

Middleware for the IoT

Desportes Kilian - Imekraz Yanis - B1

Middleware for the IoT	1
TP1	1
Questions	1
Practical part	2
TP3	5
Infrastructure tree	5
Infrastructure creation	5
Questions	9
TP4	10
Get sensors values and display them	10
Sensors and activators	11
Results :	13
Dashboard	15
Benefits and drawbacks of Node-Red	15

TP1

Questions

- What is the typical architecture of an IoT system based on the MQTT protocol?

The typical architecture of an IoT system based on the MQTT protocol is a computer which will heberge the broker, and this broker will allow connected objects, MQTT clients, such as sensor, smartphone, other servers,..., to subscribe to topics and send messages between thelselfes. This is a publish/subscribe message pattern.

- What is the IP protocol under MQTT? What does it mean in terms of bandwidth usage, type of communication, etc ?

The IP protocol under MQTT is TCP. It means it'll use more bandwidth than an UDP connection, due to aknowledgment. It also means the communication will be lossless and the messages are reliable.

- What are the different versions of MQTT?

The different versions of MQTT are :

- MQTT v3.1.0
- MQTT v3.1.1 (common use)
- MQTT v5 (currently limited use)
- MQTT-SN , working over UDP

- What kind of security/authentication/encryption are used in MQTT?

To be secured, MQTT is using a client authentication, where the broker can verify the identity of a client with a client id, username/password or client certificates.

MQTT is also using TLS/SSL security and Payload encryption to protect the content of MQTT messages.

- Suppose you have devices that include one button, one light and luminosity sensor. You would like to create a smart system for your house with this behavior:

- you would like to be able to switch on the light manually with the button
- the light is automatically switched on when the luminosity is under a certain value

What different topics will be necessary to get this behavior and what will the connection be in terms of publishing or subscribing?

We'll need a broker to allow subscription and publication under topics which will be named 'luminosity' and 'button'.

The button will publish on the 'button' topic when we switch it manually and will send 'on' or 'off'.

The luminosity sensor will publish on 'luminosity' topic and will send the luminosity every X second on the topic.

The light will be subscribed to both luminosity and button topics. The light would analyse arriving messages and would switch on 'on' if it received an 'on' from button topic, or if the luminosity sends a value under a given amount.

If the light cannot make information treatment (maybe it's not its role), we can use a computer instead of the lamp, set a 3rd topic 'lamp onoff' and the computer will treat both informations from 'button' and 'luminosity' and will send the final information on 'lamp onoff' which will allow to manage the lamp (the lamp would have subscribed to this topic first, of course).

Practical part

Characteristics of NodeMCU :

- Communication : USB with laptop
- Programming language : C
- I/O Capabilities : 13 pins

Code analysis :

Setup methods will be used to give objects their parameters.

We'll initialize every object needed like MqttClient and WiFi.

We also have the loop() method which will check the connection status of mqtt and will establish the network and mqtt connection if it is not already established.

If there is an error somewhere, it will print it and return.

At the end of the loop, it will idle for 30 seconds.

Code writing and testing :

We'll add subscribe call to topics needed :

```
const char* MQTT_TOPIC_LIGHT_SENSOR = "test/" MQTT-ID "/light_sensor";
const char* MQTT_TOPIC_LAMP = "test/" MQTT-ID "/lamp";
const char* MQTT_TOPIC_BUTTON = "test/" MQTT-ID "/button";
```

And, into loop function :

```
MqttClient::Error::type rc = mqtt->subscribe(
    MQTT_TOPIC_LIGHT_SENSOR, MqttClient::QOS0, processMessage
);
if (rc != MqttClient::Error::SUCCESS) {
    LOG_PRINTFLN("Subscribe error: %i", rc);
    LOG_PRINTFLN("Drop connection");
    mqtt->disconnect();
    return;
}
rc = mqtt->subscribe(
    MQTT_TOPIC_LAMP, MqttClient::QOS0, processMessage
);
if (rc != MqttClient::Error::SUCCESS) {
    LOG_PRINTFLN("Subscribe error: %i", rc);
    LOG_PRINTFLN("Drop connection");
    mqtt->disconnect();
    return;
}
rc = mqtt->subscribe(
    MQTT_TOPIC_BUTTON, MqttClient::QOS0, processMessage
);
if (rc != MqttClient::Error::SUCCESS) {
    LOG_PRINTFLN("Subscribe error: %i", rc);
    LOG_PRINTFLN("Drop connection");
    mqtt->disconnect();
    return;
}
```

```
val = digitalRead(buttonPin);
Serial.println("Value");
Serial.println(val);
if(val == HIGH){
    digitalWrite(ledPin,LOW);
}else{
    digitalWrite(ledPin,HIGH);
}
```

TP3

Infrastructure tree

Describe the deployed infrastructure and the interactions between the different components.

