Department of Information Engineering
Artificial Intelligence and Data Engineering
Internet of Things course

Smart Data Center Maintenance

E. Ruffoli, T. Baldi

Academic Year 2021/2022

Contents

1 Introduction

It is essential that a Data Center has an automatic regulation system that controls the humidity and temperature levels in the air in order to maintain the precise environmental conditions for the safekeeping of the server: if the temperature is too high, the server components will not be able to function properly and in the worst case, the extreme temperature will damage the processor; a high humidity level can form water that puts server hardware and other equipment at risk.

Data Centers also have different technical fire protection measures, like oxygen reduction system: the air inside a Data Center has a lower percentage of oxygen in order to prevent the starting of fires. Such an environment, can be dangerous for humans especially for those that have to pass hours inside it, like technicians that have to perform a repair.

The goal of this project is to design and implement an IoT solution for autonoumous managing a *Data Center* environment and making server maintenance safer for the technicians.

In particular, the application must independently manage and regulate the temperature, the humidity and the oxygen values inside the data center. It also must provide support for *Smart Health Bands*: lightweight and portable IoT devices that have multiple sensors capable of capturing vital signs values, like blood pressure and heartbeat, of the person wearing them. By employing these devices, the system can continuosly keep track of the health conditions of the technicians and alert them if a critical situation occurs.

The document is organised as follows: Section 2 discusses the design of the IoT system and its components, with particular attention to the implementation of the MQTT and CoAP network, the data encoding techinques employed, the collector and the database; Section 3 reports the testing results obtained reproducing the behaviour of vital signs monitors in a simulated environment (Cooja) and in a real environment (testbed).

The entire code of this project is available in the following repository:

`https://github.com/balditommaso/Smart-DC-Maintenance`

2 Design

2.1 Overview

The Smart Data Center Maintenance system is composed by two different network of IoT devices, one network of devices that use MQTT to report data, the other uses CoAP as the application protocol. The system also includes a collector, that must be be installed in a Data Center, a SQL database, that can be installed on premises or on a cloud, and a Grafana web interface to view the data.

The device networks are LLNs (Low Power and Lossy Networks) that exploit the 802.15.4 and IPv6 protocols, where the multihop communication is implemented by RPL protocol and the traffic with the collector is enabled by a dedicated border router, one for each Data Center on which the system is deployed.
There are two types of IoT devices, both equipped with the Contiki-NG operating system, that periodically send data to the collector and receive commands and alerts from it:

- *Environment Sensors*: they are in charge of sense envirnomental values such as the **temperature**, the **oxygen** level and the **humidity** of the Data Center; they also are supplied with a simple alert system that reproduces a blinking on the set of LEDs; they use CoAP as protocol to communicate with the collector.

- *Smart Health Bands*: they are supplied with multiple sensors that sense the **body temperature**, the **oxygen saturation**, the **respiration**, the **blood pressure** and the **heart rate** of the wearer; they also have an alarm system that is able to reproduce a warning by activate a blinking the set of LEDs that it has, so that a life-threatening situation can be signalled visually; they use MQTT as application protocol to exchange data with the collector.

As far as the collector concerns, it is responsible for receiving data from both the *Environment Sensors* and the *Smart Health Bands*. It is also in charge of implementing the control logic in order to enable/disable the actuators and activating the alert on the devices; more in detail:

- if the collector receives a sample from one of the *Environmental Sensors* that it is not within the pre-established thresholds for that quantity, it turns on the alarm system and (if present) turns on the related actuator in order to re establish the standard values.

- if the collector detects a worrying trend of one of the vital signs values sensed by a band, it activates the alarm on the related band, in order to warn the technician so that he can leave the data center as soon as possibile.

The collector also stores all the data into a MySQL database for historical collection and to be viewed by the Grafana web interface. The collector is thought to be executed on a computer or a server on premises, to guarantee low latency responses, especially for the *Smart Health Bands* alerts. The overall system structure is represented in Figure 1.

