Benchmarking IoT Middleware Platforms

João Cardoso*†, Carlos Pereira*†, Ana Aguiar*†, Ricardo Morla*‡
*Faculdade de Engenharia, Universidade do Porto, Portugal
†Instituto de Telecomunicações, Portugal
‡INESC TEC, Portugal
Email: jmmesquitacardoso@gmail.com, dee12014@fe.up.pt, anaa@fe.up.pt, rmorla@fe.up.pt

***Abstract*—Middleware is being extensively used in Internet of Things (IoT) deployments and is available in a variety of flavors – from general-purpose community-driven middleware and telco-developed Machine-to-Machine (M2M) middleware to middleware targeting specific deployments. Despite this extensive use and diversity, little is known about the benefits, disadvantages, and performance of each middleware platform and how the different platforms compare with each other. This comparison is especially relevant to help the design and dimensioning of IoT infrastructure. In this paper, we propose a set of qualitative dimensions and quantitative metrics that can be used for benchmarking IoT middleware. We use the publication-subscription of a large dataset as use case inspired by a smart city scenario to compare two middleware platforms. The methodology enables us to systematically compare the two middleware platforms. Further, we are able to use our approach to identify inefficiencies in implementations and to characterize performance variations throughout the day, showing that the metrics may also be used for monitoring.**

I. Introduction

The promises brought forth by Internet of Things (IoT) have fueled the deployment of large scale sensing and actuating infrastructures in diverse areas of application, like smart cities, logistics or healthcare. Middleware platforms are intermediaries between sensors, services, and applications, managing the flow of data and allowing them to interoperate. Different flavors of middleware are being used in IoT deployments to speed up deployment, by providing a set of common functionalities, and to allow interoperability between devices and services/applications that consume the data and make it useful [1]. Because data flows through the middleware at all components of the system, a particular implementation of the chosen middleware may not provide the features that a given deployment requires or may have a detrimental impact on the performance of the applications. Despite this, we have failed to find a systematic study and comparison of the performance, benefits, and disadvantages of the different middleware platforms that can be used in IoT.

Our goal is to define a set of qualitative and quantitative dimensions along which middleware platforms that allow the development of IoT applications and services can be compared. To achieve this, we chose a typical IoT scenario, smart cities [2], for identifying communication models and load scenarios. A typical communication model in this context is publish-subscribe [3] for accessing dynamic data as it allows greater scalability and flexibility than the request-response model. Load varies greatly depending on the process and type of data being sensed. We consider the motivating use case of the periodic publication-subscription of a large dataset. The data that our application publishes is the average speed of traffic in each street of the city of Porto in a hourly basis, corresponding to 19998 data points every hour, e.g., as would be required for a routing service using real-time data. This data might be extracted from a wide range of the mobile and static sensors spread throughout the city, and mapped to the edges of the OpenStreetMaps' city graph. The qualitative and quantitative analyses proposed can provide insights on which middleware is better suited for each scenario, bringing rational arguments to the problem of choosing which middleware platform to use.

In this paper, we perform a qualitative and quantitative evaluation of two middleware platforms using the load described. We consider the middleware platforms as black boxes. We use them only passively in the measurements and we do not use information about internal implementation or behavior on the analyses. We believe that this will be the most common situation, with platforms being owned and operated by third parties, as seen from the application developers or sensing/actuating operators. The qualitative analysis identifies middleware functionalities and characteristics relevant for an IoT application. The quantitative evaluation defines a set of performance metrics on specific communication scenarios. Further, we develop a toolbox to automate the collection of the relevant metrics that quantify performance.

We apply the methodology to two different middleware platforms (FIWARE[1] and ETSI M2M[2]). Both middleware platforms are based on HTTP requests, are RESTful [4], and rely on a broker. We chose these platforms as use case because we had access to both. We explore the publish-subscribe use case described above, as it is more challenging than request-response. Despite the concrete instantiation, the proposed methodology can be applied to benchmark other middleware

This article is a result of the project NanoSTIMA – Macro-to-Nano Human Sensing: Towards Integrated Multimodal Health Monitoring and Analytics, NORTE-01-0145-FEDER-000016, supported by Norte Portugal Regional Operational Programme (NORTE 2020), through Portugal 2020 and the European Regional Development Fund.

