Lessons Learned and Challenges on Benchmarking Publish-Subscribe IoT Platforms

Ana Aguiar
University of Porto, Instituto de Telecomunicações
Porto, Portugal
ana.aguiar@fe.up.pt

Ricardo Morla
University of Porto, INESC TEC
Porto, Portugal
rmorla@fe.up.pt

ABSTRACT

The Internet of Things (IoT) emerged simultaneously in various research fields and application domains, leading to the appearence of a multitude of hardware, software technologies, horizontal platforms and data models. Lost in this diversity, adoption decisions are made to a large extent based on familiarity with a technology, or because there is a broad community support for it. In this context, bechmarking different technologies at different horizontal levels would provide a more solid justification for the adoption of a specific technology in a specific context. In this paper, we reflect on our previous work on benchmarking publish-subscribe IoT platforms as a middleware for IoT applications, reporting lessons learned and identifying challenges, thus contributing to the discussion on open topics and relevant downstream work.

CCS CONCEPTS

• **General and reference** → **Performance**; *Evaluation*; *Experimentation*;

KEYWORDS

Internet of Things, benchmarking, platforms

ACM Reference Format:
Ana Aguiar and Ricardo Morla. 2018. Lessons Learned and Challenges on Benchmarking Publish-Subscribe IoT Platforms. In *CPS-IoTBench 2019: 2nd Workshop on Benchmarking Cyber-Physical Systems and Internet of Things, April 15, 2019, Montreal, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As the Internet of Things (IoT) emerges in various research fields and application domains, little is known about the benefits, disadvantages, and performance of each IoT technology or specific implementations. Benchmarking different technologies at different horizontal levels would provide a more solid justification for adoption in a specific context. In prior work, we followed a systematic approach at benchmarking different IoT publish-subscribe middleware platforms motivated by a smart city scenario. In this paper, we make a meta-analysis of that work, highlighting lessons learned from these benchmarking efforts that . Although other benchmarking tools for publish-subscribe [3] and IoT middleware [6] [7] exist, we emphasise publish-subscribe communication middleware as interoperability enabler for composing IoT applications from heterogeneous components. Our systematic approach and the use of an archetypal IoT scenario makes us confident that the lessons learnt are valuable and to some extent can be generalised, providing useful end-yo-end system insights that speed up development and make the learning curve less steep to performance evaluators.

We cover different challenges identified supported by illustrative examples from our work, whereby the results are not important per se, but serve as illustration and support for the points we make in Section 8. We start by discussing important quantitative dimensions in Section 3 and metrics in Section 4, and explore initial results to show relevant observations. Section 6 follows from the duality we experienced between using real deployments on the wild or using laboratory settings with comparable computing infrastructure and network conditions [1]. Section 7 shows how we addressed the problem of abstracting components to reduce the cost of implementing a new platform and further improve comparability [5]. We conclude the paper with a summary of lessons learned and other challenges in Section 8. Although our examples use only two publish-subscribe IoT platforms, we believe the lessons we learned can be applicable to other publish-subscribe IoT platforms and other massive sensing IoT scenarios.

2 SCENARIOS AND WORKLOAD

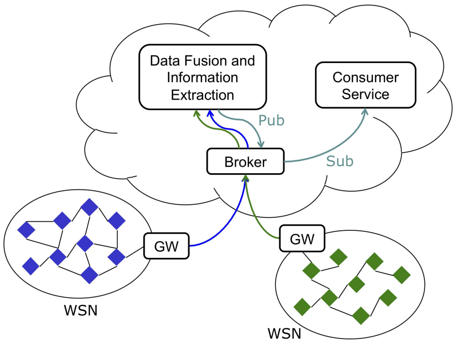

Figure 1: A massive sensing IoT scenario

One challenge of benchmarking IoT systems is to have a shared understanding of scenarios and benchmarked IoT systems. This is important when replicating results from others and making sure we are comparing apples to apples.

