recipient's student ID and wait for user input to send messages.

Code description

Here we also use the `paho.mqtt.client` library to handle MQTT communication. The program establishes a connection to an MQTT broker running on **localhost** at port **1883**. Additionally, by embedding the sender's ID (my student ID is 309247) in each message, the receiver can easily identify which publisher sent the data."

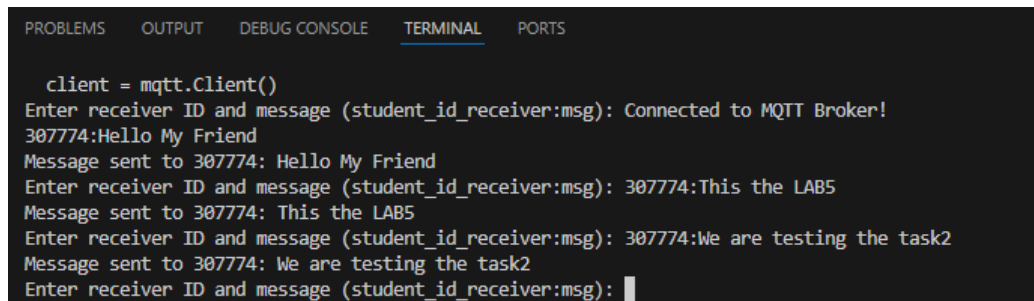
After establishing a connection to the broker and starting a network handling loop, the script repeatedly prompts the user for input in a specified format, indicating a recipient's ID and a message. Once input is received, it splits the input to extract the recipient's ID and message, forms a specific MQTT topic based on the recipient's ID, and constructs a message payload. This message, containing the sender's ID and the actual message text, is then published to the constructed MQTT topic. After publishing, the script outputs a confirmation to the console, indicating successful message transmission to the specified recipient. This process loops indefinitely.

A screenshot of a code editor window titled 'LS.py'. The editor shows a Python script for an MQTT publisher. The script imports the 'paho.mqtt.client' library as 'mqtt'. It defines a 'sender_id' as '309247', a 'broker_address' as 'localhost', and a 'port' as 1883. It creates an MQTT client instance and defines an 'on_connect' callback function that prints a success message if the connection is successful or an error message with the return code if it fails. The script then connects to the broker, sets the 'on_connect' callback, and starts the client loop. Finally, it enters a 'while True' loop where it prompts the user for input in the format 'student_id_receiver:msg', splits the input, constructs a topic 'IP4IoT/lab5/{student_id_receiver}', and publishes a message with a JSON payload containing the sender ID and the message. A confirmation message is printed after each publication.

```
1  import paho.mqtt.client as mqtt
2
3  sender_id = "309247"
4  broker_address = "localhost"
5  port = 1883
6
7  # Create an MQTT client instance
8  client = mqtt.Client()
9
10 def on_connect(client, userdata, flags, rc):
11     if rc == 0:
12         print("Connected to MQTT Broker!")
13     else:
14         print("Failed to connect, return code %d\n", rc)
15
16 # Connect to the MQTT broker
17 client.connect(broker_address, port, 60)
18 client.on_connect = on_connect
19
20 client.loop_start()
21
22 while True:
23     # User input in the format <student_id_receiver>:<msg>
24     user_input = input("Enter receiver ID and message (student_id_receiver:msg): ")
25     student_id_receiver, msg = user_input.split(':', 1)
26     topic = f"IP4IoT/lab5/{student_id_receiver}"
27     message = {"sender": sender_id, "msg": msg}
28
29     # Publish the message
30     client.publish(topic, payload=str(message))
31     print(f"Message sent to {student_id_receiver}: {msg}")
32
```

