Received: 21 January 2020 | Revised: 27 March 2020 | Accepted: 14 April 2020

DOI: 10.1002/nem.2115

SPECIAL ISSUE PAPER

Classification and Evaluation of IoT Brokers: A Methodology

Eddas Bertrand-Martinez*[1] | Phelipe Dias Feio[1] | Vagner de Brito Nascimento[1] | Fabio Kon[2] | Antônio Abelém[1]

[1]GERCOM - Research Group on Networks and Multimedia Communications, Federal University of Pará, Belém, Brazil

[2]Department of Computer Science, University of São Paulo, São Paulo, Brazil

Correspondence
Eddas Bertrand-Martinez, GERCOM - Research Group on Networks and Multimedia Communications, Federal University of Pará, Belém, Brazil.
Email: eddasjbertrand@gmail.com

Present Address
LabTIC - Federal University of Pará (UFPA) - Guamá, Belém - PA, 66075-110, Brazil

Funding Information
Coimbra Group of Brazilian Universities (GCUB); Coordination Agency for the Improvement of Higher Education Personnel (CAPES); National Council for Scientific and Technological Development (CNPq); National Education and Research Network (RNP); National Science Foundation (NSF); Organization of American States (OAS); Research and Development Center in Digital Technologies for Information and Communication (CTIC); São Paulo State Research Support Foundation (FAPESP); National Science Foundation and the Brazilian Ministry of Science, Technology, Innovation, and Communication (MCTIC)

Abstract

Since the term Internet of Things (IoT) was coined by Kevin Ashton in 1999, a number of middleware platforms have been developed to cope with important challenges such as the integration of different technologies. In this context of heterogeneous technologies, IoT message brokers become critical elements for the proper function of smart systems and wireless sensor networks (WSN) infrastructures.

There are several evaluations made on IoT messaging middleware performance. Nevertheless, most of them ignore crucial aspects of the IoT context that also need to be included, such as reliability and other qualitative aspects. Thus, in this article, we propose a methodology for classification and evaluation of IoT brokers to help the scientific community and technology industry on evaluating them according to their interests, without leaving out important aspects for the context of smart environments.

Our methodology bases its qualitative evaluations on the ISO/IEC 25000 (SQuaRE) set of standards, and its quantitative evaluations on Jain's process for performance evaluation. We developed a case study to illustrate our proposal with 12 different open source brokers, validating the feasibility of our methodological approach.

KEYWORDS:
Internet of Things, message-oriented middleware, quality, metrics, MQTT, SQuaRE

1 | INTRODUCTION

The Internet of Things (IoT) has gained increasing attention during the last decade, with continuous innovations in hardware, software and connection solutions. Initiatives in different countries such as China[1], Hong Kong[2], United States[3] and various countries in Europe among others[4] reflect the worldwide interest in the topic. In Brazil, initiatives such as InterSCity[5], FIoT[6], SmartMetropolis on the city of Natal[7] or smart campuses as implemented in the State University of Campinas[8] reflect the interest of the country on the development of smart environments and smart cities.

This worldwide interest has been followed by the development of middleware platforms with the intent of integrating heterogeneous technologies. In this context, middleware platforms and wireless sensor network (WSN) infrastructures use protocols such as MQTT[I], CoAP[II], AMQP[III], STOMP[IV], and HTTP for exchanging messages[9].

Message brokers, a central component present in middleware platforms, are responsible for exchanging data among network nodes, such as sensors, actuators, services or even other platforms. They are in charge of publishing data from network nodes and making them available to other nodes. This kind of communication model is known as the publish/subscribe pattern, in which sensors and actuators act as publishers and subscribers, respectively, and application users can act as publishers or subscribers depending on the logic of each application[10,11].

According to application requirement definitions, features such as quality of service (QoS), robustness, performance, and adaptability can be fundamental when selecting an IoT broker. Due to the great number of available middleware solutions, it can become a daunting task to choose the most suitable ones in a given scenario. Our study of the research literature about evaluation processes with IoT messaging middleware showed only a handful of studies that evaluated performance, the majority of them not taking into consideration important aspects such as reliability and other qualitative aspects, crucial for the adoption of the technology.

Taking as a purpose to fill the gap of not having a standardized way to do IoT messaging middleware evaluations without leaving important aspects, we developed a methodology to evaluate them. We took considerations from the *ISO/IEC 25000*[12] (SQuaRE) family of standards and Raj Jain's process definition on performance evaluation[13]. Taking them as a base, we defined a process consisting of logical steps starting from the definition of clear goals until arriving at results evaluation about the benchmarked solutions.

The remainder of this paper is structured as follows: Section 2 presents an overview of theoretical concepts concerning message brokers on smart environments. Section 3 presents a literature review, contrasting papers with the purpose of our research. Section 4 describes the evaluation and classification methodology, complementing it with a case study we presented in Section 5, describing the entire evaluation process. Finally, we close the paper in Section 6 with concluding remarks and future work.

2 | MESSAGING BROKERS ON SMART ENVIRONMENTS

The Internet of Things is a paradigm that describes a complex interaction between the physical and the virtual world where ordinary things like televisions, cars, thermostats and other objects can communicate with people, with other things, or with services. Fig. 1 depicts different possibilities for IoT scenarios and how a digital ecosystem can be created by integrating a plethora of domains.

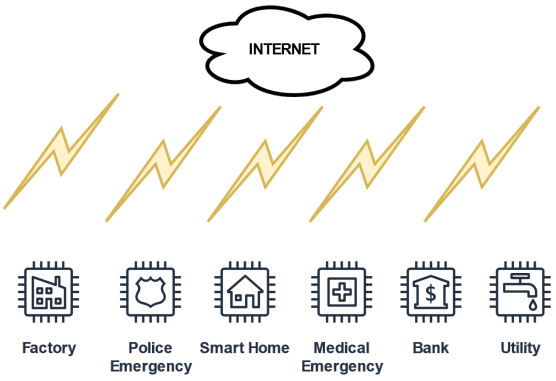

FIGURE 1 Possible application domains for the Internet of Things

[I]MQTT - Message Queuing Telemetry Transport, more information can be found on https://mqtt.org/faq
[II]CoAP - Constrained Applications Protocol, more information can be found on https://tools.ietf.org/html/rfc7252
[III]AMQP - Advanced Message Queuing Protocol, more information can be found on https://www.amqp.org/
[IV]STOMP - Streaming Text Oriented Messaging Protocol, more information can be found on https://stomp.github.io/

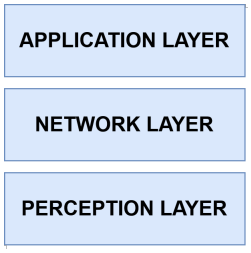

FIGURE 2 The 3-layered architectural model for the Internet of Things

As a consequence, there is no common agreement on the reference architecture or adopted standards of communication protocols. One of the major challenges to address is to choose the appropriate protocol for their specific IoT system requirements. A considerable number of architecture proposals include models consisting of three to five layers[14], which are equivalent in addressing the same needs but differ on the level of granularity.

In Fig. 2 we illustrate a widely used reference architecture consisting of three layers, namely, *perception layer*, *network layer*, and *application layer*. The perception layer is the one responsible for capturing data and dealing with aspects of the physical world, including the delivery of the captured data to be distributed by the network layer and processed into valuable information at the application layer.

The aforementioned abstraction is mainly carried out by the network layer, where IoT middleware platforms integrate all the data received by different sources and big data processes begin. A particularly important kind of middleware, due to its extensive use on distributed systems and in web services, is the Message Oriented Middleware (MOM). MOMs are event-based middleware, i.e., they act according to the messages they receive[15].

Brokers are the major agent on the MOMs implementation and are in charge of the dissemination of data among nodes; Fig. 3 illustrates their generic mechanism. They are based on the publication-subscription model, which obeys the principle of providing information only for those components that have previously subscribed to a particular data type. The usage of brokers leads to efficient data manipulation and more scalable architecture, at the expense of adding an additional degree of complexity. Some examples of brokers for IoT are ActiveMQ, RabbitMQ, Mosquitto, ZeroMQ, Orion Context Broker and YAMI4.

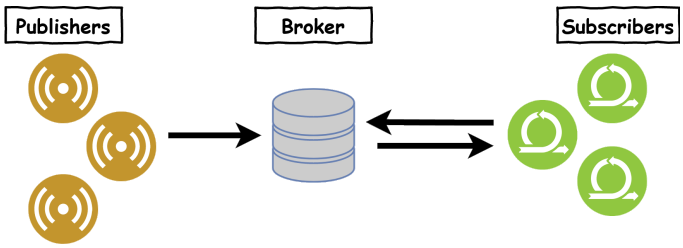

FIGURE 3 Message broker communication model

3 | RELATED WORK

The research literature contains works that evaluate IoT brokers. In fact, in most proposals concerning new mechanisms for optimization, enhancement on security or so on, a benchmark is done to ensure that the proposed solution does not compromise performance. Efforts as the ones made by Esposte et al.[5], Bhawiyuga et al.[16], Giambona et al.[17], Pipatsakulroj et al.[18], Blackstock et al.[19], Antonić et al.[20], and Vandikas & Tsiatsis[21] propose middleware that was subject to performance analysis, thus proving the performance improvements and benefits for IoT applications. However, they do not provide a systematic guide to reproduce the evaluation, so the experiment is somehow limited to reproducing it and/or modifying it.