CPS-IoTBench 2019, April 15, 2019, Montreal, Canada

ACM 978-1-4503-6693-9/19/04…$15.00
https://doi.org/10.1145/3312480.3313174

We focus on an archetypal case of publication-subscription of massive amount of data, motivated by a smart city application, e.g. adaptive vehicle routing. This scenario is depicted in figure 1. Heterogeneous sensor networks collect massive amounts of heterogeneous data, e.g. inductive loop counts, buses and taxis positions, or their observed travel time statistics. A data fusion and information extraction service must subscribe to the various data sources to provide useful information, e.g. to predict the travel time in each edge. Then, it publishes the processed data for a wide geographic ares every x minutes. A mobility service, e.g. adaptive routing, then subscribes to this data. Publish-subscribe middleware provides interoperability among these different IoT application components, both at the protocol and data layers.

The illustrative question that motivated our work was which IoT middleware on the path to standardisation, FIWARE[1] or ETSI M2M[2]/ OneM2M[3], would be the most appropriate to support this application. The load, which was constant throughout our efforts, was the publication of 20 thousand [4] 4 byte values.

Another scenario for which it should be sensible to benchmark IoT systems is where data is not massive but delay sensitive. That scenario is out of scope of this paper, but was previously explored in [2].

3 QUALITATIVE DIMENSIONS

When presented with different sets of capabilities of competing platforms, qualitative metrics can be used to assess how well they comply with *functional requirements*. The most systematic part of these metrics rely on the extensive framework of IoT-A[5]. Relevant requisites within this framework concern interoperability at protocol and data level, security, access control and anonymity, name-based access and self-description, and support for queries (semantic, location) [4].

Additionally, it is important to considered dimensions related to the *support to developers*: availability and clarity of the documentation, available tutorials, quality of the support and livelihood of developer communities, like StackOverflow[6]. These dimensions are not technically relevant, but play a big role in the learning curve and maintainability of the implementation. Standardisation efforts are of major importance for sustainability of the platforms.

Our qualitative analysis showed that FIWARE does not fulfill IoT-A 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). We also found out that FIWARE's Orion Context Broker imposes a 1 MByte limit on the publish request size and 8 MByte limit on the subscription/notification size.

4 QUANTITATIVE METRICS

Quantitative performance metrics measure speed and efficiency of the platforms, whereby here the specific implementations must be considered. As speed, we mean the time to spread data among subscribers. Efficiency measures the overhead imposed by the platform, 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 defined metrics at different levels of the stack [4]: publish and subscribe times, as defined in figure 2; time between first publication and last confirmation; size of marshalled data; size of TCP payload; total amount of data to publish a resource, measured at network level. One challenge is to understand how rich this set is, how relevant each of the metrics is for different benchamrking goals, workloads and scenarios, and which sets of metrics should be added to cover other areas like power and security.

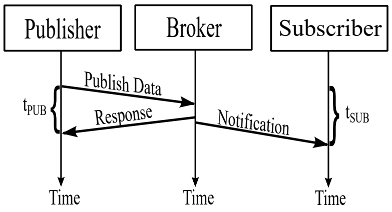

Figure 2: Publish and subscribe time metrics.

5 ETSI M2M VS. FIWARE

We summarise here the initial results obtained from observing the defined metrics for the two IoT platforms for brokers not under our control—*on the wild*. These results have the purpose of illustrating the insights that can be obtained from such an experiment, rather than the actual values.

Publishing a large dataset as individual data elements can be done in two ways: sequentially, or in parallel, and we tried both.