The infrastructure we created is composed of 3 application entities that will heberge 3 sensors :

- SmartMeter
- LuminositySensor
- TemperatureSensor

Each sensor will have a DESCRIPTOR and a DATA container instance.

The goal is to be able to monitor every data that are retrived by the sensors by the subscription mecanism.

Infrastructure creation

Create a report that includes the main requests that you have created and describes the application you have implemented with some code examples.

First we'll create the ACP to allow specific rights to specific ressources.

URL	http://127.0.0.1:8080/~in-cse/
Methode	POST
En-tête	X-M2M-Origin: admin:admin Content-type: application/xml;ty=1
Corps	<pre><m2m:acp xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="ACP_SENSOR"> <pv> <acr> <acor>guest:guest</acor> <acop>3</acop> </acr> <acr> <acor>admin:admin</acor> <acop>63</acop> </acr> </pv> <pvs> <acr> <acor>admin:admin</acor> <acop>63</acop> </acr> </pvs> </m2m:acp></pre>

We will do the same again with 'guest:guest' acop of 2 and name "ACP_MONITORINGG".
Then we create the sensors AE.

URL	http://127.0.0.1:8080/~in-cse/
Methode	POST
En-tête	X-M2M-Origin: admin:admin Content-type: application/xml;ty=2
Corps	<m2m:ae xmlns:m2m = "http://www.onem2m.org/xml/protocols" rn = "Sensor_Name"> <api>app-sensor</api> <lbl>Type/sensor</lbl> <rr>false</rr> </m2m:ae>

Sensor_Name being the name of the sensor (we'll have to change for each).

Then we'll create, for each sensor, the DESCRIPTOR and DATA container.

URL	http://127.0.0.1:8080/~in-cse/in-name/Sensor_Name
Methode	POST
En-tête	X-M2M-Origin: admin:admin Content-type: application/xml;ty=3
Corps	<m2m:cnt xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="DATA"> </m2m:cnt>

Then the same request with DESCRIPTOR and not DATA in 'rn' field.

We'll then attach the ACP to these containers.

URL	http://127.0.0.1:8080/~in-cse/in-name/Sensor_Name/
Methode	PUT
En-tête	X-M2M-Origin: admin:admin Content-type: application/json;ty=3
Corps	{ "m2m:cnt": { "acpi": ["/in-cse/acp-585995077", "/in-cse/acp-534323853"] } }

We'll then create content instances

URL	http://127.0.0.1:8080/~in-cse/in-name/Sensor_Name/DATA
Methode	POST
En-tête	X-M2M-Origin: admin:admin Content-type: application/xml;ty=4
Corps	<pre><m2m:cin xmlns:m2m="http://www.onem2m.org/xml/protocols"> <cnf>message</cnf> <con> &lt;obj&gt; &lt;str name="Category" val="Light"/&gt; &lt;str name="Data" val="300" /&gt; &lt;str name="Unit" val="Lux" /&gt; &lt;str name="Location" val="Home" /&gt; &lt;/obj&gt; </con> </m2m:cin></pre>

and

URL	http://127.0.0.1:8080/~in-cse/in-name/Sensor_Name/DESCRIPTOR
Methode	POST
En-tête	X-M2M-Origin: admin:admin Content-type: application/xml;ty=4
Corps	<pre><m2m:cin xmlns:m2m="http://www.onem2m.org/xml/protocols"> <cnf>message</cnf> <con> &lt;obj&gt; &lt;str name="Type" val="Sensor" /&gt; &lt;str name="Category" val="Light" /&gt; &lt;str name="Unit" val="Lux" /&gt; &lt;str name="Model" val="1142_0" /&gt; &lt;str name="Location" val="Home" /&gt; &lt;str name="Manufacturer" val="PHIDGETS" /&gt; &lt;str name="Consumption Max" val="27 mA" /&gt; &lt;str name="Voltage Min" val="4.8 V DC" /&gt; &lt;str name="Voltage Max" val="5.3 V DC" /&gt; &lt;str name="Operating Temperature Min" val="0 C" /&gt; &lt;str name="Operating Temperature Max" val="70 C" /&gt; &lt;/obj&gt; </con> </m2m:cin></pre>

Each sensor will have specific characteristic into <obj> part.
We'll finally subscribe to the luminosity sensor stream to receive data from it.