The works of Banno et al.[22], Jutadhamakorn et al.[23], and Neumann et al.[24] proposed mechanisms for data dissemination, clustering, and update, respectively, where they evaluated brokers' performance to ensure the feasibility of these new mechanisms. However, similarly to the previous works, the performance evaluation presented there was mainly a tool to strengthen their contributions and, thus, their intention was not exposing an explicit measurement or evaluation methodology.

The works carried out by Mishra[25], Scalagent[26], and Ismail et al.[27] analyze and study multiple publicly available MQTT brokers. The first one measures subscription throughput (productivity) and the time it takes to send messages from the broker (responsiveness); both aspects relate to the speed of the broker. The second work measures productivity taking the publication throughput of the broker, and responsiveness, taking the time it takes a message to get from the publisher to the subscriber, adding resource utilization analysis on the research. The third one measures the same general aspects of the previous one, i.e., responsiveness metrics, productivity metrics, and resource utilization metrics. These works emphasize the importance of comparing MQTT brokers, but there are aspects such as security or reliability that could be analyzed too as key aspects of message brokers and are not considered. Although each benchmarking process needs to be designed differently according to the researcher's needs, it is instrumental to know which aspects are not being evaluated and why.

Finally, the contributions made by Happ et al.[28] and Pereira et al.[29] mention the importance of analyzing both qualitative and quantitative aspects of middleware for smart solutions. The first work mentions functional aspects that are crucial for MOM to be chosen as ideal solutions, i.e., the messaging pattern they use, the filtering techniques, QoS semantics, etc. The second work proposes non-functional aspects to be considered such as availability and clarity of technical documentation so to assure support continuity; also the compliance of the middleware platforms to follow the IoT-A[30] reference model architecture for IoT solutions. They both mention performance metrics and unfold comparison cases. But, as also happens with previous works, they do not propose a formal methodology or framework so to permit researchers to implement different comparisons. A summary table of related works can be found in Table 1.

All these previous works analyzed performance on IoT middleware. However, few studies considered qualitative aspects such as reliability[V]. Just the studies of Pereira et al., and Happ et al., mentioned the relevance of qualitative evaluation, recognizing that it has an impact on the adoption of these technologies.

TABLE 1 Related works on IoT message brokers evaluation and benchmarking

Paper	**Quantitative Evaluation**			**Qualitative Evaluation**	**Experimental Methodology**	
	Efficiency	Reliability	Availability		Flexible framework	Standard based
Esposte (2017)	✓		✓			
Bhawiyuga (2017)	✓					
Giambona (2018)	✓					
Pipatsakulroj (2017)	✓					
Blackstock (2010)	✓					
Antonić (2014)	✓					
Vandikas (2014)	✓					
Banno (2017)	✓		✓			
Jutadhamakorn (2017)	✓					
Neumann (2016)	✓					
Mishra (2018)	✓					
Scalagent (2015)	✓					
Ismail (2018)	✓					
Pereira (2018)	✓		✓	✓		
Happ (2017)	✓			✓		
Our research paper	✓	✓	✓	✓	✓	✓

[V]Reliability mechanisms are fundamental given the lossy nature of sensor networks and IoT in general.

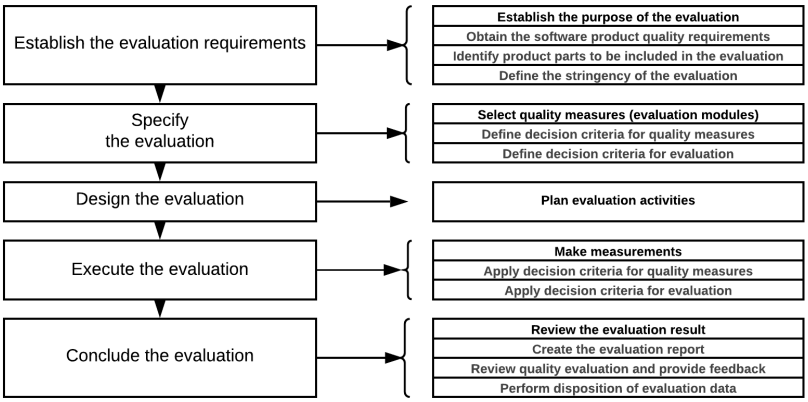

FIGURE 4 *ISO/IEC 25040:2011* software quality evaluation process

Under this context, our research aimed to develop an evaluation methodology for IoT message brokers in both quantitative and qualitative approaches. We propose a conceptual framework for scientists and technology engineers to have enough flexibility to assess IoT brokers quality according to their best interests, without leaving aside other aspects that need to be considered.

4 | EVALUATION AND CLASSIFICATION METHODOLOGY

In software engineering, the quality of a software can be analyzed according to two complementary perspectives: internal quality and external quality[31]. Internal quality, as the name implies, assesses the aspects concerned with source code. Its main objective is to evaluate aspects such as source code complexity, size, correctness, density of code, ease of maintenance, understandability, and modifiability, among other aspects. Internal quality has an impact on external quality, hence internal quality measurements can also be used to predict external quality characteristics such as reliability and portability. Internal quality assessment tends to be an essential part of software development processes.

On the other hand, external quality is concerned with the aspects perceived by the stakeholders and end-users[32], such as ease of installation, interoperability, accessibility, quality of documentation, support by a commercial entity or a developer community, among other aspects. Unlikely and unfortunately, external quality assessment is not always a mandatory practice in the software development industry[33]. Accordingly, our proposal addresses this situation and presents a methodological form to evaluate the external quality of IoT message brokers.

To define the evaluation process in our methodology, we based our approach both on the *ISO/IEC 25040:2011* norm[34] and on the benchmark process defined by Raj Jain[13]. The *ISO/IEC 25040:2011* norm defines the whole evaluation process to assess the external quality of a software. The evaluation process is comprised of five stages: requirements establishment, evaluation specification, evaluation design, evaluation execution. As presented in Fig. 4, these stages are subdivided further into 15 sub-steps. Depending on the step, the process requires the use of other norms in conjunction. For instance, when selecting quality measures (Step 2.1), the norm suggests to use the reference model defined by the *ISO/IEC 25010:2011* norm to select the relevant features. Also when planning the evaluation (step 3.1 in Fig. 4), it is suggested to use the metrics definition of the *ISO/IEC 25023:2016* norm; and so on. However, we considered that some steps in this evaluation process were redundant and the process itself could be regarded as complex. Further, according to Schneider[35], ISO standards generally reference other standards, but they rarely reference works outside the “standard domain", which is seemingly acceptable for the technology industry but maybe not plausible enough for the academic domain. Therefore, from the SQuaRE family of standards, only three standards were selected for the construction of the methodological process:

- *ISO/IEC 25040:2011*, which defines the evaluation process model.
- *ISO/IEC 25010:2011*, which defines the software product quality model (characteristics and sub-characteristics).
- *ISO/IEC 25023:2016*, which defines the metrics calculation model.

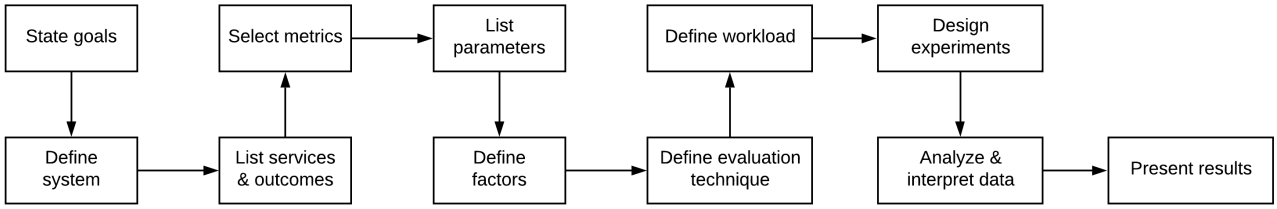

FIGURE 5 Process for systems' performance evaluation according to Raj Jain

The process defined by Jain for performance analysis, is divided in 10 steps, but in Fig. 5 we illustrated it in 11 steps to make explicit the first two tasks required on the first step. The first two tasks focus on defining what are the goals of the evaluation (similar to the process in *ISO/IEC 25040:2011*) and to define characteristically what constitutes "the system" to be assessed. The following stage constitutes the planning of the evaluation process, where the researcher defines the metrics, services, parameters, factors, workload, evaluation technique. This process is widely used in academia for performance analysis, but as the name of the book suggests, it focuses specifically on performance analysis. Thus, some steps of this process were taken in consideration to the construction of our quality evaluation process, but the ones that were specific to performance analysis, were adapted to be more general.

By comparing the two previously presented processes, studying carefully the purpose of each of the steps, and removing or merging steps that where considered redundant or too performance-specific, we derived a new process for the methodology. The resulting process with its steps is explained then, and illustrated in Fig. 6.

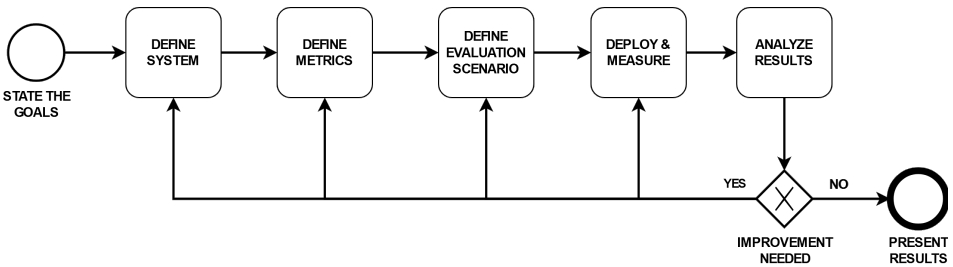

FIGURE 6 Quality evaluation process of the methodology

1. **State the goals:** The most important step on quality evaluation, although one of the most understated, is the establishment of clear and unbiased goals. Goals are not trivial as they often can change with a better understanding of the problem[33]; that is why clear goals could benefit from proper preliminary research. In this step, it is of extreme importance to define which features we want to evaluate. Possible goals for IoT broker evaluation could be to analyze the efficiency of scalability mechanisms for MQTT brokers, evaluate the reliability with different QoS levels on AMQP brokers, analyze main memory usage with different protocols like CoAP, STOMP, and HTTP on multi-protocol brokers among other possibilities. As a general rule, it is recommended to have a enough knowledge of how the communication model used by MQTT brokers works, and also in what type of configurations they can be installed, in order to establish more realistic goals and plausible results; otherwise, the results can be misleading[13].

