



UNIVERSITÀ DI PISA

Artificial Intelligence and Data Engineering

Internet of Things

Industrial Health Monitor

Project Documentation

AUTHOR
Matteo Pierucci

Academic Year 2021/2022

Contents

1	Introduction	2
2	Architecture	3
2.1	Data Encoding	3
2.2	CoAP network	4
2.2.1	Temperature sensor	4
2.2.2	Cooler actuator	5
2.3	MQTT network	6
2.3.1	Vibration sensor	6
3	Command Line Interface	7
4	Database Storage	8

1 Introduction

Industrial Health Monitor is an application that support the monitoring phase of industrial machinery. It creates a network of sensors and actuators that respectively monitor the workflow and take actions to keep the piece of machinery in good working condition. It exploits sensors that collect data about **Temperature** and **Vibration** of the machine where they are applied and they send them to a **Collector** application that stores them into a MySQL database. On the other end an actuator is responsible for enabling a **Cooling system** when the temperature threshold, chosen by user, is overcame.

Collected data can be useful to create **Machine Learning** models capable of predicting machine failures related to Vibration and Temperature. In this way the application can help in organizing predicted maintenance work, that minimizes downtime, improves efficiency and reduces costs.

2 Architecture

The application is composed by two networks: in the first one devices use **CoAP** to communicate with the Collector and in the second the sensor publish data using **MQTT**, which are then retrieved by Collector application through MQTT broker.

Data received are stored in a MySQL database. A user can interact with the Collector using a **Command Line Interface** changing some parameters of the control logic and visualizing the data stored.

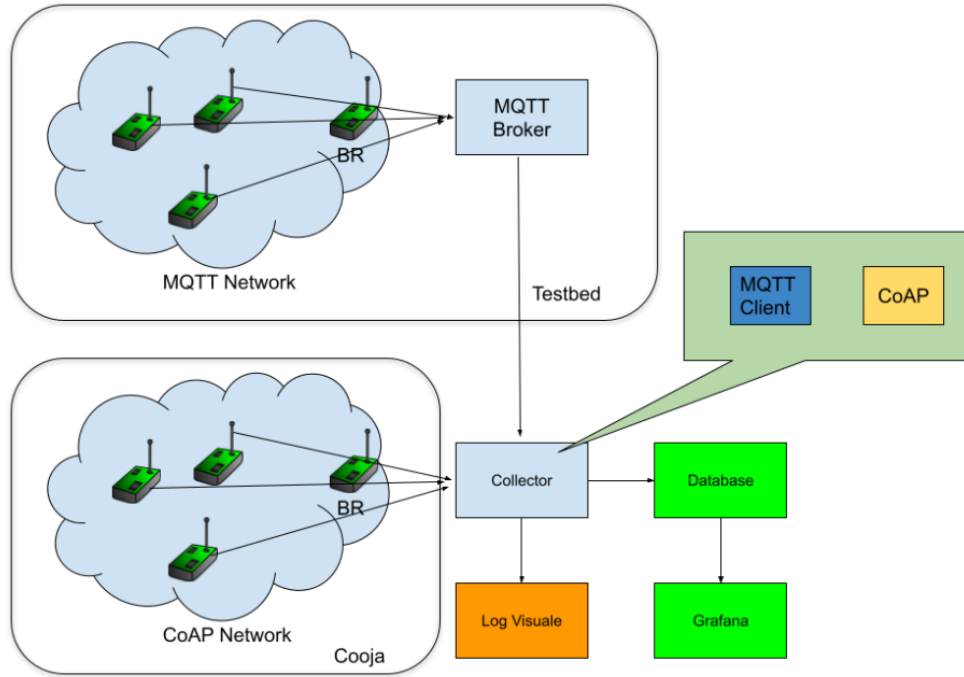


Figure 1: System scheme and structure

2.1 Data Encoding

The Collector and Control Logic were developed using **Java**. Data was encoded using **JSON** format, as the application domain and the kind of data exchanged don't require a more structured encoding language as XML.

