EYRAUD Côme – RIO201 Project

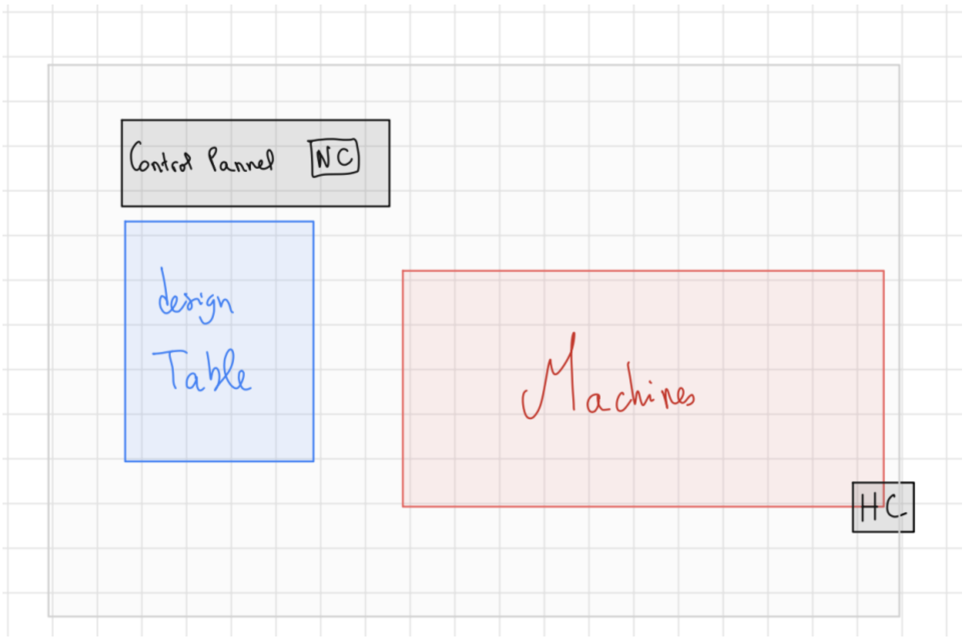
Final Project – Factory Metrics Control

1. Introduction

Factory Metrics Control is an IoT application that aims the wellbeing of machines and worker in factories.
Our first use case is a simple warehouse with machines working next to workers, the machines have special needs in term of humidity level, the factory also needed to monitor the loudness of the warehouse for the wellbeing of the workers, to be able to certify that the environment is safe for the workers.

It uses sensors to monitor the loudness and the humidity level. I'll explain the architecture just under.

2. Architecture



This is a schematic of the warehouse, with the location of the two sensors, the Humidity Sensor, and the Noise Sensor.
On the Control Panel, there is a LED, that is in case of an issue, so that workers know when something is up.
For the status of the LED, the Noise Sensor (NC), read its noise sensor then asks the humidity sensor with a request, then checks if the humidity level or noise is too high.

Then there is a server in the warehouse that requests to the NC and HC to have an history and create a Graph.

3. Implementation

A. NC (Noise Controller)

The NC is a coap client and server and has a noise sensor.
As a server, it has this route: /get-noise, where the Graph Server (GS) can read the value of the sensor.

As a client it queries the HC (Humidity Controller) at the route /get-humid to read the value of the sensor.

It periodically asks the HC for the value, then check if it is above the threshold, then read its noise value then check if it is above the threshold, if yes, then light up the LED.

B. HC (Humid Censor)

The HC is a coap server that has a humidity sensor.
As a server, it has the route: /get-humid so the GS and NC can ask it for the value of the sensor.

C. GS (Graph Server)

The Graph Server runs on a server and loops every x second to request HC/get-humid and NC/get-noise. To be then able to have a graph with all this data.

D. BR (Border Router)

It makes the interface between the GS and the HC & NC network.

4. FIT IOT-LAB Specific

A. Introduction

There is no Humidity or Noise sensor on the available controllers, so I will use light sensor to "replace" them

B. The functions

- Function 1:

The HC has his route available.

