Serviços de Comunicações
Middleware evaluation in an IoT application

Rodrigo Caldas (up201708987), Tiago Ribeiro (up201708988)

ABSTRACT

Internet of Things (IoT) applications are nowadays widely used both for domestic and industrial purposes. However they still represent a rather large implementation challenge.
On this project a temperature sensor will be used to automatically send out warnings about the air quality. This sensor is connected to a NodeMCU ESP8266 that has an internet connection and in addition two different middlewares will be used which are a software that provides services and resources common to applications.
Once the design and implementation of the IOT application is done, a comparison of the two middlewares used will be made.

1. INTRODUCTION

A middleware is a software that provides common services and resources to applications and helps developers creating applications on an easily and efficiently way since this software has the purpose of connecting applications, data and users. Includes everything from web servers to authentication systems and messaging tools.
MQTT and FIWARE are the two software that will be used throughout the project and later compared with each other showing the advantages and disadvantages that each one presents and that were evidenced during the project.

2. MIDDLEWARE EXPLANATION

2.1 MQTT

MQTT stands for Message Queuing Telemetry Transport is a publish-subscribe network protocol that transports messages between different devices. This protocol runs over TCP/IP but any network protocol that provides ordered, lossless and bi-directional communications can support MQTT[4]. Although TCP/IP provides guaranteed data delivery, MQTT adds 3 quality of service (QoS) levels on top of TCP[4]:

- QoS 0: where messages are assured to arrive at most once, hence can be lost when connection problems occur.
- QoS 1: where messages are assured to arrive but duplicates can occur. This level requires a 2-way handshake for each sent message.
- QoS 2: where messages are assured to arrive exactly once. This level requires a 4-way handshake for each sent message.

The protocol defines two types of network entities: a broker and the clients.
Broker is a server that receives messages from the clients and then routes the messages to the destination clients having many advantages such as increased security eliminating insecure client connections and can easily scale from a single device to a large number of devices.
A client is any device that runs an MQTT library and connects to an MQTT broker over a network[1][3].
All the information is organized in a hierarchy of topics. Whenever a client publisher has a new data to distribute, it sends a control message with the data to the broker. The broker then distributes the information to any subscriber clients that have subscribed to that topic.

2.2 FIWARE

FIWARE is a curated framework of open source platform components to accelerate the development of smart solutions enabling the connection to IoT with Context Information Management.
The project will only make use of one FIWARE component - the Orion Context Broker[2].
Figure 2 shows that the Orion Broker depends on Mongo Database open source technology to maintain the consistency of the context data information it holds such as data entities, subscriptions and registrations.
This broker is an implementation of the Publish/Subscribe Context Broker GE, providing an NGSI interface. Using this interface, clients can do several operations such as query or update context information, register context provider applications and get notified when changes on context information take place.

Both broker installations were done in a Docker container that is a lightweight, executable and secure software package that includes everything needed to run a software unit.
Using the server in a Docker container and not locally has some advantages such as it does not affect other services that are running and also allows scalability.

3. EXPERIMENT SETUP

3.1 Hardware Setup

All the setup was implemented in Linux OS 20.04 where both brokers were installed. In the context of this experience it was also considered convenient to perform it in a closed environment.
The ESP8266 WiFi module produced by Adafruit was used as the IOT device and the temperature sensor was LM35DZ.

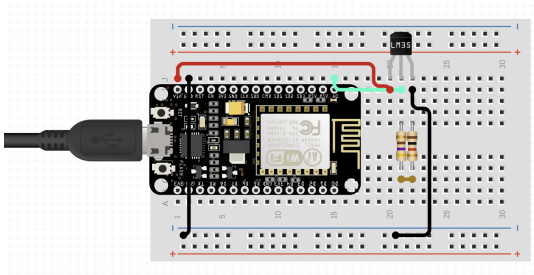

Fig. 1: Hardware setup configuration

3.2 Software Setup

Regarding the MQTT protocol was used Mosquitto which is an open source message broker.
Once the broker was installed it was necessary to configure the server so that the IP address of the server corresponds to the address of the computer that was doing the server.
Then it was necessary to define a topic to save the information regarding the temperature. After this topic definition it is now possible for a publisher to publish the content in that same topic and for a subscriber to receive that same information (if he has subscribed that same topic). The ESP8266 microcontroller was then defined as a publisher and as a subscriber a Python language script was developed to subscribe to the topic in order to receive the temperature values. To publish the temperature value was used the library ”PubSubClient.h” that provides a client for doing simple publish/subscribe messaging with a server that supports MQTT. To subscribe the topic corresponding to the temperature value the resources of the ”paho.mqtt.client” library were used.

Regarding the FIWARE, the Orion Context Broker which is one of many FIWARE components was used.
As previously mentioned, mongoDB which is a database that the broker uses to store all the information received was installed.
It was also necessary to define a data structure to store the temperature information, it should be remembered that this structure is database oriented, therefore a JSON structure was used.

To issue requests to the broker, we use the curl command line tool. This way it is possible for a publisher to publish the content in the Orion broker through an HTTP PUT request and for a subscriber to receive the information through an HTTP GET request. The creation of the entity is done through an HTTP POST request, in this case the entity is the temperature sensor.
In the publisher was implemented a INO script (software program created with Arduino) that publishes the temperature value by creating a HTTP PUT request in order to send the data to the broker.
From the subscriber’s side, a Python script was implemented that subscribes the information about the desired entity through HTTP GET requests.