URL	http://127.0.0.1:8080/~in-cse/in-name/MY_SENSOR/DATA
Methode	POST
En-tête	X-M2M-Origin: admin:admin Content-type: application/xml;ty=23
Corps	<m2m:sub xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="SUB_MY_SENSOR"> <nu>http://localhost:1400/monitor</nu> <nct>2</nct> </m2m:sub>

Then once the subscription is done, when the sensor will receive something on DATA container, the monitor will receive it too.

```
Received notification:
<?xml version="1.0" encoding="UTF-8"?>
<m2m:sgn xmlns:m2m="http://www.onem2m.org/xml/protocols">
  <nev>
    <rep rn="cin_572332578">
      <ty>4</ty>
      <ri>/in-cse/cin-572332578</ri>
      <pi>/in-cse/cnt-925967122</pi>
      <ct>20201102T100833</ct>
      <lt>20201102T100833</lt>
      <st>0</st>
      <cnf>message</cnf>
      <cs>196</cs>
      <con>
        <obj>
          <str name="Category" val="Light"/>
          <str name="Data" val="300"/>
          <str name="Unit" val="Lux"/>
          <str name="Location" val="Home"/>
        </obj>
      </con>
    </rep>
    <rss>1</rss>
  </nev>
  <sud>>false</sud>
  <sur>/in-cse/in-name/LuminositySensor/DATA/SUB_LUMINOSITY</sur>
</m2m:sgn>
```

Questions

Give the main difference between MQTT and oneM2M after the TP1 and TP3.

MQTT is based on a broker to allow sending and subscription while oneM2M is based on a resource architecture and a subscription mechanism to given resources.

oneM2M is based on multiple layers (application, services,...) that can be deployed on objects, gateway, while MQTT is a client/server mechanism that can be used on objects.

oneM2M is based on REST requests and a resources tree while MQTT is a publish/subscribe mechanism through different topics.

Compare the solution seen in the previous lab with OM2M with a clear positioning.

In the previous lab, we've seen how to use the Sierra Wireless solution with mangOH Yellow.

The difference with OM2M is that with mangOH Yellow we already had access to the whole architecture with every sensor on the board.

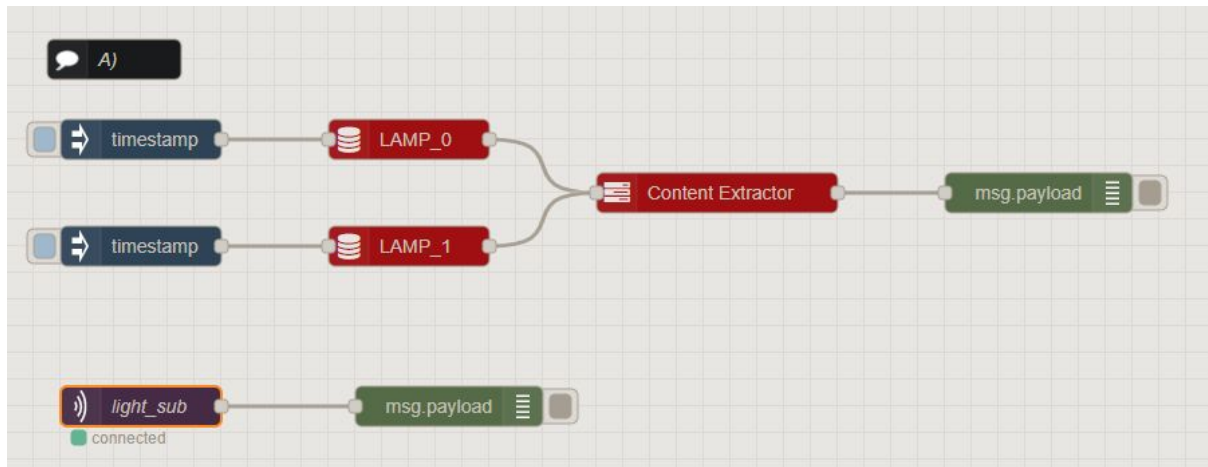
We only had to activate them then create a code to monitor the values.

The interface was also better with a graphical view of the values (we can also do that with OM2M but it's a bit harder to implement).

TP4

In this TP we will use both MQTT and oneM2M at the same time with node-red application.

Get sensors values and display them



The application will look like this in node-red.

We'll have timestamps which will trigger content extraction from LAMPs objects in the mn-cse tree. It's configured to show last content instance of DATA container.