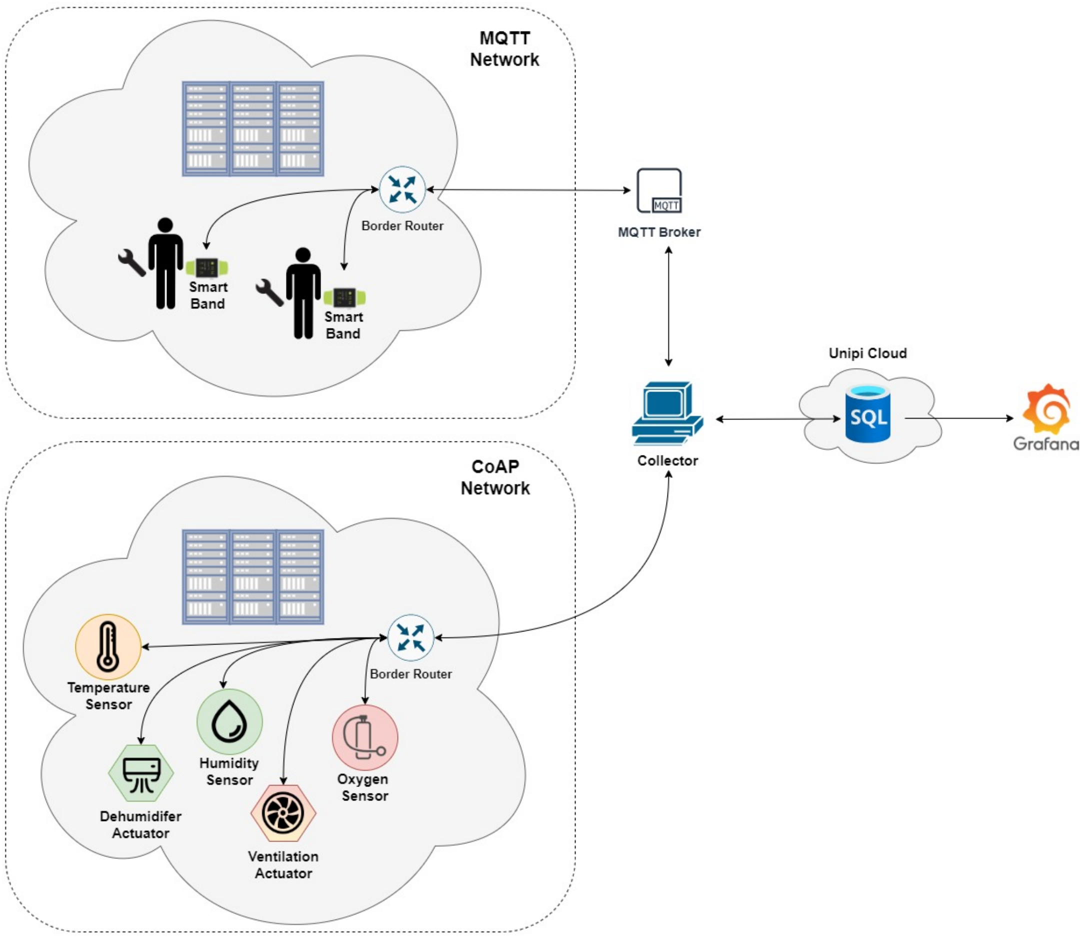


Figure 1: High level architecture of the IoT system.

2.2 CoAP Network

Each device in the CoAP Network register itself to the Collector Server, thus allowing the server to take trace of which devices are active and what kind of signals they are sampling. Once registered, the server creates a dedicated process, for each device, responsible for observing the data and uploading it to the database. Processes control that the data collected by the sensors are within the thresholds, otherwise they have to set the larm status and send to the relative actuator the command to activate itself until the values do not return within the limits. This closed-loop control logic is foundamental in order to avoid dangerous situation for Data Center devices and workers.