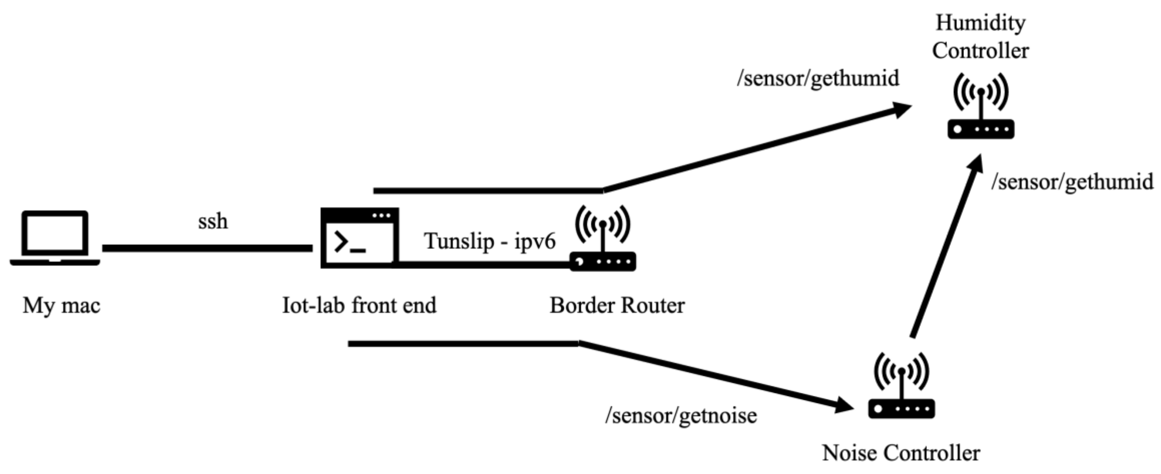
- Function 2:

The NC Asks the HC and reads his sensor

- Function 3:

The GS asks the HC and the NC periodically and stores the result.

C. The Architecture



D. Code

This is the architecture of the code :

10-humid-censor/
11-noise-censor/
auto.py
GS.py
Readme.md

1. The python scripts

I have two python script, the auto.py one is a « helper » that helps setup the environment and the nodes. It is a program that asks for information then print the needed commands to run.

GS.py is a script emulate the Graph Server, it queries every second the NC and HC and stores its information.

2. The node code

I have three nodes, the BR, the NC and the HC. The BR is already compiled, but the NC and HC are coded by me.

For the HC, it is a simple COAP server, with two routes, the /test/hello that sends back his name and the /sensor/gethumid that sends the sensor information.

For the NC, it is a COAP client/server, as a server it has two routes, /test/hello that sends back his name and the /sensor/getnoise that sends the sensor information.
As a client, it asks periodically HC/sensor/gethumid and print it to the terminal.

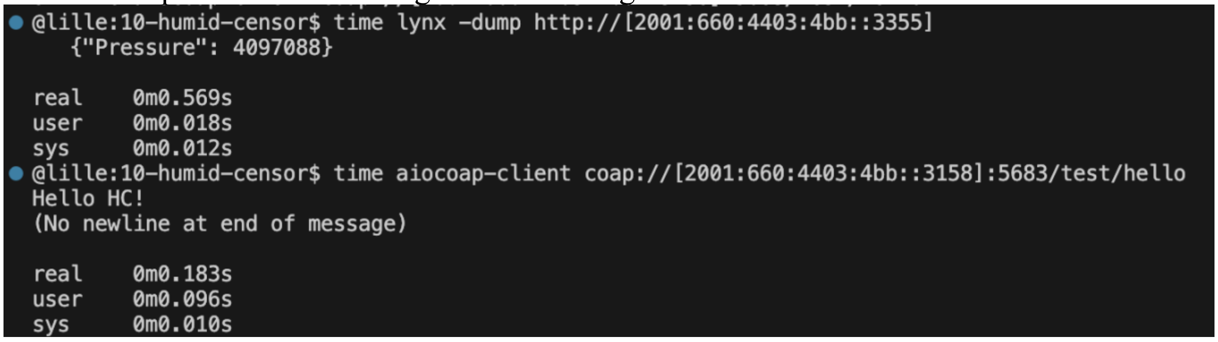
To make this work, you need to put all the code I provided in the exemple/iotlab directory to be able to compile.

