Lab: IoT Experiment

In this lab assignment, we shall experiment with a typical-yet-simplified example of performing data analysis on an Internet of Things (IoT) sensor.

Through this lab, you will:

1. Understand the Publish-Subscribe architecture of (MQ Telemetry Transport) MQTT, a network protocol commonly used in IoT networks
2. Using python, attempt to connect to an MQTT broker to receive sensor readings
3. Process the received sensor readings to arrive at a simple conclusion

INTRODUCTION TO MQTT

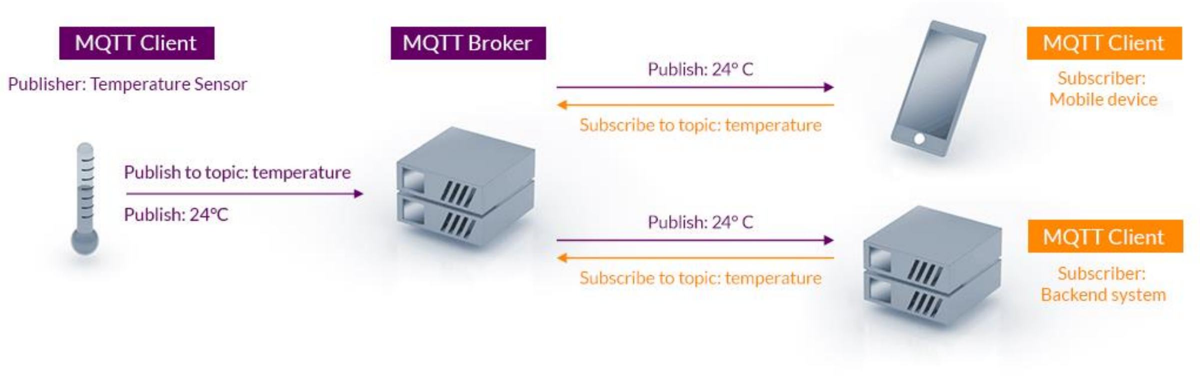


Figure 1 - Overview of a typical MQTT network with MQTT clients (or devices) communicating through an MQTT broker [Image source: mqtt.org]

MQTT, short for MQ Telemetry Transport (of which MQ is short for Message Queue), is a very lightweight and simple protocol for allowing short pieces of information (messages) to be received from and delivered to various devices of varying capabilities.

It is important that the protocol is both lightweight (meaning, data transfers are kept minimal) and simple (meaning, no complicated logic and no large memory is required to run the protocol) because IoT devices typically run on batteries, which limit the amount of processing power and wireless bandwidth available. As a result, you will notice that the features of MQTT that we talk about in the following sections are designed around meeting this ‘lightweight and simple’ goal.

MQTT Broker

At the core of the MQTT system is the MQTT broker (server). All devices (clients) connect to the broker and the broker manages the complexities of routing messages from device to devices. The use of a broker, which is typically a (relatively) powerful server running in a datacentre, allows for the complexities of routing messages and the data bandwidth required to broadcast a message to multiple devices, to be offloaded away from the devices. In this arrangement, each device only knows of and talks to the broker and does not need to be concerned with the existence of other devices.

Publish-Subscribe Architecture

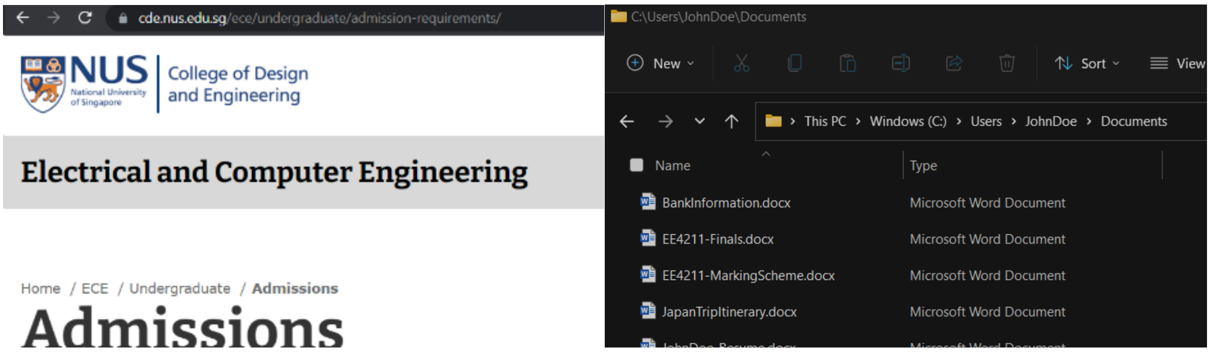
In order for the Broker-Client arrangement to work however, the client needs to be able to send and receive messages in a manner that does not depend on the existence of other devices. MQTT thus uses the publish-subscribe architecture (or pub-sub in short). A device sending data simply publishes, and a device interested in the data simply subscribes. The MQTT broker handles the complexity of replicating and forwarding messages published by devices to devices subscribed to it.

