Dissertação de Mestrado

Inatel

Performance Evaluation of IoT Middleware

Mauro Alexandre Amaro da Cruz

Dezembro/2017

Performance Evaluation of IoT Middleware

Mauro Alexandre Amaro da Cruz

Dissertação apresentada ao Instituto Nacional de Telecomunicações, como parte dos requisitos para obtenção do Título de Mestre em Telecomunicações.

ORIENTADOR: Prof. Dr. Joel José Puga Coelho Rodrigues

Santa Rita do Sapucaí 2017

Cruz, Mauro Alexandre Amaro da

C957p

Performance Evaluation of IoT Middleware. / Mauro Alexandre Amaro da Cruz. – Santa Rita do Sapucaí, 2017.

67 p.

Orientador: Prof. Dr. Joel José Puga Coelho Rodrigues.

Dissertação de Mestrado em Telecomunicações – Instituto Nacional de Telecomunicações – INATEL.

Inclui bibliografia.

1. Arquitetura de middleware para IoT 2. Middleware 3. Internet das Coisas 4. Métricas de avaliação do desempenho 5. Plataforma 6. Qualitativa 7. Quantitativa 8. Mestrado em Telecomunicações. I. Rodrigues, Joel José Puga Coelho. II. Instituto Nacional de Telecomunicações – INATEL. III. Título.

CDU 621.39

Performance evaluation of IoT middleware

Dissertação apresentada ao Instituto Nacional de Telecomunicações – Inatel, como parte dos requisitos para obtenção do Título de Mestre em Telecomunicações.

Trabalho aprovado em______/______/______ pela comissão julgadora:

Santa Rita do Sapucaí-MG – Brasil

2017

“Intelligence is a gift that must be used to help mankind”.

Spider-Man 2 (2004)

Dedicatória

Aos meus Pais e irmãos que me ensinaram o caminho da verdade com o objetivo de me tornar uma pessoa capacitada e responsável.

AGRADECIMENTOS

Primeiro quero agradecer a Deus pelo dom da vida, e pelas sucessivas benções.

Agradecimento especial ao Ministério das Telecomunicações e Tecnologias de Informação de Angola pela concessão da bolsa de estudo. Em especial ao Ministro das Telecomunicações e Tecnologias de Informação, José Carvalho da Rocha, bem como ao Director do fundo de apoio às telecomunicações, Américo dos Santos.

Agradeço ao Prof. José Marcos Câmara Brito, que acreditou no meu potencial quando mais ninguém acreditava, e num dia em que tudo parecia perdido disse: Não fica triste, você é inteligente, vai conseguir.

Agradeço ao meu orientador Prof. Joel José Puga Coelho Rodrigues que sempre foi muito paciente, usando métodos para que eu pudesse aprender. Obrigado pelos conselhos, ensinamentos, confiança e valiosa orientação.

Agradeço a Gisele Moreira dos Santos, da seção de registros acadêmicos. Sem sombra de dúvidas a pessoa mais importante do Mestrado.

Agradeço ao Instituto Nacional de Telecomunicações-Inatel, Brazil pelo apoio neste trabalho, bem como a todos Professores pelos preciosos ensinamentos, em especial aos Professores Antônio Marcos Alberti e Luciano Leonel Mendes. Agradeço também ao Leonardo Luciano de Almeida Maia, coordenador do escritório internacional do Inatel.

Agradeço aos meus amigos e colegas do Mestrado. Em especial, Wanler Gaspar, Elvira Diogo, Branislav Coseiro, Ivo Guimarães, Natacha de Carvalho, Eliet Cardoso, Danielly Bravin, Bruna Ferreira, Erica Santos, Chandala Carvalho, Malenga Landama, Indira Marília, Diego Amorim, Ellen Paradello e Bianca Magri.

Agradeço à minha família que mesmo distante esteve sempre presente.

Agradeço de forma especial a minha namorada Sheila Janota pelo apoio e compreensão na finalização deste trabalho.

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

SOA	–Service-Oriented Architecture
OWL	– Web Ontology Language
QoS	– Quality of Service
IPv4	– Internet Protocol version 4
IPv6	– Internet Protocol version 6
OMA	– Open Mobile Alliance
W3C	– World Wide Web Consortium
HTTP	– Hypertext Transfer Protocol
HTTPS	– Hypertext Transfer Protocol Secure
MQTT	– Message Queing Telemetry Transport
SDK	– Software Development Kit
CoAP	– Constrained Application Protocol
JSON	– JavaScript Object Notation
PnP	– Plug and Play
IoT	– Internet of Things
CRM	– Customer Relationship Management
ERP	– Enterprise Resource Planning
OS	– Operating System
LVM	– Logical Volume Manager
PaaS	– Platform as a Service
RTT	– Round-trip Time
LAN	– Local Area Network
XML	– eXtensible Markup Language
JSON	– JavaScript Object Notation
SenML	– Sensor Markup Language
GUI	– Graphical User Interface

RESUMO

A Internet de Coisas (do Inglês, *Internet of Things* – IoT) é um termo usado para descrever um ambiente em que Bilhões de objetos que possuem restrições de recursos ("coisas”) estarão conectados à Internet e interagindo de forma autônoma. Com tantos objetos interagindo de forma autônoma em soluções IoT, o ambiente no qual eles estão inseridos torna-se mais inteligente. Um software, chamado *middleware*, desempenha um papel fundamental pois é responsável por parte da inteligência em IoT, atuando como um "cérebro", integrando dados de dispositivos, permitindo que eles se comuniquem e tomem decisões com base em dados coletados. Por natureza, os ambientes inteligentes são heterogêneos. Com uma infinidade de tecnologias disponíveis, o *middleware* pode prosperar, desempenhando um papel ainda mais relevante em ambientes amplos e extremamente complexos como cidades inteligentes. Esta dissertação explora os requisitos das plataformas para IoT, propõe um modelo de arquitetura de referência para *middleware* IoT e detalha o melhor método de operação de cada módulo proposto. O documento também propõe métricas, tanto qualitativas quanto quantitativas, para avaliação de soluções de *middleware*, que se pretende o mais objetiva possível. A seguir, efetua-se um estudo de avaliação comparativa do desempenho de soluções de *middleware* de código aberto (*open-source*) e de uma solução proprietária desenvolvida pelo Inatel no cenário do Inatel Smart Campus. Efetua-se a análise dos resultados e conclui-se que as métricas propostas estão muito bem ajustadas a este tipo de soluções e podem desempenhar um papel extremamente importante na escolha das melhores soluções tanto em trabalhos de pesquisa como em ambientes reais e para a indústria. A plataforma Sitewhere é a solução de *middleware* que obteve melhor desempenho no estudo realizado.

Palavras chave – Arquitetura de middleware para IoT, Internet das Coisas, IoT, Middleware, Métricas de avaliação do desempenho, Plataforma, Qualitativa, Quantitativa.

ABSTRACT

The Internet of Things (IoT) is a term used to describe an environment where Billions of objects that are constrained in resources ("things") are connected to the Internet, and interacting autonomously. With so many objects interacting in IoT solutions, the environment in which they are inserted becomes smarter. A software called middleware plays a key role since it is responsible for most of the intelligence in IoT, acting as a "brain", integrating data from devices, allowing them to communicate, and make decisions based on collected data. Smart environments are heterogeneous by nature, considering the plethora of available technologies, and middleware can thrive, playing even a more relevant role in large scenarios, such as smart cities. This dissertation explores the requirements of IoT platforms, proposes a reference architecture model for IoT middleware, and details the best operation method of each proposed module. The document also proposes metrics, both qualitative and quantitative to evaluate middleware solutions objectively. Then, a performance evaluation study of open-source middleware solutions, as well as a proprietary solution developed by Inatel for the Inatel Smart Campus scenario. The results are analyzed and it is concluded that the proposed metrics are well adjusted for this type of solution and can play an important role when choosing the best solutions for a research work, as well as for real-life environments and industry. Sitewhere is the middleware solution that obtained better performance in the conducted study.

Keywords – Internet of Things, IoT, Middleware, Middleware architecture for IoT, Performance evaluation metrics, Platform, Qualitative, Quantitative.

Chapter 1: Introduction

1.1. Motivation

The increasing miniaturization of electronic components and technologies has enabled the development of connected objects to the Internet for various applications. The term Internet of Things or IoT (Internet of Things) has been used to characterize this class of new products, which is mostly composed of sensors and actuators, that are connected to the Internet. Several segments of the industry are using wireless sensor networks (WSNs), and machine-to-machine (M2M) communication, changing the way business and processes are managed and optimized. Entire systems are composed of several sensors connected to a network, which collect and provide updated data. These sensors allow companies to look at their products and processes from an entirely new perspective. It is estimated that the number of Internet-connected devices will reach 50 Billion by 2020 [1]. This significant number of connected devices calls the attention of academia, industry, and regulators, since the total annual economic impact due to IoT is estimated to range from 2.7 to 6.2 trillion USD (United State Dollars) by 2025 [2]. It is a common misconception to think that the value of IoT consists on being able to remotely control objects through a mobile application. The real value of IoT relies on the data collected by the objects, especially after it is processed according to a context [3]. A software called middleware will be the focus of this dissertation because it is responsible for gathering data from devices, allowing them to communicate, and make decisions based on collected data.

1.2. Problem definition

When the Internet was envisioned, the primary concern was connectivity among computer networks and human-generated data. This paradigm prevailed for many years but is shifting with the uprising of IoT. These things make machine-generated data more relevant, since the number of Internet-connected devices contrasts with the 7.7 billion human population that is estimated in the same period [4]. The number of IoT-enabled devices is expected to grow exponentially, so the amount of machine-generated data will follow the same path. As the number of devices with a computational capacity increase, the surrounding environment will be smarter [5].

Human-generated data refers to the data that are produced through the recording of human choices or is pre-processed by a human (e.g., when clicking a “like” button on social media, computer stores the data). In another perspective, machine-generated data refers to data that are produced entirely by a machine as a result of its decisions. These data can also be generated by observing humans instead of recording only their actions (e.g., data analytics and big data). Figure 1 illustrates two scenarios of human-generated and machine-generated data. In (a), a camera records a video and saves the data to a hard disk, this counts for human-generated data. In (b), a camera is recording video, stores it in a hard drive and, then, analyses the footage, extracting meaning and context from the recording, this counts for machine-generated data.

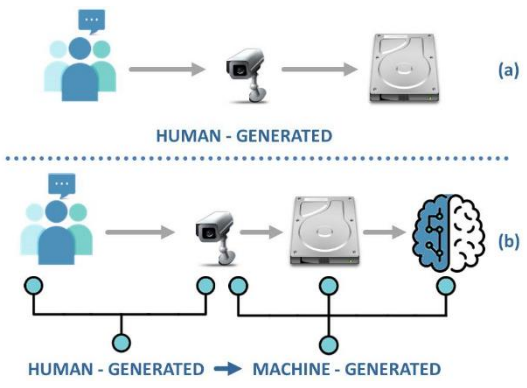


Figure 1 – *Illustration of (a) human-generated data, and (b) human-generated data transformed into machine-generated data.*

The amount of data produced by large IoT environments, such as Smart Cities, is unprecedented because they heavily rely on machine-generated data. These data increasing pose significant challenges for researchers and industry. One of these difficulties is related to M2M communications. The main principle of communication implies that each collocutor must “speak” the same language. In IoT, this is a big issue since there is a plethora of devices, each one with its own language that does not follow the standards [6] (as well as vendors that are not concerned with the compatibility of their product devices with others). Human history shows that different regions adopt different standards. Power sockets are a notable example of how difficult it is to select a standard, they exist for at least a century, and various standards are used across the globe. In IoT, this compatibility problem is solved through middleware [7][8], a software that provides interoperability between incompatible devices and applications. Otherwise, they should not communicate. IoT environments, such as smart cities, are tremendously heterogeneous, and IoT middleware is one of the technologies that enables them [9][10]. In the literature, IoT middleware solutions are sometimes referred to as IoT platforms or IoT middleware platforms because generally, the middleware is a platform, but it is not the only type of IoT platform. Developing/selecting an IoT middleware is not a simple task since there are many architectural and service requirements that even when developers agree upon, the final implementation may not cover specific scenarios or even parts of everyday situations. Another problem is that there are many middleware solutions, both open-source and proprietary offered by technology companies, all very similar to each other regarding the provided features; and no performance metrics, or even guidelines to objectively compare this type of software are defined in the literature.