Parallel Publications. Our quantitative analysis showed that 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 used that did not allow for more than 2 simultaneous TCP connections. 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. Publish times for both middleware platforms can be observed in Figure 3. The ETSI M2M average publish times for a single resource range from 5992 ms to 72858 ms for 50 and 500 requests in parallel, respectively. In both cases, we observe outliers. These are likely due to queueing at the broker's database, because we are sending requests in parallel, but the corresponding database transactions do not occur in parallel. We also observe only excess outliers in FIWARE, but both types in ETSI M2M. Although it would have been interesting to look deeper into this, that was not possible in these experiments because we did not have access to the broker except via the public interfaces. We explore similar behaviours in our controlled experiments reported in the next section. 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 limited scalability when there are several subscriptions to a given resource, since the broker will send all the subscription

[1] https://www.fiware.org
[2] http://www.etsi.org
[3] http://www.onem2m.org
[4] This is the size of the Open Street Maps graph for the city of Porto, a middle-sized city.
[5] http://www.meet-iot.eu/iot-a-requirements.html
[6] http://stackoverflow.com

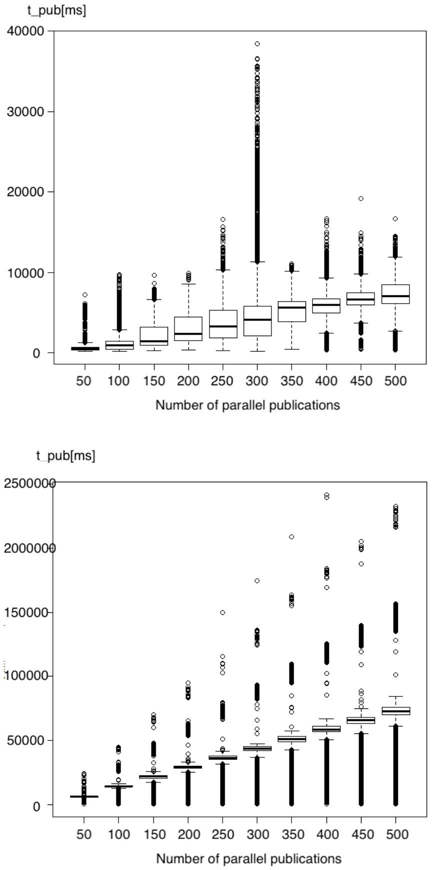

Figure 3: Publish times for different number of parallel requests. Top: FIWARE, bottom: ETSI M2M.

notifications before sending the HTTP response to the publish requests, 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.

Sequential Publication. Publish times were smaller in this case for both platforms. FIWARE's average publish time was 175 ms, while ETSI M2M's was 282 ms which was almost 61% greater. Subscribe times were also smaller for both when publishing sequentially. FIWARE's average subscribe time was 227 ms with a performance similar to ETSI M2M with 237 ms. Remarkably, FIWARE took nearly 64 minutes to publish all data, an increase of 1180% compared to parallel publication. ETSI M2M that took nearly 98 minutes, an increase of 98% in the total time to publish all data, when compared to the parallel publication.

Although there is a clear advantage of publishing large datasets using request parallelisation, this is not part of any client library and must be implemented by the developer.

Other Observations. 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 an implementation inefficiency: 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.

We also observed that the FIWARE's authentication proxy returned HTTP errors in the 500 range. According to the team operating the FIWARE broker, it should not have.

We observed that the number of retries in the FIWARE experiments grew throughout the day. We contacted the FIWARE team and were told that the retries were due to the authentication proxy—Steelskin PEP—and not due to the broker load itself.

Take Aways. The quantitative benchmark highlighted unexpected or undesired behaviour of specific implementations, rather than support for the choice of a middleware. Nevertheless, the undesired behaviour would otherwise only have been found after costly implementation efforts. Moreover, they provided many lessons about the platforms and IoT application caveats.

6 CONTROLLED VS. ON THE WILD

The quantitative results shown in section 5 were obtain by benchmarking broker deployments on the wild. This is important, e.g. to help in the process of deciding which vendor platform to plug your smart city nodes to. The brokers were installed in different locations, and were operated by different entities: public Barcelona FIWARE broker in Barcelona, vs a reference ETSI M2M broker installed in our premises but developed and operated by a telco. Thus, we defined several network metrics to support result interpretation: HTTP retries, round trip time to broker, TCP re-transmissions and delays. We are able to make an initial comparison among the two standard candidate platforms. We identified changes related to the different specifications, but also inefficiencies in implementations, and characterized performance variations throughout the day. The results showed variability that could not be explained because the brokers were running on different computing platforms and at different network distance. A challenge here is to identify methods for narrowing down causes for the variability of results, for example by additionally collecting traffic performance in the same path between the clients and the broker.

