Machine Learning for IoT
Lab 6 – MQTT

Exercise 1: Mosquitto

Become confident with the Mosquitto clients:

- Use *mosquitto_pub* to publish json messages with a certain topic
- Use different (more than one) *mosquitto_sub* to receive the messages published by *mosquitto_pub*. Play with the wildcards ('+' and '#') when you subscribe for the topic.

You can use the message broker provided by eclipse (mqtt.eclipseprojects.io at port 1883).

SUGGESTION: Use the mosquitto clients (both *mosquitto_pub* and *mosquitto_sub*) to debug the communication among your applications.

Exercise 2: MQTT Clients in Python

2.1. Develop an MQTT publisher to send:

- every 5 seconds the date and time following the format dd-mm-yyyy hh:mm:ss.
- every 10 seconds the POSIX timestamp.

The two messages must be sent in JSON with two different topics.

2.2. Develop a **first** MQTT subscriber to receive only the messages about date and time and print the information in a user-friendly format (not the full JSON).

2.3. Develop a **second** MQTT subscriber to receive only the messages about unix timestamp and print the information in a user-friendly format (not the full JSON).

You can use the message broker provided by eclipse (mqtt.eclipseprojects.io at port 1883).

SUGGESTION: Define topics in a hierarchical way, including a unique identifier.

Exercise 3: MQTT Data Collection

3.1. On your board, develop a **first** MQTT client, that works as publisher and subscriber, to:

- publish information about the temperature sensor every 10 seconds
- publish information about the humidity sensor every 20 seconds
- receive, as subscriber, commands to record an audio signal
- publish the recorded audio signal

3.2. On your notebook, develop a **second** MQTT client to subscribe, receive, and print all the information about temperature and humidity.

3.3. Develop a **third** MQTT client that publishes a command to record an audio signal, receive the recorded signal, and store it on disk.

Use the SenML+JSON format to exchange the information among the MQTT clients.
You can use the message broker provided by eclipse (mqtt.eclipseprojects.io at port 1883).

Exercise 4: MQTT Inference

4.1. Develop an MQTT publisher that every minute send six temperature and humidity samples (use the SenML+JSON format).

4.2. Develop an MQTT subscriber that receive the six temperature and humidity values, predicts the future temperature and humidity values (with a model from Lab3), and prints the prediction in a human-friendly format.

Exercise 5: IoT Catalog

5.1. Develop a RESTful style Catalog of a distributed platform for general purpose services. Identify the most suitable HTTP methods (among GET, POST, PUT and DELETE) and develop the web services to:

- Retrieve information about IP address and port of the message broker in the platform.
- Add a new device with the following information:
  - unique device ID
  - end-points (i.e. Rest Web Services and/or MQTT topics).
  - available resources (e.g., Temperature, Humidity, and Microphone).
  - "insert-timestamp" when this device was added.
    (**SUGGESTION:** to avoid synchronization issues, this attribute is managed and updated only by the Catalog according to its system clock)
- Retrieve all the registered devices.
- Retrieve a specific device with a device ID.
- Register a new user with the following information:
  - unique user ID
  - name
  - surname
- Retrieve all the registered users.
- Retrieve a specific user with a certain user ID

This information is stored in a JSON file and all the information among the actors in the platform must be exchanged in JSON

Implement an additional feature of the Catalog to remove all the devices with "insert-timestamp" higher than two minutes. The Catalog must take this action periodically (e.g., every 1 minute).

5.2. Develop a **client** Python application for invoking the RESTful Catalog developed in *5.1*. This application must retrieve information about:

- the message broker.
- all the registered devices.
- device with a specific device ID given as input.
- all the registered users.
- device with a specific user ID given as input.

5.3. Develop a **client** Python application, that emulates an IoT device, to invoke the RESTful Catalog developed in *5.1*. This application must periodically (e.g., every 1 minute) either register a new device or refresh the old registration by updating its “insert-timestamp”. During the refresh of an old device registration, the Catalog must update also the “insert-timestamp”.

5.4. Extend the functionalities of the Catalog developed in 5.*1* to work as MQTT subscriber either to register a new device or to refresh the old registration by updating its “insert-timestamp”. The Catalog must subscribe to a specific topic used for this purpose only.

5.5. Develop a Python MQTT publisher, that emulates an IoT device, to periodically (e.g., every minute) either register a new device or refresh the old registration in the Catalog developed in *5.2.* During the refresh of an old device registration, the Catalog must update also the “insert-timestamp”.

SUGGESTION: Use the *threading* Python package to implement the synchronization between the different components of the Catalog.