Figure 5. Implemented code for the Publisher

Results



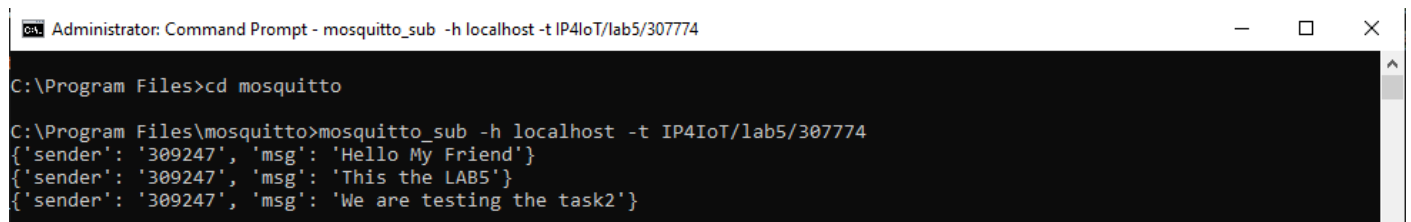
```
client = mqtt.Client()
Enter receiver ID and message (student_id_receiver:msg): Connected to MQTT Broker!
307774:Hello My Friend
Message sent to 307774: Hello My Friend
Enter receiver ID and message (student_id_receiver:msg): 307774:This the LAB5
Message sent to 307774: This the LAB5
Enter receiver ID and message (student_id_receiver:msg): 307774:We are testing the task2
Message sent to 307774: We are testing the task2
Enter receiver ID and message (student_id_receiver:msg):
```

Figure 6. Sending and Confirming Messages

Upon running the provided code, if the message 'Connected to MQTT Broker!' is displayed, it confirms a successful connection to the message broker, allowing us to proceed with sending data. The script then prompts the user to input the receiver ID and message in the format of:

"student_id_receiver:msg"

For testing purposes, we have sent three distinct messages: **'Hello My Friend'**, **'This the LAB5'**, and **'We are testing the task2'** to a specific subscriber identified by the student ID 307774. After each successful publication, the script outputs confirmation messages, indicating that each message has been sent to the specified receiver ID.



```
Administrator: Command Prompt - mosquitto_sub -h localhost -t IP4IoT/lab5/307774
C:\Program Files>cd mosquitto
C:\Program Files\mosquitto>mosquitto_sub -h localhost -t IP4IoT/lab5/307774
{'sender': '309247', 'msg': 'Hello My Friend'}
{'sender': '309247', 'msg': 'This the LAB5'}
{'sender': '309247', 'msg': 'We are testing the task2'}
```

Figure 7. Receiving Messages by Subscriber Terminal

As illustrated in Figure 7, the **mosquitto_sub** command (in the Command Prompt) subscribes to MQTT messages published to a specific topic. In this instance, the topic is **'IP4IoT/lab5/307774'**, meaning that messages published to this topic will be received by this subscriber. The three messages appear in the output, demonstrating that the connection is functioning correctly. Furthermore, **'sender': '309247'** identifies the message's sender, which in this case is the publisher we configured.

