4 Milestone IV – communicate sensor data (MQTT)

In an Internet of Things context, a critical component is the communication layer: it connects devices (or "things") with each other, and with the any other system. For this, a messaging protocol is required. In the IoT world, the messaging protocol called MQTT is often used.

> It is important to note that there are *many* different, or sometimes rather similar, messaging or communication protocols out there, each with their own optimal use cases and applications.

The concept of MQTT includes the following main "components", which is moreover illustrate in Figure 4.1.

- It follows a publish-subscribe pattern
- All message pass through a message **broker** (server)
- **Topics** can be created on the MQTT broker that allows MQTT clients to share information on a specific subject (~topic), e.g. "temperature"
- Clients can **publish** (sensor) data to a topic
- Clients can **subscribe** to a topic, in order to receive the published data on that topic

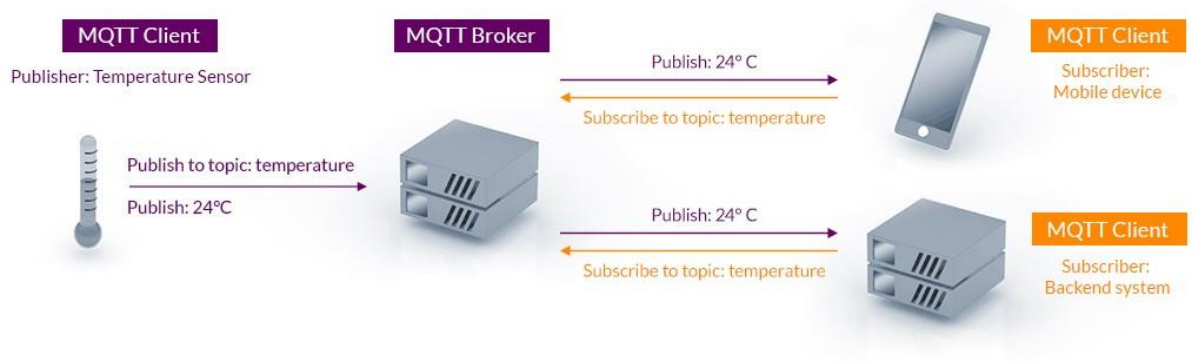

Figure 4.1: *MQTT messaging workflow*

4.1 MQTT broker

A MQTT broker is already set up (on a KU Leuven server); so, you do *not* need to worry about that. In order to use this broker, however, a client *does* need to know the broker's credentials and IP address in order to *publish* and *subscribe* to corresponding topics. All relevant information is provided in Table 4.1.

> The broker can be used both *with* and *without* message encryption (TLS/SSL encryption protocol). It is **strongly advised to implement the secure method** in all your **scripts**, i.e., *with* TLS encryption. For testing purposes however, possibly with a command line tool, the nonencrypted option is fine, and can be very useful.

NO TLS
username: ee2-all **password** : ee2-all **host:** 192.168.1.127 **port:** 8884
TLS
username: ee2-all **password:** ee2-all **host:** set-p-gt-01-mqtt.bm.icts.kuleuven.be **port:** 1883

Table 4.1: *MQTT credentials*

4.2 Tools and libraries

There is plenty of third-party software available that supports MQTT, both commercial and free. One of the open source mqtt brokers is test.mosquitto.org. In this course, we will mostly use command line tools for testing, and Python libraries for scripting.

4.2.1 Command line tools for testing

The free and open-source MQTT framework mosquitto provides command line tools for publishing and subscribing to MQTT topics, i.e.:

- **mosquitto_pub**
- **mosquitto_sub**

You need to install *mosquito-clients* to execute these commands

4.2.2 MQTT Explorer

This opensource application is a MQTT client to visualize, publish, subscribe and plot topics. Demonstrate these visualizations in milestone video.

These tools can come in handy for testing data flow between devices. These can be installed on a local PC, and/or on the Raspberry Pi, or on any other device.

- https://testclient-cloud.mqtt.cool/
- http://www.steves-internet-guide.com/mqtt-tools/

Some examples on how to use these tools , *without* **SSL, in terminal 1**:

```
$ mosquitto_sub -h 192.168.1.127 -t test -u "ee2-all" -P "ee2-all" -p 8884
```

and **in terminal 2**:

```
$ mosquitto_pub -h 192.168.1.127 -t test -u "ee2-all" -P "ee2-all" -p 8884 -m \
"Hey, how are you?"
```

Similarly, *with* **SSL, in terminal 1** :

```
$ mosquitto_sub -h set-p-gt-01-mqtt.bm.icts.kuleuven.be  -p 1883 --cafile \
"/path/ca.crt" -t test -u "ee2-all" -P "ee2-all"
```

and **in terminal 2:**

```
$ mosquitto_pub -h set-p-gt-01-mqtt.bm.icts.kuleuven.be  -p 1883 --cafile \
"/path/ca.crt" -t test -u "ee2-all" -P "ee2-all" -m "Hey, how are you?"
```

Download the SSL certificate "ca.crt" from Toledo(MS4). When your mosquitto executables are not available system-wide, you need to navigate to the mosquitto directory (on your PC), and then open a terminal/CLI in order to execute `mosquitto_sub` and `mosquitto_pub` commands. Or, you can put the `PATH` before the command, e.g., on a Windows system, `C:\some\path\to\mosquitto sub [...]`.