1.3. Objectives

The main objective of this dissertation is to conduct a performance evaluation study of open-source middleware solutions, as well as a proprietary solution from Inatel, using objective metrics. The study will also act as a guideline for those trying to choose or create a middleware for their IoT solution. To attain this main objective, the following partial objectives were defined:

- Review the IoT middleware solutions and performance evaluation mechanisms for middleware systems available in the literature.
- Identification of functional and non-functional requirements of IoT middleware solutions;
- Proposal of qualitative and quantitative metrics for IoT middleware;
- Selection of solutions for the comparison study;
- Performance evaluation and results analysis of the studied IoT middleware considering the identified requirements and proposed metrics according to the Inatel Smart Campus scenario.

1.4. Thesis statement

The choice of an IoT middleware is highly critical for real IoT solutions, given the complexity and diversity involved in IoT environments. A performance assessment based on objective metrics can substantially contribute to selecting the best middleware for each scenario under study. Without middleware solutions in IoT, it can get the point where only devices from partner brands can interact with each other.

The middleware solutions can be evaluated in a real environment where it is possible to determine which is best for a given IoT solution. The performance evaluation scenario considered for this study is based on the Inatel Smart Campus project, since “the best” middleware is conditioned by the environment in which it is deployed.

1.5. Document organization

The remainder of this dissertation is organized as follows. Chapter 2 provides an overview of the IoT technologies, displaying sizable challenges it faces, highlighting the vital role played by IoT platforms, and more specifically, IoT middleware platforms. A reference architecture model is proposed, that details the best operation method of each module of an IoT middleware platform. Chapter 3 proposes qualitative and quantitative performance metrics to evaluate middleware solutions. Chapter 4 evaluates open-source middleware solutions using the proposed qualitative and quantitative metrics. Chapter 5 concludes the dissertation, displaying invaluable learned lessons, main conclusions, and suggestion for further studies.

1.6. Publications

During this research work, three scientific papers were published.

Publication in International Journals:

1. **Mauro A. A. da Cruz**, Joel J. P. C. Rodrigues, Jalal Al-Muhtadi, Valery Korotaev, Victor Hugo C. Albuquerque, "A Reference Model for Internet of Things Middleware," in *IEEE Internet of Things Journal*, IEEE, ISSN: 2327-4662, DOI: 10.1109/JIOT.2018.2796561 (online; in press). ISI Journal Citation Report with impact factor 7.596 in 2016; Scimago journals ranking: Q1; Qualis B2.

2. **Mauro A. A. da Cruz**, Joel J. P. C. Rodrigues, Arun K. Sangaiah, Jalal Al-Muhtadi, Valery Korotaev, "Performance Evaluation of IoT Middleware," in *Journal of Network and Computer Applications*, Elsevier, ISSN: 1084-8045 (online; in press). ISI Journal Citation Report with impact factor 3.500 in 2016; Scimago journals ranking: Q1; Qualis A1.

Publication in International Conference:

3. **Mauro A. A. da Cruz**, Joel J. P. C. Rodrigues, Kashif Saleem, Andre L. L. Aquino, "Towards Ranking IoT Middleware Platforms Based on Quantitative and Qualitative Metrics," *1st IEEE International Summer School on Smart Cities* (IEEE S3C 2017), Natal-RN, Brazil, Aug. 6-11, 2017.

Chapter 2: Internet of Things Middleware

2.1. Introduction to Internet of Things

The term Internet of Things (IoT) was credited by Kevin Ashton when he started a presentation entitled "That 'Internet of Things' Thing", in 1999 [11]. From then, enormous contributions were made on the topic. The Internet of Things is sometimes also referred as the Internet of Everything (IoE) [12]. The IoT is currently considered a relevant topic for researchers, consumers, and service providers. Since its beginning, the term has suffered minimal modifications. Nevertheless, the basics are still the same. IoT can be described as a fancy term for a scenario where anything may be inserted in a network, be uniquely identified, and interact with minimal human intervention [13] [14]. These things can belong to the real world (physical things) both from inanimate pieces and to living animals, or the virtual world (virtual "things") that only exists in a simulation environment [14]. To simplify, a "thing" is an ordinary device that can be uniquely identified and connected to the Internet. Then, if users or applications have access to the information and communicate with these things (objects) through the Internet, it can be considered IoT scenario. It is expected that by 2020, about 50 billion objects may be connected to the Internet [1]. At first glance, it might seem an exaggerated number (and, maybe, it can be), but history has shown that, as the physical size and price of certain technologies reduce, more people can access to them and, consequently, their presence becomes ubiquitous in daily life [15]. For instance, since 2015, the smartphone has surpassed the laptop as the most important device for connecting to the Internet in the UK [4] and, from 2008, there are more devices connected to the Internet than all the world population [1]. Also, 84% of the world population lives in areas where Internet services are offered [16].

Considering the IoT definition, it is easy to conclude that IoT follows the basic principle of things "speaking" the same language, using technologies that perform a good communication among them. To illustrate it, imagine the following scenario: an interesting woman profile is spotted on a social network, and a conversation is initiated through the chat. Both realize that one speaks English and the other Russian. The conclusion is simple. Despite having a direct way to communicate, they do not understand each other, because they are just sending/receiving meaningless data (content). Therefore, none of them can make meaning of it. The same principle applies when “things” interact. Regardless they have an Internet connection, if they cannot interpret each other, the communication will be futile and does not exist.

In computer science, middleware software mediates these interactions. Without middleware solutions, programmers must read a new software specification every time they integrate new software packages, turning these tasks difficult and very time-consuming. In this regard, numerous organizations struggle and prefer integrated solutions from the same vendor, even when they are insufficient or too complicated for their needs because of the simplicity provided by solutions from the same vendor. In IoT, organizations and users will use multiple (and incompatible) software. Then, middleware will be one of its enabling technologies [13][17]. Considering the enormous importance of middleware in IoT, the dissertation studies the requirements of IoT platforms, and notices that in literature IoT middleware are sometimes referred to as IoT platforms even though they are not the only type of IoT platforms. Then, the document proposes a reference architecture model for IoT middleware that details the best operation method of each module. The dissertation also proposes qualitative as well as quantitative metrics to objectively evaluate middleware solutions. Then, it makes use of the proposed metrics to evaluate open-source middleware solutions, as well as a proprietary solution from Inatel.

2.1.1 The standards competition

There will be different devices from different brands and vendors in IoT. Currently, most IoT devices are only compatible with devices from the same brand, or partner brands. For this reason, several standardization initiatives such as IPSO Alliance, AllSeen Alliance, OneM2M, Openconnectivity, Fiware, OpenFog, OpenDaylight, and many more were created. All of these initiatives are developing reference architectures or standards for all IoT layers with the purpose of delivering a more efficient and sustainable IoT. The problem with standards is that history proves that different regions adopt different standards because of many factors that can range from price, implementation complexity, or even political reasons. Power sockets are a notable example, they exist for at least a century, and different standards are adopted across the globe. Big tech companies appear on the member list of more than one of the mentioned initiatives: Intel (5), Cisco (4), Ericsson, Microsoft, Qualcomm, and LG (3), Bosch (2).

Take the Open connectivity foundation, for example, it supports IoTivity [18] and Alljoyn [19], despite both being frameworks that are addressing device connectivity. It is easily inferable that tech companies are not sure what standard will prevail and are not willing to fully commit. Another aspect of the standards competition is that besides the mentioned initiatives, other traditional standardization entities, such as IEEE, 3GPP (3rd Generation Partnership Project), among others, are developing standards for IoT. With so many entities developing competing standards, another question emerges, what is the longevity of such standards, also, what happens when a standard is established, and another that is superior is developed. Therefore, expecting to reach interoperability among devices by enforcing a universal standard is somewhat innocent.

2.1.2 Connecting to the Internet in IoT

In IoT, most objects are constrained in resources. For this reason, nearly everything that works on the current Internet requires a lightweight IoT version [20]. A rapid analysis of the most common wireless methods of accessing the Internet reveals that the current Internet protocol stack does not take the limitations of IoT into account. Wi-Fi (IEEE 802.11 a/b/g/n/ad/ac) is one of the most common ways to access the Internet, and its protocol stack is not suitable for IoT, it does not provide low power consumption on end-devices, or supports a high number of end-devices. For this reason, alternatives have been developed and are being used on IoT, such as the Bluetooth 5 and the IEEE 802.15.4 that is part of both ZigBee and 6LoWPAN (IPv6 over Low Power Wireless Personal Area Networks) protocol stack. Bluetooth 5 is the latest iteration of the popular Bluetooth standard. Similar to Bluetooth 4.2, Bluetooth 5 also supports IP networks [21]. Unfortunately, users rarely explore the IP capabilities provided by Bluetooth. IEEE 802.15.4 is a standard for Low-Rate Wireless Personal Area Networks (LR-WPANs) that specifies the physical and MAC layers of the OSI model [21]. 6LoWPAN and ZigBee deployments use IEEE 802.15.4. 6LoWPAN is an Internet Engineering Task Force (IETF) approach that compresses and encapsulates the IPv6 headers and accommodates them on the frame IEEE 802.15.4 [21]. ZigBee was developed and maintained by the ZigBee Alliance, and it is mostly known for its mesh topology, but it supports other topologies, such as star and tree [21]. The advantage of 6LoWPAN is the use of the well-known Internet Protocol (IP) and, unlike ZigBee or traditional Bluetooth, a complex gateway is not needed when communicating with the

Internet, which requires further security mechanisms and increased overhead. A ZigBee gateway is illustrated in Figure 2; a similar concept applies to any technology that does not support IP natively. Recognizing the importance of IP networks, a modification of ZigBee, called ZigBee IP, was released. ZigBee IP uses many 6LoWPAN concepts, especially the header fragmentation and compression scheme [21].

Another wireless method of accessing the Internet is through 3G/4G networks. Both have the same problems as Wi-Fi. For this reason, wireless long-range network solutions such as Sigfox, LoRa, and IEEE 802.11 ah (HaLow) were developed [22]. As the name suggests, they consume less battery and provide broad area coverage. Both LoRa and Sigfox need a gateway that interfaces with end devices. This gateway connects to a backhaul that provides a connection to the Internet [23], this is depicted in Figure 3. One of the differences between LoRa and Sigfox is that Sigfox operates similarly to a traditional ISP, where the user has to subscribe to the service in order to use it, while LoRa offers technology that any user can purchase, install the infrastructure, and use the network at will. The advantage of IEEE 802.11ah over the other LoRa and Sigfox is that as an IEEE 802.11 standard, it natively supports IP networks [24]. Another promising method of accessing the Internet through IoT is 5G technology, expected to be released to the public around 2020 [25]. 5G presents difference performance requirements for distinct scenarios and IoT is one of them.