2. **Define system:** The second step is to define what constitutes the system and what services of the system will be evaluated. When defining the system, clear boundaries need to be specified as performance measurements could vary depending on the point in which they are taken. For instance, a system could be defined as the whole middleware layer including the message broker and the client software on a pub/sub model. In this type of system definition, the metrics should describe aspects both about the broker quality as well as the client quality. The system could as well be defined as solely the broker software or an individual component of the broker, given that message brokers are the main part of MOMs.

The system services that are going to be evaluated must be clearly defined and listed, as performance metrics need to be associated with them. For instance, a type of service that a message broker performs is the publication of messages sent by its publishers. Consequently, the metrics associated with this service should describe how well the broker publishes the messages it receives.

3. **Define metrics:** The third step requires us to construct the metrics set that is going to be used. This task needs to be sub-divided in three steps as it can become complex:

   (a) **Map the features with a software quality model.** The features defined previously, in the goals on Step 1, should be mapped with a software quality model. This methodology proposes the use of the reference model defined by the *ISO/IEC 25010:2011* norm shown in Fig. 7. However, any quality model could be used as well, e.g., McCall and Matsumoto[36] or Quamoco[37], among others.

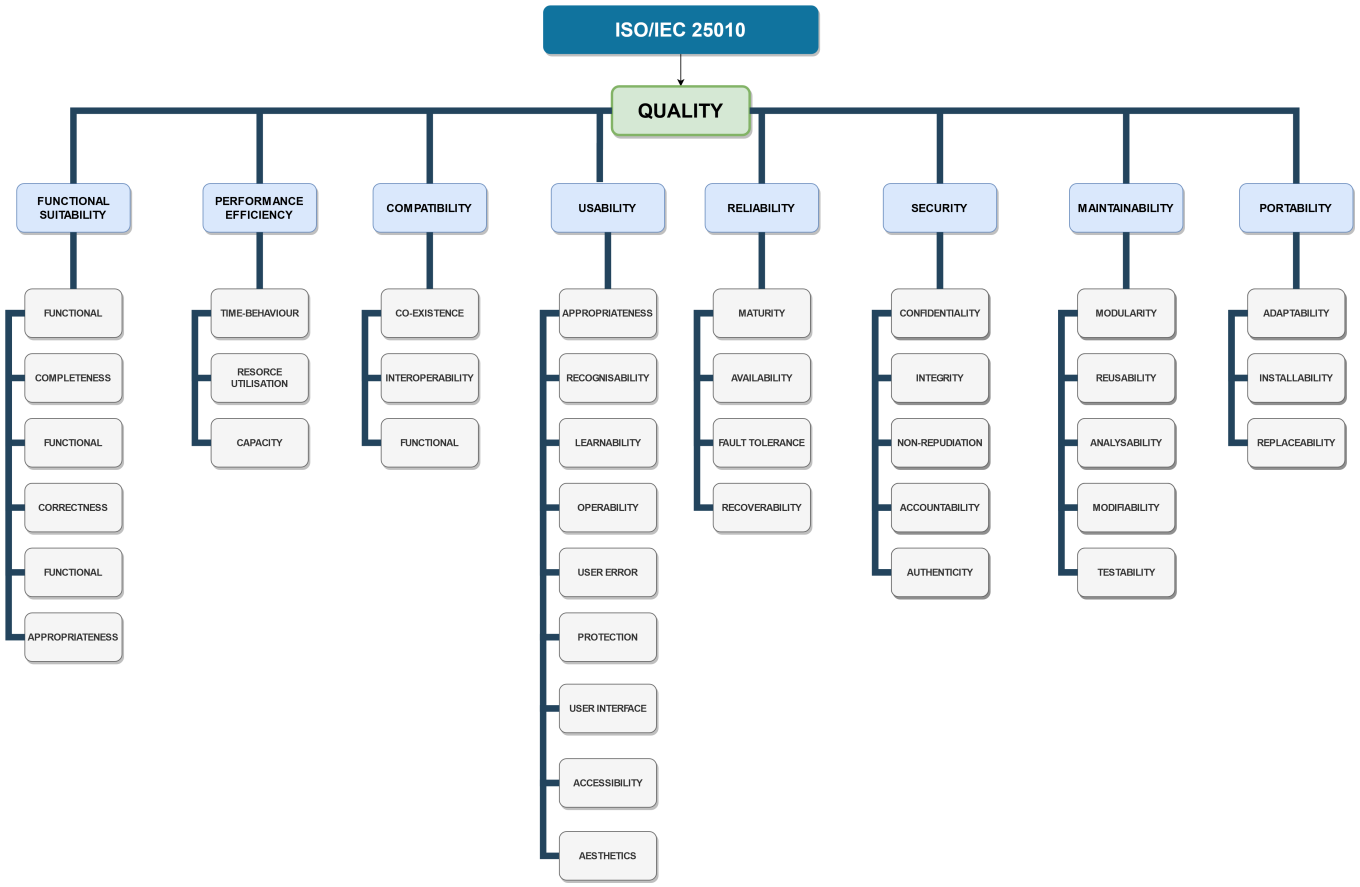

FIGURE 7 *ISO/IEC 25010:2011* Software product quality model, characteristics and sub-characteristics

   (b) **Construct feature metrics.** According to the mapped features, the appropriate quality metrics can be constructed in two ways:

      i. Choose metrics defined according to the *ISO/IEC 25023:2016* norm[38].
      ii. Construct metrics based on the possible outcomes of each feature. For instance, availability or absence could be possible outcomes when evaluating the existence of encryption mechanisms; completeness or incompleteness could be possible outcomes when evaluating software documentation. It becomes necessary to list all the possible outcomes (or range of outcomes if dealing with continuous outcomes rather than discrete) that a feature may have.

(c) **Construct performance metrics apart from the features metrics.** The performance should be measured for each of the offered services by the broker, following the definition made in Step 2. In this terms, IoT broker services can have three possible outcomes, the service (1) runs correctly, (2) runs incorrectly, or (3) does not run [13]. Differently from the features metrics, performance metrics for each of the three possible outcomes should be defined, for each service. The type of metrics related to three outcomes are called efficiency metrics (runs correctly), reliability metrics (runs incorrectly), and availability metrics (does not run). The construction of them is done in the same way as described on Step 3.b.ii.

4. **Define evaluation scenario:** In this step, the scenario on which the experiments are to be made is defined. The defined aspects are (a) the characterization of the population or workload, (b) the definition of the variables that can impact the measurements and, finally, (c) the logical topology on which the experiment is going to be performed.

   (a) According to each feature to be evaluated, a population or workload needs to be defined to do a proper evaluation. For qualitative evaluation, e.g., product usability, installation easiness or user experience, a population of different network managers tend to be employed and their responses are recorded to make up the quality evaluation data. In the case of quantitative evaluation, e.g., memory usage or publication throughput, a workload needs to be stressed on the broker so to have the corresponding performance evaluation data; data sets or databases can be used as workloads for testing the broker's capacity.

   (b) It also becomes necessary to define which will be the variables to be changed on the experiment, so to make a comparison on the system. Variables should impact directly on the feature that we want to analyze. Examples of variables for the experiments are: the scalability mechanism if measuring performance on the same broker, the message broker when comparing ease of installation among different brokers, the QoS level if measuring the reliability, etc.

   (c) Last, to have a complete scenario definition, we need to define which will be the logical topology under which the system will be evaluated.

5. **Deploy and measure:** The quality evaluation will be implemented according to the evaluation scenario defined in the previous step. It is necessary to define, in this step, how many times each measurement is going to be taken for each experiment, so to achieve a confidence level among the captured data. The physical deployment shall comply with the logical topology defined in the previous step.

6. **Analyze results:** After the measurements are taken, the whole evaluation process must be audited and analyzed so to determine if it was satisfactory as to draw conclusions about the system quality. Otherwise, it can be necessary to repeat the process from a previous step to do a modification on the definitions until it produces the desired results.

7. **Present results:** If the results from the evaluation process present a clear insight about the concerning quality aspects, then a conclusion can be made and help in decision making or knowledge creation.

In this way, the 7-step created process defines a quality evaluation methodology. Hereby, this proposal provides a theoretical framework for the academic and industry domain, in order to do systematic quality assessments, classification and comparison among IoT message brokers.

5 | CASE STUDY FOR IMPLEMENTATION OF THE METHODOLOGY

In this section, we describe a case study where 12 different message brokers where evaluated and classified. We followed the steps described previously and give a detailed explanation on each stage. This is in order to have a clearer understanding of how the process was done and guarantee reproducibility and modifiability of the process for further research. We will see how we chose two goals for the evaluation, and consequently how the remaining definitions were made according to the goals. The metrics are described in mathematical notation, and the results are presented in a statistical way, considering both qualitative and quantitative aspects.