978-1-5386-2723-5/17/$31.00

[1] https://www.fiware.org
[2] http://www.etsi.org

platforms for different use cases and load.

The proposed analysis supports decision making by enabling systematic comparison of middleware platforms and offering solid and comparative evidence on performance. In addition, the metrics proposed may be used to monitor operation of such platforms, by offering insights on the internal behavior, which may allow us to uncover problems with specific implementations.

II. Benchmarking IoT Middleware

A. Qualitative Dimensions

To define the qualitative dimensions along which to compare middleware, we took the perspective of an IoT infrastructure operator that would like to see his data used by services/ applications. In this sense, besides requirements analysis, it is also important to take into account aspects that foster the adoption of platforms by developers, like quality of documentation and support. Thus, we arrived at the following aspects:

- support for the desired communication model (pub-sub, request-response, ...);
- IoT application requirements, based on the requirements defined by the IoT-A[3];
- availability and clarity of the documentation, as well as available tutorials;
- quality of the support and livelihood of developer communities, e.g., Stack Overflow[4].

B. Quantitative Dimensions

To evaluate their performance, we chose speed and efficiency. As speed, we mean the time to send or retrieve data. The efficiency measures the overhead imposed by the middleware, including the increase in the size of the data sent through the network and the total number of bytes needed to send the data.

We propose the following metrics:

- publish time: elapsed time between sending the publish HTTP request and receiving the publish HTTP response; t_{PUB} in Figure 1;
- subscribe time: elapsed time between sending the publish HTTP request and receiving the subscribe notification; t_{SUB} in Figure 1;
- total time: elapsed time between sending the first publish HTTP request and receiving the last publish HTTP response;
- size of marshalled data: data serialization overhead measured as the content-length [5] header of the HTTP protocol, in bytes;
- size of the publication: size in bytes of the TCP payload of the publish HTTP packet;
- total amount of data used to publish a resource: measured as the sum of the network level sizes of all packets exchanged (publisher to broker and reverse direction), in bytes;

[3]http://www.meet-iot.eu/iot-a-requirements.html

[4]http://stackoverflow.com

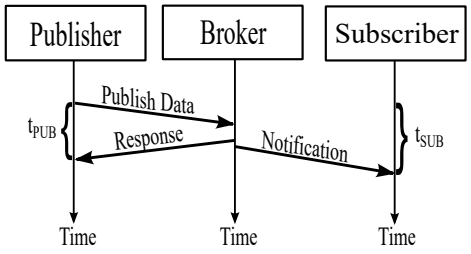

Fig. 1. Sequence diagram exemplifying the publish and subscribe times.

- goodput: the useful number of bytes sent in a given period of time, measured as the size of the serialized data divided by the publish time.

During analysis of performance results, we defined an additional set of metrics which is useful to validate results and provide support for explaining behaviors observed in the main metrics. These auxiliary metrics are:

- number of HTTP request retries, in cases where the publish HTTP request fails due to a problem with the broker, which accounts for errors in the 500 range HTTP status code [6] or where the TCP RST flag is sent;
- round trip time to broker [7], obtained through the difference between the timestamp of the TCP SYN packet and its response;
- number of TCP and HTTP packet re-transmissions [7] and the delay verified for the re-transmission.

By combining main and auxiliary metrics, we can infer whether:

- an increase in the publish or subscribe times is related to an increase in the round trip time (network congestion), or to reduced broker performance;
- an increase in the total time to publish all data is related to an increase in the number of HTTP retries, packet re-transmissions, or both.

III. IoT Middleware Platforms

A. FIWARE

The FIWARE middleware has a series of components known as Generic Enablers. These components aim to ease the development of complex applications in areas including security and data management. One of the data management components is the Orion Context Broker. Orion uses the publish-subscribe model and generic data structures, known as Context Elements, in order to represent information. This information can be, for example, the temperature of a given room. These context elements are represented using JSON, and they have a predefined structure (NGSIv2). This platform is RESTful, and therefore all operations use one of the HTTP CRUD methods. The broker has two APIs, v1 and v2[5]. The APIs differ in the data structure and in the fact that only the second version allows string or geographical filters to be applied to entity queries. The latter is a useful feature for

[5]https://github.com/telefonicaid/fiware-orion

location-based services, as often used in smart city applications and in our use case.