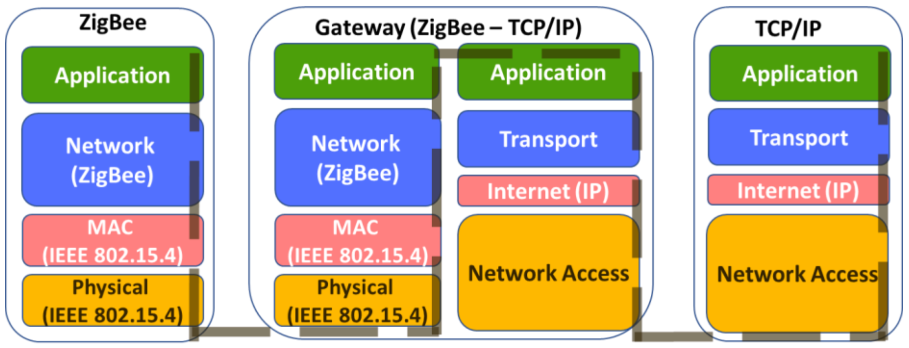


Figure 2 – *Illustration of the ZigBee architecture connecting to the Internet through a TCP/IP gateway.*

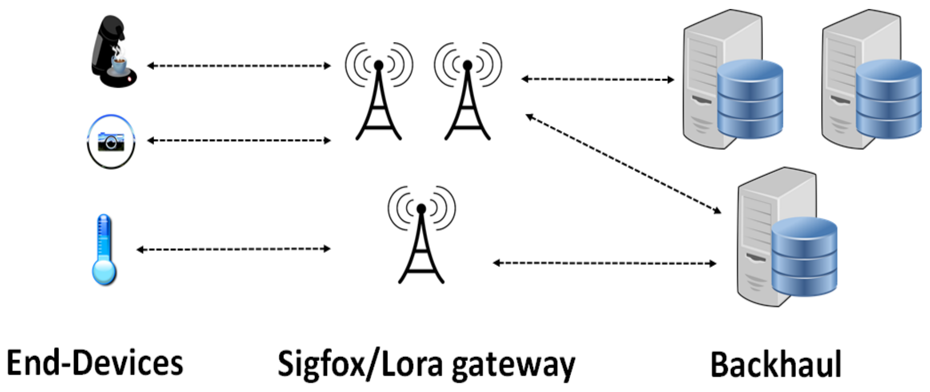


Figure 3 – *Illustration of Sigfox/LoRa overall architecture.*

The current Internet architecture uses the Hypertext Transfer Protocol (HTTP) in the presentation layer (referring to the OSI model), but common HTTP requests consume too much energy for IoT. For this reason, alternative lightweight protocols that are more efficient and practical for end-devices have been proposed for IoT [26]. Two protocols that emerged in this sense, and are being used on IoT are the Constrained Application Protocol (CoAP) and Message Queing Telemetry Transport (MQTT), both expecting a TCP/IP stack. CoAP runs over UDP while MQTT runs over TCP [27] [28]. CoAP is based on the REST model, which allows resource-constrained devices to access resources through REST methods. MQTT relies on the Publish/Subscribe (Pub/Sub) model; therefore, it needs a message broker. Among other aspects, the broker is responsible for sending the message to all subscribed clients. For instance, the Messenger from Facebook uses MQTT. A variation of the MQTT protocol for networks that are not based on TCP/IP is called MQTT-SN [29]. CoAP generates less overhead than MQTT for all message sizes when the packet loss is low; when the packet loss is higher, CoAP produces less overhead only when the message size is small [28]. When the message is large, the probability that TCP loses the message is smaller than UDP, which causes MQTT to retransmit the entire message fewer times than CoAP [28]. Another aspect of IoT is data representation. Currently, the most used encoding technique is JSON, but one of its biggest strengths (easily readable to humans) implies more computational capacity when encoding or decoding as well as transmitting. However, JSON is far superior to its competitor XML [30]. In the current Internet, this inefficiency is worth the advantages, but in IoT every Byte counts. Therefore, binary encodings such as Apache Thrift and Google's Protocol buffers are better suited for most IoT devices [31]. Despite JSON inefficiency in IoT, many devices in IoT environments still use it. However, to maximize efficiency, they should only use JSON encoding when strictly necessary.

2.1.3 Simplified IoT architecture

In computer science and engineering, an architecture describes the general organization of a system, abstracting from restraints such as implementation technology [32]. It goals to understand and describe a system behavior. A significant amount of architectural proposals for IoT can be found in related literature. To summarize the different approaches, the most relevant layers that are available in most solutions are illustrated in Figure 4. They are as follows: *i*) Users or applications, *ii*) IoT platform, and *iii*) devices and infrastructure.

Users or applications: This upper layer addresses the users and auxiliary applications such as decision support tools or social media.

IoT platform: Is a software package that integrates devices, networks, and applications. The platforms hide implementation complexity from the user, because they support and enable IoT solutions by providing an ecosystem where things are built upon [33].

Devices and infrastructure: At the low layer, the physical IoT infrastructure is located. It includes network devices (including "things"), multiple access, and modulation techniques.

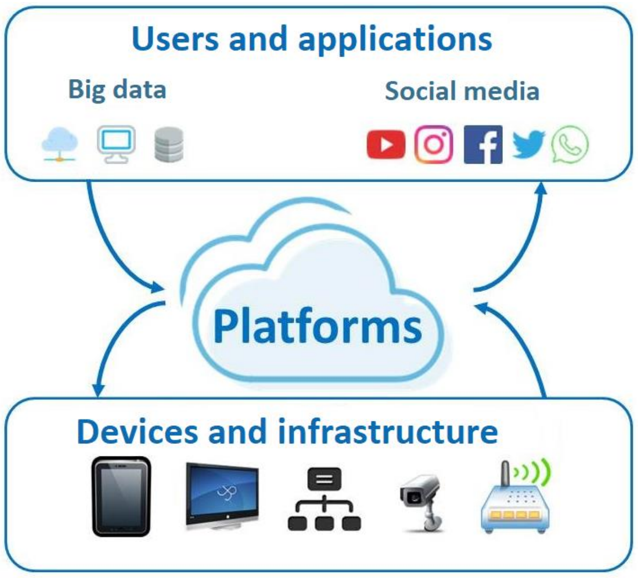


Figure 4 – *Simplified IoT layered architecture.*

2.2. IoT platforms

An IoT platform is a software package that integrates devices, networks, and applications. These platforms optimize business performance by hiding implementation complexity from the user, because they support and enable IoT solutions. These software are called platforms because they provide an ecosystem where everything is built upon [33]. As a software, platforms possess requirements, software engineering states that requirements are divided in functional and non-functional [12].

2.2.1 Functional requirements

Functional requirements are functionalities that describe what a system should be qualified to perform (what should be done) [34]. There are cases where functional requirements state what systems should not do [12]. Either functional requirements are met or not, there is no objective way of quantifying them. The functional requirements of IoT platforms are described as follows.

Resource discovery: If an individual does not know what are his capabilities he cannot advertise them to the others. The same principle is applied in IoT, where it is crucial for things to be aware of their abilities and limitations, so they can announce to peers what resources they offer. Expecting a human to complete this task for every IoT device manually is not only unrealistic but impractical, so discovery mechanisms need to scale well. Resource discovery is the process used by a device to search for the desired resources, where the entire network is probed for services [10].

Resource management: Every application requires QoS (Quality of Service) to be reliable, and that is only possible through fair resource allocation. Platforms should be able to estimate device battery-time, current memory usage, and other relevant internal data to facilitate resource allocation and satisfy application needs. An efficient resource management can guarantee that a device that is handling many requests or is low on battery is requested less often if other devices are able to perform the same task.

Data management: Data are critical in every application; It holds a big part of IoT value, so it should be appropriately handled. In this paragraph, data refers to what is sensed by the thing, or any other information that is interesting to the application. Data management consists of acquiring data, storing in a database, and processing through analytics. When data is processed and interpreted in accordance with a context it is called information.

Event Management: IoT applications can generate a massive number of events. Event management is an extension of data management. After storing data, other applications make use of it; meaning that accurate decisions can be made in real-time with the information provided by the data, and the proper events are generated.

Code Management: Updating every device in person is unpractical, and IoT will have a plethora of them. Platforms should facilitate updating operations since they possess a connection to devices.

2.2.2 Non-functional requirements

Non-Functional requirements are certain aspects that a system should ensure, to guarantee QoS (Quality of Service) [34]. These requirements are described as follows.

Scalability: An IoT platform needs to be scalable, since the things connected to a network grow exponentially, so will the amount of data. Platforms should provide a similar QoS as time passes and more devices are added.

Real-time or Timeliness: Most applications will rely on real-time data, so data must continuously be updated. In computer science, the term real-time means that the user barely perceives the delay between sending data, and the amount of time the computer takes to receive and process the data.

Reliability: Is the likelihood that the software will experience no failures in a specified timeframe. The specified timeframe depends on the scenario. This means that the timeframe can be the duration of a single task or even the entire software lifecycle.

Availability: Platforms supporting critical IoT applications must be available at all times. The platform should remain operational when executing tasks, even if it is experiencing failures. Reliability and availability should work together to ensure some level of fault tolerance.

Security: One of the most significant concerns in every application is always security. In IoT, that aspect is even more critical since a compromised object could perform all sorts of attacks such as DoS (Denial of Service) or even disclose sensitive information such as user location, regular schedule, or even live video. The implications of such data being exposed are limitless, and platforms should do their best to protect user data, while also providing intrusion detection mechanisms.

Privacy: A substantial amount of Facebook and Google revenue comes from collecting user data and selling to advertisers (users consent to this practice in the service agreement). However, there is no way of being sure what data they collect. Privacy issues are related to the willing disclosure of data are an enormous concern. This problem is even more severe when VoiceLabs (devices that are always listening) [35], such as Amazon Alexa and Google assistant are used. An IoT platform escalates the risks further with the amount of collected data. A business model that could be popular in the future is for users to consume cloud middleware systems for free with the tradeoff of the data being sold to advertisers or other interested parties.

Ease of deployment, maintenance, and use: These platforms will be handled by users, who might not have technical expertise. The average user should be able to install, maintain, and use the platform easily. Software that are easy to use are generally preferred by the public and usability without compromising security will probably be one of the key aspects of successful IoT solutions.

Interoperability: The platform should be compatible with various devices and applications with minimal effort from developers. If the Platform supports many devices, it will gain a boost in popularity and will indirectly turn the solution more scalable. A way of reaching interoperability is if besides the popular HTTP(S), the platform also supports common IoT communication protocols such as CoAP and MQTT.

Spontaneous interaction: New devices will continuously be added to the network, or even repositioned. These changes in the network will occur at any time. Platforms should help devices discover and interact each other with minimal human interference.

Multiplicity: Multiple devices are expected to communicate simultaneously; when various devices offer the same service, platforms should help other IoT intervenients decide which one provides the best service. If instead of querying a single entity, the device merely broadcasts a service solicitation to the entire network, the device would then have to decide which is the best (in the case that more than one entity provides the desired service). If a single entity is enquired for the best device for a service, the decision of the most suitable service is delegated to a “smarter” player. The problem with querying a single entity is that better devices will be prioritized. Therefore, better devices will not always be able to provide the best service due to memory constraints (too many requests being processed), or even constraints from the physical world such as distance. These are issues related to multiplicity [36], and platforms should take them into account when replying.

Adaptability and Flexibility: The platform should be able to adapt to long-term changes, as well as be flexible enough for short-term alterations. The platform should also be viable across multiple scenarios.

2.2.3 IoT platform categories

The best would be for IoT platforms to support all the mentioned requirements. Instead, most IoT platforms are built to support some of the previous requirements and fall under three categories that are described as follows *i*) Device management, *ii*) application development, and *iii*) application enablement. Table 1 displays a list of IoT platforms in alphabetical order, and it also displays which categories each one targets.