Benchmarking in a controlled environment is a different approach that provides the degree of freedom required to test and compare different configurations of the systems being assessed. In a controlled environment the variability related to the network and computing infrastructure is removed, and a fairer comparison is possible for protocols, marshalling and implementation options, e.g., the chosen database engine. It is also possible to select and compare alternative configurations at the broker. We used servers in our lab to install an Orion Context Broker instance to evaluate FIWARE's performance and the OM2M broker to evaluate oneM2M's performance. FIWARE uses HTTP as application protocol. The OneM2M contains bindings for HTTP, CoAP, and MQTT[7] application protocols. In this controlled environment, we were able to find that broker performance can depend on different components like the database and underlying communications protocol, but also on the verbs used. For example, FIWARE performance with the same load degrades with single versus multiple entities in the database and the OneM2M performance is much better with HTTP and CoAP protocols than with MQTT and its intermediate broker. Further, while the execution of several parallel publications can in fact decrease the overall transmission time, in average each publication will be delayed. We provide quantitative performance results for these conclusions in the rest of this section. Further details of this work can be found in [1].

6.1 Example – protocol performance

OM2M HTTP and OM2M CoAP have much better average publish times than FIWARE. FIWARE's publish times range from 21251 to 98,635 ms for 1 and 5 requests, respectively. The average publish times of OM2M CoAP are lowest, ranging from 5873 to 8696 ms. The use of CoAP represents an average decrease of 1.1% of the publish time when compared with the use of HTTP in OM2M, which shows small benefits of using UDP in addition to the smaller, binary base header format. The use of MQTT as application protocol, including the Mosquitto broker as proxy between the client and the OM2M broker, represents an increase of the publish time of nearly 9.6 times when compared with the use of HTTP and an increase of approximately 9.7 times when compared with the use of CoAP. The publish times for the platforms with the different configurations can be observed in figure 4.

The average total time to publish all data was nearly 395 s for FIWARE, 58 s for OM2M HTTP, 58 s for OM2M CoAP, and 567 s for OM2M MQTT. The latter represents an increase of nearly 8.8 times to the fastest (OM2M HTTP and OM2M CoAP). We observe that although the average publish time increases with the number of parallel requests, the total time to publish all data decreases, except for FIWARE. We discuss below the reasons for this not happening with FIWARE. We further observe that the use of Mosquitto considerably decreases the performance. Figure 5 shows the marshalling overheads, highlighting the efficiency of CoAP and MQTT vs HTTP.

6.2 Example – Impact of CRUD Methods

OneM2M complies with technical standards that strictly define how and where on the resource structure should be published. The flexibility in FIWARE allows us to compare three different approaches of publishing data using its Orion Context Broker, which we could not compare in the previous section on the wild. The first approach is the one we have been using up to here in this section: we previously create an entity for each edge and upon incoming edge data we update the values of all entities. In the second approach, upon incoming edge data we create a new set of entities—each entity with the new value for its edge. This requires additional mechanisms for entity cleanup and subscription. In the third approach, a single entity is used to publish new data for all edges. We update the edge id attribute together with edge data in the entity, once for every edge and upon incoming edge data.

The results are shown in figure 6. There is a clear decrease in performance for updating the values of several entities (first approach) and for creating entities (second approach) when compared to updating the attributes of a single entity (third approach). Publishing data using the third approach results in FIWARE having slightly better performance than OM2M HTTP and OM2M CoAP. We relate this variation in FIWARE's performance to the database. We observe that in the boxplots of the first approach there are no publish requests with small publish times. As we are doing updates to entities in a database with a rather large number of entities, each update takes longer, and more parallel requests mean larger times due to concurrency. In the second approach, we see that the boxplots have measurement points spread across the entire Y-axis. The database locks writing updates to the different entities. The requests that can access the lock to create a new entity will have low publish times; the rest of the requests must wait and experience larger publish times. As a final argument for the impact of the database on the FIWARE broker performance, we stress the fact that all reported measurements were done in a clear broker and empty database.