MQTT Topics

To differentiate between different data sources, MQTT uses the concept of 'topics'. A topic is hierarchical and looks like this:

sensors/living_room/temperature

It is very similar to the URL and directory structure most of you should be very familiar with, where different levels of the hierarchy are separated by a forward slash '/' and the highest hierarchical level is on the left:



MQTT topics however support two types of wildcards[1]: the single-level wildcard '+' and the multi-level wildcard '#'.

With the wildcards, this hierarchical organisation allows for very intuitive selection of information a device wishes to *subscribe* to. (A device can only *publish* to a specific topic; thus, the topic to *publish* to cannot contain wildcards).

For example, a device subscribing to the topic '*sensors/living_room/#*' or '*sensors/living_room/+*' will receive readings from all sensors under '*living_room*'. If the device instead subscribes to the topic '*sensors/#*', it will receive readings from all sensors in all rooms.

To receive only temperature readings from all rooms, the device can subscribe to the topic '*sensors/+/temperature*'.

As we have seen above, a positive effect of the use of topics is that they can be designed to describe real-life hierarchies, thus making it more intuitive for humans to understand an MQTT network's organisation.

Quality of Service

A final important feature of note is the Quality of Service (QoS) level. We have already mentioned that MQTT is designed to be lightweight and simple. As such, certain guarantees regarding message delivery can be traded off in order to minimise complexity and data usage. This trade-off is controlled via the QoS level.

MQTT offers three levels of Quality of Service: 0 (at most once), 1 (at least once) and 2 (exactly once). The differences are subtle but significant[2].

[1] A wildcard in this context describes a placeholder that is interpreted with the meaning 'anything'. If you are still unsure of what this means, read ahead a little more, examples will be provided.

[2] If you are interested in understanding the details of how the QoS levels are achieved and when to use which level, the following link provides clear explanations with illustrations: https://web.archive.org/web/20220806150250/https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/

- Level 0 does not guarantee delivery (basically send once and move on) but uses the least data and is least complex
- Level 1 guarantees delivery but duplicates can occur (basically retries sending until an acknowledgement, which can be lost, is received) and thus uses significantly more data than level 0 with some additional complexity tracking retries and acknowledgement
- Level 2 guarantees delivery and no duplicates by an additional confirmation cycle which handles the duplication that occurs due to a lost acknowledgement, and is the most data and resource intensive QoS level

Summary

With an understanding of the important features of MQTT, we can think of an MQTT network as something similar to the magazine publishing industry: The magazine publisher is the MQTT broker; magazine writers are like the sensors on an MQTT network, they *publish* their works through the publisher under certain magazine titles (i.e. MQTT topics); magazine readers *subscribe* to certain magazine titles with the publisher and receive them whenever a new issue is published. In terms of QoS, both the magazine writer and reader can choose different postal services to send and receive their magazine with higher cost for higher reliability delivery.

Q1 – This pub-sub architecture is actually very prevalent in day-to-day activities, notably in social media and media consumption. Fill in the table below with one social media (e.g. Instagram, WeChat, Twitter, Weibo) and one media (e.g. YouTube, Bilibili) platform as parallels to the pub-sub architecture. The magazine publisher analogy given in the summary is filled in as an example.

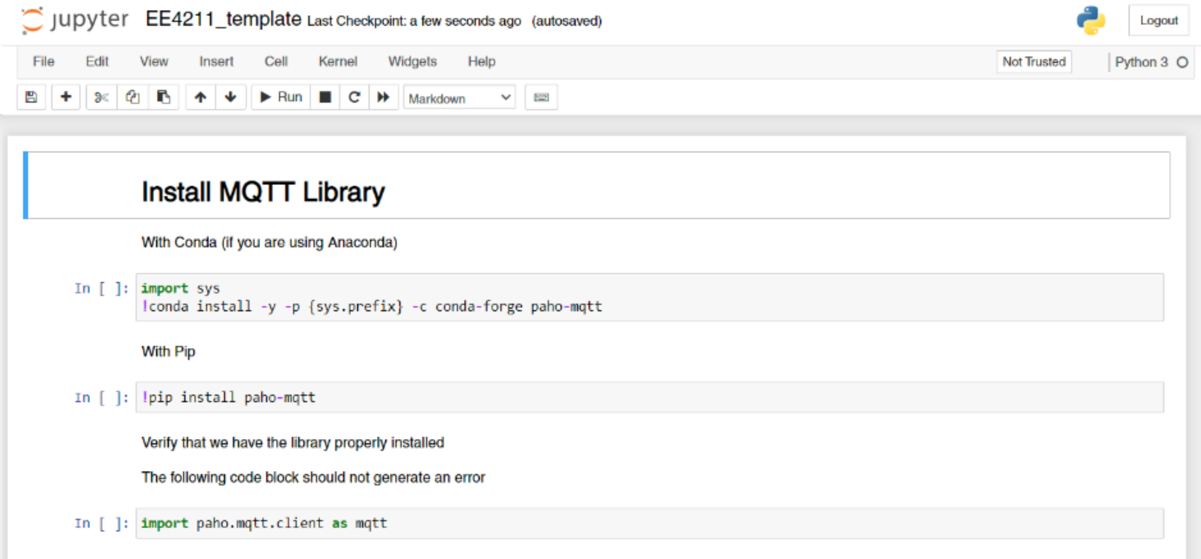
	Broker	**Publish Action**	**Subscribe Action**
Example **Magazine Publishing**	Magazine Publisher	Magazine writer sends magazine to publisher	Magazine reader subscribes to magazine through publisher
Social Media Platform			
Media Platform			

IOT EXPERIMENT

In this experiment, we shall log live sensor data published through an MQTT broker and run a simple analysis on the logged data to arrive at a simple conclusion.