Access control can be provided when Orion is combined with the Steelskin PEP[6] component, which is an authentication mechanism independent of the broker that verifies whether the client has permissions to access a resource by intercepting the request before it reaches the broker. The information verified in order to allow access is the following:

- an OAuth[7] token generated by the authentication server, which is generated once and then included in the requests via x-auth-token header;
- a ServiceId, obtained through the Fiware-Service header and that identifies the component protected by this mechanism;
- a SubServiceId, obtained through the Fiware-ServicePath header, that identifies futures sub-divisions of the service;
- the desired action.

B. ETSI M2M

Machine-to-Machine (M2M) communications allow wireless and wired devices and services to exchange or control information without the need for human intervention [8]. M2M communications are a key enabler of IoT by, for example, making data from several sensors available publicly or connecting devices and sensors to the Internet (IoT) in order to process the data collected by these.

The ETSI M2M standard is the reference for global and end-to-end M2M communications in terms of service-level. It settles on an M2M architecture with a series of generic capabilities for M2M services, and it defines a resource model, easing the device's integration and interoperability, as well as the development effort of horizontal applications. On the other hand, it is easier to develop M2M applications due to the homogeneity the standard provides. Without standards, the interaction between objects using M2M applications would be harder.

ETSI M2M is RESTful, thus it adheres to the REST principles, and uses HTTP methods, defining how communications are made and how the information is represented with this architecture. The information is represented on the form of resources that are structured in a tree-like way [9].

ETSI M2M allows for synchronous communications, using the request/response communication model, and asynchronous communications, using the publish-subscribe model.

Finally, ETSI M2M does not have any reference implementation, as it is only defined by the standard; hence it is necessary to implement the middleware (broker and library/API).

IV. Qualitative Analysis

A. Communication Model

FIWARE's Orion Context Broker implements the publish-subscribe model. We also used the ETSI M2M broker and library [10], which implements the main features of the ETSI M2M standard, including the publish-subscribe communication model.

B. IoT-A Requirements

We do not have space to go over all IoT-A requirements. However, we identified the following as relevant for an architectural analysis: UNI.001, UNI.002, UNI.005, UNI.008, UNI.016, UNI.022, UNI.023, UNI.030, UNI.036, UNI.047, UNI.048, UNI.067, UNI.071, UNI.073, UNI.092, UNI.094, UNI.240, UNI.245, UNI.405, UNI.406, UNI.426, UNI.607, UNI.608. These requisites concern interoperability at protocol and data level, security, access control and anonimity, name-based access and self-description, support for queries (semantic, location). Of these, FI-WARE does not fulfill UNI.405 (support for multiple coordinate systems), and ETSI M2M does not fulfill UNI.016 (support geographic coordinates), UNI.240 (unified query interfaces), UNI.405 and UNI.406 (support for geographic queries).

C. Viability and Limitations

FIWARE's Orion Context Broker imposes a 1 MByte limit on the publish request size and 8 MByte limit on the subscription/notification size. Our example dataset contains 19884 data points. It is not feasible, or desirable, to publish all this data as a single resource. On one side, the notification size sent to subscribers would exceed the limit. But more importantly, filter queries would not be possible, and each subscriber would only have the possibility to receive the whole dataset. This might not be desired, e.g., for mobile applications. We thus decided to publish a single resource per street (graph edge). We defined the following structure for the resource name: *average_speed_edgeId*. This way we can apply regular expressions to select all the resources whose name starts with 'average_speed' (*average_speed_.**) and then apply filters that, for example, return the edges where the attribute 'speed' is higher than a given value. We also use this mapping in the ETSI M2M middleware.

D. Documentation and Support

FIWARE is an open standard with a reference implementation and thus it has a variety of online documentation. The documentation includes an API walkthrough that describes each version of the API, detailing each operation that the Orion Context Broker component supports, as well as explaining how to set up an instance of the broker and how to run it. Furthermore, each version of the API has a specific web page directed at detailing each operation and also offering code snippets of these operations in several programming languages. As the second version is still in beta state, there are two web pages for this version, one with the latest changes and the other with a stable version of the API.

