Lab 04: MQTT

Release Date: 11th February, 2025 (Tuesday Section) & 13th February, 2025(Thursday Section)
Demo and Submission Date: 18th February, 2025(Tuesday Section) & 20th February, 2025(Thursday Section)

Files to submit:

- vm_pub.py
- vm_sub.py
- vm_A_chain.py
- vm_B_chain.py
- README.md

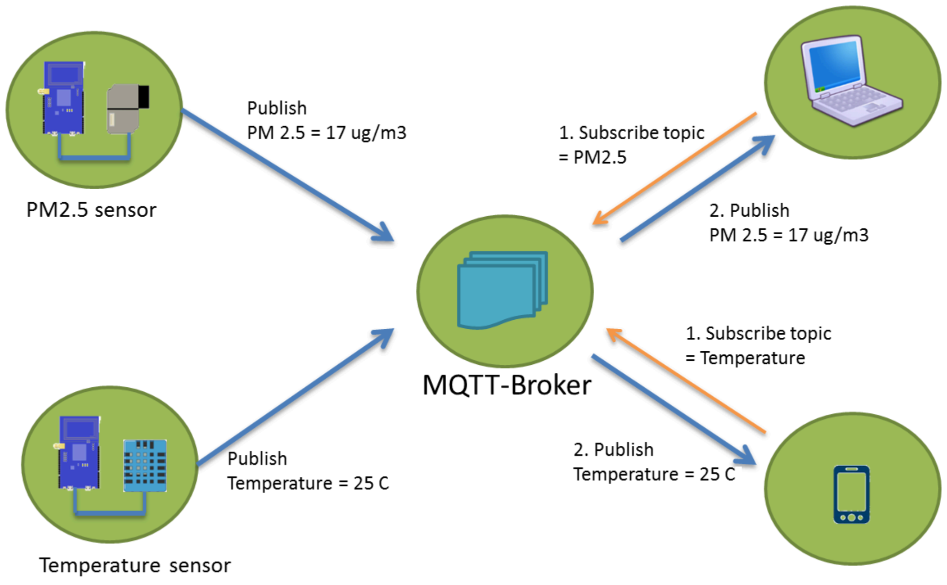
1. Introduction

A. Overview

The objective of this lab is to explore the use of the Publish-Subscribe messaging design pattern with the MQTT protocol. We have already explored the Client-Server architecture in the TCP, UDP, and Socket Programming lab. Unlike a Client-Server architecture that we saw in the previous labs, MQTT is a Publish-Subscribe architecture. This allows for decoupling of the data sources/generators from the data consumers/sinks. Also MQTT allows for an asynchronous data flow, i.e the data consumer does not have to make polling requests to receive the data from the data sources. Instead, whenever new data is available, an entity called an MQTT Broker takes the responsibility to deliver it to the data consumers. It saves a lot of network bandwidth that might have gotten wasted if new data is generated based on events rather than uniform sampling in time.

B. MQTT Architecture

In a publish-subscribe (or just "pub-sub") architecture, we have publishers that produce information and subscribers that consume that information. The communication between these agents is orchestrated by a third party called a **broker**. We are going to use the open-source implementation of the MQTT protocol that is provided by two projects: **Mosquitto** (broker) and **Paho** (publisher and subscriber client), both from the Eclipse Foundation.



Source: https://www.amebaiot.com/en/ameba-arduino-mqtt-upload-listen/

Figure 1 - MQTT example with two publishers and two subscribers

Figure 1 shows a simplified view of the MQTT architecture. We have two types of sensors, PM2.5 (particulate matter) and temperature. Each sensor is connected to an embedded system, such as a Raspberry Pi, and publishes the information to the broker. Each sensor information is associated with a specific topic. The broker has a list of subscribers associated with each topic and forwards the information to the appropriate subscribers. We see that the laptop receives only the PM 2.5 information, while the cell phone receives only the temperature. If there were no subscribers, the broker would accept the published message but would immediately throw away the data since there are no subscribers.

C. Broker Node

The great advantage of MQTT compared to other client-server protocols is its scalability and very simple interface for the application programmer. As we will see in the next sections, the publisher and subscriber only have to know the broker's IP address and port in order to be able to exchange data. The whole information management is abstracted from the programmers so they do not need to know the number of hosts, port numbers, etc. used to create a large-scale distributed application.

3. Tips on Testing

Since the broker IP is the only address you need, you can also prototype your code between two terminals on your VM before working with your raspberry pi. Focus on the MQTT messaging first and once that works, the Python code you wrote should be portable to the RPi.