Note: The MQTT broker can only be accessed from within the NUS network. You will have to be on campus via Wi-Fi or nVPN[3] from home in order to access it. <u>The experiment will require about an hour of waiting in order for sufficient data to be logged.</u>

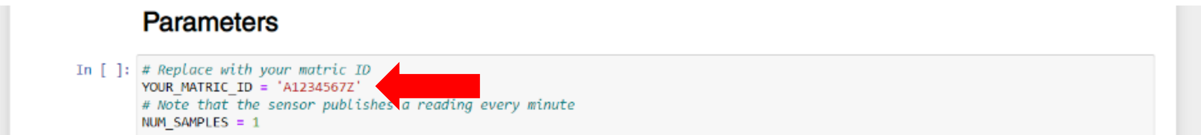
1. Open the EE4211_template.ipynb notebook file in Jupyter. You should see a notebook that looks like the following image:



2. In this experiment, we will be using the Paho MQTT library for python. Run the first three cells to install the library if it is not already installed. The third cell should not generate an error if the library has been installed properly. If you encounter an error, do contact your tutor for assistance.

3. In the next cell, fill in your matriculation ID including the last letter. This is used to authenticate with the MQTT broker and to decide which sensor you will be receiving data from. **<u>Note that each student will be allocated with a different sensor with different behaviour, which will generate a different final result.</u>**

The 'NUM_SAMPLES' parameter will determine how many sample readings we will log before stopping. We start with 1 to test that everything is working before increasing it to a suitable value.



[3] Read more about nVPN here: https://nusit.nus.edu.sg/services/wifi_internet/nvpn/

4. Run the rest of the notebook. If you are on the NUS network and your matriculation ID is correctly entered, you should see a single reading being logged. If not, double check that you have followed the above instructions correctly and contact your tutor for assistance.

```
print(MQTT_TOPIC)
```

```
Subscribed to topic: ee4211/xxxxxxxxxxxxxxxx/light_sensor_0/brightness_lux
```

Wait for the required number of readings to accumulate and disconnect.

If you do not see any readings being collected within 5 minutes, something is wrong. Check that you have your matric id filled in correctly.

```
In [9]: from time import sleep
        while len(readings) < NUM_SAMPLES:
            sleep(1.0)
        client.disconnect()
        client.loop_stop()
        print('Number of readings obtained:', end=' ')
        print(len(readings))
```

```
Got reading: {'timestamp': '2022-08-06T00:00:00.000000', 'brightness_lux': 99999}
Number of readings obtained: 1
```

Dump the readings to a JSON file for offline processing

```
In [10]: import json
         with open('readings.json', 'w') as f:
             json.dump(readings, f)
```

Data Processing

Write your data processing functions to determine the **distribution** of the readings and its **parameters**.

Your normal data visualisation and processing libraries such as seaborn, pandas and numpy are available.

```
In [11]: import seaborn as sns
         import pandas as pd
         import numpy as np

         import json

         # Sample that just imports from json and prints all the readings
         with open('readings.json', 'r') as f:
             readings = json.load(f)

         for reading in readings:
             print('Timestamp:', end=' ')
             print(reading['timestamp'], end=' ')
             print('Brightness (lux):', end=' ')
             print(reading['brightness_lux'])
```

```
Timestamp: 2022-08-06T00:00:00.000000 Brightness (lux): 99999
```

5. After you have verified that the notebook is working, change the ‘NUM_SAMPLES’ parameter to a suitable value (of your determination) and re-run the notebook to begin data collection. If no error occurs, the logged data should be saved in a file named ‘readings.json’ in the same directory as where the Jupyter notebook is located.

6. Modify the last cell in the notebook to process the logged data to answer the following questions.

Q2a – Plot the data you have collected. What statistical distribution best fits the collected data?

Q2b – What are the 4 parameters of the statistical distribution that best fits the collected data? Please provide substantiation on how the chosen distribution and its parameters fit the data well.

(Knowing how to substantiate your conclusions is part of the evaluation for this assignment, the teaching staff will not be able to tell you what constitutes as sufficient substantiation.)

CONCLUSION

In this lab, you have learnt about the basics of the MQTT protocol, namely: 1) its design goals, 2) the pub-sub architecture, 3) topics and 4) quality of service levels. You also used code that logs and analyses sensor data through a MQTT broker in a simplified example of data analysis on an IoT sensor.