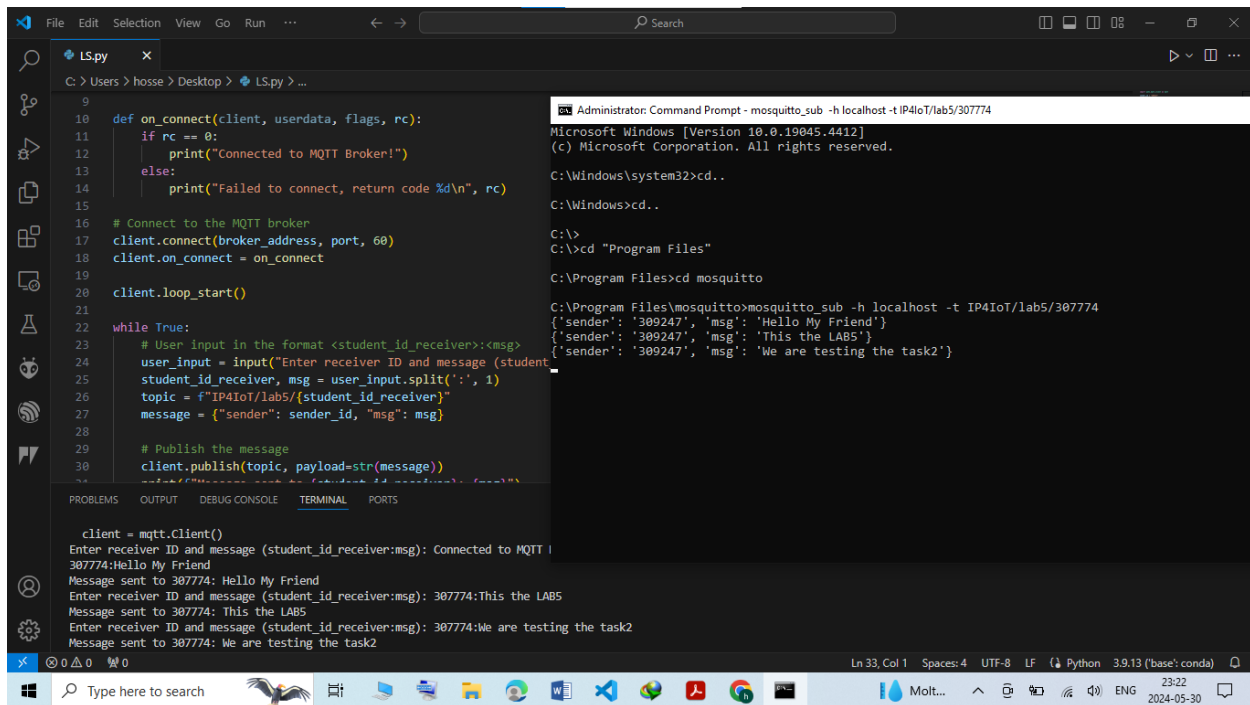


Figure 8. Complete Overview of the system

Task 3

This Task demonstrates the simulation of a temperature sensor using Python, MQTT, and the Cayenne Low Power Payload (CayenneLPP) format. The system periodically generates random temperature values, encodes them in CayenneLPP format, and publishes them to an MQTT broker. A separate script subscribes to these messages and decodes them, displaying the temperatures.

MQTT Configuration

The MQTT Broker Configuration for this project involves setting up a local MQTT broker to facilitate the communication between the temperature sensor simulator and the subscriber receiving temperature data. The chosen port for the MQTT broker is 1883 port. This port allows for straightforward connectivity between publisher and subscriber

The topic used for publishing and subscribing to temperature data is "309247." By using this unique topic, the system ensures that messages are correctly routed to processes that are set up to receive specific types of data, in this case, temperature readings.

Sensor simulation

The temperature sensor simulation is configured with a temperature range of 0.0°C to 20.0°C and a high resolution of 0.1°C. It sends updates every 10 seconds.

Data Format

CayenneLPP Encoded temperature data in CayenneLPP format includes a channel identifier (0x01), type identifier (0x67 for temperature), and the temperature value split into two bytes (MSB and LSB).

Publisher

```
C: > Users > hosse > Desktop > Task3.py > ...
1  import paho.mqtt.client as mqtt
2  import time
3  import random
4  import struct
5
6  # MQTT broker settings
7  broker_address = "localhost"
8  port = 1883
9  topic = "309247"
10
11 # Create an MQTT client instance
12 client = mqtt.Client()
13
14 def on_connect(client, userdata, flags, rc):
15     if rc == 0:
16         print("Connected to MQTT Broker!")
17     else:
18         print("Failed to connect, return code %d\n", rc)
19
20 # Connect to the MQTT broker
21 client.connect(broker_address, port, 60)
22 client.on_connect = on_connect
23
24 client.loop_start()
25
26 try:
27     while True:
28         # Generate a random temperature between 0.0°C and 20.0°C
29         temp_c = random.randint(0, 200)
30
31         payload = bytearray([0x01, 0x67]) + struct.pack("!h", temp_c)
32         client.publish(topic, payload)
33         print(f"Published temperature: {temp_c/10:.1f}°C to {topic}")
34         time.sleep(10) # Send data every 10 seconds
35
36 except KeyboardInterrupt:
37     print("Program terminated by user")
38
39 finally:
40     client.loop_stop()
41     client.disconnect()
42
```

Figure 9. Publisher side of the system

The publisher code in this task has the primary job of generating, encoding, and transmitting temperature data at regular intervals to an MQTT broker, where it can be accessed by subscribing clients.