2.2 CoAP network

The **CoAP network** consists of a **Temperature sensor** and a **Cooler actuator**, they connect with a **Border Router** and then register themselves to the Collector. The two devices aim to **automatically** detect overheating and cool down the machinery if necessary. After registration of the devices the Collector continuously receive temperature data from Temperature Sensor, then it checks if the received temperature is above or below critical temperature threshold. If some conditions are met the Collector enables or disables the Cooler actuator.

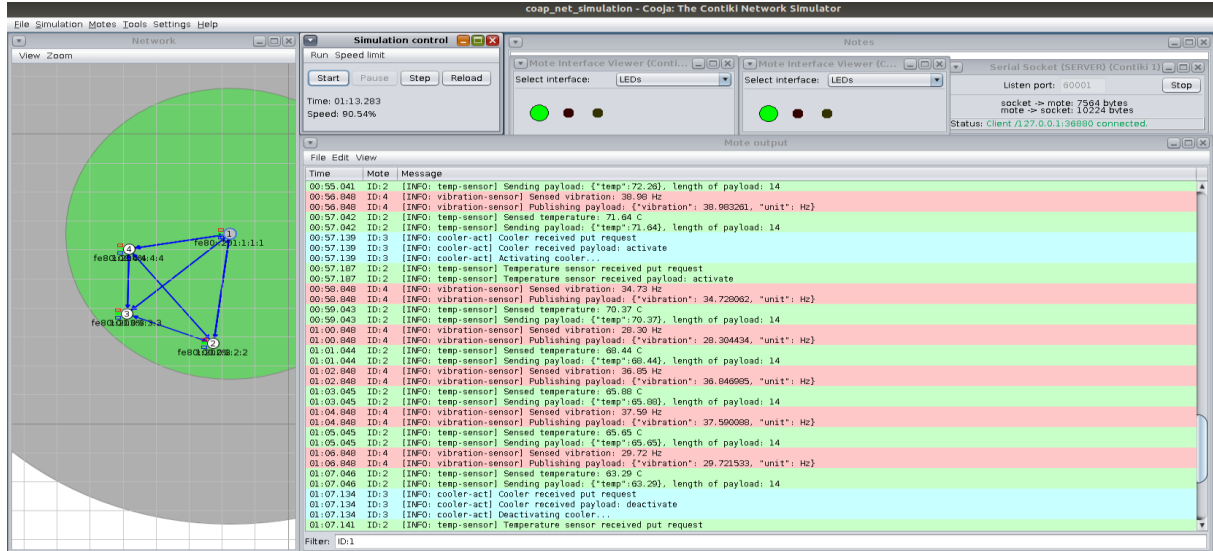


Figure 2: Network simulation in Cooja

2.2.1 Temperature sensor

Since machines that work at high temperatures can be damaged, the insertion of a Temperature Sensor can **avoid failures** and **minimize downtime**.

Temperature sensor connects to the Border Router and then **registers** itself to the Collector with a CoAP request. After the registration, the sensor exposes an **observable resource** which accepts **GET requests** and notifies all observers when new data are available using **JSON format**. In order to **simulate** the sensed environment the sensor also replies to **PUT requests** to simulate the temperature drop when the Cooler System is activated.

The Temperature device is equipped with LED that indicate the **status** of the Sensor:

- **RED** led if the sensor is not connected to the Border Router
- **YELLOW** led if the sensor is connected to the Border Router but it is not registered to the Collector yet
- **GREEN** led if the sensor is connected and registered and it is sending data to the Collector

2.2.2 Cooler actuator

The Cooler Actuator is responsible for decreasing the machine working temperature when the Collector identifies an episode of **overheating**. Cooler actuator connects to the Border Router and then registers itself to the Collector with a CoAP request. After the registration, the actuator exposes a resource that accepts **PUT requests** to activate or deactivate the **Cooling System**. The Collector is responsible for sending **activation/deactivation** requests when sensed temperature changes **above/below** Critical Temperature Threshold.