There are various documents specifying all aspects of the ETSI M2M standard, such as the binding with protocols like HTTP or its functional architecture [9]. Unlike FIWARE, ETSI M2M only defines the standard and it does not offer any reference implementations. Therefore, it is necessary to

[6]https://github.com/telefonicaid/fiware-pep-steelskin

[7]http://oauth.net

implement the standard, a highly complex and time consuming task, or resource to a specific implementation, which might not be fully compliant.

In terms of support, FIWARE has a community on the Stack Overflow web site with 380 questions tagged at the moment of writing this paper. There is also a FAQ (Frequently Asked Questions) available about the general FIWARE platform[8]. Additionally, there are mailing lists directed at each kind of problem, such as technical issues with some of the Generic Enablers, that was used to clarify some doubts in the obtained results. On the other hand, there is no ETSI M2M community on Stack Overflow nor any kind of platform to clarify doubts or offer insights about this middleware.

V. Quantitative Analysis

A. Benchmark Toolbox

We developed a toolbox in order to automate the collection of metrics for the quantitative evaluation. The toolbox has 5 components:

- a database, with PL/pgSQL scripts to generate the test dataset;
- the publisher, which publishes the data and calculates metrics at the application level, such as the publish time or the number of retries, and stores them on CSV files. This component is implemented in Node.js for FIWARE and in Java for ETSI M2M. Additionally, a packet capture process runs the tcpdump[9] tool in the background;
- the subscriber, which subscribes to the several entities being published and receives notifications whenever new data is published. It creates a CSV file with the metric that this component collects, namely the time when each notification is received;
- the pcapAnalyser, which parses the packet captures made in the publisher and subscriber components, extracts metrics at the network level and stores them in CSV files;
- the finalLogger, which aggregates all these CSV files in a single file so that the metrics can be easily analyzed.

B. Setup

The experiments were done on a DigitalOcean virtual machine located on their London data center and running Ubuntu 14.04 x64, on a two core *Intel(R) Xeon(R) CPU E5-2650L v3 @ 1.80GHz* with 2 Gbytes of RAM, and a 1 Gbit/s Ethernet card shared between the virtual machines running on the same physical machine.

We used a FIWARE account on the public Barcelona FIWARE broker. This broker was chosen as it is similar to a production one and no production brokers were available to us. This broker does not implement HTTPS.

The ETSI M2M broker is available to our group in the Faculty of Engineering in Porto. The broker implements HTTPS; due to this, the payload TCP size of the TLS Application Data packet is evaluated instead of the payload TCP size of the HTTP protocol.

C. Methodology

Publishing a large dataset as individual data elements can be done in two ways: sequentially, or in parallel. For sequential publication, we performed 4 publish cycles as the amount of time required to perform each one limited the total number of measurements. For the parallel publication, each experiment consisted of 10 publish cycles with a number of parallel requests ranging from 50 to 500, in increments of 50. We limited the number of parallel requests to 500 because brokers were not able to cope with more parallel requests. This variation on the number of parallel requests allowed us to observe differences in the behavior of brokers, as the load on the broker increases, including the number of retries, the publish-subscribe times, and the number of packet re-transmissions. FIWARE experiments were done in the morning, afternoon, and night. In each part of the day, we repeated each experiment 3 times. In the case of ETSI M2M, only one experiment was done in the whole day, since the ETSI M2M's excessive publish time did not allow for more than 4 publish cycles in each part of the day. No synchronization was necessary between publisher and subscriber as they were located in the same machine.

D. Measurement results

1) Parallel Publication: ETSI M2M publish, subscribe, and total publish times were approximately 850% greater than FIWARE times. We believe it was caused by an error in the implementation of the ETSI M2M library we used that did not allow for more than 2 network connections at the same time. As such, it was not possible to publish more than 2 resources at the same time, no matter what number of parallel requests was chosen. This also led to the absence of retries.

Publish times for both middleware platforms can be observed in Figures 2 and 3. The ETSI M2M average publish times for a single resource publication range from 5992 ms to 72858 ms for 50 and 500 requests in parallel, respectively. This is much larger than FIWARE publish times ranging from 630 ms to 7400 ms, again for 50 and 500 requests in parallel, respectively.