5.1 | State the goals

The first step in the methodological process is to define a clear goal for the evaluation, hence we performed an initial research on the Internet about available MQTT-supporting brokers, as this application-level protocol is one of the most widely used. We chose the brokers to be open-source as they are easier to acquire and to compare than commercial options. The selected options were twelve different brokers: Mosquitto[39], ActiveMQ[40] & ActiveMQ Apollo[41], RabbitMQ[42], muMQ[43], Moquette[44], Emqttd[45], HiveMQ Community Edition[46], HBMQTT[47], Mosca[48], Emitter[49] and GridServer[50].

Because they implement MQTT, they are also expected to implement different levels of QoS as stated in the definitions of MQTT v3.1.1[51] and MQTT v5[52], being them:

- **QoS-0, At most once**: known as fire-and-forget, as it gives no guarantee of the message being published on the other side and the receiver does not acknowledge the reception of messages at MQTT level.
- **QoS-1, At least once**: the receiver acknowledges every received message. The sender must store each sent message until it is acknowledged by the receiver that the message was delivered. If the sender does not receive a message with the PUBACK (Publication Acknowledge) flag on a reasonable amount of time, it will publish again the message until it receives an acknowledgment.
- **QoS-2, Exactly once**: the highest level of reliability. A four-part handshake at MQTT level is made by both communicating ends to ensure that messages are delivered exactly once. First, the sender publishes a message with the PUBLISH flag, then the receiver must respond with a PUBREC (Publish Received) message. Finally, the sender sends a PUBREL (Publish Release) message to the receiver and it responds back to the sender with a PUBCOMP (Publish Complete) message. This is repeated for each one of the messages sent with QoS-2. It is the most reliable level, although the one that consumes more resources.

Taking in consideration the common aspects that we found on each software documentation and also taking performance as a fundamental non-functional aspect for message brokers, we defined two goals:

1. Evaluate qualitative aspects of these 12 MQTT brokers, specifically the completeness of their documentation, the configuration options they provide, installation and configuration experience, correct functioning, QoS level implementation and portability aspects such as programming language bindings, operating system portability, and compatibility with other protocols.
2. Choose the best three qualitatively evaluated brokers, and analyze the performance aspects of efficiency, reliability and availability.

5.2 | Define system

As the evaluation is focused on external aspects, the brokers were treated as black-boxes; our intention was not doing an in-depth analysis of the internal components of the brokers. We defined the system to be comprised solely by the message brokers, discarding the publishing and subscribing clients from the system structure. Thus, the performance metrics focus on the broker itself, ignoring any factors that could impact the client side performance. The illustration of the system is shown in Fig. 8.

We defined the message brokers to provide two services:

1. Publish a message onto its own topics or publish-on-queue.
2. Publish a message to subscribed clients or publish-to-client.

This means that, when establishing the metrics for performance measuring, we need to define metrics for publish-on-queue service and for publish-to-client service as well.

5.3 | Define metrics

In this subsection, we describe how the metrics were constructed. As stated by the process, both metrics for qualitative features and performance were constructed. The qualitative features, for this case study, were further divided in two groups: network

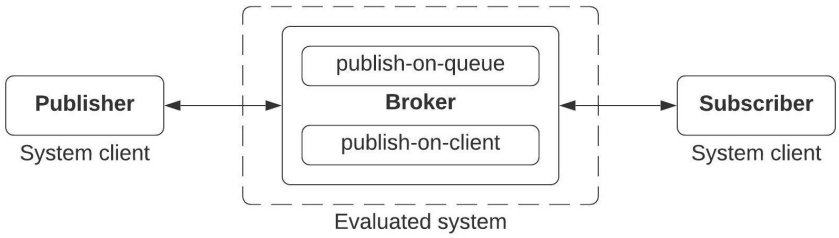

FIGURE 8 Message broker system definition for the case study

manager evaluation metrics and technical evaluation metrics. The network manager evaluation metrics assess the experience that a network manager would have with the usage of the message brokers as part of his/her routine work. Technical evaluation metrics refer to aspects that could be evaluated according to the brokers' technical documentation sites and/or software communities. Finally, we present the performance metrics that we used to benchmark the top three brokers.

5.3.1 | Map features

In the first step, we defined what was the goal of our evaluation and what features we intended to examine, specifically on the first goal. Therefore, we made a mapping of these features to the software product quality model shown on the *ISO/IEC 25010* standard. In Table 2, we can see to which characteristic/sub-characteristic pair we mapped each one of them.

TABLE 2 Mapping of features according to the *ISO/IEC 25010:2011* standard

Feature	Characteristic	Sub-Characteristic
Documentation completeness	Usability	Learnability
Configuration versatility	Usability	Appropriateness recognizability
Configuration and usage experience	Usability	Operability
Correct functioning	Functional Suitability	Functional Correctness
QoS levels implementation	Functional Suitability	Functional Completeness
Installation easiness	Portability	Installability
Operating systems supported	Portability	Adaptability
Programming languages bindings	Compatibility	Interoperability
Compatibility with other protocols	Compatibility	Interoperability

As the services of our system, and also the concerning features, were defined and mapped, we constructed quality metrics for the features and the system services in the following subsections.

5.3.2 | Construct features metrics

To construct the feature metrics we categorized them in two groups, network manager evaluation metrics and technical evaluation metrics.

Network manager evaluation metrics

We grouped in this category the features of documentation completeness, configuration versatility, configuration/usage experience, correct functioning, and installation easiness. For each feature, we implemented a Likert scale ranging from 1 to 5 and used the following formula to calculate the weight of every network manager feature evaluation:

$$w_{user} = L/5 \tag{1}$$

where L was the Likert scale value; so the minimum weight a broker could have for each feature was 0.2 and the maximum weight was 1. These calculations were defined by our own, none of them used the *ISO/IEC 25023* standard.

Documentation completeness. The feature was evaluated according to the existence or absence of technical manuals; taking also in consideration how understandable, complete and clear they were. The values ranged from (1) very unsatisfactory to (5) very satisfactory.

Configuration versatility. Evaluates the versatility and flexibility that the broker had itself to adapt to different configuration scenarios, i.e., use scalability mechanisms, ability to persist messages, ability to enable/disable TLS encryption on messages. The values for user evaluation ranged from (1) poor versatility to (5) high versatility.

Configuration/usage experience. Evaluates the general user experience on the configuration and usage of the broker, principally navigating on the options for customizing the broker and the intuitiveness of the software itself. The values for user evaluation ranged from (1) very unsatisfactory to (5) very satisfactory.

Correct functioning. Evaluates the user perception on the proper functioning of the broker's basic functions as publishing a service and subscribing new clients. The values ranged from (1) very unsatisfactory to (5) very satisfactory.

Installation easiness. The feature evaluates the easiness of installation, according to the user documentation guidelines if they had, and the customization capacity at installation time. This feature had correlation with the *Documentation Completeness* feature. The values also ranged from (1) very difficult to (5) very easy.

Technical evaluation metrics

Into this category, we grouped the features of QoS levels implementation, operating systems supported, programming languages bindings and compatibility with other protocols. All these measurements were based on the *ISO/IEC 25023* standard, except the *operating systems supported* calculation, which was defined by our own. The general formula for calculating the weight for the technical feature evaluation was the following:

$$w = L/r \tag{2}$$

where L was the Likert scale value and r the roof for the possible outcomes. Both the range of the Likert scale and the roof were defined according to each metric. Similarly to the network manager evaluations, the maximum weight that a broker could have was 1, but the minimum weight depended on the metric definition itself.

QoS levels implementation. This feature evaluated which levels of QoS did the broker implement, being the possible outcomes: (1) QoS-0, (2) QoS-0 & 1, and (3) QoS-0, 1 & 2. The roof for possible outcomes was 3, so the calculation was

$$w_{qos} = L/3 \tag{3}$$

Operating systems supported. We evaluated the different platforms in which a broker could be installed, being them in general Microsoft Windows, GNU/Linux and macOS Server. The possible outcomes for this measurement were (1) one platform, (2) two platforms, or (3) three platforms. The roof for the possible outcomes was also 3, so the calculation was similar to the previous one

$$w_{os} = L/3 \tag{4}$$

Programming language bindings. This feature evaluated if the broker, being open-source, had bindings with solely one programming language or if it was capable of binding with more the one language; consequently our possible outcomes were: (1) unique language, and (2) multiple languages, being the calculation

$$w_{lang} = L/2 \tag{5}$$

Compatibility with other protocols. Finally, this feature analyzed if the broker was compatible with other application-level protocols besides MQTT. This is desirable as it enables the broker to communicate with even more technologies. The possible outcomes were: (1) MQTT-only, and (2) multi-protocol; the calculation was

$$w_{proto} = L/2 \tag{6}$$

5.3.3 | Construct performance metrics

To construct the performance metrics according to the methodological flux, we need to define three types of metrics: efficiency metrics, reliability metrics, and availability metrics.

Efficiency metrics

According to Jain[13], efficiency is by itself defined by three additional types of metrics, i.e., responsiveness, productivity, and resource utilization metrics.

Broker responsiveness. This metric refers to the time it takes for the broker to publish a message onto a subscribed client, so the measurement is basically the mean time taken since a publisher sends a message until it arrives to the subscribed clients. The calculation is the following:

$$t_{brk} = (\sum_{i=1}^{N} t_{sub_i} - t_{pub_i})/N \tag{7}$$

where *t_sub* is the time when the message was received by the subscriber, *t_pub* is the time when the message was sent by the publisher and *N* is the number of published messages.