This piece of code is consisted of several main parts including:

- **MQTT Setup**

The code initializes a connection to an MQTT broker using the `paho.mqtt.client` library. It sets the broker's address (localhost) and port (1883), allowing the publisher to communicate with the broker over the network.

- **Connection Management:**

The `on_connect` function is defined to handle the connection event. It provides feedback on whether the connection to the broker was successful or not. If the connection is successfully established, the message "Connected to MQTT Broker!" is displayed in the publisher's terminal.

- **Data Generation**

In the main loop, the script generates a random temperature value within the specified range of 0°C to 20°C. The random value simulates real sensor output, which would typically fluctuate within a certain range.

- **Data Encoding**

Once a temperature value is generated, it is encoded in CayenneLPP format. This involves creating a bytearray with a predefined channel and type identifier for temperature (0x01 for the channel and 0x67 for temperature). The temperature value is then scaled (multiplied by 10 to match the resolution of 0.1°C) and packed into two bytes using `struct.pack("!h", temp_c)`, ensuring it adheres to the network transmission standards.

- **Data Transmission**

The encoded data is published to the MQTT broker under a specific topic. This allows any subscribing clients to receive updates at the set interval, which in this case is every 10 seconds.

Subscriber:

Similar to the publisher, the subscriber code initializes a connection to an MQTT broker using the `paho.mqtt.client` library. Then the `on_connect` callback function is crucial for handling the connection status with the broker. It confirms a successful connection by subscribing to the specific topic (309247) where temperature data is published.

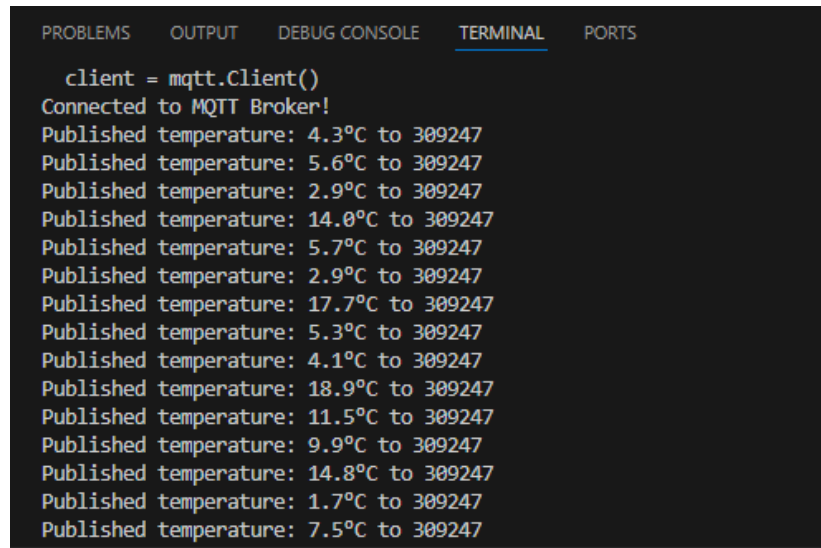
Also the `on_message` callback function is defined to process incoming MQTT messages. This function checks if the received message is formatted correctly (CayenneLPP format) and extracts the temperature data. The data consists of a channel identifier, a type identifier for temperature, and the actual temperature value encoded in two bytes. It then decodes the bytes into a temperature value, converting it back into a human-readable format. Upon receiving a valid message, the decoded temperature is extracted by combining the most significant byte (MSB) and least significant byte (LSB) to form the complete temperature value, which is then divided by 10 to adjust for the resolution scaling.

```
1  import paho.mqtt.client as mqtt
2  import struct
3
4  # MQTT broker settings
5  broker_address = "localhost"
6  port = 1883
7  topic = "309247"
8
9  # Define the callback for when the client receives a CONNACK response from the server
10 def on_connect(client, userdata, flags, rc):
11     print("Connected with result code " + str(rc))
12     client.subscribe(topic)
13
14 # Define the callback for when a PUBLISH message is received from the server
15 def on_message(client, userdata, msg):
16     # Decode the CayenneLPP message
17     if len(msg.payload) >= 4:
18         channel = msg.payload[0]
19         type = msg.payload[1]
20         msb = msg.payload[2]
21         lsb = msg.payload[3]
22
23         if channel == 0x01 and type == 0x67:
24             temperature_raw = (msb << 8) | lsb # Combine the two bytes
25             temperature = temperature_raw / 10.0 # Convert to temperature
26             print(f"Received temperature: {temperature}°C")
27         else:
28             print("Received message with unexpected format.")
29     else:
30         print("Received message with insufficient length.")
31
32 # Create an MQTT client instance
33 client = mqtt.Client()
34
35 # Assign the callback functions
36 client.on_connect = on_connect
37 client.on_message = on_message
38
39 # Connect to the MQTT broker
40 client.connect(broker_address, port, 60)
41
42 client.loop_forever()
```