4.2.3 Client libraries

The Paho MQTT client library is a library with bindings for several programming languages, including Python. It can easily be installed using PIP, *both* **on your local machine** and/or **on the Pi**, i.e.:

```
$ python3 -m pip install paho-mqtt
```

Note that depending on your editor of choice, you might be able to install python packages with PIP through the IDE, for example, with the Thonny editor.

Now you should be able to import this library in a Python script, on any machine, on any device, as long as library is installed.

4.2.4 Client scripts for publishing and subscribing to topics

After installing the Paho library, you can start by creating a **script to publish and subscribe messages to MQTT topics**. Similar to the aforementioned *Mosquitto tools*, this can also be done with or without TLS; however, since these scripts will be a stepping stone towards more advanced application or sensor

integrations, it is strongly advised to **use the secure TLS protocol** here. In terms of implementation, it is best to approach this step by step, for example:

1. Create one or more script(s) **on your local machine** (e.g. your laptop), and publish some **random data** to a **random topic** (no sensor readings or Raspberry Pi needed here)
2. If this works, copy the script where you publish data to a topic **to the Raspberry Pi** (or simply create a new one on the Pi), so you can test the following communication layer:
   - Client script 1 (Raspberry Pi): publishing random data to MQTT topic
   - Client script 2 (Laptop/PC): subscribing to that MQTT topic

In order to connect to the encrypted broker (with TLS/SSL) using the Paho library, one needs to tell the library to set the TLS context, e.g., with:

```
client = mqtt.Client("MyClient-01")
client.tls_set()
# rest of your code ...
```

- https://docs.faircom.com/docs/en/UUID-cb67a0f7-170b-352a-39eb-3307a423b7d4.html
- https://github.com/eclipse/paho.mqtt.python
- http://www.steves-internet-guide.com/publishing-messages-mqtt-client
- http://www.steves-internet-guide.com/subscribing-topics-mqtt-client
- http://www.steves-internet-guide.com/python-mqtt-publish-subscribe
- http://www.steves-internet-guide.com/receiving-messages-mqtt-python-client

- https://stackoverflow.com/a/62480957
- http://www.steves-internet-guide.com/mqtt-python-callbacks

4.2.5 Publish sensor data to a MQTT topic

After successfully implementing previous client scripts, you can now create a **publish-subscribe application** where you publish sensor data (from the Pi) to a MQTT topic. As such, you have to **integrate** the code for reading out a sensor into the publisher client script on the Pi (in one script), and communicate *that* sensor data (e.g. in a one line string) to another device, such as your laptop/PC (that runs a subscriber script). While, or after, receiving the data on your local machine you can do anything you want with it (post process, write to CSV, aggregate, visualise, ...).

4.3 The JSON format

Previously, you have been communicating unstructured (text) strings over a topic, however, this is **not very efficient**, for instance, when you want to share your data with third-party apps and external devices. These "endpoints" (or users) do not know how to interpret or decode your message, and will most likely fail to make any sense of it. Instead, we want to use a **pre-defined data format** such as **JSON**. Such language-independent, standardized data formats (sometimes also called data schemes) introduce a set of "rules" to structure this interchanged data.

An example is:

```
{
  "sensors" : ["thermometer", "light sensor"],
  "timestamp": 145164960051,
  "values": {
    "temperature": 20,
    "lightIntensity": 90
  }
}
```

There are many, many data formats available, including both human-readable and binary formats. As usual, each data format has its pros and cons. JSON is a human-readable one, and is widely adopted for MQTT messages, so, that is why it is used in this course.

4.3.1 Communicate JSON data from the Pi to your PC

Modify your scripts from Section 4.2.4 in order to **publish** JSON data strings (instead of unstructured text). Python has a built-in JSON library for working with JSON data (parse/serialize from/to python variables).

- http://www.steves-internet-guide.com/send-json-data-mqtt-python
- https://www.w3schools.com/python/python_json.asp
- https://pythonexamples.org/python-dict-to-json

4.4 Extend your smart home

Now that you have experimented with a useful range of software, tools, libraries, among other things, you can appeal to your creativity. There are **much more sensors available**, and other teams probably have **other data available** that might be interesting for you. So, feel free to **extend your smart home**. Here area couple ideas to give you some inspiration:

Validation through multiple sensors

A surveillance system can be a **stepwise process**, for instance:

1. A motion sensors detects movement, and this first movement triggers a LED light.
2. A second movement within a timeframe of, say 30 seconds, automatically sends you an email that "there might be an intruder".
3. An ultrasonic sensor reading in the same area (e.g. door/window/driveway) as where themotion sensor is pointed to confirms this movement via a "change in distance" to the object. This, combined with (1), triggers an alarm (buzzer).

Other sensor combinations can of course be used here.

Share data – interact with other groups

Say your friends live in the same city (or village) as you, and they are equipped with humidity and temperature sensors while you are not. This data might be useful to you as well; so, why not create a topic and share data between groups? Generally speaking, creating topics and discussing a message schema for **sharing information between groups** is a very useful **interoperability exercise**.

- https://matplotlib.org/stable/tutorials/index.html
- https://pythongeeks.org/send-email-using-python/
- https://youtu.be/GIywmJbGH-8
- https://matplotlib.org/

Custom (realtime) visualisation

Visualize your data programmatically, using the Matplotlib library, preferably in real time.