Broker productivity. The broker productivity metric was defined by two calculations, namely, (1) the throughput of the broker when storing on its own database when it receives a message from publishers (publish-on-queue), and (2) the throughput of the broker when publishing messages to subscribed clients (publish-to-client). The calculations were defined as

$$T_{pub \rightarrow brk} = TotalMessages_{rcv}/ElapsedTime \tag{8}$$

$$T_{brk \rightarrow sub} = TotalMessages_{snd}/ElapsedTime \tag{9}$$

where Equation 8 shows how to calculate the publish-on-queue throughput taking the total messages that the broker received from its publishers, and then calculating its quotient with the elapsed performance test time. Equation 9 shows the calculation for the publish-to-client throughput, taking instead the total sent messages by the broker to its subscribers. It is important to note that this measurements are to be taken on the broker itself as defined on Step 2.

Resource Utilization. The resource utilization metrics were defined by two measurements, i.e., CPU mean utilization and main memory mean utilization. Both measurements were taken with a specialized tool for analyzing high-performance computing applications. There was no defined calculation needed for these metrics.

Reliability metrics

The reliability of our system was analyzed by implementing one measurement, the failure rate, defined as the percentage of failed messages in an elapsed time. Failed messages in this context meant the messages that were received by the broker but were not sent by the broker to the subscribed clients. The calculation for this metric was the following:

$$R_{\%} = (TotalMessages_{rcv} - TotalMessages_{snd})/ElapsedTime \tag{10}$$

Availability metrics

The availability was defined as the rate of messages that were not published by the broker as defined on the publish-on-queue service definition; this could be due to server saturation, communication problems and so on. The calculation for defining this metric was the rate of dropped messages, obtained by

$$D_{\%} = TotalMessages_{pub}/TotalMessages_{rcv} \tag{11}$$

5.4 | Define evaluation scenario

Our study had two different goals, the first one analyzed qualitative aspects that could be evaluated by end users. To evaluate the network manager features, we had to search and read the whole documentation provided for each of the brokers, install the brokers, configure them and navigate among their different configuration files and menus. To evaluate the technical features, we searched for the specifications throughout the self-documentation and compared them with the installed versions. We scored

each of these evaluated features on a questionnaire. The variable on this goal was the broker itself; so we changed the broker and applied the questionnaire on each of them.

The second goal was to analyze performance, so we had to define a workload for it. We used a dataset comprised of location information for all the bus lines on the city of Teresina in the Brazilian state of Piauí. The information contained over 200K GPS location registries that corresponded to one day of traffic on November 2019; this dataset was extracted from Kaggle[VI]. The variable on this goal was the QoS level on the connection between the publisher-broker pair and broker-subscriber pair, varying it between QoS-0, QoS-1, and QoS-2.

For the logical topology, we adopted a basic scenario connecting one publisher to one instance of the message broker, and to which only one subscriber was subscribed. We captured data on the three components so to calculate correct performance metrics. In Fig. 9 we illustrate the scenario; this configuration is simple but it is also suitable enough to do specific performance analysis.

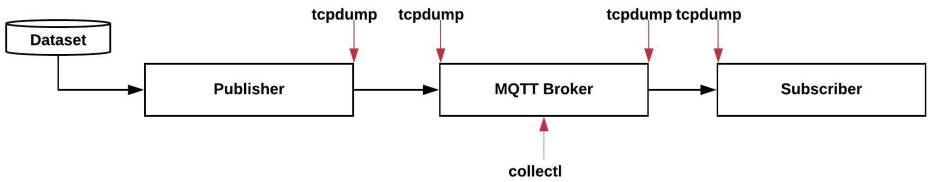

FIGURE 9 Diagram of the evaluation scenario and data capturing tools

5.5 | Deploy and measure

The hardware and software configurations used to evaluate the qualitative aspects were differently from those used for performance, as the evaluation requirements were different, i.e., qualitative aspects did not a need high-performance infrastructure, whereas performance evaluation needed some robustness on the deployed scenario. The hardware configuration for both types of evaluations can be found in Table 3.

TABLE 3 Hardware configuration for qualitative/quantitative evaluation experiments

	Qualitative Evaluations		**Quantitative Evaluations**	
Hardware Component	Publisher/Subscriber (physical machine)	MQTT Server (virtual machine)	Publisher/Subscriber (virtual machine)	MQTT Server (virtual machine)
Network adapter	Intel PRO/1000 MT		Broadcom NetXtreme BCM5720-2P	
Processor	Intel Core i5-4260U, 1.4 GHz		Intel Xeon E5-2650, 2.00GHz	
# Cores	2	1	8	8
# Threads	4	2	16	16
Memory	4 GB RAM	512 MB RAM	8 GB RAM	8 GB RAM
Storage	128 GB HDD	8 GB HDD	15 GB HDD	15 GB HDD

For the qualitative evaluations, the MQTT publisher and subscriber clients were installed in a MacBook Air notebook, with one 1.4 GHz Intel Core i5-4260U processor, running MacOS High Sierra 10. The MQTT server was deployed on a virtual machine hosted in the same notebook with the Oracle VirtualBox v5.0.26 hypervisor. The MQTT server virtual machine had Ubuntu Server 16 as operating system.

For performance evaluation, the MQTT publisher, subscriber, and server were all deployed on bare-metal virtualization running on Xen Server hypervisor. The physical machine, in which the virtual machines were hosted, had two 2.00 GHz Intel Xeon

[VI] https://www.kaggle.com

E5-2650 processors that provided 8 cores each, running 2 threads per core, totaling 32 threads. The total main memory was 62 gigabytes RAM and the total storage was 2 terabytes of HDD. Both the publisher and subscriber shared the same virtual machine with CentOS 7. The MQTT server was deployed in a separate virtual machine running CentOS 8. We used *collectl*[VII] to capture CPU, main memory, swap memory, and network traffic data in the MQTT broker on a periodic base, capturing every second and storing the information on a file for later analysis; this is illustrated in Fig. 9. For MQTT message analyses, we used *tcpdump*[VIII] to capture only MQTT messages with the PUBLISH flag, so to analyze only the information concerning publication. It was placed in the publisher to analyze the outgoing PUBLISH messages, in the MQTT broker to capture the incoming and outgoing PUBLISH messages and in the subscriber to capture the incoming messages; all these captures were stored in files for later analysis.

5.6 | Evaluation results

As previously mentioned, our analysis had two different goals, the first one was the evaluation of the qualitative aspects of the brokers and the second one was a quantitative evaluation concerning aspects of the brokers' performance.

5.6.1 | Qualitative aspects

We started our analysis comparing the brokers' qualitative features, taking first the network manager features of *documentation completeness*, *installation easiness*, *configuration/usage experience*, *correct functioning*, and *configuration versatility*.

Concerning documentation completeness, we researched the technical documentation provided by the developers of each of the brokers. According to Table 4, 33% of the documentation was regarded as Very Satisfying, meaning they had complete guides about their configuration files, installation customization, a troubleshooting wiki or community for bug handling, and version control system. Also, 25% of them were regarded as Satisfactory, meaning they had complete guides and troubleshooting channels but lacked versioning schemes; 17% of them were regarded as Regular, meaning they had only essential content concerning configuration and installation guides. The rest of the statistics represent brokers that did not have satisfactory documentation giving at least customization guidelines besides superficial README file texts.

On installation easiness, Table 4 shows that 58% of the brokers' installation processes were considered Satisfactory, taking account that the process took a few steps or the proper documentation was very helpful and gave clear guidance. A recurrent problem found on the installation process was the absence of some commands related to software dependencies. Additionally, some of the step-by-step guides were not precise or clear enough.

Configuration and user experience evaluated the overall installation, configuration, and navigation experience. According to Table 4, 33% of the brokers were qualified as Very Satisfactory in general use. Nonetheless, 50% were considered Regular due to problems happening during the installation process; 8% of dissatisfaction was due to some brokers that presented difficulty in the learning process, as their configuration was not intuitive.

Regarding broker functionality correctness, most of the brokers met the user expectations. Based on the publisher/subscriber model the brokers achieved 58% of high satisfaction according to Table 4; the Regular ones (17%) meant that the broker did not

TABLE 4 Qualitative features of the brokers - Network manager evaluation metrics

Satisfaction Level	Documentation Completeness	Installation Easiness	Configuration and usage experience	Functionality Correctness	Configuration Versatility
Very Satisfactory	33%	8%	33%	58%	33%
Satisfactory	25%	58%	8%	25%	8%
Regular	17%	25%	50%	17%	50%
Unsatisfactory	17%	-	8%	-	8%
Very Unsatisfactory	8%	8%	-	-	-

[VII] http://collectl.sourceforge.net
[VIII] https://www.tcpdump.org/

work as expected after several configurations; the remaining 25% meant that the functionalities had no problem, but a careful attention was required by the users to make them work appropriately.

The results for configuration versatility are shown in Table 4. Emqttd and HBMQTT were part of the best 33% evaluated as Very Satisfactory because they supported clustering/scalability mechanisms, persistence, MQTT v3, and v5 support. A 50% was regarded with Regular versatility; RabbitMQ and Apollo were among the evaluated as regular ones because of clustering mechanisms and persistence capabilities they offer.

After the network manager features, we evaluated the technical features of *QoS levels implementation*, *operating systems supported*, *programming languages bindings*, and *compatibility with other protocols*. When an MQTT client establishes a connection request with the broker, the QoS level is set for enduring communication. Table 5 shows the proportional evaluation made on the brokers according to the QoS levels they provided.