2.2.1 Oxygen

The level of oxygen inside the Data Center is lower than the normal in order to avoid wildfires, but it shouldn't goes under a certain limit, beacuse it could be really dangerous for the health of workers. That is why we equipped the racks with devices able to monitor this signal and control an actuator in case of necessity.

The device equipped with the oxygen sensor maps its resource at the following address: **coap://<BAND_ID>:5683/oxygen**. It is possible to issue GET request to receive the current level of oxygen, it also sends notification to all the CoAP observers whenever a change is detected. The device is also equipped with an actuator in order to regulate the level of oxygen in the enviroment which can be triggered by a PUT request at the following address: **coap://<BAND_ID>:5683/ventilation**. The payload of the request must contains "ON" or "OFF", the status of the actuator is indicated by the flashing red LED.

2.2.2 Temperature

The temperature at which a server works greatly affects its performance, for this reason it is essential to monitor that it is always kept at optimal levels. In addition, any overheating can lead to breakdowns and therefore to the interruption of service.

The device equipped with the temperature sensor maps its resource at the following address: **coap://<BAND_ID>:5683/temperature**. It is possible to issue GET request to receive the current temperature, it also sends notification to all the CoAP observers whenever a change is detected. The device is also equipped with an actuator in order to regulate the temperature in the enviroment which can be triggered by a PUT request at the following address: **coap://[BAND_ID]:5683/ventilation**. The payload of the request must contains "ON" or "OFF", the status of the actuator is indicated by the flashing red LED.

2.2.3 Humidity

The humidity of the Data Center is monitored in order to avoid malfunctioning of the servers.

The device equipped with the humidity sensor maps its resource at the following address: **coap://[BAND_ID]:5683/humdity**. It is possible to issue GET request to receive the current percentage of humidity in the environment, it also sends notification to all the CoAP observers whenever a change is detected. The device is also equipped with an actuator in order to regulate the percentage of humidity in the enviroment which can be triggered by a PUT request at the following address: **coap://<BAND_ID>:5683/ventilation**. The payload of the request must contains "ON" or "OFF", the status of the actuator is indicated by the flashing red LED.

2.3 MQTT Newtork

2.3.1 Smart Health Band Operation

A *Smart Health Band* connects as a client to an MQTT broker publishing and subscribing to dedicated topics to communicate with the collector. Each *Smart Health Band* is characterised by its global IPv6 address, that is used as ID. Each band has associated a charger that is thought to be placed outside the data center. The bands will be unplugged from the charger from the technicians that are about to enter the data center, and they will be plugged back when the technicians will exit. More in details, a *Smart Health Band* operates in the following way:

1. when a new band is introduced in the system it has to be attached to its charger in order to be initialized. Once plugged, it waits until it is connected to the network, namely until it obtains a global IPv6 address and a default route;
2. once connected to the network, the band tries to connect to the MQTT broker. If the connection is established, the band subscribes to the alarm topic that concerns it.
3. if the subscription succesfully occured, the band publishes a registration message on the related topic. The registration will be received by the collector, that will instantiate all the resources dedicated to the band and will insert it in the database. The band will be in non-active mode.
4. during the **non-active mode**, the band turns on the red led and starts recharging its battery. When the band is unplugged from the charger it passes to the active mode.
5. during the **active mode**, the band turns on the green led and periodically acquires health information by means of its sensors. The samples are then published in the topic reserved to the band. If the band is attached to the charger, it returns to the non-active mode. If the band battery falls below a minimum threshold, the band goes in the battery-low state.
6. when the band has low battery, it stops sending the periodical health updates and turns on both the red and green leds. If the band is attached to the charger, it will pass in non-active mode.
7. if the band receives an alert message from the alarm topic that concerns it, it starts a led blinking in order to warn the wearer of the worrying state of health and stops sending periodic updates. When the wearer exits the Data Center and attaches the band to its charger, it goes in non-active mode.

Note that, in order to realize the functionalities of plug and unplug from the charger, it is employed the button of the launchpads.