The subscribe times are very similar to the publish times. A difference is that the ETSI M2M broker we used always sends the subscription notification before sending the HTTP response to the publish request. This results in a scalability problem when there are several subscriptions to a given resource, since the broker will send all the subscription notifications first before sending the HTTP response to the publish requests, and, ultimately, leading to larger than necessary publish times. On the contrary, FIWARE sends the subscription notifications asynchronously, and therefore sometimes the subscription notification arrives before the HTTP response to the publish request and at other times it arrives after. This behavior is illustrated in Figures 4 and 5 that show the differences between publish and subscribe times for both middleware platforms.

[8]http://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE_Frequently_Asked_Questions_(FAQ)

[9]http://www.tcpdump.org

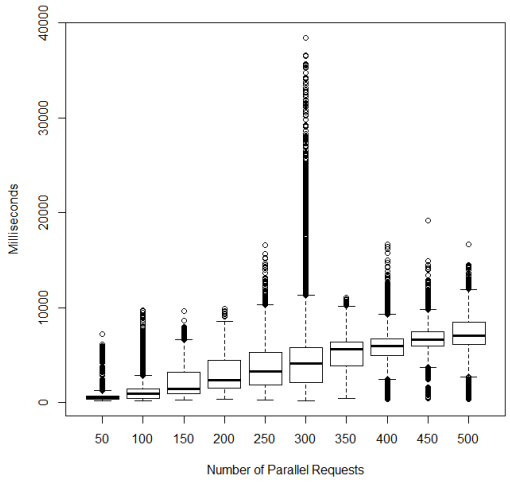

Fig. 2. FIWARE publish times for different number of parallel requests.

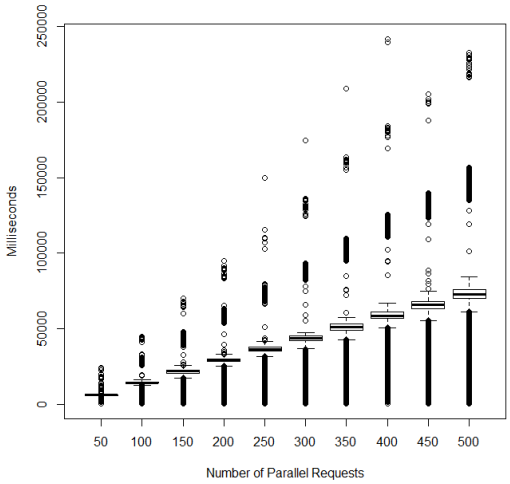

Fig. 3. ETSI M2M publish times for different number of parallel requests.

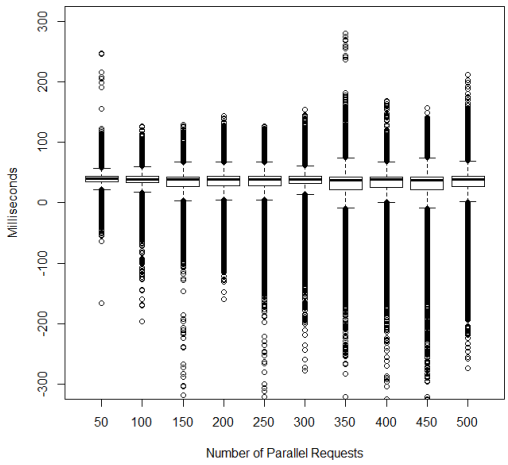

Fig. 4. Observed difference between FIWARE publish and subscribe times for different number of parallel requests.

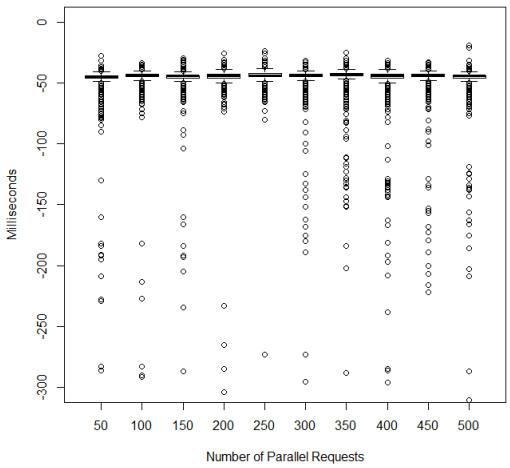

Fig. 5. Observed difference between ETSI M2M publish and subscribe times for different number of parallel requests.