The output will look like this :

```
11/17/2020, 6:33:34 PM node: e0a75fd6.4fe55
msg.payload : string[136]
"<obj><str name='type' val='LAMP'/><str
name='location' val='Home'/><str name='lampId'
val='LAMP_0'/><str name='state' val='true'/>
</obj>"

11/17/2020, 6:33:34 PM node: e0a75fd6.4fe55
msg.payload : string[136]
"<obj><str name='type' val='LAMP'/><str
name='location' val='Home'/><str name='lampId'
val='LAMP_1'/><str name='state' val='true'/>
</obj>"
```

We also have an application to show what we receive on a specific topic through MQTT.

We both have the choice to retrieve last DATA cin from luminosity sensor or to receive a measure from a MQTT topic.

CSE	MN-CSE	
AE	LAMP_0	
Container(s) Name(s)	DATA	
Content Instance	Latest	
Name	LAMP_0	

Property	Other
To indicate	Data
Name	Name

We could also have chosen to use MQTT with a sensor sending simulation like this :

```
C:\Program Files\mosquitto>mosquitto_pub.exe -t topic_light -m "200"
```

Then, in both cases, we will compare the value with our threshold, set at 350 Lux.

→ Test if

msg. payload < 350

Name

LuminositySensor data :

OM2M CSE Resource Tree

http://127.0.0.1:8080/~in-cse/cin-572332578

in-name

- acp_admin
- ACP_MONITORINGG
- ACP_SENSOR
- acpae-300844771
- acpae-280643604
- acpae-494477620
- SmartMeter
- LuminositySensor
 - DESCRIPTOR
 - DATA
 - cin_572332578
 - SUB_LUMINOSITY
- TemperatureSensor
- mn-cse

Attribute	Value
ty	4
ri	/in-cse/cin-572332578
pl	/in-cse/cnt-925967122
ct	20201102T100833
lt	20201102T100833
st	0
cnf	message
cs	196

Attribute	Value
Category	Light
Data	300
Unit	Lux
Location	Home

Then we will create a new content instance for LAMPs objects and set them to true or false, depending on the result of the condition :

Platform: MN-CSE

Application: LAMP_0

Container: DATA

Labels:

Type	Name	Value
str	location	Home
str	lampId	LAMP_0
str	state	true

+ add

Name: LAMP_0 ON

Results :

- LuminositySensor retrieving

```
11/17/2020, 6:38:54 PM node: 874e03dc.ca7a7
msg.payload : string[3]
"300"

11/17/2020, 6:38:54 PM node: 874e03dc.ca7a7
topic72f02d57.c89714 : msg.payload : boolean
true

11/17/2020, 6:38:54 PM node: 874e03dc.ca7a7
topic_lamp : msg.payload : string[4]
"true"

11/17/2020, 6:38:54 PM node: 874e03dc.ca7a7
topic72f02d57.c89714 : msg.payload : string[580]
  ▶ "<?xml version='1.0' encoding='UTF-8'?><m2m:cin
  xmlns:m2m='http://www.onem2m.org/xml/protocols' rn='cin_963064378'>
  <ri>/mn-cse/cin-963064378</ri>
  <pi>/mn-cse/cnt-987853131</pi>
  <ct>20201117T183854</ct>
  <lt>20201117T183854</lt>
  <st>0</st>
  <cnf>message</cnf>
  <cs>136</cs>
  <con><obj><str name='type';
  val='LAMP';/>
  <str name='location'; val='Home';/>
  <str name='lampId'; val='LAMP_1';/>
  <str name='state'; val='true';/>
  </obj></con></m2m:cin>"

11/17/2020, 6:38:54 PM node: 874e03dc.ca7a7
topic72f02d57.c89714 : msg.payload : string[580]
  ▶ "<?xml version='1.0' encoding='UTF-8'?><m2m:cin
  xmlns:m2m='http://www.onem2m.org/xml/protocols' rn='cin_843184373'>
  <ri>/mn-cse/cin-843184373</ri>
  <pi>/mn-cse/cnt-904717067</pi>
  <ct>20201117T183854</ct>
  <lt>20201117T183854</lt>
  <st>0</st>
  <cnf>message</cnf>
  <cs>136</cs>
  <con><obj><str name='type';
  val='LAMP';/>
  <str name='location'; val='Home';/>
  <str name='lampId'; val='LAMP_0';/>
  <str name='state'; val='true';/>
  </obj></con></m2m:cin>"
```

- MQTT receive

11/17/2020, 6:37:34 PM node: 1f8029a5.64d9d6

topic_light : msg.payload : string[3]