7 EASE OF INTEGRATION

Early work showed that ensuring common ground across different experiments is very cumbersome and error prone. To address these difficulties, we designed a tool that enables multiple comparisons across different platforms in an efficient manner. We propose a generic architecture from which will stem a modular tool in which the benchmarks can be run. The aim of the tool is twofold: to provide a common ground in which the different platforms can be benchmarked to ensure an equal playing field and replicability, and to reduce the cost of trying out subsequent platforms.

We created a modular architecture by factoring the common elements in the benchmarking applications, whose general plan can be seen in figure 7. A user should be able to swap instances of a block as necessary. To achieve this, we defined interfaces (inputs and outputs) for all blocks.

The *load block* will enable different types of IoT scenarios to be programmed and dynamically changed, so that we can attempt to mimic varied real world scenarios. Again, this should be totally independent from any other block so that the same workloads can be used throughout all middlewares and protocols, providing a basis for comparison and ensuring high flexibility.

The *data block* is where the middleware specific functions reside, and each of these is responsible for implementing its data structure and bridging the gap to the protocols. When a new middleware is to be tested, one can leverage the existing functions, thereby speeding up the implementation process. A new middleware will be added as a new instance of the *data block* so as to not interfere with the previous middlewares.

Next, we have the *communication block* where the protocols, such as HTTP, CoAP, MQTT, are lodged. Each has its methods implemented, e.g. POST or GET, so that they are middleware independent and can be reused. If a new protocol is required, its methods can be implemented without interfering with the remaining structure.

[7] The use of MQTT is only possible with the use of an external MQTT broker as intermediary. For that, we used the Mosquitto broker (https://mosquitto.org/).

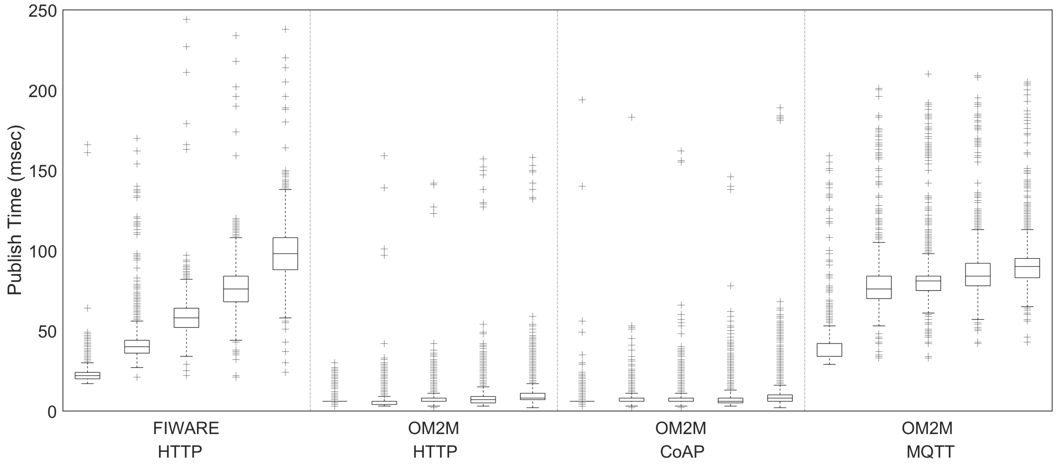

Figure 4: Publish times for different number of parallel requests. Number of parallel requests increases from 1 to 5 from left to right.