The average content-length for ETSI M2M is 390 bytes, 55% greater than that of FIWARE which is 251 bytes. This is unexpected as FIWARE data structure format is JSON which tends to be lengthy. We believe the reason behind this larger than expected ETSI M2M content-length is another implementation inefficiency of the library: we verified that the ETSI M2M library sends unnecessary attributes in the payload, such as the content-type which should only be in the headers of the HTTP packet. The data itself only accounts for 146 bytes, which is lower than in FIWARE that requires the JSON overhead.

An average of 511 bytes of TCP payload size was observed for FIWARE, whereas for ETSI M2M this value was 728 bytes; we did not include the 40 bytes average TLS overhead in this value. This difference in size is due to the increase in size of the content-length, already discussed above.

ETSI M2M registered an average of 1118 bytes for the HTTP response to the publish request, whereas FIWARE registered an average of 417 bytes. There is quite a difference in size between these two middleware platforms, and that again is due to the specific implementation we used, since the response to the publish request of the ETSI M2M library is not a simple OK with no payload, but instead contains a copy of the published data which only adds inefficiency to the communication.

The goodput measured in ETSI M2M was on average 2662 bytes/second, while in FIWARE it ranged from 351 bytes/second (lowest value registered) to 1700 bytes/second (highest value registered). This is due to the fact that the ETSI M2M library we used does not allow for more than 2 network connections active at the same time, and therefore the goodput is higher because the lower number of active requests there is the higher the rate at they are processed. These values are depicted in Figures 6 and 7. We observed that the number of retries in the FIWARE experiments grew throughout the day, as can be observed in Figure 8. Therefore, a hypothesis was formulated for the existence of a memory leak in the broker. In order to verify this hypothesis, additional measurements

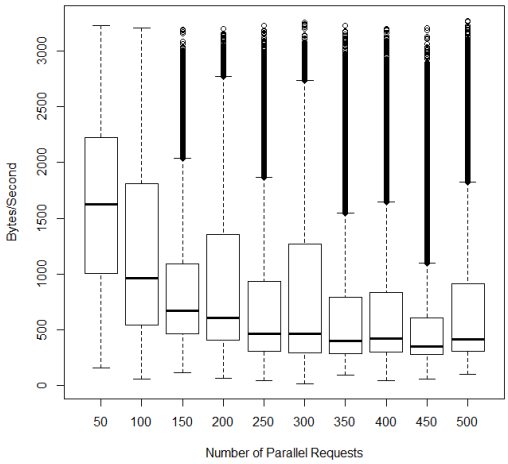

Fig. 6. FIWARE goodput for different number of parallel requests.

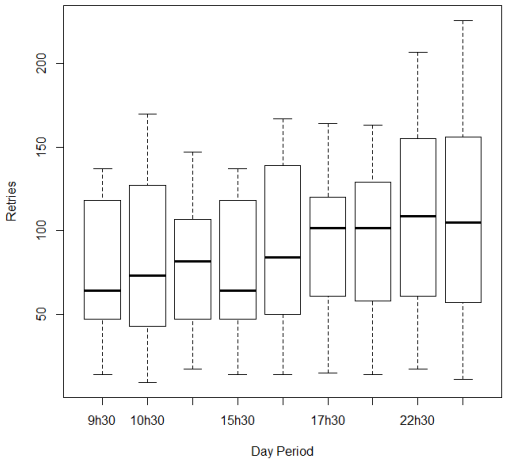

Fig. 8. Number of retries in FIWARE throughout a day.

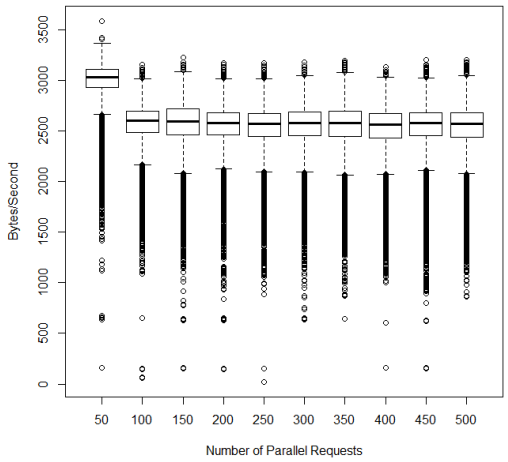

Fig. 7. ETSI M2M goodput for different number of parallel requests.