Device management platforms are focused on remote device management and the optimization of network resources. The definition of device management that is going to be used in this dissertation is inspired in OMA DM (Open Mobile Alliance Device Management) specification. According to this standard, device management consists (**but is not restricted**) to setting initial configuration (**provisioning**), changing parameters or settings (**maintenance**), delivering updates (**upgrading**), query device status, diagnostics, error reporting (reporting), and event processing [37]. These platforms also focus on connectivity, as well as optimizing the usage of network resources. They collect the network capabilities and optimize the network resources by offering tools that facilitate data delivery, device detection, and network diagnostics. If a specific device in the network is overloaded or is short on battery, the platform should notice and take proper actions. Plug and play is another concern for this type of platform, so when new devices enter the network or get repositioned, little configuration by the user is necessary. It is important to notice that device management usually requires that additional software is installed on the device. Notice that some software frameworks that enable D2D connectivity will also be included in this category.

Application development platforms are focused on developing secure applications that can scale to many users, and deal with the heterogeneity present in IoT environments. This type of platforms also offers built-in tools to integrate with popular service providers allowing the developed applications to be compatible with them. Platforms that merely provide basic SDKs (Software Development Kit) to send/receive data on their platform will not be included in this category. However, software development frameworks and toolkits specifically for IoT will be included in this group.

Application enablement platforms are focused on enabling and integrating external applications. They provide means to manage and visualize data, which accelerates application development and facilitates integration with enterprise systems such as CRM (Customer Relationship Management) and ERP (Enterprise Resource Planning). Additionally, these platforms also secure user data and enable information exchange among various devices/applications. This type of platform is also called IoT middleware platform, or IoT middleware and is the focus of this dissertation. It is very common for this kind of platform to also advertise themselves as supporting device management. However, most do not offer ways of delivering updates. From here on, the terms middleware, IoT middleware, and IoT middleware platform will be used interchangeably. The middleware is one of the enabling technologies for IoT [10][9] Further details regarding IoT middleware platforms can be found sub-section 2.3. Around 43 middleware platforms were identified in the literature. However, not all of them are included in Table 1 because they were discontinued a long time ago.

Table 1 – *Available IoT platforms and corresponding targeted areas.*

Name	**App enablement**	**App Development**	**DM and Connectivity**
Alljoyn (Framework) [38]			X
Amazon IoT platform [39]	X		
Artik Cloud [40]	X		X
Autodesk Fusion Connect [41]	X		
Carriots [42]	X		X
Chorevolution [43][44]		X	
CloudPlugs [45]	X		
Devicehive [46]	X		
EVRYTHNG [47]	X		
Fiware (Orion+STH) [48][49]	X		
GroveStreams [50]	X		
InatelPlat	X		
Iotivity (Framework) [51]			X
Kaa [52]	X		
Konker [53]	X		
Linksmart [54]	X		X
Losant [55]	X	***	X
M2Mlabs (Framework) [56]		X	
Microsoft Azure IoT Suite [57]	X		
Nimbits [58]	X		
Nitrogen [59]	X		
OpenIoT [60]	X		X
Sitewhere [61]	X		
Stack4Things (Framework) [62][63]			X
Tago [64]	X		
Telit IoT platform [65]	X		X
Temboo (Toolkit) [66]		X	
ThingSpeak [67]	X		
Thingworx IoT platform [68]	X		
Ubidots [69]	X		
WSO2 IoT server [70]			X
Webinos [71]	X		X
Xively [72]	X		X

*** – Although the development for Losant is for the Losant platform, the tools are very advanced.

2.3. IoT middleware platforms

As the name suggests, middleware is a software that is located in the middle (between two things). The primary goal of a middleware is bringing different systems together so they can interact with each other [73]. The role of middleware is not only to enable communication but to facilitate it. No middleware can be applied to every scenario, so they are generally built for specific or set of scenarios. In the literature, IoT middleware solutions are sometimes referred to as IoT platforms or IoT middleware platforms because generally, the middleware is a platform, but it is not the only type of IoT platform.

In IoT, middleware acts as a translator. To illustrate it, imagine a scenario where three people from different nationalities debate. If they do not have a common language among them (the standardization option), they would need a translator mediating the conversation. Now imagine that the three people are different applications (APPs). APPs communicate through APIs (the language), each APP has its own API. Without a middleware (the translator) each APP must understand every other API. This simple idea allows users to focus on the problem and is illustrated in Figure 5, because instead of knowing how each application works, users manipulate data from one application (the middleware).

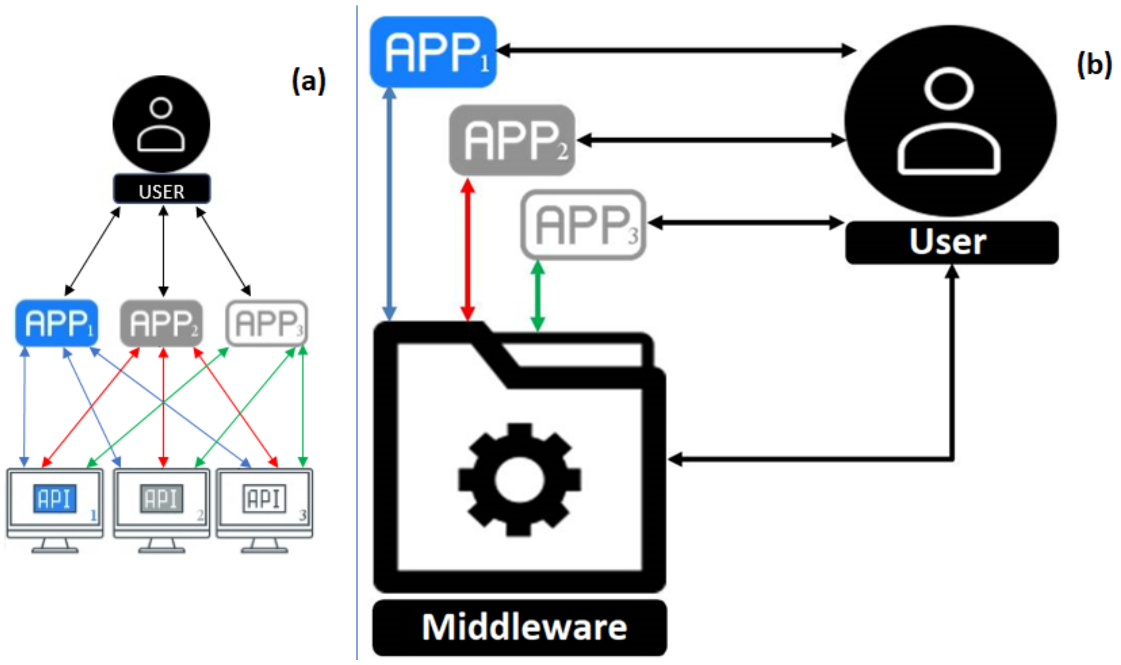


Figure 5 – *Illustration of the communication (a) without middleware and (b) with middleware.*

There are many IoT middleware solutions available in the literature as well as the market. Some of these solutions are open-source and free to download, trial, like most open-source, the code can be altered at will. Other solutions are closed-source, and are only available in the cloud in the form of PaaS (Platform as a Service). The advantage of PaaS solutions is that they are located in the cloud, and authenticated users can access the data located on the server from anywhere around the globe without having to worry about deploying or managing the infrastructure [74]. Both open-source and closed-source middleware solutions from Table 1 are described below.

Amazon IoT platform is an IoT middleware platform developed by Amazon. It supports MQTT, REST, and Websockets communications with its server. One of the biggest advantages of Amazon IoT is that it easily allows interaction with other Amazon services such as S3, Machine learning, CloudWatch, and many more. Their business model is PaaS.

Artik Cloud is a platform developed by Samsung. It provides application enablement as well as device management. It supports MQTT, REST, Websockets, and CoAP communications with its server. One of the advantages of Artik Cloud is that popular IoT apps and devices such as Amazon echo and Google Home can be easily integrated with it. Their business model is PaaS.

Autodesk Fusion Connect is an IoT middleware platform developed by Autodesk. It is marketed as supporting all M2M protocols and vendor specific technology from over 50 devices. One of its biggest strength is the fact that it provides comprehensive analytics tools. Their business model is PaaS.

Carriots is a platform developed by Carriots. It provides application enablement as well as device management. It supports MQTT and REST communications with its server. Their business model is PaaS and it can integrate with external systems such as Dropbox.

Cloudplugs is an IoT middleware platform developed by Cloudplugs. It supports MQTT, REST, and Websockets communications with its server. Their business model is PaaS.

Devicehive is an open-source middleware platform created by DataArt and is distributed under Apache license 2.0. It supports MQTT, REST, and Websockets communications with its server. Although it is open-source, an online version is available as PaaS where users can trial for free, or expand to a paid version. To successfully deploy the solution, users must install PostgreSQL, Apache Kafka, and

Java 8 or above. The downside of Devicehive (when deploying a private server) is that measurement data from devices is cached, meaning that if the server is restarted, or runs out of memory all data are lost. If the user desires this feature, it is necessary to create an additional connector or modify backend logic. However, Devicehive plans to support this feature in next releases. More information regarding Devicehive can be found in their official website [46].

EVRYTHNG is an IoT middleware platform developed by EVRYTHNG. It supports MQTT, REST, Websockets, and CoAP communications with its server. An interesting feature is that it allows integration with external Business Intelligence systems. Their business model is PaaS.

Fiware (Orion+STH): It is common for Fiware to be referred as a middleware platform. In reality, Orion Context broker is the middleware. Orion is an open-source middleware platform created and maintained by Fiware and is licensed under Affero General Public Licence (GPL) version 3. It is a publish/subscribe implementation of the NGSI-9 and NGSI-10 Open RESTful API specifications. It only supports REST communications with its server. To successfully deploy the solution, users must have MongoDB installed. The downside of Orion (when deploying a private server) is that its specification states that only the last collected value is stored in the database, meaning that chronological data consultation is not possible. Recognizing the limitations of Orion, Cygnus and STH (Short Time Historic) were developed by Fiware. They both subscribe to Orion notifications, and when values are changed, they are persisted to the database. The main difference between Cygnus and STH is that Cygnus only stores data, and no consultation is possible, while STH allows both. Fiware officially supports both Cygnus and STH. The REST API version that was used in the experiments is v1, instead of v2 Due to incompatibility with STH. In theory, v1 is less efficient than v2. More information regarding Orion and STH can be found in their official documentation [48][49].

InatelPlat is a middleware platform created in August 2017, at INATEL's (Instituto Nacional de Telecomunicações) ICC (INATEL Competence Center). The goal is to provide PaaS for interested buyers. Currently, it only supports REST communications with its server, but the intention is to support other protocols by early 2018. No further information regarding implementation was provided because INATEL desires to keep that information private. The name InatelPlat is temporary, and the final version will have a different name.

Kaa is an open-source middleware platform created and maintained by KaaIoT and is licensed under Apache license 2.0. Although it is an open-source, users can expand to a paid version by contacting the KaaIoT [75]. It supports REST communications with its server, and SDKs can be deployed to devices. To successfully deploy the solution, users must have Oracle Java SDK, either MariaDB or PostgreSQL, MongoDB or Cassandra, and Zookeeper. The downside of Kaa (when deploying a private server) is that it is not possible to inquiry the stored data from the server through the REST API, meaning that the user has to develop another application for this feature. To those who are interested, it is possible to build a REST API that returns data from a MongoDB database using free tools such as Spring tool suite [76]. More information regarding Kaa can be found in their official website [52].

Konker is an open-source middleware platform created and maintained by the Brazilian KonkerLabs. It is licensed under Apache license 2.0. Although it is an open-source, an online version is available as PaaS where users can trial for free, or expand to a paid version. It supports REST and MQTT communications with its server. To successfully deploy the solution, users must have Java SDK, MongoDB, Cassandra, an application server that supports servlets. More information regarding Konker can be found in their official website [53].