Cooler device is equipped with LED that indicate the status of the Actuator:

- **YELLOW** led if the actuator is connected to the Border Router but it is not registered to the Collector yet
- **RED** led if the actuator is connected and registered, but is not running
- **GREEN** led if the collector is connected and registered and is currently running

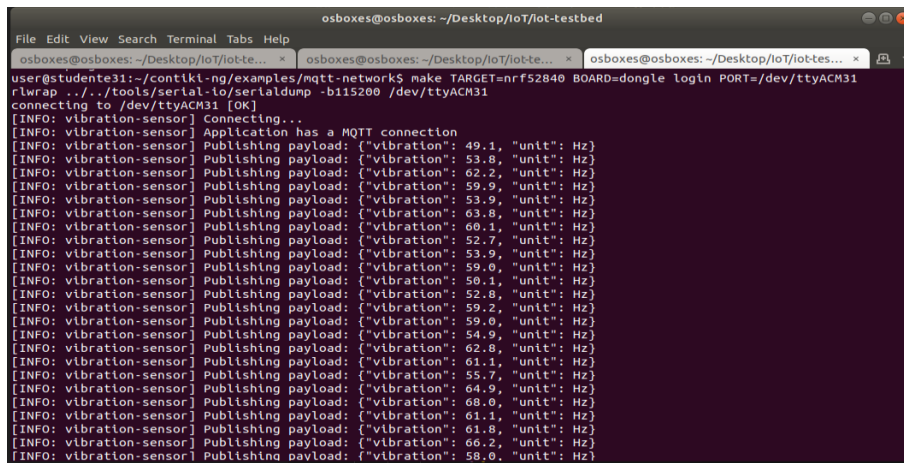
2.3 MQTT network

The MQTT network is composed by a **Vibration Sensor**, which is connected to **MQTT Broker** (mosquitto) through a Border Router. The Sensor and BR are deployed on the Testbed and are connected with the local machine using **SSH**.

Since vibrations can determine **failure of machinery parts**, collecting such data can help during maintenance tasks reducing faults and downtime. Furthermore, vibration data, can be collected and associated with failure episodes in order to train **Machine Learning** models capable of predicting machine faults when vibration sensor registers specific vibration patterns.

2.3.1 Vibration sensor

Vibration sensor connects to the Border Router and then establishes a connection with MQTT broker. When connection is completed the Sensor starts to **publish** sensed data on *vibration* topic using **JSON** format.



```
osboxes@osboxes: ~/Desktop/iot/iot-testbed
File Edit View Search Terminal Tabs Help
osboxes@osboxes: ~/Desktop/iot/iot-te... osboxes@osboxes: ~/Desktop/iot/iot-te... osboxes@osboxes: ~/Desktop/iot/iot-te...
user@student31:~/contiki-ng/examples/mqtt-network$ make TARGET=nrf52840 BOARD=dongle login PORT=/dev/ttyACM31
r/urap ../tools/serial-io/seriaIdump -b115200 /dev/ttyACM31
connecting to /dev/ttyACM31 [OK]
[INFO: vibration-sensor] Connecting...
[INFO: vibration-sensor] Application has a MQTT connection
[INFO: vibration-sensor] Publishing payload: {"vibration": 49.1, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 53.8, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 62.2, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 59.9, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 53.9, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 63.8, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 60.1, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 52.7, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 53.9, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 59.0, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 50.1, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 52.8, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 59.2, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 59.0, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 54.9, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 62.0, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 61.1, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 55.7, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 64.9, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 68.0, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 61.1, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 61.8, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 66.2, "unit": "Hz"}
[INFO: vibration-sensor] Publishing payload: {"vibration": 58.0, "unit": "Hz"}
```