TABLE 5 Implemented QoS levels

Implemented QoS Levels	
Level 0-1-2	42%
Level 0-1	8%
Only 0	8%
No Type	8%
Undefined	33%

Regarding portability across platforms, 58% of MQTT brokers supported three different operating systems, i.e., Windows, Linux, and macOS. This is beneficial as it means they can be deployed in different facilities, improving their portability. The details can be seen in Table 6.

TABLE 6 Operating systems supported

Operating Systems Supported	
Windows / Linux	17%
Undefined	17%
Windows / Linux / macOS	58%
Linux	8%

Concerning the bindings to programming languages, a good portion of the brokers were able to implement APIs with two or more programming languages. Table 7 illustrates that the majority support more than one programming language, 50% of them can be implemented with C, C++, Ruby, Java, Python, Go, .NET, JavaScript and PHP. However, 33% of them supported only one programming language and 17% did not provide substantial information on their documentation.

TABLE 7 Programming languages supported

Programming Languages Supported	
Single Language	33%
Two or More	50%
Undefined	17%

Finally, concerning the network manager and technical features evaluation process, we reviewed if the brokers supported different application protocols other than MQTT. In Table 8, we can see that 67% of the sample supported MQTT only as a protocol. Although CoAP is a well-known protocol for IoT applications, none of the chosen brokers seemed to support it. In

addition, there were brokers that supported both MQTT and AMQP plus another communication mechanism like STOMP, web-socket and/or CoAP, they were Emitter, RabbitMQ, ActiveMQ, and Apollo.

TABLE 8 Supported communication protocols

Supported Communication Protocol	
MQTT	33%
Two or More	67%

5.6.2 | Quantitative aspects

According to the previous results, the best qualitatively evaluated brokers were Mosquitto, HiveMQ Community Edition, and RabbitMQ. As a result, we deployed our second goal scenario with these three brokers.

Our first task was to evaluate the efficiency metrics of *Responsiveness*, *Productivity*, and *Utilization* according to the definitions made previously. We used the MQTT clients provided by Mosquitto to load the data set, the publisher tier, and to subscribe to the topic, the subscriber tier. The data file had 218,772 registries corresponding to the 30th of November, 2019 in the whole 24 hour cycle. We ran the experiments with HiveMQ, Mosquitto and RabbitMQ varying for each one of them between QoS-0, QoS-1 and QoS-3, having each combination of broker+QoS experiment ran 3 times.

We evaluated the Responsiveness of the three of them by measuring the time a message took from the publisher client side to arrive to the subscriber client side; on Fig. 10 we can see the results of those measurements. The broker that maintained a low publisher-subscriber time on the three different QoS levels was RabbitMQ, having that most of its messages were sent from one side to the other below 10 milliseconds. In absolute terms, Mosquitto was the one that reached the lowest latency with messages of 250 microseconds at QoS-0, but in the same QoS level, it reached a median of 340 milliseconds. HiveMQ was the broker that reached higher latency times with pub/sub times of up to 1.9 seconds, but we took them as exceptions given that the third quartile stood below 3 milliseconds. We can say, in general, that these brokers maintain responsiveness of fewer than 3 milliseconds per message when configured on QoS-1 and QoS-2. When using QoS-0, lower times can be reached in absolute terms, but it increased instability and scattered results.

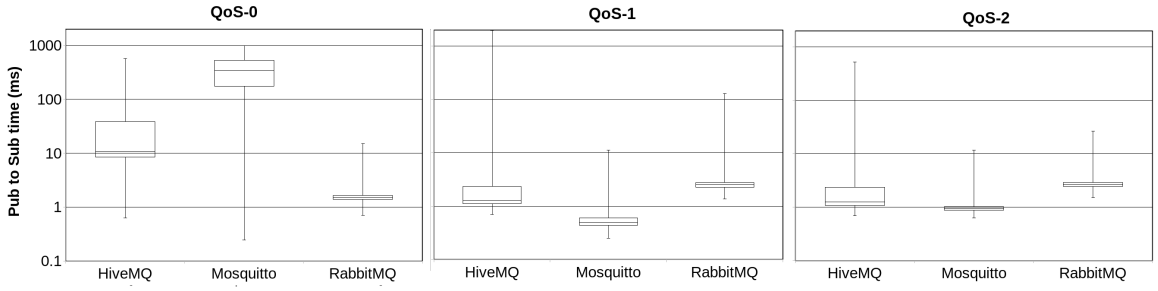

FIGURE 10 Processing time for a message to get from the publishers to the subscribers

The second metric for efficiency was Productivity, which was determined by the throughput of the broker on its second service: publish-to-client. We measured throughput by the number of messages each broker could process taking in consideration: the reception of the message by the broker, the publication of that message on the topic, and the sending of that message to its subscriber. As expected, the level of QoS used was inversely proportional to the throughput achieved by the broker, higher level of QoS required a higher message exchange to assure reliability on communication. On QoS-2, i.e., *exactly-once* delivery, the throughput values were the lowest, with RabbitMQ reaching low values of 17 messages per second, HiveMQ with 82 messages per second and Mosquitto with 140 messages per second as their lowest. The highest throughput was reached by Mosquitto on QoS-0 with 163 processed messages per second, as it is shown in Fig. 11.

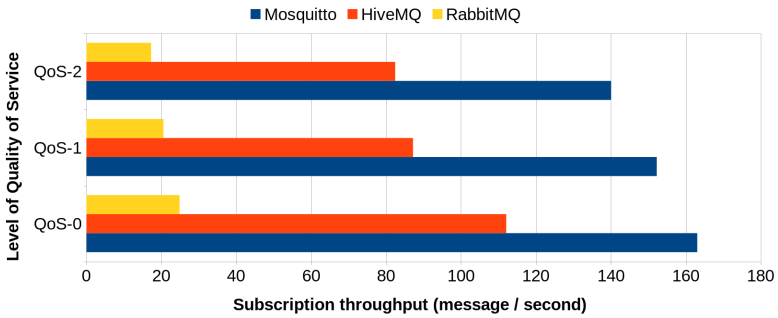

FIGURE 11 Throughput comparison, varying QoS levels among the three brokers

The third metric was Utilization, in which we measured the CPU utilization and the main memory utilization as well. In general, the CPU utilization was not relevant among the brokers as HiveMQ had a 3% of busy time, Mosquitto had 2% and RabbitMQ a 5% busy; different QoS levels did not cause a significant impact on the processor. However, a more relevant behavior was appreciated on main memory usage, with HiveMQ using almost the half of the available RAM memory, Mosquitto oscillated between 15% and 18% on RAM memory, and RabbitMQ ranged between 7% and 34%, as it can be seen on Fig. 12. Unlike processor results, higher levels of QoS required higher memory usage. This behavior could possibly be because higher levels of reliability require higher increments on the heap; additionally, message persistence is used to assure that the messages arrive to their destination, which in turn can occupy more heap space with the message queues growth.

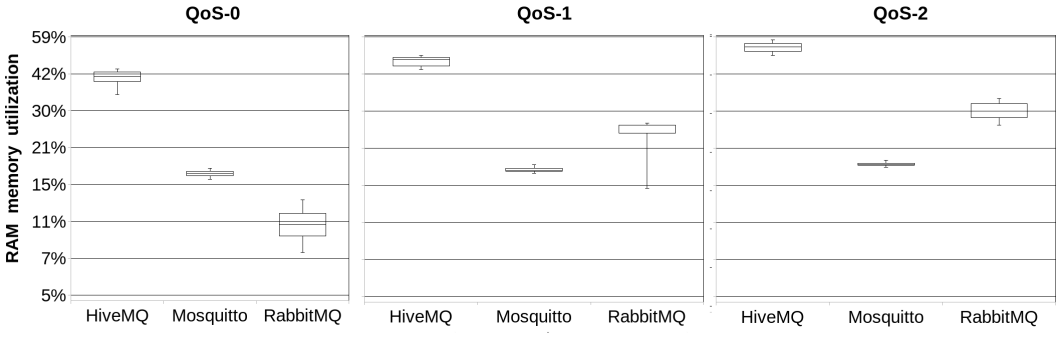

FIGURE 12 Main memory utilization comparison, varying QoS levels

Finally, we calculated the metrics for Reliability and Availability. For Reliability, the failure rate of messages not delivered to the subscribers was 0% with all three brokers when using QoS levels 1 and 2. This means that the three of them have good reliability when using QoS-1 and 2. However, when using QoS-0, Mosquitto had a failure rate of 8.72%, which is not acceptable on critical systems because it could mean more than 250K messages not delivered monthly to traffic monitoring systems for mid-sized cities, or more than 60 hours without synchronization on a monthly basis for real-time systems; therefore QoS-0 on Mosquitto could not be considered a safe option when using systems that need high reliability. HiveMQ had a failure rate of 0.26% on QoS-0 which could mean 7800 undelivered messages or 2 hours without synchronization monthly; these values could be considered acceptable depending on the application scenario. Concerning Availability, all of them presented 100%, which was measured with the rate of dropped packages by the broker when receiving messages from its publisher. Therefore, all three appeared to be solutions with high availability.