Linksmart, formerly known as Hydra [77], is a complete IoT platform that supports device management, as well as application enablement. The app enablement module is called **Linksmart HDS** (Historical Datastore). HDS is an open-source middleware platform that is licensed under Apache license 2.0. It supports REST communications with its server, and data visualization is made through grafana. To successfully deploy the solution, users must have either influxDB or MongoDB installed. Regarding the platforms that were experimented, it is the only one that uses SenML [78] instead of JSON More information regarding Linksmart can be found in their official documentation [54].

Losant is a platform developed by Losant. It provides application enablement as well as device management. It supports MQTT and REST communications with its server. Although the application development tools offered by them are to communicate with their own middleware, the tools are very advanced. One of its biggest advantages is that besides analytics it can also be used on the edge of IoT devices. Their business model is PaaS.

Microsoft Azure Iot Suite is an IoT middleware platform developed by Microsoft. It supports MQTT, AMQP (Advanced Message Queuing Protocol), and REST communications with its server. One of the biggest advantages of Azure IoT suite is that it easily allows interaction with other Azure services such as machine learning, Data warehousing, and much more. Their business model is PaaS.

Nitrogen is an open-source middleware platform. Some of its modules are licensed under MIT license, while others are under the Apache license 2.0. To successfully deploy the solution, users must have Nodejs installed. The disadvantage is that only Nitrogen enabled devices (devices that run Nitrogen software) can communicate with the server. The project has received no updates to its Github repository since March 2015, and the official website domain (nitrogen.io) is for sale [79]. Which leads the dissertation to conclude that the project was terminated. More information regarding Nitrogen can be found in their official documentation [59].

Nimbits is an open-source middleware platform created and maintained by Nimbits; it is licensed under Apache license 2.0. It supports MQTT and REST communications with its server. Although it is an open-source, an online version is available as PaaS where users can trial for free. To successfully deploy the solution, users must have Java, Redis, a java server application, and Mosquitto MQTT installed. The problem with Nimbits is that it is going through a restructure and all documentation related to usage was erased from the official documentation, and the public cloud is down with no estimated date of return.

OpenIoT is an open-source platform that supports device management, as well as application enablement. Created and maintained by the OpenIoT consortium, it is licensed under Apache license 2.0. It supports REST and GSN (Global Sensor Network) communications with its server. To successfully deploy the solution, users must have Java, Maven, JBoss, and Local Virtuoso installed. Although it is a fascinating project, it has received no updates to its Github repository since November 2015. More information regarding OpenIoT can be found on their official documentation [60].

Sitewhere is an open-source middleware platform created and maintained by Sitewhere and is licensed under CPAL-1.0 (Common Public Attribution License Version 1.0). Although it is open-source, users can expand to a paid version by contacting Sitewhere. It supports MQTT, AMQP, and REST communications with its server. To successfully deploy the solution, users must have Java, MongoDB, HiveMQ, and Apache Tomcat. More information regarding Sitewhere can be found in their official website [61].

Tago is an IoT middleware platform developed by Tago. It supports MQTT and REST communications with its server. Their business model is PaaS.

Telit IoT platform is an IoT platform developed by Telit. It supports MQTT and REST communications with its server. It provides application enablement as well as device management. One of its biggest advantages is that besides analytics it can also be used on the edge of IoT devices.

ThingSpeak is an IoT middleware developed by ThingSpeak. It supports REST communications with its server. The differential of this platform is that it offers MATLAB analytics. ThingSpeak started as an open-source project, but currently offers its service in the form of PaaS, although the old version of the server is still up in the Github repository.

Thingworx IoT platform is an IoT platform developed by PTC. It supports REST communications with its server, and additional connectors are available in its marketplace. It provides application enablement as well as device management. One of its biggest advantages is that besides analytics it can also be used on the edge of IoT devices.

Ubidots is an IoT middleware platform developed by Ubidots. It supports MQTT and REST communications with its server. Their business model is PaaS.

Xively is an IoT platform developed by LogMeIn. It provides application enablement as well as device management. Xively supports MQTT and REST communications with its server. One of its biggest advantages is that besides analytics it can be easily integrated with Amazon web services, Salesforce Device Bridge, and custom integrations with external CRM and ERP tools are also possible. Their business model is PaaS.

Webinos is an open-source service platform that supports device management, as well as application enablement. It was developed as part of the EU FP7 ICT Programme and is licensed under Apache license 2.0. Webinos uses the concept of Personal Zones, which allows communication between services and devices. Personal zones are divided into two parts: i) PZH (Personal Zone Hub) and ii) PZP (Personal Zone Proxy). A PZH possesses a public IP address and runs in the cloud [80]. The PZP is a device that is able to run Webinos services. To successfully deploy a personal zone, users must have Nodejs installed. The disadvantage is that only Webinos enabled devices (devices that run Webinos software) can communicate with the server, besides that, it is not suitable for real systems, because many critical features are still unimplemented. It has received no updates to its Github repository pzp module since February 2014, and pzh since March 2015. More information regarding Webinos can be found in their official website [71].

2.4. Middleware reference architecture modules

When IoT is publicized, beautiful scenarios are presented where devices learn from the user habits and react to them, improving quality of life and experience. All the scenarios that are presented finish with a sentence similar to this one: "all of this with minimal human intervention." These scenarios are only possible because of middleware platforms that integrate data from all the devices and acts upon it. For this reason, Middleware are present in most IoT scenarios. Collecting data and react accordingly is a crucial feature in IoT because most devices are small, and resource constrained to make complex decisions. Therefore, the middleware platforms are responsible for part of the intelligence in IoT. To fulfill their goals, the modules of an IoT middleware platform architecture should reflect IoT requirements as follows: *i*) interoperability, *ii*) persistence and analytics, *iii*) context, *iv*) resource and event, *v*) security, and *vi*) Graphical User Interface (GUI). The modules of a considered ideal IoT middleware are presented in Figure 6 and described as follows.

Interoperability module: The IoT is a heterogeneous environment, and the middleware platform is the integrator. Therefore, it should provide an API (Application Programming Interface), that allows software to expose functionalities to other applications and things without sharing actual code [7]. API requests made by things/applications can be performed through any protocol, so the middleware should at least support the most popular IoT application protocols, such as CoAP, MQTT, and HTTP(S) [7]. The module should also support standard data representation methods, like XML (eXtensible Markup Language) and JSON (JavaScript Object Notation), as well as binary encodings (Apache thrift, Google protocol buffer), another data representation that is emerging for IoT is SenML (Sensor Markup Language) [78]. To further extend interoperability, the middleware should provide basic SDKs, so the code can quickly be deployed to devices, and they can send/receive data to/from the middleware platform. SDKs can be vital, because adding new devices to the middleware is relatively easy, but is not scalable in the sense that it is tedious for the user to add various devices at once. Then, adding new devices should be further simplified (without compromising security). This module is intended for future expansions, and is ideal for new and unforeseen technologies to be integrated here.

Persistence and Analytics module: IoT produces a massive amount of data, which needs to be quickly and continuously stored for chronological consultation and further processing [81]. IoT Middleware should use NoSQL databases to store data since they are generally faster than SQL databases because their data model is simpler [82]. It is commonly said that in IoT, Things learn from user habits. In practice, devices are constrained in resources, and the middleware is the one who learns from collected data. Therefore, middleware least it should provide basic analytics, such as simple graphs, averages, or min/max values [8]. However, the best is further data processing through data warehousing, big data, or even feeding these data to deep/machine learning algorithms because the collected data are highly valuable, especially after being processed [3].

Context module: In a communication, context provides meaning to a conversation. IoT environments are expected to adapt to surroundings and context will play a significant role in this regard [13]. A system is context-aware if it is capable of providing relevant information or services according to the task demanded by the user [83]. Regarding user interaction, systems are classified into three levels of context-awareness [83]: *i*) Personalization, *ii*) Passive, and *iii*) Active. Context-awareness personalization is when the user states to the system precisely what he wants, and the system merely follows what was programmed [83] (e.g., user programs the lights to go on when he enters the room). Passive context-awareness is when the system monitors the environment and suggests actions according to the monitored data [83] (e.g., a user walks into a room, and the system asks if it should turn on the lights). Active context-awareness is when the system monitors the environment and acts on the changes to the environment autonomously [83] (e.g., a user walks into a room, and the system autonomously identifies if the user can navigate through the room and turns on the light with the right degree of luminosity). Context-awareness affects the ability to adapt to new circumstances or environments, and is deeply connected to event detection/management. For context-awareness to be achieved, it has to be modeled. In recent years the ontology-based modeling has become mainstream, spawning different standards. A popular standard is OWL (Web Ontology language) that is backed by W3C (World Wide Web Consortium). More information regarding other context modeling techniques, as well as context in general can be found in [83]. Semantic interpretation and ontologies are expected in this module because people communicate semantically and the same is expected when humans interact with machines in IoT environments. For the IoT that is envisioned the best is artificial intelligence in this module (one of the most challenging fields in this technology), but the middleware platform can use external APIs to achieve this goal. Currently, some middleware proposals such as Linksmart and OpenIoT rely on ontologies to reach semantic interoperability between the sensed data [84].

Resource and Event module: For devices to be efficient in their actions, they must know what they can perform and their internal operation status (battery level, internal/external temperature, current memory usage), so they can advertise their resources and discover resources from others. Multiple devices are expected to communicate with each other simultaneously; they can even offer the same service, and better devices are supposed to be requested more often than the others. This means that they will not always be able to provide the best service, due to memory constraints (too many requests being processed), or even constraints from the physical world such as distance. These issues are a concern related to the multiplicity of actions and the limitations of the tiny device [36]. Middleware platforms can minimize these problems by managing and optimizing these interactions. When connecting for the first time to a middleware platform, devices and external applications should announce their capabilities through some sort of text message (e.g., in JSON). Then, the context module semantically interprets the capabilities, and when a device or application needs an individual service, it can query the middleware for nearby devices that are able to fulfill the task. The middleware understands all capabilities provided by the environment and can generate the proper events. Middleware should also facilitate events update through devices [7], as it is not expected a person can manually manage every single device in large environments such as smart cities.

Graphical User Interface: A graphical user interface (GUI) is a must for every modern software, as it makes applications user-friendly. In IoT middleware, the GUI is often referred as Dashboard, because many data will be exchanged, and dashboards present data in a way that is easy to read. Despite GUIs being so important, it is common for open-source middleware platforms do not possess a native GUI, relying instead on integrations with third-party applications such as Freeboard [85] or Grafana [86] to provide dashboards. These third-party applications can be deployed on private instances, are very powerful and relatively easy to use, as the hardest part is having to configure data-sources when using them.

Security module: IoT will not become popular without plug-and-play. This means that middleware should be flexible enough for the average user to handle. Unfortunately, ease of use (usability) is difficult to achieve with the level of security needed by middleware. If the data could be tampered or retrieved by a malicious user or application, the threats would be limitless. IoT devices are not known for their security, middleware platforms should not follow the same trend because they are the brain of IoT. The amount and value of the collected data are significant and must be secure, but the solution is not simple for any IoT scenario including middleware, because devices are very constrained in resources. Encryption, for example, is costly (regarding processing), so lightweight encryption tools or algorithms must be used for this goal, along with a lightweight cryptographic protocol [87]. Public keys require that certificates are updated when they expire, and propagating these updates for every device is not a simple task. Both cryptography and public keys are basic security features that are common on the current Internet, and their limitations in IoT display the problem in hand, so every security aspect that is efficient and can be included exclusively on a powerful server is welcome. With that in mind, the dissertation proposes four essential security aspects for middleware security in IoT. They are: *i*) Per device authentication, *ii*) The credentials to consult and publish data should be different, *iii*) devices should access other device data using their own credentials, and *iv*) middleware should know device habits and store their MAC and IP. More details regarding the proposed security measures, and the reasons behind them can be found in Sub-section 3.2.1. "With great power comes great responsibility," the iconic quote from the famous movie Spider-Man that defines middleware platforms, as they should be updated using the state-of-the-art security mechanisms.