2.3.2 Topics

A *Smart Health Band* publishes to the following topics:

- **SDCM/collector/band-registration**, to register itself to the database. A message is sent every time a new band is connected to the network. The format of the message is {"bandId": "<BAND_ID>", "active": false, "alertOn": false}.
- **SDCM/band/<BAND_ID>/status**, to inform the collector about the activation/disactivation of the band. The format of the message is {"active": true/false}.
- **SDCM/band/<BAND_ID>/sample**, to notify the collector of the current values of the vital signs of the wearer. The format of the message is {"batteryLevel": VALUE, "oxygenSaturation": VALUE , "bloodPressure": VALUE, "temperature": VALUE, "respiration": VALUE, "heartRate": VALUE}.
- **SDCM/band/<BAND_ID>/alert**, from which the band receives commands to turn on the alarm system, when a worrying trend of the wearer's health is detected. The format of the message is "alertOn": true. Note that the deactivation of the alarm must be done physically by exiting the Data Center and plugging the band to its charger.

2.4 Collector

The collector is responsible for receiving data from both MQTT *Smart Health Bands* and CoAP *Environmental Sensors*, activating the alarm on the bands whenever a worrying trend is detected and activating the actuators whenever the environmental values are not within the established range. The thresholds can be set using the config.properties file. The collector is also in charge of handling the connection with the database.

To receive information from the MQTT *Smart Health Bands*, the collector subscribes to the following topics:

- **SDCM/collector/+**, to receive new band registrations;
- **SDCM/band/+/status**, to receive updates regarding the status (active/inactive, alarmOff) sent by all the bands;
- **SDCM/band/+/sample**, to receive vital signs data sent by all the bands.

To send information to the MQTT monitors, the collector publishes to the topics:

- **SDCM/band/%s/alarm**, to turn on the alarm system of the band specified in the topic itself.

The message formats are the ones described in the MQTT network section.

For what concerns the CoAP protocol, the collector acts both as a client and server. It exposes a single resource, making it available at the following endpoint:

- **/registration**, this endpoint accepts only POST requests, where the latter is issued with payload "type": "RESOURCE_TYPE". In this way the server can allocate the correct CoAP client and associate it to the IPv6 of the registered device. All those information are recordered in data structure mantained by the CoAP server process.

The registration is fundamental when CoAP is used, since it is the only way to let the collector understand that a new monitor has joined the network: the collector is deployed in a traditional network, separate from the LLNs, and so it is not possible to exploit the CoAP discovery capabilities. After the registration, the collector sets up the observe relations to receive updates about the telemetry resources of the device; the observe relations are instantiated only once and they are turn down only if no data arrived since a certain amount of time, in this way the Collector can independently understand when a resource is no longer active.

The Collector analyses the data of sensors if the values go out of some thresholds an alarming situation is detected, the Collector triggers the alarm system of the involved monitor issuing a PUT to its /ventilation endpoint with the following payload "type": "COMMAND", the command could be ON or OFF, when the values received by the sensor returned within the limits the Collector can issue another PUT request to tunn off the actuator.

Thanks to this implementation the control of the enviroment of the Data Center is completaly autonomous.

2.4.1 Package Structure

The collector of the Smart Data Center Maintenance is composed of the following packages and classes.

- **it.unipi.dii.iot.coap**: this package contains the classes required to handle the registration of new CoAP resources and to observe the data exposed by the CoAP servers.
- **it.unipi.dii.iot.config**: this package contains utility classes to manage the configuration parameters.
- **it.unipi.dii.iot.model**: this package contains the classes required for the model. These classes are the Java bean of our application.
- **it.unipi.dii.iot.mqtt**: this package contains the classes required for implementing the MQTT client.
- **it.unipi.dii.iot.persistence**: this package contains the classes to interface with the MySQL database.
- **Collector.java**: it contains the *main* function.

2.5 Data Encoding

The exchanged data between sensors and the collector is encoded in JSON, in the form of JSON objects. The choice was lead by the fact that the sensors have constrained resources in terms of memory, battery and computation capability and XML has a too complex structure and interoperability is not required.