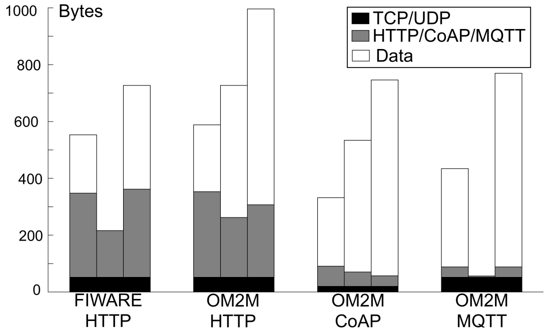

Figure 5: Publish request (left bar), response (center bar), and notification (right bar) sizes for the different configurations.

During the test cycle, a set of defined values, such as times and publish request sizes, will be stored and fed into the *metrics block*, which will extract metrics such as average publish time or generated traffic. New metrics can be added without affecting those that are already implemented, also keeping result compatibility.

We iteratively developed this tool, by implementing the benchmark for OneM2M, and then extending it to FIWARE, re-utilising as much as possible of the existing code. Later, we extended it to Ponte[8].

Class	Lines of code
Main	51
Load	119
Middleware	16
Metrics	136
Ponte	88
FIWARE	193
OM2M	198

Table 1: Lines of code for each class

We can attempt to quantify the impact of such a tool on the changes needed for a new implementation by looking at the lines of code. The distribution of lines per class can be seen in Table 1. This adds up to a total of 801. At first glance, we can see that for the implementation of a single platfrom the effort required varies, with Ponte only needing 88 lines of new code. Nevertheless, there is a great deal that can be reused across all of them, greatly easing the process. Not only are the structures and protocols similar, but some of the methods are similar as well, such as with the publish() methods between Ponte and FIWARE.

8 LEARNT LESSONS AND OUTLOOK

- It is important to understand communication protocol and data overhead as there are significant differences, especially for cyber-physical systems.
- IoT platform benchmarks need to consider parameters like: network latency at the time of the measurements; publish-subscribe protocol parameters like quality of service; database state and CRUD method; degree of parallelism while publishing data; infrastructure specification, including network configurations of the used machines and execution environments.
- Overall time for publication of a massive dataset is faster if publications are parallelised, because the network latency time can be filled with new requests while the broker is actually processing each request.
- Overall time for publication depends on how the dataset is mapped to the data structures of the IoT platform, because of the different database operations involved.
- Measuring the number of publications per unit time is very challenging and must be carefully interpreted. If measured on the lab it does not generalise to other deployments due to different computation capabilities. If measured on the wild, the impact of network throughput and latency variations cannot be easily de-correlated and requires additional measurements.

[8] https://www.eclipse.org/ponte/

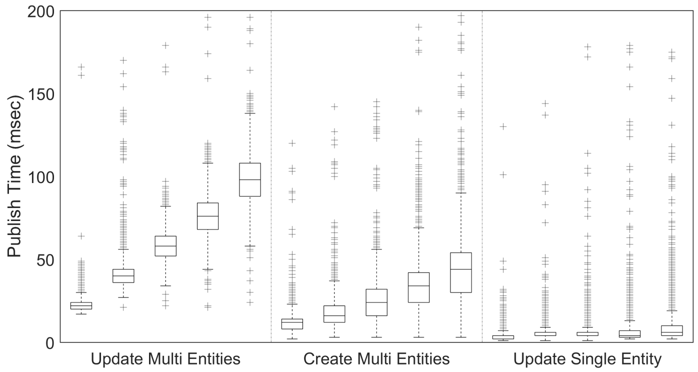

Figure 6: Publish times for FIWARE with three different ways to publish data and for different number of parallel requests (Left: 1; Right: 5).

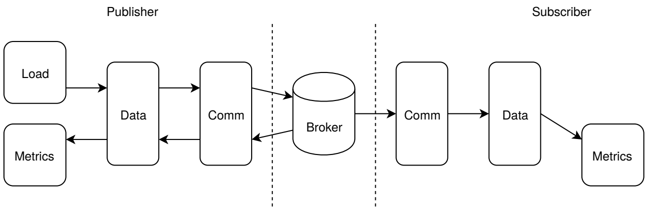

Figure 7: Main building blocks of the benchmarking tool.

