Machine Learning for IoT
Homework III

Lorenzo Melchionna
Politecnico di Torino
Data Science and Engineering
s304651@studenti.polito.it

Edoardo Marchetti
Politecnico di Torino
Data Science and Engineering
s303873@studenti.polito.it

Edgar Gaytán
Politecnico di Torino
Data Science and Engineering
s300164@studenti.polito.it

***Abstract*—In this report, we describe the design of an MQTT publisher that runs locally and sends messages in JSON format about the battery status of the device. This data will later be read and stored in REDIS time series by an MQTT subscriber running on the cloud. Still working with this data we also designed a REST server that exposes it through several endpoints, making use of query and path parameters of the HTTP protocol. Finally, we discuss the design of a REST client capable of plotting and updating these records.**

1. Data Collection, Communication and Storage

This section describes the design of the MQTT publisher that every second sends the mac address, timestamp, battery level and whether or not the power is plugged for the device running the script. We also discuss the design of the of the script that subscribes to the publisher's topic and receives its data. The choice of MQTT as the protocol over REST can be justified for an easy and simple reason: The publisher script is supposed to be run on different devices at the same time, so we are dealing with a one-to-many communication. Using a publish-subscribe system such as MQTT simultaneous communication with many devices becomes easier to manage because the subscriber script is able to receive messages from all publishers as soon as the MQTT broker receives them without creating a stable connection with all devices as would happen using REST. This allows having a smaller message header, which renders MQTT preferable over REST for projects that have battery and power usage constraints. However this also can mean drawbacks in terms of security, but since we are not communicating sensible data the use of this protocol is the best choice.

1.1. Data Collection and Communication with the MQTT protocol

The publisher first needs to connect with a message broker and to a specific port and topic. For this script we will be using the eclipse broker since is the most accessible off-the-shelf broker to implement. After this, we collect the data of interest from the device and pack it as a Python dictionary in order to publish it in JSON format under a user-defined topic, i.e. id number of one of the team members, which will need to be the same for the subscriber to 'listen' to this publisher. All of this is carried out inside an infinite loop and published with an interval of one second.

1.2. Data Storage

By defining the MQTT client for the subscriber we need to implement the methods on_connect and on_message, to tell the script what to do in case of a failed or successful connection and, after this, what to do with the incoming messages. Here the on_message method checks if the REDIS time series exists and appends the messages to it or creates a new one otherwise. It is very important to use the same port and topic used by the publisher since these are the only keys that guarantee a successful connection.

2. Data Management & Visualization

2.1. REST Server

We built an API following the given documentation inside the notebook *rest_server.ipynb*. In particular, we implemented a Devices class, corresponding to the */devices* endpoint, in which we defined the **GET** method to obtain the list of available *mac_addresses* within Redis. After that, a new Device class was defined to represent the */devices/mac_address* endpoint. Within it, a **GET** function, to retrieve battery status information of the device using the specified MAC address in the specified time range, and a **DELETE** function, to delete the time series associated with the specified MAC address. In the first two cases, we choose GET since we need only to retrieve the data without modifying anything, while DELETE function is used since it is the only one that allows the remotion of a resource. Finally, the cherrypy library and its configuration methods were used to launch the server.

2.2. REST Client for Data Visualization

In this last part, we have created a Python notebook which uses separate cells to invoke the three functions

Method	Endpoint	Description
GET	**/devices**	Retrieve the list of MAC addresses of the monitored devices.
GET	**/device/{mac_address}**	Retrieve battery status information of the device with the specified MAC address in the specified time range.
DELETE	**/device/{mac_address}**	Delete the time series associated to the specified MAC address.

TABLE 1. DESCRIPTION OF THE METHODS USED FOR THE REST CLIENT

mentioned in the previous section. Before returning the output to the user, we check that the response code of the request is the desired one, i.e. 200, otherwise the text of of the response object is printed.