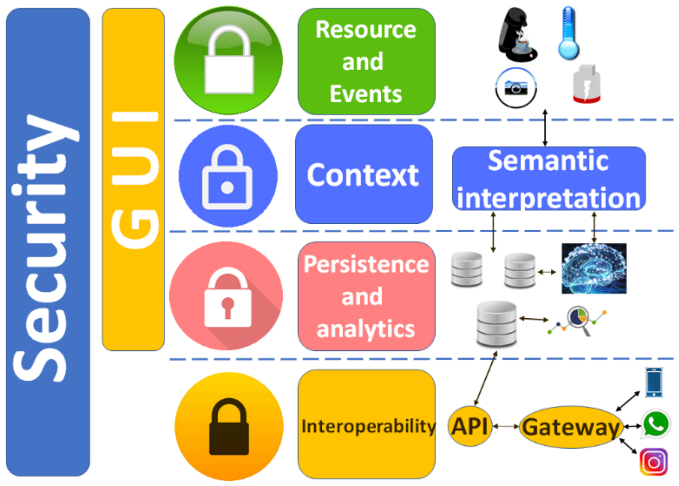


Figure 6 – *Definition of a model for IoT platform modules.*

An IoT environment is characterized by its heterogeneity considering different technologies and data collected will be used across many IoT verticals. However, some scenarios are broader than others. Small solutions like weather stations will just consider data collection and storage, as most of their data are predictable and repetitive; then, it will most likely perform basic analytics and expose data for external consultation. In big verticals, such as smart cities, that can include energy management, smart parking, smart transportation, mobility, etc., data are unpredictable. The middleware platform should be equipped with AI mechanisms to analyze broader scenarios. In practice, this means that not all possible scenarios require all the presented modules since in small scenarios such as a weather station, a simple middleware platform that facilitates data consultation and storage might suffice.

Chapter 3: Performance Metrics for IoT Middleware

The first step towards selecting an IoT middleware platform is defining the scenario since a considerable part of the available middleware platforms are built for specific scenarios. The same principle applies when weighting performance metrics, some scenarios are more sensitive to certain parameters than others. Performance metrics may be both qualitative and quantitative. Qualitative metrics are metrics that are difficult to translate into numbers because the way they are perceived depends on the person that is analyzing (e.g., beauty). Quantitative metrics are easily converted into numbers and can be quantified. Individuals can disagree on who is more beautiful (qualitative), but if one is taller than the other, or possesses longer hair, it is undeniable (quantitative). There are many available IoT middleware platforms; therefore, qualitative metrics are used to filter the middleware that will be present in the quantitative comparison, and quantitative to objectively compare and even rank the tools under study. In both qualitative and quantitative metrics, this dissertation will also display honorable mentions, which are metrics, that although exceptional, do not deserve top honors. Currently, no performance metrics, or even guidelines to objectively compare IoT middleware are defined in the literature. Table xxx summarizes the proposed qualitative metrics that are described below.

3.1. Qualitative (filter) metrics

IoT middleware platforms are so similar in features that the traditional qualitative comparison where a table containing a list of features (reflecting their requirements, or detailing architectural aspects), does not help when deciding which middleware to use. Imagine for example that a solution is marked as not scalable, the reasoning behind such statement should be detailed to its fullest. The qualitative metrics in this dissertation are proposed with the goal of reducing subjectivity, as well as aid users that are trying to select an IoT middleware for a given solution.

3.1.1 Security

Security is an essential aspect of any system, and it seems IoT developers are relegating it to second plan, so products can be developed faster, and the exploits can be later resolved. It is this dissertation view that IoT middleware should not follow the same path, and should ensure data security. For this reason, four fundamental aspects are proposed in this dissertation with the intention of increasing security in IoT middleware, and are based on the assumption that device credentials were somehow compromised. They are the following: *a*) Per device authentication, *b*) Devices should use different credentials to publish and consult data from the middleware, *c*) Devices should access other device data using their own credentials, *d*) Middleware should know device habits and store their MAC (medium access control) and IP (Internet protocol) addresses.

a) Per device authentication is crucial for the safety of middleware data. Every device should have its individual credentials when accessing the middleware platform. If credentials get compromised, and the user notices, the threat is eliminated by revoking or updating the device credentials. However, if all devices share the same authentication, besides revoking or updating the credentials, the user also has to insert them into every other device. Some middleware platforms already follow this guidance.

b) Devices should use different credentials to publish and consult data from the middleware. Some users already comply with the guidance that every device should have its own authentication. However, the implementation is limited, as the same credentials to publish device data are the same that are used to consult. This means that an organization cannot safely expose its device data to external users, without risking that data is tampered. For this reason, authentication per device is not enough and different credentials should be used to publish and retrieve data. To the best of author's knowledge, none of the existing middleware platforms includes this security measure.

c) Devices should access other device data using their own credentials. The former scenario is an excellent example of a weather station, where device data can be retrieved by any interested party, but makes it difficult to discover which device credentials were compromised. Imagine that one day a close friend visits the user house and says he hacked one of the devices and now he always knows what is in the refrigerator. The solution would be to change the consultation credentials of the refrigerator and propagate them to every device that needs it (and that is the problem). A few days later the same friend is back, and compliments for changing the password but says he can consult it again. The friend also says that changing the refrigerator password is pointless, because he hacked another device to get the password. The cycle would go infinitely, because the user cannot determine which device was breached. However, if it is possible for devices to access other device data using their own credentials, by checking the middleware logs, one can determine which device credential was used from an external source and the user can take proper actions. When configuring devices, users should be able to determine what other devices or pre-defined group of devices have access to consulting rights. Also, some devices that simply sensor data and never retrieve it should not have rights to either consult other devices, or its own data. To the best of authors' knowledge, none of the existing middleware platforms deploy this security measure.

d) Middleware should know device habits and store their MAC and IP address. All past scenarios assume that the user notices credential theft, but in real life it is hard to notice such breaches, especially if the middleware does not comprehend the devices habits. For this reason, middleware should know device habits and store their MAC and IP addresses. If the middleware notices that a device is consulting or publishing in different intervals than it regularly does, or is consulting devices that it usually does not, it is an indicator that the device was compromised, and the user should be alerted of the anomaly to take proper actions. However, if the attacker knows this security feature, he can just disable the original device and keeps sending tampered data from any part of the globe. The middleware platform can counter this if it can extract the MAC and IP addresses directly from the HTTP header, and alert the user. In the Internet, IP changes, so the middleware has to detect if the device IP has changed in a reasonable range. The single scenario where the credential theft is not detected with this security features is if the attacker manages to spoof the device regular IP address, clone the MAC address, and keeps the device transmission habits. To the best of authors' knowledge, none of the existing middleware platforms implements this security measure.

3.1.2 Supported application protocols

To maximize compatibility with devices, the middleware platform should support a plethora of application protocols, as it ensures that a vast product range is compatible with the middleware platform. However, the mandatory should be the HTTP(S) (HyperText Transfer Protocol), as the Internet application layer is based on it. Other standard IoT protocols are MQTT (Message Queue Telemetry Transport) and CoAP (Constrained Application Protocol). Here, supported protocols refers to the protocols that can be used to communicate with the IoT middleware platform (i.e., application layer).

3.1.3 Provided Standard Development Kits

An SDK is a set of development tools written in a programming language, which allows programmers to create applications for a specific system (the system could be an application, platform, operating system, embedded system, and more). SDKs that are offered by middleware platforms offer libraries and methods, so devices can interface with the middleware API. Most IoT devices are constrained in resources, SDKs provide an easy and quick way to deploy code into devices so that they can communicate with the middleware platform. Usually, one programming language is enough, but “the more, the merrier”.

3.1.4 Number of updates

Using discontinued software is a challenging and costly operation, and it is not unusual for open-source solutions to be discontinued. Also, it is not uncommon for paid solutions to terminate services due to financial reasons. In these cases, the only alternative is to backup data and migrate to another service provider, or hire specialized staff to maintain the middleware (in the case of a local instance). Observing the number of updates per year, users can have an indicative if the software is being maintained. It is assumed in this paper that, if a middleware platform is updated, at least, once every 2 months (in a year) it is a reasonable number. In the case of open-source solutions, the number of updates can refer to the number of releases located in the respective project Github repository. Some projects have many releases in a single day, so only releases with a date spacing of a reasonable amount of days (further on 10 days is used later on) among them should be counted. In the case of middleware platforms that are only available as a service in the cloud, this resource is not available and observing the changelog is a valid alternative.

3.1.5 Honorable mentions

Popularity: The idea of popularity is somewhat abstract, as it is nothing more than attempting to measure how widespread, known, or accepted a particular topic/thing is. To measure popularity, a poll is generally started, where participants are given specific choices (this is very common in political campaigns). Since it is harder to conduct a scientific poll regarding most research topic, a possible solution would be to check the Google PageRank of each middleware platform and compare (because Google is the most popular search engine). However, since 2016 the Google PageRank information is no longer public [88]. The next best option to Google PageRank are other SEO (Search Engine Optimization) tools such as MOZ [89] and Alexa Page Rank [90]. It is essential to keep in mind that even with an accurate PageRank, some middleware platforms do not have a website dedicated exclusively to them. So, if a corporation develops a middleware platform and places related documentation in a subdomain of the root site, the PageRank is for the root site. An alternative would be to perform a manual Google search using the name of each middleware platform and manually check if the first results are relevant (according to [91], 91% of users do not go beyond the first three results). When performing a manual Google search, one must have in mind that Google considers the user location and previous browser history when returning results.

Support tiers: When organizations adopt a software, they also hire a specialized workforce to deal with possible problems that occur during the software lifecycle. Although the workforce is qualified, some issues require external support from developers, making support a big issue, and some information is too sensitive and cannot be exposed in a public forum. For this reason, direct support is interesting for bigger organizations, and open-source solutions without a “freemium” support option may not suffice.

Mobile App: In all the scenarios, a mobile App for data visualization, and interaction with devices is a good additive. Nevertheless, in some situations, it is mandatory. Taking a smart house for example, turning on a computer to consult temperature in a room is not convenient.

Spam and unsolved issues: Everyone hates spam, so if there are lots of undeleted spam in forums, it is an indicator that moderators probably do not visit the forums regularly. The same is valid if there are many unsolved issues. Both are indicators that the project is dying.

Documentation quality: It is hard to ensure that documentation is good or bad without actually trying to follow the steps. However, if a documentation provides video, images, and examples in their tutorials, it is a good indicator. Also, if it is versioned (every version of the software has its documentation), is an indicator of good document organization.

Table 2 – *Proposed qualitative metrics for IoT Middleware evaluation*

Qualitative metrics	Brief description
Per device authentication SF	Device should have individual credentials
Different credentials to publish and consult data from the middleware SF	This feature allows organizations to publicly expose device data without risking that such data is tampered with
Devices should access other device data using their own credentials SF	It is hard to know which device was compromised without this security feature
Middleware should know device habits and store their MAC and IP address SF	Detecting a compromised device is difficult, and if a device is not doing what it should, the user should be alerted
Number of Standard Development Kits (SDKs)	SDKs provide an easy and quick way to deploy code into devices so that they can communicate with the middleware platform
Supported application protocols	A middleware can boost interoperability by supporting a multitude of application to communicate with it. The most common are HTTP(S), MQTT, and CoAP
Number of updates	Discontinued software are challenging and costly. It is not unusual for open-source solutions to be discontinued
Popularity HM	How widespread, known, or accepted a topic is
Support tiers HM	Some issues require external support from developers. A “freemium” option is necessary for some organizations
Mobile app HM	Mobile apps are mandatory in some scenarios. In others, they are simply a good additive

SF – Security feature.