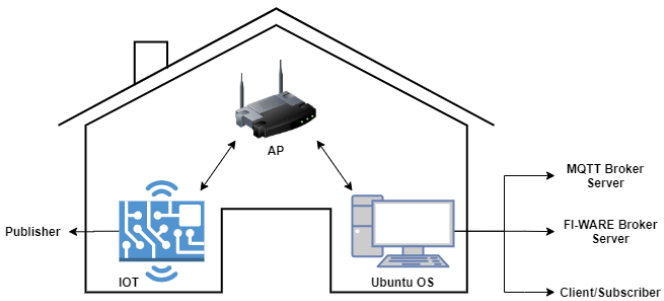

Fig. 2: Project setup configuration

4. EXPERIMENTAL RESULTS

4.1 Complexity and Message Size

As it is possible to check from the following table (Figure 3), the MQTT protocol has a total value of bytes lower than the FIWARE. MQTT has very short specifications when compared to FIWARE that uses HTTP requests like PUT and GET. In the first protocol there are only the CONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE and DISCONNECT types while the HTTP specifications are much longer[5].

Number of Bytes	MQTT - Mosquitto Broker	FI-WARE - Orion Broker
Establish Connection	376	178
Per message published	80	639
Disconnect	272	216
SUM	728	1033

Fig. 3: Message size sent during each of the transfer states

4.2 Measurement of delivery speed

According to the time values obtained in the traffic captures of the first experimental study (Chapter 4.1), the following table shows us the delivery speed values for both MQTT and FIWARE messages. The latency measurement was made based on the initial time the client makes the request for information until the moment he receives it.

	MQTT Delivery Speed (ms)	FI-WARE Delivery Speed (ms)
	0.101	2.98
	0.097	3.03
	0.106	2.98
AVERAGE RESULT	0.101	2.99

Fig. 4: Delivery speed values: MQTT broker vs FIWARE Broker

This leads to the conclusion that the delivery speed of MQTT is 29 times faster than FIWARE. Thus MQTT Protocol is preferable when response time and throughput are a priority.

Another study that was elaborated was taking into account the impact of virtualization on delivery time, having the MQTT on the local machine and then on a container.
The time values obtained are represented on figure 5.
Also here the latency measurement was made based on the initial time the client makes the request until the moment he receives the information.

	MQTT Delivery Speed on a local machine (ms)	MQTT Delivery Speed using a Docker container (ms)
	0.047	0.101
	0.013	0.097
	0.024	0.106
AVERAGE RESULT	0.037	0.1013

Fig. 5: Delivery speed values: MQTT local broker vs MQTT virtual broker

Thus, it is possible to conclude that the delivery speed of MQTT on a local machine is 2,73 times faster than MQTT using a Docker container.
Therefore server virtualization in a container is also a good option instead of having the server running on our own machine since the differences in latencies presented are not significant.

5. CONCLUSION

Once the experimental tests have been conducted and taking into account all the difficulties and advantages registered during the execution of the project in relation to the two protocols, it can be stated that MQTT is better in situations where the delay of the message is important since it provides smaller latencies.
In an earlier part of the project we noticed that MQTT was easier to work with and has an easier installation process because over the internet there is a lot of more information about it while FIWARE only had the broker documentation and a few examples of JSON structures.
Thus we concluded that MQTT Protocol is preferable when response time and throughput are a priority while FIWARE is better on the organization of the data since it is database oriented.

REFERENCES

1. AFRUIT. Webpage. https://learn.adafruit.com/mqtt-adafruit-io-and-you/overview.
2. FIWARE. Webpage. https://fiware-orion.readthedocs.io/en/master/.
3. MOSQUITTO. Webpage. https://smosquitto.org/.
4. WIKIPEDIA. Webpage. https://en.wikipedia.org/wiki/MQTT/.
5. YOKOTANI, T., AND SASAKI, Y. Comparison with http and mqtt on required network resources for iot. In *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)* (2016), pp. 1–6.

APPENDIX

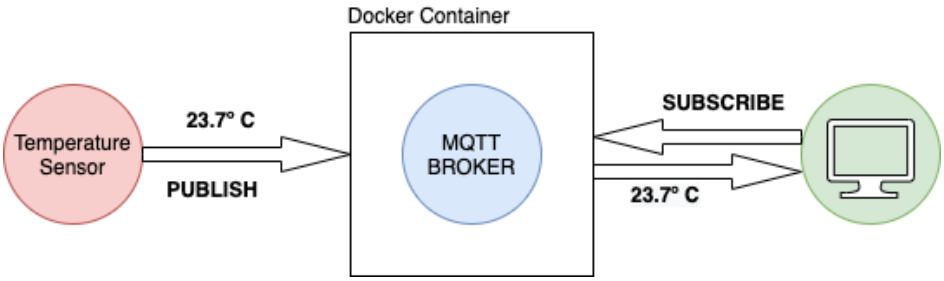

Fig. 6: MQTT Mosquitto Broker implementation

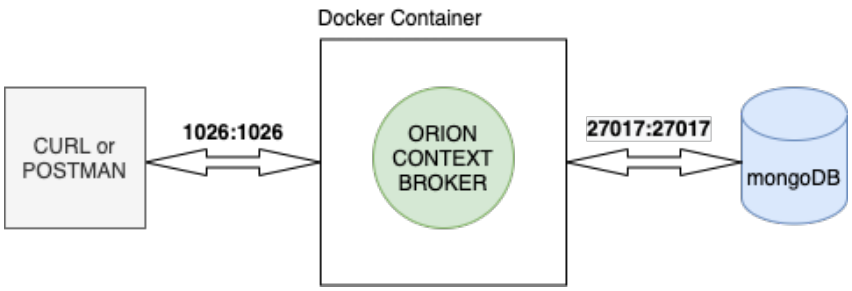

Fig. 7: FI-WARE Orion Broker implementation