- Protocol implementations play a significant role in the performance. On the one hand, benchmarking can highlight limitations; on the other, for benchmarking it is advisable to abstract from such details by using common protocol implementations.
- The broker is a key component of high complexity with significant impact on system performance.
- Measuring the broker as a blackbox has several caveats: an external entity can measure either a broker implementation in a lab, or a broker deployment on the wild, but the effect of the network cannot be removed and it is challenging to account for it. Only the operator of the broker can measure the broker deployment directly, and only with more work may we be able to infer broker characteristics from observed behaviour.
- Database is one major bottleneck of the broker.

Although we did not explore them in detail yet, we believe the following are relevant additional challenges for a credible benchmarking of IoT/CPS systems.

- Quality and diversity of the workload, specifically considering samples from real deployments and modeling, and IoT testbeds[9]
- Understand the impact of different platform on energy constrained nodes, by means of measuring and modeling node energy consumption
- Consider known vulnerabilities of libraries and components of the benchmarked system in the qualitative analysis
- Assess the impact of security protocol overhead in the performance of the IoT platform, and chart against security guarantees
- Assess how resilient the platforms are to side channel attacks on the communication patters that could endanger user privacy in privacy-sentitive scenarios

Our goal was to share our experience and know-how and raise awareness to caveats of benchmarking complex multi-layer computer systems as IoT enabled cyber physical systems are bound to be. We believe that our experience can fuel discussions about platform benchmarking and also about benchmarking other IoT-CPS components.

ACKNOWLEDGMENTS

This work is a result of the project MobiWise (POCI-01-0145/FEDER-016426), funded by FEDER, through COMPETE 2020 and national funds through Fundação para a Ciência e Tecnologia and UID/EEA/50008/2019 and UID/EEA/50014/2019.

REFERENCES

[1] A. Aguiar R. Morla C. Pereira, J. Cardoso. 2018. Benchmarking Pub/Sub IoT middleware platforms for smart services. *Springer Journal of Reliable Intelligent Environments* 4, 1 (April 2018), 2537. https://doi.org/10.1007/s40860-018-0056-3

[2] D. Ferreira A. Aguiar C. Pereira, A. Pinto. 2017. Experimental Characterisation of Mobile IoT Application Latency. *IEEE Internet of Things Journal* 4, 4 (April 2017). https://doi.org/0.1109/JIOT.2017.2689682

[3] Philippe Dobbelaere and Kyumars Sheykh Esmaili. 2017. Kafka Versus RabbitMQ: A Comparative Study of Two Industry Reference Publish/Subscribe Implementations: Industry Paper. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (DEBS '17)*. ACM, New York, NY, USA, 227–238. https://doi.org/10.1145/3093742.3093908

[4] R. Morla A. Aguiar J. Cardoso, C. Pereira. 2017. Benchmarking IoT Middleware Platforms. In *Proc. IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE. https://doi.org/10.1109/WoWMoM.2017.7974339

[5] A. Aguiar L. Zilhão, R. Morla. 2018. A Modular Tool for Benchmarking IoT Publish-Subscribe Middleware. In *2018 IEEE 19th International Symposium on 'A World of Wireless, Mobile and Multimedia Networks' (WoWMoM)*. IEEE. https://doi.org/10.1109/WoWMoM.2018.8449774

[6] A. Shukla, S. Chaturvedi, and Y. Simmhan. 2017. RIoTBench: An IoT benchmark for distributed stream processing systems. *Concurrency and Computation: Practice and Experience* 29, 21 (Nov. 2017). https://doi.org/10.1002/cpe.4257

[7] K. Zhang, T. Rabl, Y.P. Sun, R. Kumar, N. Zen, and H.-A. Jacobsen. 2014. PSBench: A Benchmark for Content- and Topic-based Publish/Subscribe Systems. In *Proceedings of the Posters & Demos Session (Middleware Posters and Demos '14)*. ACM, New York, NY, USA, 17–18. https://doi.org/10.1145/2678508.2678517

[9] https://www.iot-lab.info