**Water Heater Detection**

**Basic Description**

In this section we will study the implementation of the version of the code written for disaggregation but aiming to disaggregate water heaters. The basic idea won’t be different. What that means is that there will be a section collecting data, a section where the rules are applied and a section where the final form of the data of detected events is stored.

**Scripts developed and deployed**

* getEvents.py ***[1]***
* uploadToTB2.py ***[2]***
* delOverlapping.py ***[3]***

Each of these python scripts is accompanied by a bash executable tasked to call the corresponding python script.

In the following section we will discuss the role each of these files plays.

**Each script**

1. getEvents.py: This script utilizes the preexisting functions (from fridge detection) to collect data from the entirety of the previous day, plus one hour from the day before and the day after. When all that data is collected, the following rules are applied to the dataset:
2. An event starts when a measurement has taken place more than three hundred seconds after the previous one and the second next one has taken place earlier than that.
3. An event continues while measurements are frequent.
4. If two events are 5 minutes apart, they are considered one and are joined.
5. Events shorter than 2 minutes are disregarded since they are most likely random peaks.
6. Events with pwrA fluctuating with mean deviation larger than 1000W are also deleted from the dataset since a water heater’s signal should generally be stable.
7. Events that average a pwrA smaller than 2000W are ignored as well.
8. Finally, any events that remain and have occurred close to each other are merged.

Now, it is important to examine if it is the first time the script runs. If that is the case, then a json file is saved in the same folder that contains the following information: a) The mean value of every event’s pwrA plus 10% and b) The mean value of every event’s pwrA minus 10%. That data will be playing the role of floor and ceiling for the events detected from all subsequent executions of the script. Any events detected with an average that exceed the ceiling or is lower than the floor, are deleted. Finally, the script stores the events it produced in a json file, saving **1)** average\_value ,**2)** start unix timestamp and **3)** end unix timestamp.

1. uploadToTB2.py: This script will check if there was an event for the current day. If there is, then it uploads: 1) The two timestamps corresponding to the start and the end of the event, each with a copy of the event’s average pwrA and two timestamps -10ms and +10ms respectively, accompanied by a 0 value, creating a square pulse. 2) An estimate of the total consumption of the event multiplying the duration of the event by its average pwrA.
2. delOverlapping.py: Finally, this script merges the json files produced from the execution into one which contains events that have been uploaded and so should be ignored by the uploadToTB2.py script.

**How it all comes together**

In the Linux VM, where the codebase is hosted, there are three cronjobs that run every day, one minute apart in this order: 1) wh\_exec calling ***[1]***, 2) wh\_upload\_exec calling ***[2]*** and finally 3) wh\_cleanup\_exec calling ***[3]***. The logic that’s followed goes as described below:

When the code runs for the getEvents.py script runs for the first time, then it creates two json files. One containing the values range that will dictate which events will be considered valid in future executions and one that contains the events detected. All subsequent executions will only produce the second one mentioned. After the execution of getEvents.py it is uploadToTB2.py’s turn to execute. It loops through the files inside the current folder and selects solely json files that cover a range smaller than two days. That way the json file that has been produced by all the executions so far will not be taken into account, since its data have already been uploaded. Then, the data are processed and uploaded on thingsboard as described in the previous section. Finally, the delOverlapping.py merges the json files together in order to exclude them from future executions. Any errors that might arise during the methodology described above will be written in a text file named log\_wh.txt.