4. MQTT Setup

We are going to run publishers and subscribers on both your VM and Raspberry Pi. The first thing that we need to do is install the Paho library. We are specifically going to use the Python Paho-MQTT library. If you haven't already installed `pip`, you will need it to install any python packages. To specifically install the python3 version of pip, `pip3`, run this command on both your VM and RPi.

```
sudo apt install python3-pip
```

Next, install `paho-mqtt` via `pip3` on both your VM and RPi.

```
pip3 install paho-mqtt
```

If this method does not work you can try by running the command:

```
sudo apt-get -y install python3-paho-mqtt
```

5. Test publish-subscribe

Install mosquitto-clients and use the command line examples to understand how to use `mosquitto_pub` and `mosquitto_sub`.

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

On one terminal, subscribe to a **topic** at **host** test.mosquitto.org on **port** 1883. As a topic use your unique USC username to avoid interference with classmates:

```
mosquitto_sub -h test.mosquitto.org -p 1883 -t USERNAME
```

Then, open another terminal and publish a **message** to that topic. Note that the backslash is to indicate to your bash shell to interpret the next line to be in-line. That is, “hello” should not be interpreted as a separate line but instead it should be interpreted inline right after “-m”.

```
mosquitto_pub -h test.mosquitto.org -p 1883 -t USERNAME  -m
\“Hello”
```

To better understand the commands better, you can always use `man mosquitto_pub` or `man mosquitto_sub`.

6. Part 1: VM publish and subscribe

For this entire lab, you will use the uniqueness of your usc username to determine your topic names to make sure students do not interfere with one another's topics. **If you are working as a team**, you can choose one of your usernames.

Edit vm_pub.py

On your VM, you will run vm_pub.py. Edit this file so that the IP address, date and time are published on their own separate subtopic, print some text indicating which date is being published each time.

Edit vm_sub.py

In a separate terminal on your VM, you will run vm_sub.py. Edit this code to subscribe to the subtopics defined on the publisher script with a custom callback that prints the IP address, a different custom callback to print the date, and the default callback to print everything else while indicating the topic. To identify each one of the 3 callbacks use different print messages before the actual data.

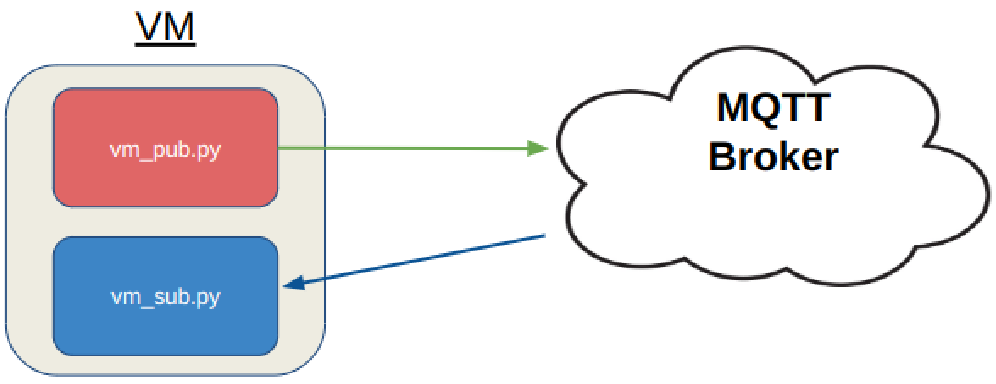


Figure 2: Illustration of Part 1 publish-subscribe structure.

For this lab you will be using a public broker server . More specifically, the brokers we are utilizing are hosted at the following addresses.

Hostname: broker.hivemq.com OR broker.emqx.io
Port: 1883
Topic: Use a unique topic like `your_username/test` to avoid conflicts with other users

7. Part 2: Implement RPI as broker

A. Broker install and configuration

Ssh into your rpi and Install mosquitto:

```
sudo apt install -y mosquitto mosquitto-clients
```

After finishing the installation enable mosquitto service to run at startup:

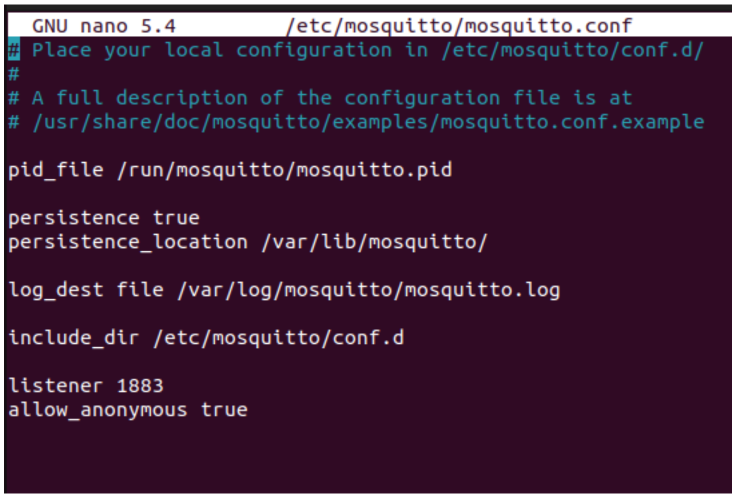
```
sudo systemctl enable mosquitto.service
```