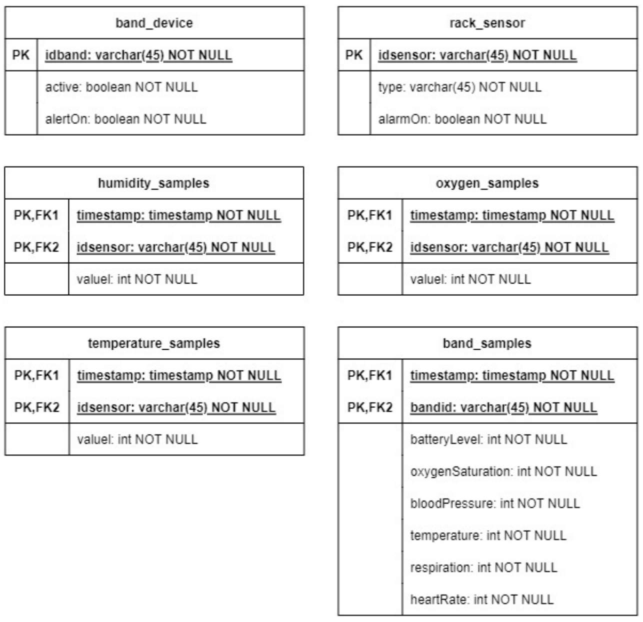
This choice allows for a more lightweight communication, faster processing times and a simpler representation of the information since JSON is more flexible and less verbose, resulting in less overhead.

Since JSON is a text-based format, probably a binary encoding like **CBOR** would have been better; unfortunately nowadays Contiki does not have any library that implements this type of encoding.

2.6 Database

The data captured by the sensors are stored in a MySQL database in order to be able to visualize them in Grafana. The database is placed on the University of Pisa cloud at the following IP address: 172.16.4.159.

The database, named *SDCM*, has the following tables:



band_device	
PK	idband: varchar(45) NOT NULL
	active: boolean NOT NULL
	alertOn: boolean NOT NULL

rack_sensor	
PK	idsensor: varchar(45) NOT NULL
	type: varchar(45) NOT NULL
	alarmOn: boolean NOT NULL

humidity_samples	
PK,FK1	timestamp: timestamp NOT NULL
PK,FK2	idsensor: varchar(45) NOT NULL
	valuel: int NOT NULL

oxygen_samples	
PK,FK1	timestamp: timestamp NOT NULL
PK,FK2	idsensor: varchar(45) NOT NULL
	valuel: int NOT NULL

temperature_samples	
PK,FK1	timestamp: timestamp NOT NULL
PK,FK2	idsensor: varchar(45) NOT NULL
	valuel: int NOT NULL

band_samples	
PK,FK1	timestamp: timestamp NOT NULL
PK,FK2	bandid: varchar(45) NOT NULL
	batteryLevel: int NOT NULL
	oxygenSaturation: int NOT NULL
	bloodPressure: int NOT NULL
	temperature: int NOT NULL
	respiration: int NOT NULL
	heartRate: int NOT NULL

Figure 2: SQL tables of the system.

2.7 Grafana

A web interface is developed with Grafana in order to be able to monitor in real time the data that is stored in the database.

The Grafana dashboard has two rows: the first row contains two panel one for monitoring the alert state of the *Smart Health Bands* and the other that displayes their current status (active/charging); the second row shows the historical data of the three environmental quantities, it also shows the thresholds so that it can be see if the measured values remain within the established range.