You'll find attached to this email a video of the working project.

E. Comparing HTTP vs COAP

To compare HTTP and coap for this project, I am going to compare the HC with http and HC with COAP, for this i have a python script name coapvshttp.py inside the 10-humid-censor that helps with the setup of the environment.

When I compare both of them I got the following result :

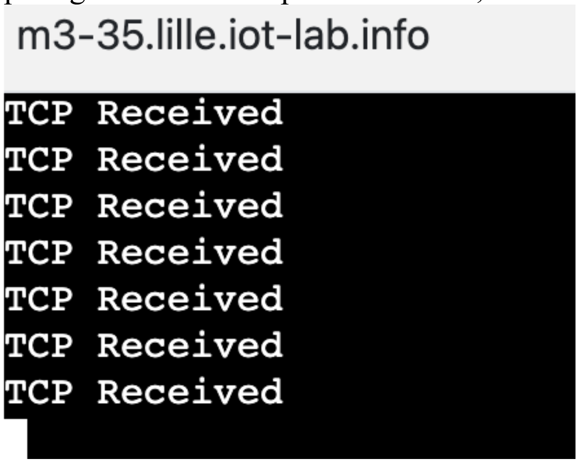


```
● @lille:10-humid-censor$ time lynx -dump http://[2001:660:4403:4bb::3355]
     {"Pressure": 4097088}

  real    0m0.569s
  user    0m0.018s
  sys     0m0.012s
● @lille:10-humid-censor$ time aiocoap-client coap://[2001:660:4403:4bb::3158]:5683/test/hello
  Hello HC!
  (No newline at end of message)

  real    0m0.183s
  user    0m0.096s
  sys     0m0.010s
```

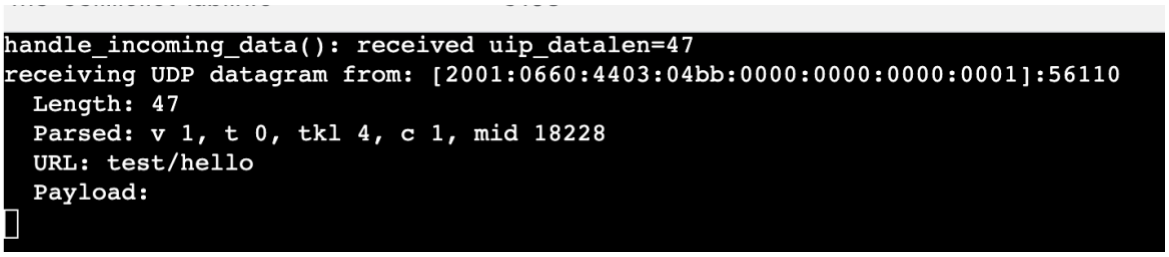
It shows that the coap is a lot faster than http.
Comparing the number of packets needed, when we look at HTTP :



```
TCP Received
TCP Received
TCP Received
TCP Received
TCP Received
TCP Received
TCP Received
```

It uses TCP, so needs to connect, and to respond to a simple request needs an exchange of at least 7 requests to the server.

Compared to the COAP



```
handle_incoming_data(): received uip_datalen=47
receiving UDP datagram from: [2001:0660:4403:04bb:0000:0000:0000:0001]:56110
  Length: 47
  Parsed: v 1, t 0, tkl 4, c 1, mid 18228
  URL: test/hello
  Payload:
```

Where only one UDP request is needed.

In conclusion, HTTP performed a lot worse in this environment, because it requires a lot of resources and more than what is necessary on this example, it is not needed to be in "connected" mode and takes more time and more resources. COAP is much more appropriate for this use as it doesn't use as much memory and time to use.

5. Conclusion

All of the needed material are on my github : https://github.com/comeyrd/rio201-project

With a README.md that gives information on how to install on the iot-lab service.