Figure 10. Subscriber part of the system

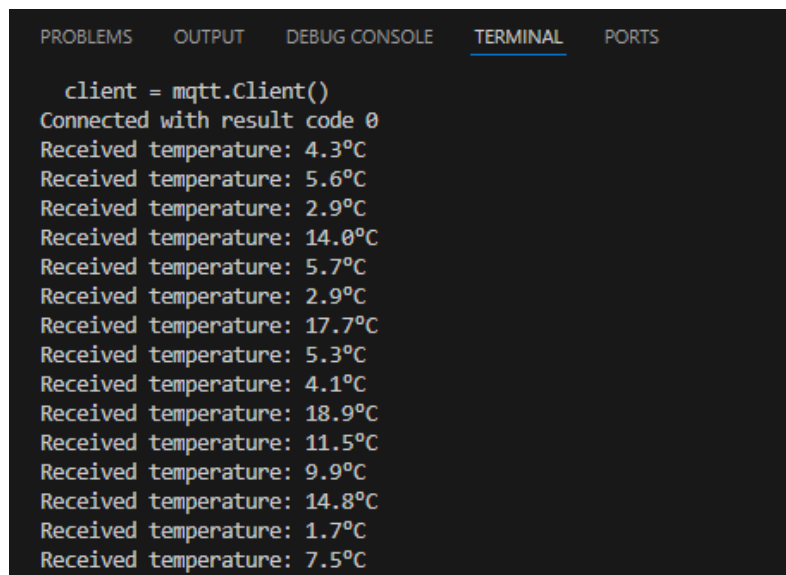
Results:

Running the codes confirms that the system successfully connected to the MQTT broker. As Figures 11 and 12 illustrate, each temperature value is listed as being successfully published to topic '309247.' This demonstrates that both scripts operated as expected, with temperatures accurately published by the publisher and correctly received by the subscriber. All published temperatures were within the predefined range of 0.0°C to 20.0°C and displayed in a human-readable format, showcasing the system's ability to efficiently handle and present data.

A screenshot of a terminal window with a dark background and light-colored text. The terminal has tabs at the top: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. The output shows the initialization of an MQTT client and a series of 15 temperature readings being published to the topic '309247'. Each line follows the format 'Published temperature: [value]°C to 309247'. The temperatures are: 4.3, 5.6, 2.9, 14.0, 5.7, 2.9, 17.7, 5.3, 4.1, 18.9, 11.5, 9.9, 14.8, 1.7, and 7.5.

```
client = mqtt.Client()
Connected to MQTT Broker!
Published temperature: 4.3°C to 309247
Published temperature: 5.6°C to 309247
Published temperature: 2.9°C to 309247
Published temperature: 14.0°C to 309247
Published temperature: 5.7°C to 309247
Published temperature: 2.9°C to 309247
Published temperature: 17.7°C to 309247
Published temperature: 5.3°C to 309247
Published temperature: 4.1°C to 309247
Published temperature: 18.9°C to 309247
Published temperature: 11.5°C to 309247
Published temperature: 9.9°C to 309247
Published temperature: 14.8°C to 309247
Published temperature: 1.7°C to 309247
Published temperature: 7.5°C to 309247
```

Figure 11. MQTT Publisher Terminal Output: Real-time Temperature Data Transmission