5.7 | Broker classification and discussion

The previous stages of the methodology gave us the results for network manager feature evaluation and technical feature evaluation. The results in this work are different from the ones presented by Bertrand-Martinez et al.[53], given that we included three

more brokers in the study, namely, HiveMQ CE, muMQ and GridServer. Additionally, the evaluations of the features (network manager features and technical features) were performed three additional times to achieve more reliable results. After carrying out the evaluation, we did a group classification and comparison of the MQTT brokers. The resulting classification is as follows:

- **Multi-platform brokers:** Mosquitto, HiveMQ, RabbitMQ, ActiveMQ, Emqttd, ActiveMQ Apollo, and Emitter.
- **Multi-language brokers:** Mosquitto, HiveMQ, RabbitMQ, ActiveMQ, Emitter, and GridServer.
- **Multi-protocol brokers:** RabbitMQ, ActiveMQ, ActiveMQ Apollo, and Emitter.
- **Multiple QoS brokers:** Mosquitto, HiveMQ, Emqttd, HBMQTT, and Moquette.

To reach a verdict for our first goal, we summed every punctuation given to the brokers and produced a descending bar graph comparison, as shown in Fig. 13. The X axis of the graph shows the message brokers, while the Y axis shows the accumulated points in the qualitative evaluation. Each feature of the qualitative evaluation had a maximum value of 1, as stated on the metrics definition. Hence, the maximum possible grade a broker could achieve for the qualitative evaluation was nine: five points for the network manager evaluation features and four points for the technical evaluation features. Based on the results, we appreciate that correct functioning (ISO: Functional correctness) was a satisfactory feature in most of the brokers, meaning that many of the brokers did what they were expected to do, i.e., publish in their own queues and forward messages to their clients. The documentation completeness (ISO: Learnability) is an aspect that needs attention by part of the development teams in general; many of the brokers lacked clear guides on how to configure and customize them. On configuration versatility (ISO: Appropriateness Recognizability), besides the functional aspects of publishing/subscribing that a broker should have, it is also expected to have some non-functional mechanisms to assure reliability and robustness, for instance, scalability mechanisms, encryption, and support to different versions of MQTT. This was an aspect not taken too much into consideration by many of the solutions, that is why we punctuated most of them with a regular qualification.

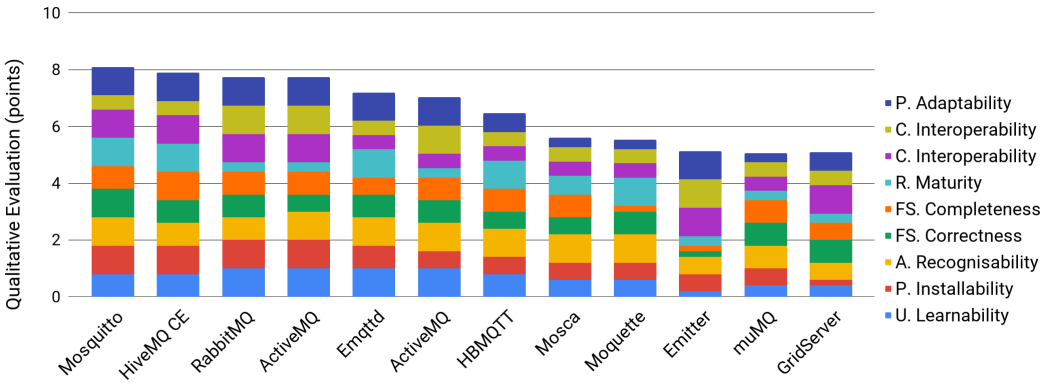

FIGURE 13 Comparison of quality aspects for the 12 MQTT open-source brokers

Concerning the portability aspects of installation easiness (ISO: Installability) and operating systems supported (ISO: Adaptability), many of them had good qualifications with Emqttd in exception. It was easy on installation but provided no information about operating systems. Emitter is portable to multiple operating systems but was not so intuitive enough, thus reducing the overall portability aspect.

Regarding usability, the evaluation varied from broker to broker. In general, Mosquitto is regarded as one of the most popular message broker implementations due to its simplicity and intuitiveness on installation, operating system portability and lightweight and simple structure measured by its source lines of code.

Recalling the results obtained by the performance analysis, we had different outcomes from each of the brokers. On responsiveness, Mosquitto and HiveMQ showed some variability when using QoS-0 as they had very low and very high processing times on the same analysis, in contrast to RabbitMQ which showed a more uniform delivery time from publisher to subscriber,

less than 10 milliseconds for the three different levels of QoS. However, when analyzing productivity, the behavior was different. Mosquitto showed the highest throughput for the three reliability levels, whereas RabbitMQ showed the lowest on the three of them, with less than 30 processed messages per second. This could mean that some brokers have efficient algorithms for delivering messages, but inefficient algorithms on handling their heap memory allocation and/or message queues management, so they can turn a bottleneck on real case scenario.

Regarding resource utilization, HiveMQ was the broker whose more memory allocated for the three QoS levels. In contrast, RabbitMQ increased its memory allocation as the QoS level on its communication was higher. Mosquitto maintained uniformity on main memory usage for all the three levels of reliability. The processor's busy time was in general low for all three of them, so it was not graphed nor analyzed. Finally, when analyzing reliability, Mosquitto was the only one to show a high failure rate when using QoS-0 as we saw. This is generally unacceptable for systems that need high reliability, as such is the requirement on smart environments, which already have a lossy nature on their communications.

6 | CONCLUSIONS AND FUTURE WORK

Defining widely accepted middleware solutions for Internet of Things is still a big challenge due to the lack of standardization on the evaluation processes and comparisons among the different proposals[29]. This is particularly true given the rising interest in smart environments as in smart cities, smart homes, and smart grids on industrial applications.

Our methodology intends to help the scientific community on setting a guideline to perform quality assessment but taking into consideration aspects that are relevant to smart applications, supported with a standardized approach based on internationally recognized processes. We wanted to develop a more complete, defined and clear methodology.

Among the main contributions of our proposal are (1) defining a process for middleware evaluation for smart environments, (2) providing a framework for the scientists to develop their own measurements and metrics system, (3) giving a practical implementation of the methodology so to have a better understanding of it.

For future work, our methodology could be applied with a more significant workload, as the workload that we used represents the traffic behavior of a mid-sized city, but is not a representative workload for a major urban city. For testing different scenarios a more elaborated and complex topology could be deployed, in which the broker would process the same workload but distributed among different gateways, or deploying the broker in a distributed fashion as well. In this type of scenarios, the broker creates more threads to attend new connections from each of the publisher clients. The purpose of the performance evaluation we did was to understand the behavior of the brokers itself, and not their maximum capacity. However, this is still a future proposal to be studied.

ACKNOWLEDGMENTS

This research was supported by the RNP-NSF joint call for research and development in cybersecurity (INSaNE—Improving Network Security at the Network Edge), funded by the National Science Foundation and the Brazilian Ministry of Science, Technology, Innovation, and Communication (MCTIC) through RNP and CTIC. It also was funded by the Partnership Program for Education and Training (PAEC) maintained by joint collaboration of the Organization of American States (OAS) and the Coimbra Group of Brazilian Universities (GCUB), through the Coordination Agency for Improvement of Higher Education Personnel (CAPES) from the Brazilian Ministry of Education. This research is part of the INCT program of Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, CAPES – Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9.

References

1. Chen Shanzi, Xu Hui, Liu Dake, Hu Bo, Wang Hucheng. A vision of IoT: Applications, challenges, and opportunities with China perspective. *IEEE Internet of Things Journal.* 2014;1(4):349-359.

2. Ma Ruiqu, Lam Patrick, Leung C.. Potential pitfalls of smart city development: A study on parking mobile applications (apps) in Hong Kong. *Telematics and Informatics.* 2018;35(6):1580-1592.

3. Ellsmoor James. Smart Cities: The Future Of Urban Development https://www.forbes.com/sites/jamesellsmoor/2019/05/19/smart-cities-the-future-of-urban-development/#7684ffa12f902019.

4. Caragliu Andrea, Del Bo Chiara, Nijkamp Peter. Smart Cities in Europe. *Journal of Urban Technology.* 2011;18(2):65-82.

5. Esposte Arthur, Kon Fabio, Costa Fabio, Lago Nelson. InterSCity: A scalable microservice-based open source platform for smart cities. In: :1-12; 2017; Porto, Portugal.

6. Nascimento Nathalia, Nascimento Carlos. FIoT: An agent-based framework for self-adaptive and self-organizing applications based on the Internet of Things. *Information Sciences.* 2017;378:161-176.

7. Batista Cesar, Silva Pedro, Cavalcante Everton, et al. A middleware environment for developing Internet of Things applications. In: :41-46; 2018; Rennes, France.

8. Unicamp . Smart Campus - Unicamp http://smartcampus.prefeitura.unicamp.br/ (Portuguese).

9. Naik Nitin. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In: :1-7; 2017; Vienna, Austria.

10. Dizdarevic Carpio F. Jukan A., Masip-Bruin X.. A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. *ACM Computing Surveys.* 2019;51:1-29.

11. Kraijak S., Tuwanut P.. A survey on IoT architectures, protocols, applications, security, privacy, real-world implementation and future trends. In: :1-6; 2015; Shanghai, China.

12. ISO/IEC . Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE2014. International Standard ISO/IEC 25000:2014.

13. Jain Raj. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling,.* Nova York, EUA: Wiley- Interscience; 1991.

14. Sethi Pallavi, Sarangi Smruti. Internet of Things: Architectures, protocols and applications. *Journal of Electrical and Computer Engineering.* 2017;2017:1-25.

15. Razzaque Mohammad, Milojevic-Jevric Marija, Palade Andrei, Clarke Siobhán. Middleware for Internet of Things: A survey. *IEEE Internet of Things Journal.* 2016;3:70-95.

16. Bhawiyuga Adhitya, Kartikasari Dany, Pramukantoro Eko. A Publish Subscribe based Middleware for Enabling Real Time Web Access on Constrained Device. In: :1-5; 2017; Phuket, Thailand.