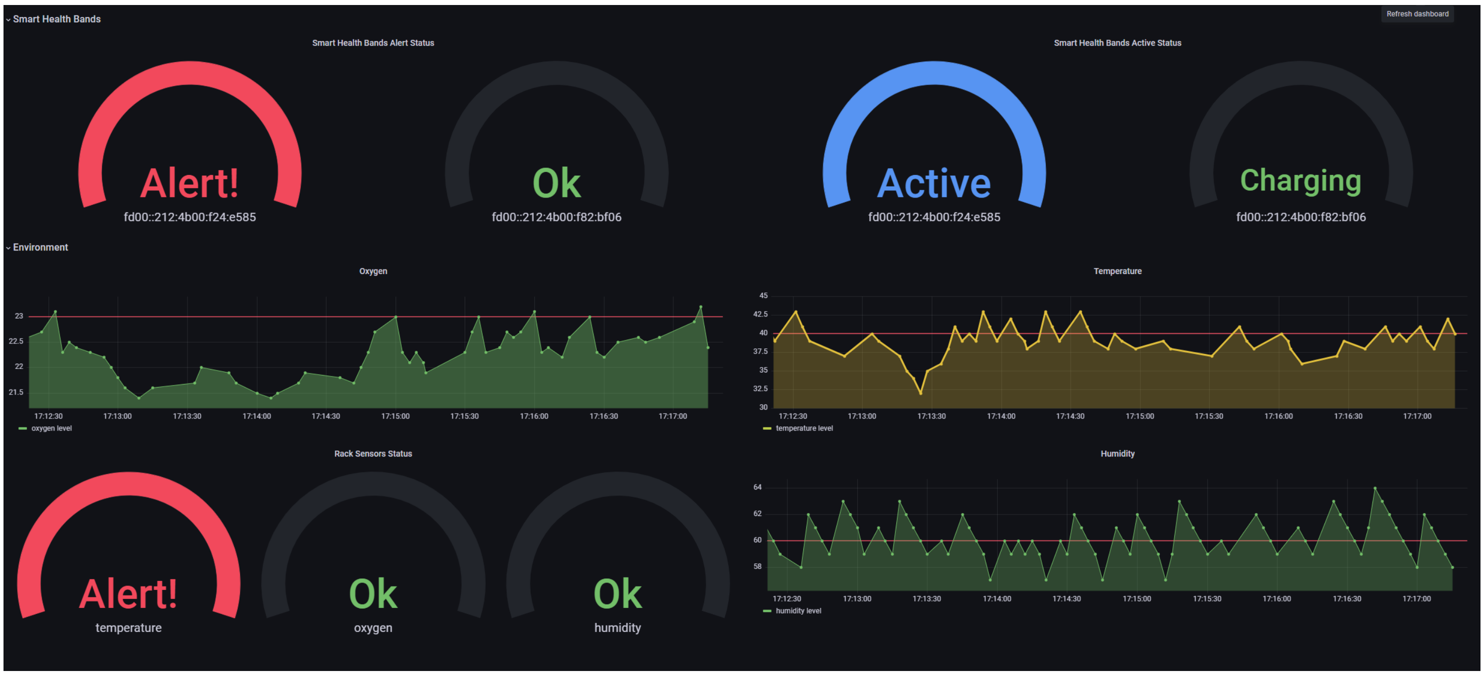
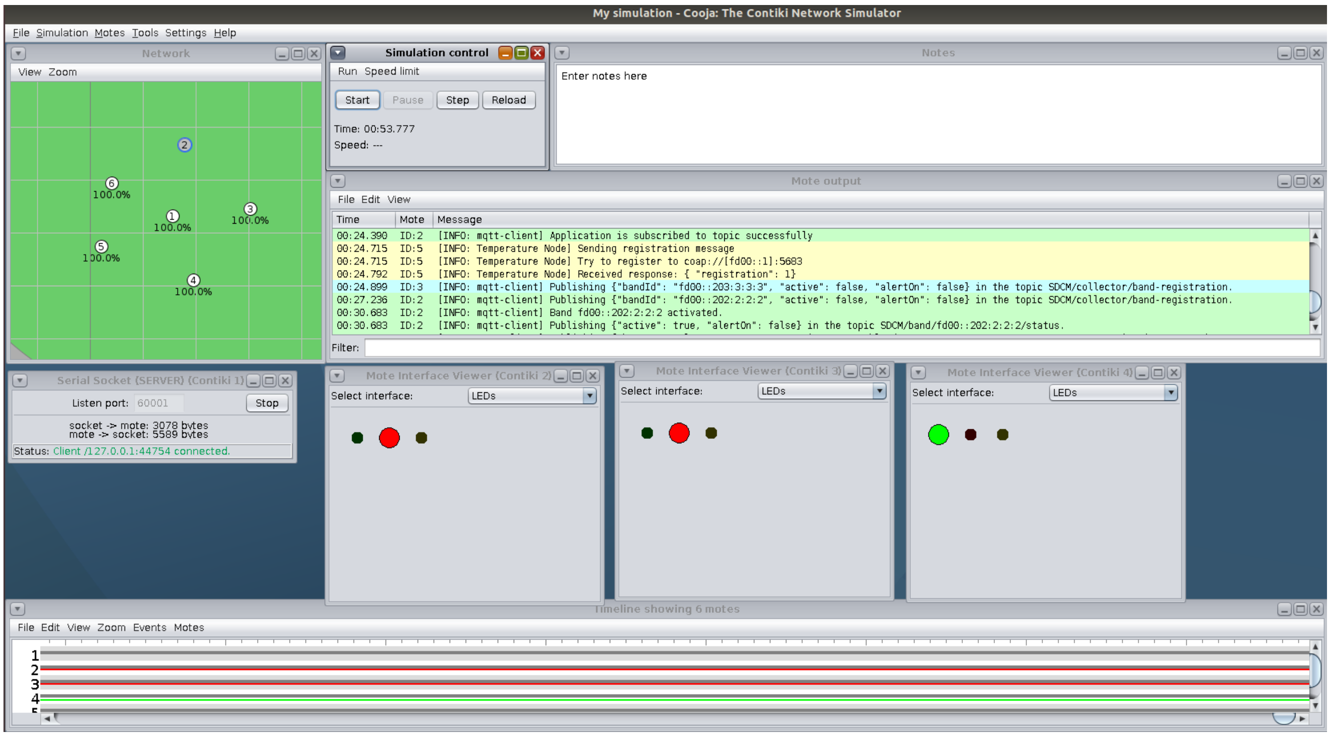