Now you need to configure your mosquitto broker, for that run the following command to edit the conf file:

```
sudo nano /etc/mosquitto/mosquitto.conf
```

To the file add at the end:

listener 1883
allow_anonymous true



```
  GNU nano 5.4          /etc/mosquitto/mosquitto.conf
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

listener 1883
allow_anonymous true
```

Restart the mosquitto service for the changes to take effect:

```
sudo systemctl restart mosquitto
```

Finally take note of rpi IP address with:

```
hostname -I
```

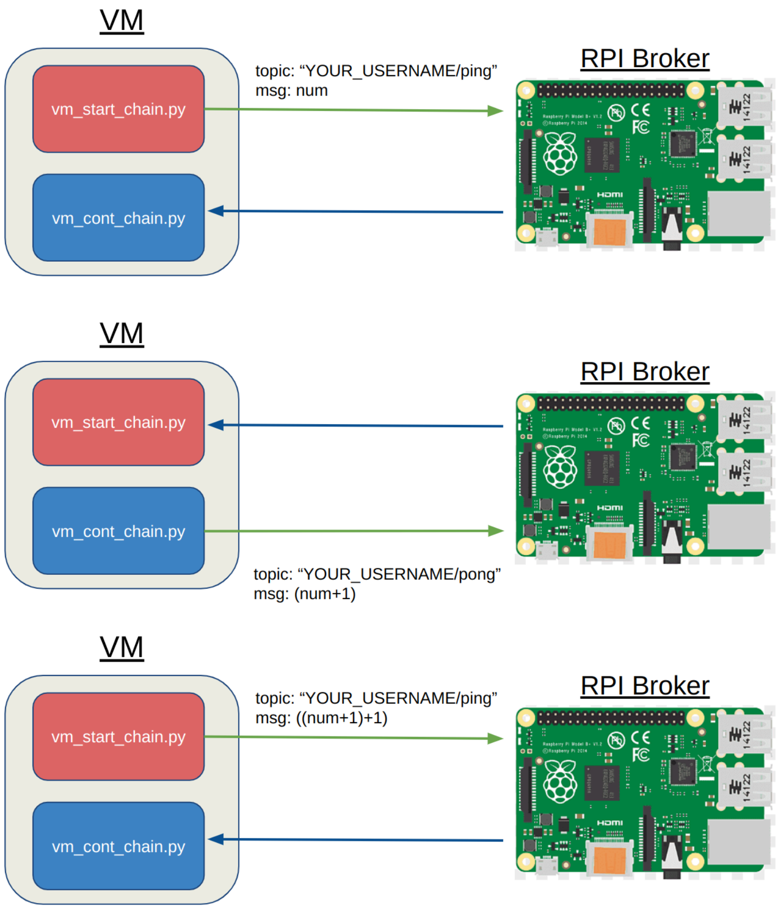
B. Test your Broker

Code a pair of publisher-subscriber python scripts (named **vm_A_chain.py** and **vm_B_chain.py**) able to fulfill the following task:

- Publish to the rpi broker a message from a terminal in your VM with the topic “YOUR_USERNAME/ping” and an integer number as payload.
- Receive your number in a second VM terminal, add one to the number you received, and publish it with the topic “YOUR_USERNAME/pong”
- Receive this message in your first terminal, add one to the number and repeat the cycle indefinitely.
- Be sure to use custom callbacks to handle subscriptions on both files.

Tips:

- Add a time.sleep(1) before publishing to avoid making the update of the number too fast in the terminals.
- Cast the payload of your received message as an integer before adding it.
- Use the following illustration to compliment the task explanation if needed.



8. Demo and Code Grading Rubric

Code files are to be submitted via Github. Include your demo video link in your README.md file.

Points	**Description**
Demo	
4	Demo of Part 1: VM publish and subscribe. ● Show both terminals in the VM. One publishing the IP, date and time through the public broker. While the other one receives and prints the information accordingly.
4	Demo of Part 2: Implement RPI as broker ● Show both terminals in the VM. They should publish and subscribe through your rpi as a broker and show how the chain message updates.
Code	
0	**List all team member names and the link to your github repo in the comments at the top of each file.**
3	**vm_pub.py** Gets IP, date and time and publishes them to the corresponding topic/subtopic.
3	**vm_sub.py** Correctly handles and prints IP, date and time with 3 separate custom callbacks.
6	**(vm_A_chain.py - vm_B_chain.py)** Handles the received number with a custom callback and publishes the same number +1 to the corresponding topic/subtopic.
	Total Possible: 20