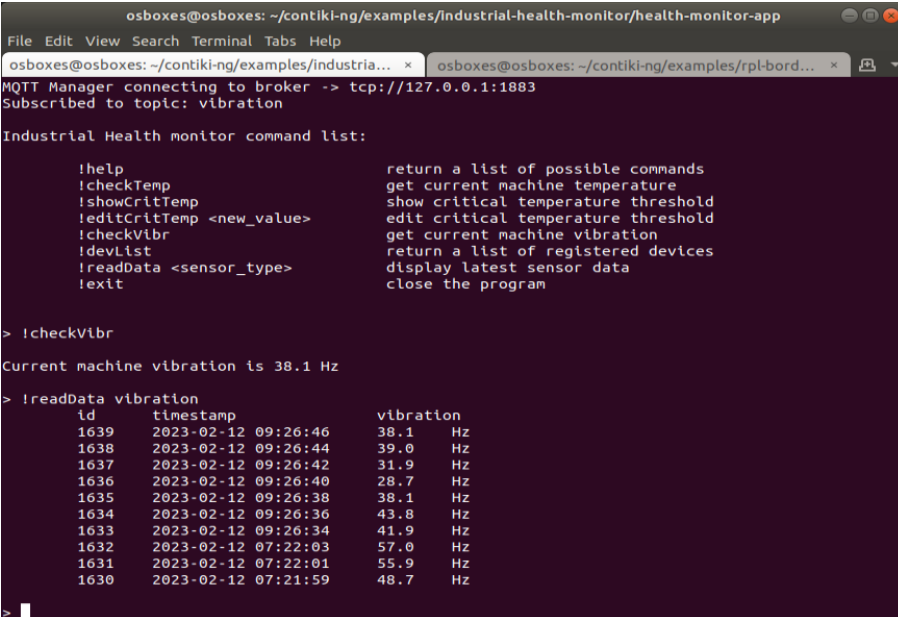
Figure 3: MQTT Vibration Sensor Log

3 Command Line Interface

The Collector is responsible for accepting connection of **CoAP** devices and receiving updates from **MQTT Broker**. Furthermore it implements **control logic** functionalities that allow to manually show and edit **actuator thresholds**, retrieve collected data stored in **DB** and automatically manage actuators.

A user can interact with the **CLI** that allows following commands:

- **!help**, return a list of possible commands
- **!checkTemp**, get current machine temperature
- **!showCritTemp**, show critical temperature threshold
- **!editCritTemp <new_value>**, edit critical temperature threshold
- **!checkVibr**, get current machine vibration
- **!devList**, return a list of registered devices
- **!readData <sensor_type>**, display latest sensor data
- **!exit**, close the program



```
osboxes@osboxes: ~/contiki-ng/examples/industrial-health-monitor/health-monitor-app
File Edit View Search Terminal Tabs Help
osboxes@osboxes: ~/contiki-ng/examples/industria... x osboxes@osboxes: ~/contiki-ng/examples/rpl-bord... x
MQTT Manager connecting to broker -> tcp://127.0.0.1:1883
Subscribed to topic: vibration

Industrial Health monitor command list:

!help          return a list of possible commands
!checkTemp     get current machine temperature
!showCritTemp  show critical temperature threshold
!editCritTemp  edit critical temperature threshold
!checkVibr     get current machine vibration
!devList       return a list of registered devices
!readData <sensor_type> display latest sensor data
!exit          close the program

> !checkVibr

Current machine vibration is 38.1 Hz

> !readData vibration
  id      timestamp          vibration
  ---      -
  1639    2023-02-12 09:26:46  38.1    Hz
  1638    2023-02-12 09:26:44  39.0    Hz
  1637    2023-02-12 09:26:42  31.9    Hz
  1636    2023-02-12 09:26:40  28.7    Hz
  1635    2023-02-12 09:26:38  38.1    Hz
  1634    2023-02-12 09:26:36  43.8    Hz
  1633    2023-02-12 09:26:34  41.9    Hz
  1632    2023-02-12 07:22:03  57.0    Hz
  1631    2023-02-12 07:22:01  55.9    Hz
  1630    2023-02-12 07:21:59  48.7    Hz

>
```

Figure 4: Command Line Interface

4 Database Storage

The Collector stores received data in a SQL database. The *industrial_health_monitor* database has the following tables:

- **temperature** (id, timestamp, temp_value, unit)
- **vibration** (id, timestamp, vibr_value, unit)

Most recent data can then be read using CLI command specifying the table where data are stored.

GitHub Repository

Full implementation code can be found at the following link:

<https://github.com/pieruccim/industrial-health-monitor>