HM – Honorable mention.

3.2. Quantitative metrics

A fair comparison among different solutions implies that the conditions are the same (for all the solutions) across all considered aspects. For software, this means that the host machine will have the same available resources (memory, processing power, disk space, etc.). This pre-condition turns the comparison between solutions that are only available in the cloud, and local instances very complicated because it is not possible to determine what resources are allocated to the cloud instance. In practice, this means that with more resources, the local instance can perform better in comparison with fewer resources. Quantitative comparisons are easily translated into numbers and graphs. The quantitative metrics proposed in this dissertation are as follows: *i*) packet size, *ii*) error percentage, *iii*) variation of response times, and *iv*) honorable mentions.

3.2.1 Packet size

Most of the energy consumption in devices, is due to communication. Therefore, by knowing the packet size that is necessary to communicate, devices can better manage critical resources such as battery level, and in advanced cases, they can even plan in what intervals to transmit. When analyzing packet size, it is important to remember that REST communications generally end with a response message (response code), so the response message should also be accounted. Depending on the scenario, the packet size can even aid in load balancing. Imagine if the devices use IEEE 802.15.4, which supports transfer rates between 20 and 250 Kbps [21] (take the higher limit into account). If the device needs 1000 Bytes (8000 Bits) to send data, the maximum number of devices that can be attached to a single gateway is 31. This is assuming that the devices are directly connected to the gateway and that the number of received Bytes (to confirm that data were successfully transferred) is less or equal than the number of sent Bytes.

3.2.2 Error percentage

Before a middleware is deployed in a real-life scenario, it is crucial to verify that it can deal with the incoming load, without experimenting errors. This verification is vital for IoT solutions because a plethora of connected devices are expected in IoT, and the middleware will eventually handle a high amount of data. A viability criterion, stating the maximum tolerated error percentage due to the server should be established according to the proposed scenario.

3.2.3 Variation of response times

Response time refers to the amount of time that a software needs to process information. Response time can be critical depending on the scenario. Therefore, knowing the response time of the middleware is crucial, especially in high load scenarios, where a significant amount of data is sent to a server. This experiment can also be performed in a wired LAN if the point is to evaluate whether the application can handle high load or not. In this case, response time can be the round-trip time (RTT).

3.2.4 Honorable mentions

Price: Is always a deciding factor for stakeholders. It should always be considered when evaluating paid, and even free solutions (free solutions still have maintenance costs). The current business model for solutions that offer PaaS (Platform as a Service) consists on charging monthly or yearly per number of requests, analysis, stored records, or sent e-mails. A careful study on the solution should be conducted in accordance with the scenario to make sure the quality-price ratio is accordingly satisfied.

Timeliness performance: Middleware solutions should assure that its performance is not impacted as the volume of stored data grows, or when the solution is running for an extended period of time. In theory, unless the developers make serious programming mistakes, the performance is not impacted as time passes. However, this is still a compelling aspect to verify. It is the old saying “rather safe than sorry”.

DoS and DDoS prevention: A DoS (Denial of Service) attack, unlike regular threats, does not consist on planting malware in the intended machine. A DDoS (Distributed DoS) means that multiple attackers (multiple machines) target a victim. There is no clear solution for DDoS attacks, but DoS attacks are relatively easy to deal with, if the server in which the middleware is hosted was configured correctly. Properly configuring the server is a recommendation that is valid for every Web application. This security tip is valid not only for DoS attacks but also for other common Web attacks. When configuring a middleware server (and every Web application), users should then test the server against some free DoS tools to make sure that the server is secure. With a properly configured server it would be interesting to verify at which point the server starts identifying legitimate requests as DoS attacks and vice-versa, or even if it can identify them at all. A good place to find free DoS tools is Sourceforge [38], but one should be aware that some may contain malware.

Table 3 – *Proposed quantitative metrics for IoT Middleware evaluation*

Quantitative metrics	**Brief description**
Packet size	Most of the energy consumption in devices is due to communication. Knowing help estimate the number of devices per gateway
Error percentage	It is crucial to verify that the server can deal with the incoming load without experimenting errors
Variation of response times	knowing the response time of the middleware is crucial, especially in high load scenarios
Price [HM]	is always a deciding factor for stakeholders and should always be evaluated because software has maintenance costs (even free software)
Timeliness performance [HM]	In theory, unless the developers make serious programming mistakes, the performance is not impacted as time passes
DoS and DDoS prevention [HM]	There is no clear solution for DDoS attacks, but DoS attacks are relatively easy to deal with, if the server in which the middleware is hosted was configured correctly. Properly configuring the server is a recommendation that is valid for every Web application. Verifying if the server interprets legitimate requests as DoS attacks and vice-versa

HM – Honorable mention.

Chapter 4: Performance evaluation of IoT middleware

4.1. Experimentation Scenario

The scenario that is described in this dissertation is similar to the one that is present at Inatel Smart Campus Project, where devices exchange information and send data to a middleware. Data can later be accessed by external users or applications (from any place around the globe). For this reason, it is essential that server continuously stores data that can be retrieved at any-time and any-where with proper credentials. At the beginning of the project, the Inatel Smart Campus used a proprietary middleware solution that was later discontinued by the developers. Maintaining the software started being an issue; for this reason, it was decided to create a middleware solution that could be available to others in the form of PaaS. An advantage of PaaS solutions comes from the fact that they are located in the cloud and authenticated users can access data located on the server from anywhere from the Internet, without having worries about deploying or managing the infrastructure [74]. The number of updates received in the current year is an important deciding factor for this scenario since it forced Inatel to develop its own solution. For the moment, the campus does not desire that only devices running specific software are able to communicate with the middleware, and established that primary method of communication with the middleware platform use REST interfaces. When sending data to the middleware, most devices on the campus send 15 variables, so this will be prioritized.

The middleware platforms that are evaluated in this chapter are open-source, the only exception is Inatel solution that was offered for free to be installed and evaluated. The first step of the performance evaluation study will consider a qualitative comparison among the available solutions in order to filter which middleware platforms are useful to the chosen scenario. The qualitative evaluation will be followed by a quantitative comparison study to objectively evaluate the solutions. Middleware that only offer the solution in the form of PaaS (Platform as a Service) were not included in either the study because a fair quantitative comparison is impossible between a local solution and a cloud instance since it is impossible to determine what resources are dedicated to the cloud instance.

4.2. Performance evaluation considering qualitative metrics

Considering the Inatel Smart Campus scenario, the most important aspects of the qualitative comparison displayed in Table III are the following: permanent data storage (the ones that do not offer this feature are identified by [6]), a REST interface to retrieve data, the number of releases in the last year, and no additional software is mandatory for devices to communicate with the server (the ones that do not possess this feature are signaled with [7]). Regarding the security features described in Section 3.1, only authentication per device is implemented (and only in some middleware). To the best of authors' knowledge, even solutions that are available in the cloud in the form of PaaS do not implement the other suggested security features. Authentication per device will be designated as security feature identified by *a*), different passwords to publish and consult data is mentioned to *b*), specific device restrictions is represented by *c*), and MAC and IP addresses storage are noted by *d*). In this section, ten (10) open-source, as well as a proprietary middleware solution developed by Inatel are studied among 43 identified in the literature. Table 4 displays a qualitative comparison between the middleware, as well as the versions of the software that were compared.

Table 4 – *Qualitative comparison considering Devicehive, Orion+STH, Kaa, Konker, Linksmart HDS, Nimbits, Nitrogen, OpenIoT, Sitewhere, Webinos, and InatelPlat.*

Name	**Communication methods with the server**	**Security**				**SDKs Programming language**	**Releases in 2017** [1]
		a)	**b)**	**c)**	**d)**		
Devicehive v3.3.3 [6]	MQTT, REST	N**	N	N	N	Java, Javascript	> 5
Orion v1.8.0 STH v2.2.0	REST	N	N	N	N	Javascript	3
Kaa v0.10.0	REST [2]	N	N	N	N	Java, C++, C, Objective C	0 [3]
Konker v4.1	MQTT, REST	Y	N	N	N	C, C++, Java, Python	> 5
Linksmart HDS	REST	N	N	N	N	No SDK	N.A[4]
Nimbits v5.0.12	REST	N	N	N	N	Java, Javascript	1
Nitrogen [7]	HTTP [5], MQTT	N.A	N	N	N	Javascript	0
OpenIoT v0.6.1	REST, GSN	N	N	N	N	N.A	0
Sitewhere v1.11.0	MQTT, AMQP, REST, Webscoket	N	N	N	N	Android, IoS	3
Webinos [7]	HTTP(S) [5]	N.A	N.A	N.A	N.A	N.A	0
InatelPlat (Beta)	REST	N	N	N	N	No SDK	1

a) – Authentication per device.

b) – Different passwords to publish and consult data

c) – Specific device restrictions

d) – Understanding device habits as well as MAC and IP address storage.

** – Devicehive uses private keys, but they are generated using the credentials of the same user.

1 – The information regarding the number of releases was extracted from the projects respective official Github repository, and only releases with a spacing of 10 days between them were counted.

2 – Kaa possesses a REST interface to communicate with the server, it is merely for administrative purposes, and it is not possible to retrieve measurement data.

3 – Kaa is preparing the release of a new version that is currently in closed tests.

4 – Linksmart code is not available in Github, and it is not possible to determine the number of releases in the year, only that it is regularly modified.

5 – Nitrogen and Webinos official documentation states that it supports HTTP communications (in the case of Webinos also HTTPS), However, no REST interfaces were found in the documentation.

6 – Devicehive does not offer permanent data storage, meaning that if the server is restarted all data are lost.

7 – Nitrogen and Webinos do not possess a REST interface and no official plugin or auxiliary application are offered

N.A – Details inexistent or not found.

Among the 11 studied solutions, the following 5 middleware platforms, were compliant with the proposed scenario and were considered for the quantitative experiments: InatelPlat, Konker, Linksmart, Orion+STH, and Sitewhere. It is relevant to mention the fact that, among these 11 studied solutions only Konker uses individual authentication for each device and, as previously mentioned, none of the solutions complies with the other proposed security measures. Regarding the communication methods with the server, REST and MQTT were the most common. Regarding SDKs programming language, Java and Javascript were dominant. The other solutions were excluded from the quantitative performance comparison study because the following reasons (per platform):

Devicehive: It was ruled out because the measurement data from devices is cached, meaning that it if the server restarts all data are lost.

Kaa: It is not possible to inquiry the stored data from the server through the REST API, meaning that the user has to develop another app for this feature, and no official plugin or auxiliary application is offered.

Nimbits: Is going through a restructure and all documentation related to usage was erased from the official documentation, and the public cloud is down with no estimated date of return.

Nitrogen: Only Nitrogen enabled devices (devices that run Nitrogen software) can communicate with the server. The project has received no updates to its Github repository since March 2015, and the official website domain is for sale.

OpenIoT: The project has received no updates to its Github repository since November 2015.

Webinos: Only Webinos enabled devices (devices that run Webinos software) can communicate with the server, besides that, it is not suitable for real systems, as many critical features are still unimplemented. It has received no updates to its Github pzp module since February 2014, and pzh since March 2015.