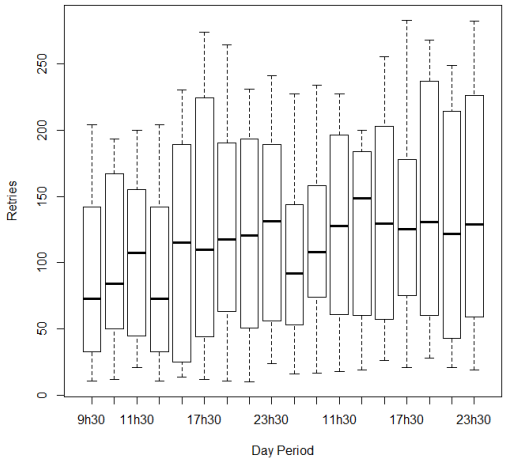

Fig. 9. Number of retries in FIWARE throughout two additional days.

were conducted on two consecutive days, and if the number of retries grew throughout these two days, then this hypothesis would be verified. According to the results obtained, the retries grew throughout the first day, but in the second day they did not pick up from the last value registered on the previous day. Instead, we observed the same growing pattern as in the day before, that is, it started low and grew as the day progressed, as shown in Figure 9. As a result, we updated our hypothesis and assumed that the broker had been restarted somewhere in the period where no measurements were taken (00:30 - 09:30). In order to confirm this new hypothesis, we contacted the FIWARE team via email and were told that the retries were due to the authentication proxy - Steelskin PEP, mentioned in Section III - and not due to the broker load itself, as was suspected. The FIWARE team also told us that they were not aware of any broker malfunction, restart, or memory leak.

2) Sequential Publication: We observed differences for both middleware platforms between this and the parallel publication scenario, though expected. The sequential mechanism had implication in the publish time, subscribe time, and total time to publish all data, as well as in the goodput. Publish times were smaller in the sequential publication for both middleware platforms. FIWARE's average publish time was 175 ms, while ETSI M2M's was 282 ms which was almost 61% greater than measured in FIWARE. Subscribe times were also smaller for both middleware platforms in the sequential publishing. FIWARE's average subscribe time was 227 ms with a performance similar to ETSI M2M with 237 ms.

ETSI M2M performed considerably better for sequential publication, as its implementation limited the performance in parallel publication. This scenario allowed a fair comparison between both middleware platforms; however, the results show that FIWARE performs better in overall. Figures 10 and 11 show the publish time and the differences between publish and subscribe times for FIWARE, respectively.

The total time to publish all data was 48% greater in ETSI M2M than in FIWARE. More remarkably is the increase of 1180%, when compared to the parallel publication, observed in FIWARE which took nearly 64 minutes to publish all data. Although smaller, there was also an increase of 98% in the

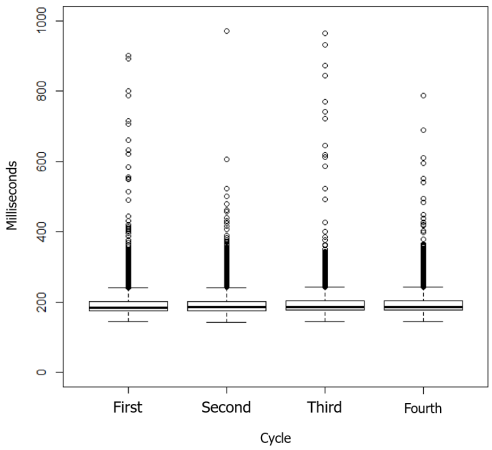

Fig. 10. FIWARE publish times for each sequential publish cycle.