A screenshot of a terminal window with a dark background and light-colored text. The terminal has tabs at the top: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. The output shows the initialization of an MQTT client, a successful connection with result code 0, and a series of 15 temperature readings being received from the topic '309247'. Each line follows the format 'Received temperature: [value]°C'. The temperatures are: 4.3, 5.6, 2.9, 14.0, 5.7, 2.9, 17.7, 5.3, 4.1, 18.9, 11.5, 9.9, 14.8, 1.7, and 7.5.

```
client = mqtt.Client()
Connected with result code 0
Received temperature: 4.3°C
Received temperature: 5.6°C
Received temperature: 2.9°C
Received temperature: 14.0°C
Received temperature: 5.7°C
Received temperature: 2.9°C
Received temperature: 17.7°C
Received temperature: 5.3°C
Received temperature: 4.1°C
Received temperature: 18.9°C
Received temperature: 11.5°C
Received temperature: 9.9°C
Received temperature: 14.8°C
Received temperature: 1.7°C
Received temperature: 7.5°C
```

Figure 12. MQTT Subscriber Terminal Output: Real-time Temperature Data Reception

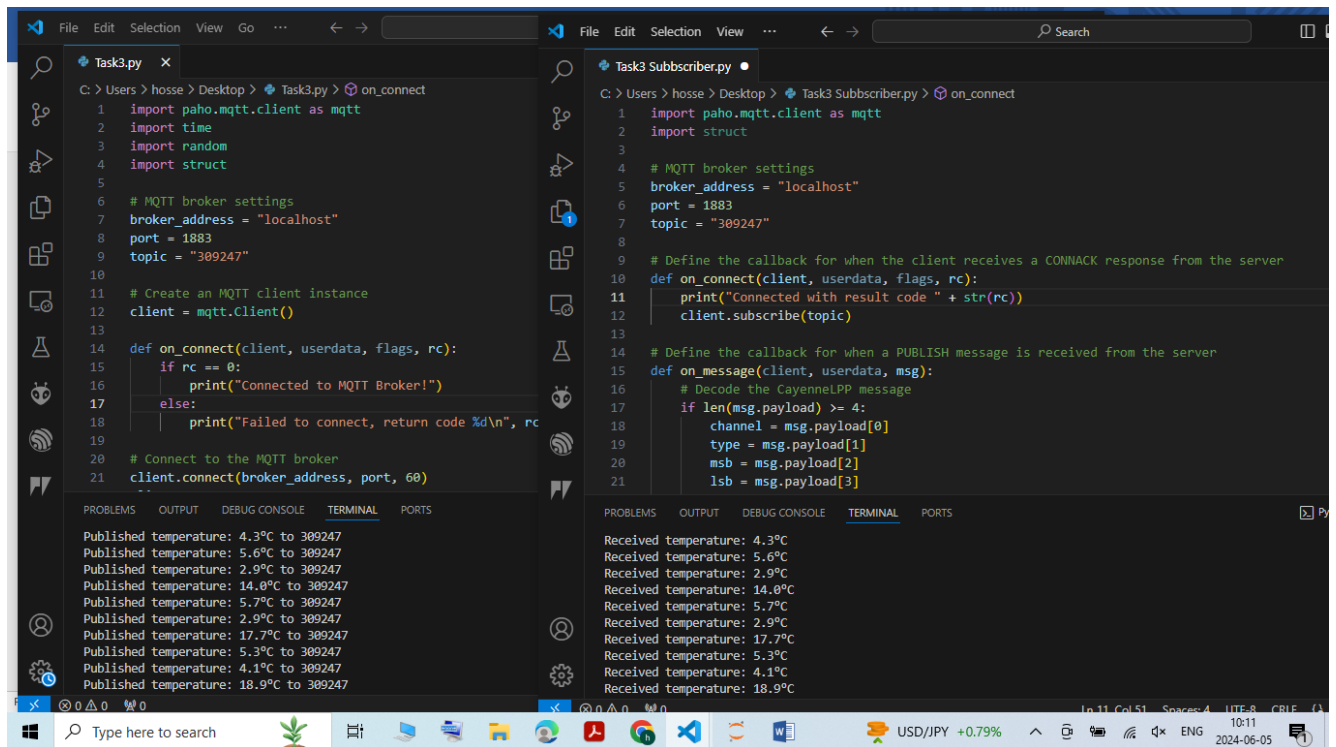


Figure 13. Complete overview of the system