17. Giambona Riccardo, Redondi Alessandro, Cesana Matteo. Demonstrating MQTT+: An Advanced Broker for Data Filtering, Processing and Aggregation. In: :357-358; 2018; Quebec, Canada.

18. Pipatsakulroj Wiriyang, Visoottiviseth Vasaka, Takano Ryousei. muMQ: A Lightweight and Scalable MQTT Broker. In: :1-6; 2017; Osaka, Japan.

19. Blackstock Michael, Kaviani Nima, Lea Rodger, Friday Adrian. MAGIC Broker 2 - An Open and Extensible Platform for the Internet of Things. In: :2-8; 2010; Tokyo, Japan.

20. Antonic Aleksandar, Rozankovic Kristijan, Marjanovic Martina, Pripuzic Kresimir, Zarko Ivana. A Mobile Crowdsensing Ecosystem Enabled by a Cloud-based Publish/Subscribe Middleware. In: :107-114; 2014; Barcelona, Spain.

21. Vandikas Konstantinos, Tsiatsis Vlasios. Performance Evaluation of an IoT Platform. In: :141-146; 2014; Oxford, United Kingdom.

22. Banno Ryohei, Sun Jingyu, Fujita Masahiro, Takeuchi Susumu, Shudo Kazuyuki. Dissemination of edge-heavy data on heterogeneous MQTT brokers. In: :1–7; 2017; Tokyo, Japan.

23. Jutadhamakorn Pongnapat, Pillavas Tinnapat, Visoottiviseth Vasaka, Takano Ryousei, Haga Jason, Kobayashi Dylan. A Scalable and Low-Cost MQTT Broker Clustering System. In: :1-5; 2017; Nakhon Pathom, Thailand.

24. Neumann Martin, Bach Christoph, Miclaus Andrei, Riedel Till, Beigl Michael. Always-On Web of Things Infrastructure using Dynamic Software Updating. In: :1-6; 2016; Stuttgart, Germany.

25. Mishra Biswajeeban. Performance evaluation of MQTT broker servers. In: :599–609; 2018; Melbourne, Australia.

26. Scalagent . *Benchmark of MQTT servers.* 2015.

27. Ahmed Ismail, Haitham Hamza, Amira Kotb. Performance Evaluation of Open Source IoT Platforms. In: :1-5; 2018; Alexandria, Egypt.

28. Happ Daniel, Karowski Niels, Menzel Thomas, Handziski Vlado, Wolisz Adam. Meeting IoT platform requirements with open pub/sub solutions. *Annals of Telecommunications.* 2017;72:41–52.

29. Pereira Carlos, Cardoso João, Aguiar Ana, Morla Ricardo. Benchmarking pub/sub IoT middleware platforms for smart services. *Journal of Reliable Intelligent Environments.* 2018;4:25–37.

30. European Comission . The Lighthouse Project IoT-A https://cordis.europa.eu/project/id/257521FP7-ICT - Specific Programme "Cooperation": Information and communication technologies; 2013.

31. Glinz Martin. Software Product Quality https://www.ifi.uzh.ch/dam/jcr:8e9cd2bb-60de-4ce7-b17e-8f994744d4cd/06_product_q.pdf2014.

32. Boehm B., Brown J., Lipow M.. Quantitative Evaluation of Software Quality. In: :592-605; 1976; San Francisco, USA.

33. Brunst Holger, Mueller Matthias. Performance Analysis of Computer Systems: Requirements, metrics, techniques, and mistakes https://tu-dresden.de/zih/ressourcen/dateien/lehre/ws1112/lars/vorlesungen/lars_lecture_02_requirements-metrics-techniques.pdf2019.

34. ISO/IEC . Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Evaluation process2011. International Standard ISO/IEC 25040:2011.

35. Schneider Florian, Berenbach Brian. A Literature Survey on International Standards for Systems Requirements Engineering. In: ; 2013; Atlanta, USA.

36. McCall J., Matsumoto M.. Software Quality Measurement Manual1980. Rome Air Development Center, RADC-TR-80-109-Vol-2.

37. Deissenboeck F., Heinemann L., Herrmannsdoerfer M., Lochmann K., Wagner S.. The quamoco tool chain for quality modeling and assessment. In: :1007-1009; 2011; Honolulu, USA.

38. ISO/IEC . Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Measurement of system and software product quality2016. International Standard ISO/IEC 25023:2016.

39. Eclipse Foundation . Mosquitto - An open source MQTT broker https://mosquitto.org/.

40. Apache Foundation . ActiveMQ - Flexible & Powerful Open Source Multi-Protocol Messaging https://activemq.apache.org/.

41. Apache Foundation . The Apollo Project https://github.com/apache/activemq-apollo.

42. Pivotal Software . RabbitMQ - Messaging that just works https://www.rabbitmq.com/.

43. Pipatsakulroj W., Visoottiviseth V., Takano R.. muMQ: A lightweight and scalable MQTT broker. In: :1-6; 2017.

44. Eclipse Foundation . Moquette MQTT Broker https://moquette-io.github.io/moquette/.

45. Fen Lee. The EMQ project - The Massively Scalable MQTT Broker for IoT and Mobile Applications https://emqtt.io.

46. Götz Christian. HiveMQ Open Source Community https://www.hivemq.com/developers/community/.

47. Jouanin Nicolas. HBMQTT https://github.com/beerfactory/hbmqtt.

48. Collina Matteo. Mosca - MQTT broker as a module http://www.mosca.io/.

49. Emitter Studios . Emitter - Scalable Real-Time Communication Accross Devices https://emitter.io/.

50. GRID System . GRID Server Open Sourced https://grids.systems/2019/08/02/grid-server-open-sourced.

51. Banks Andrew, Gupta Rahul. MQTT Version 3.1.1 http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

52. Banks Andrew, Briggs Ed, Borgendale Ken, Gupta Rahul. MQTT Version 5.0 http://docs.oasis-open.org/mqtt/mqtt/v5.0/cs02/mqtt-v5.0-cs02.html.

53. Bertrand-Martinez Eddas, Feio Phelipe, Nascimento Vagner, Pinheiro Billy, Abelém Antônio. A Methodology for Classification and Evaluation of IoT Brokers. In: ; 2019; Niterói, Brazil.

AUTHOR BIOGRAPHIES

Eddas Bertrand-Martinez. He holds a bachelor's degree in Telecommunications Engineering from the Technological University of Central America (UNITEC, Honduras), and a master's degree in Computer Science from the Federal University of Pará (UFPA) in Belém, Brazil. He has industry experience leading software development and infrastructure projects with emphasis on smart banking and general financial services. His research is focused on middleware solutions, performance analysis and privacy in smart environments.

Phelipe Dias Feio. Bachelor of Information Systems from the University Center of Pará (CESUPA), Postgraduate in Computer Network from the University of Amazonia (UNAMA) and Master in Computer Science from the Federal University of Pará (UFPA). I have worked in the private sector as an Infrastructure and Support Analyst and currently as a Software Requirements Analyst and university professor at Estácio de Sá University, I have Scrum Foundation Professional (SFPC) and Exin Green IT Citizen certifications.

Vagner de Brito Nascimento. He is studying for a PhD in Computer Science, specifically with computer networks, infrastructure and multimedia communication. He also works as a university professor in courses in the area. He has been a member of a research group since 2005, where, in addition to academic activities, he has published several articles, book chapters, short courses and conference lectures. He has also been working on national and international research projects, sometimes working in leadership roles. As a manager of infrastructure, networks and IT systems for 5 years, he became interested in horizontal, organic management, and began to use this knowledge with traditional methodologies to increase productivity. Curriculum Vitae (CNPq): http://lattes.cnpq.br/9645867210501112

Fabio Kon is Full Professor of Computer Science at the Institute of Mathematics and Statistics at the University of São Paulo (IME-USP). He carries out research in the fields of Software Engineering, Distributed Systems, Smart Cities, Free and Open Source Software, and Innovation and Technological Entrepreneurship. In 2013, he was Visiting Professor at the Technion, Israel, where he conducted research on Software Startup Ecosystems and Digital Entrepreneurship. In 2018, he was a Visiting Professor at MIT, USA, where he carried out research on Data Science applied to Smart Cities.

Antônio Abelém. He is Fellow Productivity Technological Development and Innovative Extension CNPq—Level 2. He is full Professor on the Computer Science Faculty at the Federal University of Pará (UFPA) in Belém, capital of the Brazilian state of Pará. He holds a PhD in Computer Science from the Catholic University of Rio de Janeiro (PUC-Rio, 2003). He is an associate and Member of the Board of the Brazilian Computer Society (SBC). He coordinates the Research Laboratory in Computer Network and Multimedia Communication (GERCOM) from which participates and coordinates several national and international projects. He is a TPC member and reviewer of national and international conferences and journals. He was president-director (CEO) of Guamá Science and Technology Park in Belém-PA, from 2010 to 2018, responsible for the structuring, implementation, and management of the park. His research is focused on future internet, software defined networking, network function virtualization, internet of things, performance analysis, and network security. Currently, he is spending a year as visiting professor at the University of Massachusets (UMass), in Amherst-MA, where he carries out research on Quantum Internet.

How to cite this article: Bertrand-Martinez E, Dias Feio P, Brito Nascimento Vd, Kon F, and Abelém A (2020), Classification and Evaluation of IoT Brokers: A Methodology, *Int J Network Mgmt*, *2020; e2115. https://doi.org/10.1002/nem.2115*