"200"

11/17/2020, 6:37:35 PM node: 874e03dc.ca7a7

topic8b1007ef.d179d8 : msg.payload : boolean

true

11/17/2020, 6:37:35 PM node: 874e03dc.ca7a7

topic_lamp : msg.payload : string[4]

"true"

11/17/2020, 6:37:35 PM node: 874e03dc.ca7a7

topic8b1007ef.d179d8 : msg.payload : string[580]

▼ string[580]

```
<?xml version="1.0" encoding="UTF-8"?>
<m2m:cin xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="cin_458523677">
  <ty>4</ty>
  <ri>/mn-cse/cin-458523677</ri>
  <pi>/mn-cse/cnt-904717067</pi>
  <ct>20201117T183734</ct>
  <lt>20201117T183734</lt>
  <st>0</st>
  <cnf>message</cnf>
  <cs>136</cs>
  <con>&lt;obj>&lt;str name="type" val="LAMP"/>&lt;str name="locati
on" val="Home"/>&lt;str name="lampId" val="LAMP_0"/>&lt;st
r name="state" val="true"/>&lt;/obj></con>
</m2m:cin>
```

11/17/2020, 6:37:35 PM node: 874e03dc.ca7a7

topic8b1007ef.d179d8 : msg.payload : string[580]

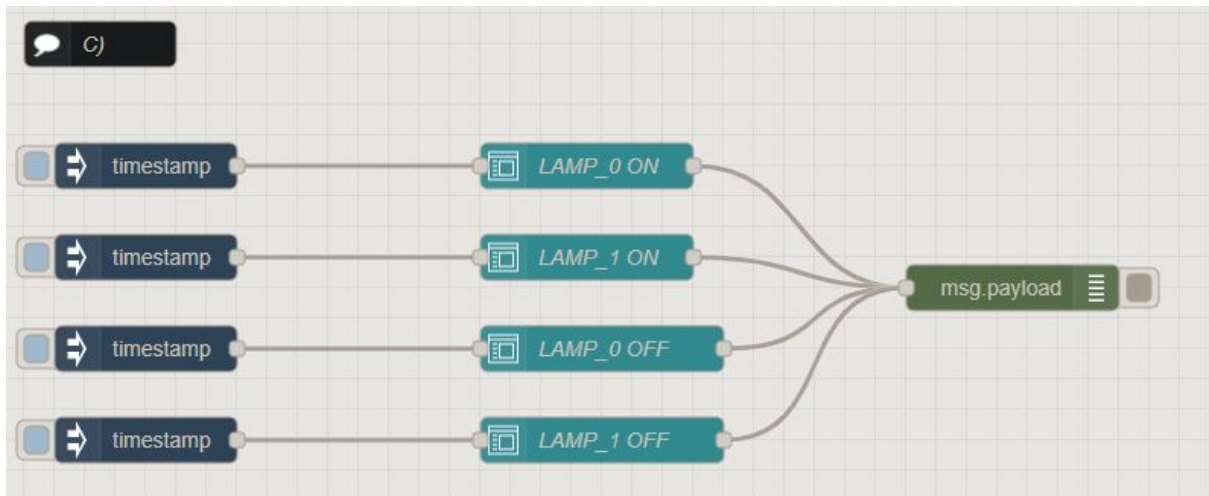
▼ string[580]

```
<?xml version="1.0" encoding="UTF-8"?>
<m2m:cin xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="cin_121122731">
  <ty>4</ty>
  <ri>/mn-cse/cin-121122731</ri>
  <pi>/mn-cse/cnt-987853131</pi>
  <ct>20201117T183734</ct>
  <lt>20201117T183734</lt>
  <st>0</st>
  <cnf>message</cnf>
  <cs>136</cs>
  <con>&lt;obj>&lt;str name="type" val="LAMP"/>&lt;str name="locati
on" val="Home"/>&lt;str name="lampId" val="LAMP_1"/>&lt;st
r name="state" val="true"/>&lt;/obj></con>
</m2m:cin>
```


Dashboard

I only created buttons to switch on/off lamps.

I didn't find a way to plot values with graphs on Node-Red.



Imagine

- Not done -

Benefits and drawbacks of Node-Red

The main benefit of Node-Red is the capacity to use different technologies in the same application very easily, like here with MQTT and oneM2M, only by installing the needed packages.

Another advantage is that the application is very 'visual', easy to set up and easy to modify. It does not require knowing how to code.

A drawback is that if the application becomes more and more complex, it will be really hard to understand what is happening (with too many 'rectangle' and 'arrows').