Figure 3: Screenshot of the Grafana web interface.

3 Testing

The Smart Data Center Maintenance system has been tested with the Contiki Cooja Simulator environment and with 6 launchpads.

3.1 Cooja


My simulation - Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Network
View Zoom
Simulation control
Run Speed limit
Start Pause Step Reload
Notes
Mote output
File Edit View
Serial Socket (SERVER) (Contiki 1)
Stop
Mote Interface Viewer (Contiki 2)
Mote Interface Viewer (Contiki 3)
Mote Interface Viewer (Contiki 4)
Timeline showing 6 motes
File Edit View Zoom Events Motes

Time: 00:53.777
Speed: ---

Enter notes here

```
Time       Mote   Message
00:24.390  ID:2   [INFO: mqtt-client] Application is subscribed to topic successfully
00:24.715  ID:5   [INFO: Temperature Node] Sending registration message
00:24.715  ID:5   [INFO: Temperature Node] Try to register to coap://[fd00::1]:5683
00:24.792  ID:5   [INFO: Temperature Node] Received response: { "registration": 1}
00:24.899  ID:3   [INFO: mqtt-client] Publishing {"bandId": "fd00::203:3:3:3", "active": false, "alertOn": false} in the topic SDCM/collector/band-registration.
00:27.236  ID:2   [INFO: mqtt-client] Publishing {"bandId": "fd00::202:2:2:2", "active": false, "alertOn": false} in the topic SDCM/collector/band-registration.
00:30.683  ID:2   [INFO: mqtt-client] Band fd00::202:2:2:2 activated.
00:30.683  ID:2   [INFO: mqtt-client] Publishing {"active": true, "alertOn": false} in the topic SDCM/band/fd00::202:2:2:2/status.
```

Filter:

Listen port: 60001
socket -> mote: 3078 bytes
mote -> socket: 5589 bytes
Status: Client /127.0.0.1:44754 connected.