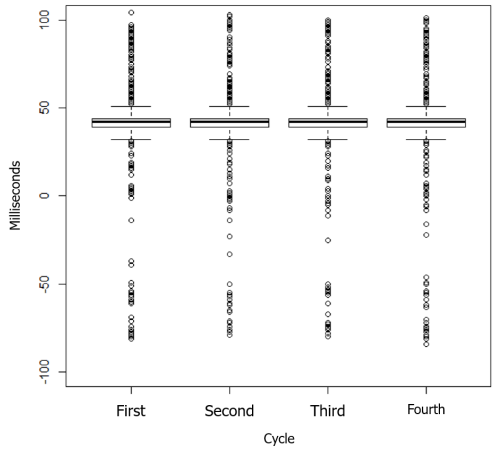

Fig. 11. Observed difference between FIWARE publish and subscribe times for each sequential publish cycle.

total time to publish all data, when compared to the parallel publication, in ETSI M2M that took nearly 98 minutes. The parallel publication is thus the recommended choice for the timely dissemination of data.

The goodput measured in ETSI M2M was on average 1444 bytes/second, while in FIWARE it was 2660 bytes/second. Both middleware platforms show different behaviors in terms of goodput when compared to the previous scenario which can be once again due to differences in the implementation.

VI. Conclusion

In this paper, we proposed a set of qualitative dimensions and quantitative metrics to compare IoT middleware. We used them together with a smart city use case with data from the Future Cities project to evaluate the performance of two well-known middleware platforms: FIWARE and ETSI M2M. For two publish-subscribe scenarios in which a user publishes large volumes of data, either in a parallel way or in a sequential way, we measured larger publish times, subscribe times, and total times to publish all data with ETSI M2M than with FIWARE. We concluded that the significant differences between the two middleware platforms observed in the parallel publication were mainly due to limitations of the ETSI M2M's implementation. On the other hand, we also observed performance variations throughout the day in FIWARE. We also concluded that parallel publication can significantly reduce the total time required to publish all data when compared to publish in a sequential way.

The results allowed us to have a better understanding of the internal functioning of these middleware platforms, showing that the metrics proposed may be used to monitor their operation, and, by discovering implementation errors in the middleware itself or in their libraries, they allowed us to contribute for possible improvements. For instance, the FIWARE's authentication proxy returned HTTP errors in the 500 range, when supposedly it should not according to FIWARE's team. The ETSI M2M middleware used had several implementation problems, namely: the subscription notifications were always sent before sending the HTTP OK response to the publish request, which can create a scalability problem in cases of large number of subscriptions for a given resource; the publish request payload contains superfluous information in addition to the data to be published, such as the content-type attribute which should only be defined in the request's header and was wrongly included in the request's payload; and the response payload relative to the publish request was bloated, as it returned superfluous information, like the data being published, which is not needed and only adds inefficiency to the communications.

Future work should focus in benchmarking and comparing more middleware platforms in different use cases, but following the same methodology and metrics.

References

[1] Philip A. Bernstein. "Middleware: A model for distributed system services". Communications of the ACM, 39(2):86–98, February 1996. doi:10.1145/230798.230809.

[2] Andrea Caragliu, Chiara Del Bo and Peter Nijkamp. "Smart cities in Europe". Journal of Urban Technology, 18(2):65–82, 2011. doi:10.1080/10630732.2011.601117.

[3] Patrick TH. Eugster, Pascal A. Felber, Rachid Guerraoui and Anne-Marie Kermarrec. "The many faces of publish/subscribe". ACM Computing Surveys, 35(2):114–131, 2003. doi:10.1145/857076.857078.

[4] Roy Thomas Fielding. "Architectural Styles and the Design of Network-based Software Architectures". PhD thesis, 2000. doi:10.1.1.91.2433.

[5] W3C. HTTP/1.1: Header Field Definitions. Available in https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html, accessed in October 2016.

[6] W3C. HTTP Status Code Definitions. Available in https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html, accessed in October 2016.

[7] Douglas Comer. "Internetworking with TCP/IP. Prentice Hall", Fourth edition, 2000.

[8] Carlos Pereira and Ana Aguiar. "Towards efficient mobile M2M communications: Survey and open challenges". Sensors, 14(10):19582–19608, 2014. doi:10.3390/s141019582.

[9] ETSI. ETSI TS 102 690 V2.1.1 (2013-10) - Machine-to-Machine communications (M2M); Functional Architecture, 2013.

[10] Carlos Pereira, Antonio Pinto, Ana Aguiar, Pedro Rocha, Fernando Santiago and Jorge Sousa. "IoT interoperability for actuating applications through standardised M2M communications". IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2016. DOI:10.1109/WoWMoM.2016.7523564.