Select interface: LEDs

Figure 4: Simulation on Cooja.

3.2 Launchpads

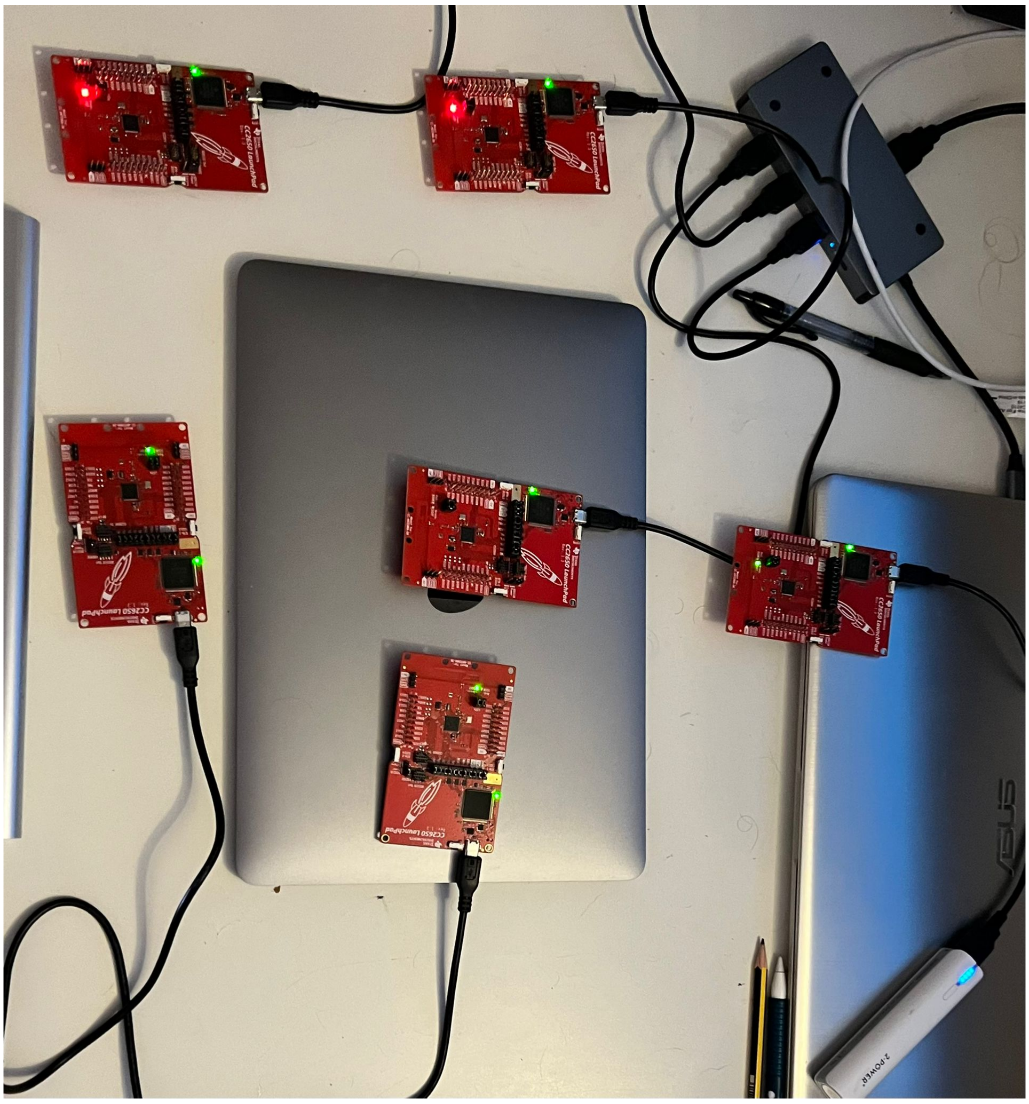


Figure 5: Picture of the launchpads.