A qualitative comparison is a good method to identify which solutions can be applied to a scenario

4.3. Performance assessment of IoT middleware using quantitative metrics

The experiments present in this chapter were performed in a real network scenario. The packets were generated through Apache Jmeter [92] in a wired LAN with 1Gbps. The experiments were performed in a wired LAN instead of a wireless IoT environment because the goal of the dissertation is to evaluate the application, abstracting from constraints of the physical world, guaranteeing that, if such a high number of requests arrives at the server, the middleware will be able to deal with them. A wired LAN Guarantees that packets successfully arrive in the server, and that network aspects such as delay, or packet loss are not due to the unstable network environment. Before conducting any experiment, it was verified that each solution considered in this study runs for an extended period without needing a restart and stores more than 180 GB of data. For the quantitative evaluation, only five (5) of the eleven (11) platforms could be considered because the others were not compliant with the considered scenario, as described in Section 4.2. For comparison purposes, it is used a simple pointing system to classify the solutions where, for a given topic under evaluation (criterion), the platform that performs better receives five (5) points, the second four (4), and so on. All the platforms start with zero (0) points in each criterion and the gathered points are only valid in each criterion. This pointing system intends to demonstrate that the best solution depends on the prioritized requirement.

The study considers the Inatel Smart Campus scenario where most devices send fifteen (15) parameters. One (1) parameter will have a weight of 0.3, fifteen (15) parameters a weight of 0.6, and one hundred (100) parameters 0.1. Although this weighting system is inspired in the Inatel Smart Campus scenario, it is observed that most devices will not send more than fifteen (15) parameters.

Devices send various variables results to middleware, meaning that an object can send the external temperature, its current location, battery level, and much more. These variables are also referred to as parameters. In the paper, the naming convention used to perform the experiments are the following: variable names = parameter (number of parameter), variable value = 10+number of parameter. To clarify, variable 1 is called parameter1, and variable 100 is parameter100. The value of variable 1 is 11 and the value of variable 100 is 110.

Details on the hardware specifications of the host machine, as well as guest machines can be found in Table 5. Information regarding the OS that each middleware ran can be found in Table 6.

Table 5 – Host and guest hardware specifications.

Host				**Guest**	
Processor	**Frequency**	**OS**	**RAM**	**Cores**	**RAM**
Intel Xeon E5-1620 v3	3.50 Ghz	Windows 10	32 GB	4	8GB

Table 6 – Operating system where each solution was running.

	InatelPlat	**Konker**	**Linksmart HDS**	**Orion + STH**	**Sitewhere**
OS	Ubuntu 16	Ubuntu 16	Ubuntu 16	Centos 7	Ubuntu 16

4.3.1 Packet size to publish data

Here, the sent and received bytes are analyzed because RESTFUL methods generally end with a reply code. This experiment is critical, because it helps to balance the number of devices per gateway. The middleware that performs better will be determined by a sum of sent and received Bytes. Figure 7 presents the packet size of a single REST request to publish data with 1, 15, and 100 parameters.

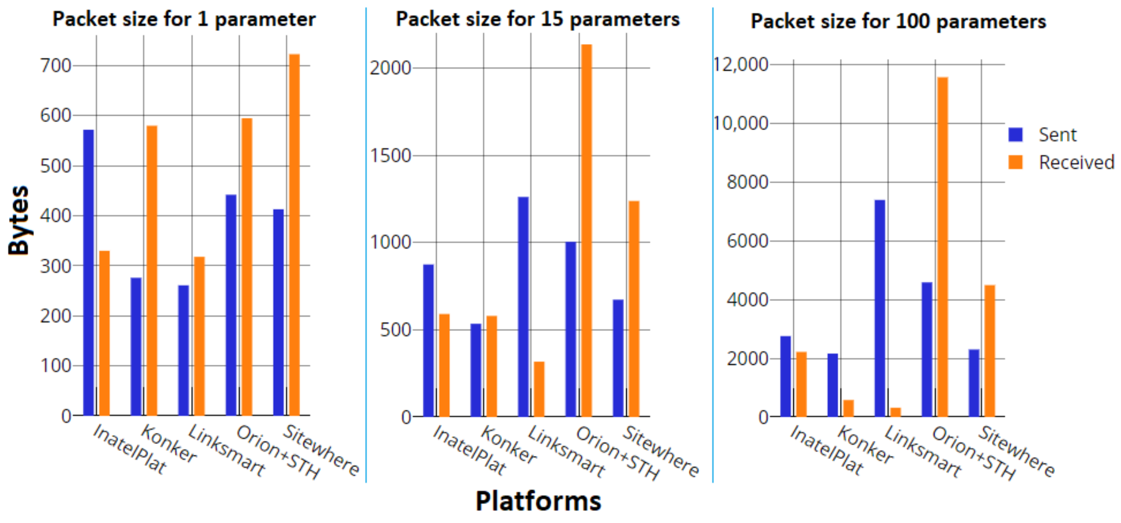


Figure 7 – *Analysis of packet size of a single request where 1, 15, and 100 parameters were sent considering the InatelPlat, Konker, Linksmart, Orion+STH, and Sitewhere middleware.*

In this category, Konker performed better than the rest, followed by Inatel, Linksmart, Sitewhere, and Orion. It is observed that when data is sent to Orion or Sitewhere, more Bytes are received than sent regardless of the number of parameters. Also, Konker and Linksmart are the only middleware platforms in which the number of received Bytes does not increase when more parameters are sent.

4.3.2 Error percentage

The goal of this experiment is to determine if the middleware platforms can deal with the incoming load without errors. To accomplish this goal, data were sent when 100, 1000, 5000, and 10000 concurrent users were present. In each experiment, the users were sending 1, 15, and 100 parameters. In this section the pointing system will be different, since the ones that do not meet the viability criteria shall gather 0 points. The viability criterion considered here states that more than 15% errors will not be tolerated. Keep in mind that all failures were due to the middleware not being able to deal with the number of requests. Figure 8 displays the analysis of the error percentage. With 100 concurrent users none of the middleware platforms encountered errors, so that data are neglected from the figure.

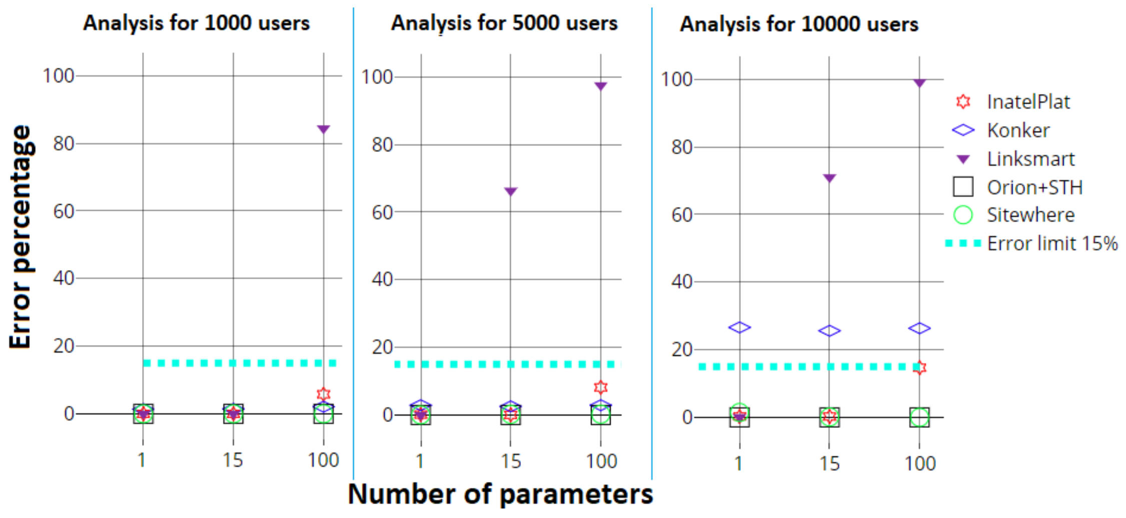


Figure 8 – *Analysis of the error percentage for 1000, 5000, and 10000 users where 1, 15, and 100 parameters were sent considering the InatelPlat, Konker, Linksmart, Orion+STH, and Sitewhere middleware.*

In terms of error percentage, Orion and Sitewhere are virtually tied in every aspect (Orion is slightly better). The next that performs better is InatelPlat, that displays errors in every experiment when 100 parameters are sent. Following, Konker is viable until 5000 concurrent users (where it presents less errors than InatelPlat), and Linksmart is the last displaying a high error rate when it was not viable. InatelPlat is slightly below the viability limit when there are 10000 concurrent users and 100 parameters (14,67). Linkmart cannot deal with many parameters, not being viable in 5 out of the 9 scenarios, and with an astounding error percentage it is unviable.

The packet size and error percentage might be related, because packets that implement more robust code correction mechanisms tend to be larger. However, it is a mere speculation because none of the Middleware provide details in that regard.

4.3.3 Response times

The goal of this experiment is to determine the amount of time each middleware platform needs to process a considerable number of requests. To accomplish this goal, data were sent with 1, 15, and 100 parameters, when there were 100, 1000, 5000, and 10000 concurrent users. All solutions with an error percentage above 15% were excluded in the respective scenario they were deemed unviable (see 4.4.2). Figures 9 to 12 display the response times. For this test, the most important statistical measures are the average and median. The median is favored over the standard deviation because since it was a real-life experiment, and not a simulation, it is normal for the server to attend some of the requests in an extremely high time. Therefore, in this particular case, the standard deviation does not accurately depict the behavior of the system.

A noteworthy aspect to this experiment is the fact that Orion only forwards data to STH if the variable value is different than the one previously registered. For this reason, when experimenting with Orion, in each parameter of each request, a random number was sent, respecting the length of the values presented in 4.4. This was done because otherwise Orion would have a slight advantage in the experiments, since the data would not have to be forwarded to STH, reducing the load on the server.

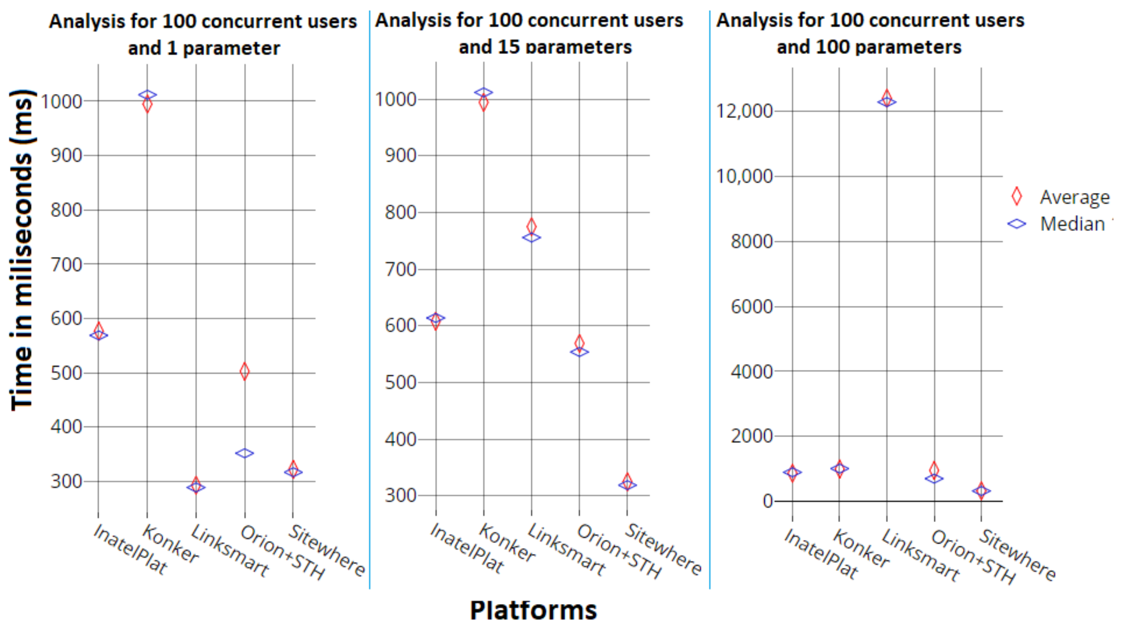


Figure 9 – *Analysis of response time for 100 concurrent users where 1, 15, and 100 parameters were sent considering the InatelPlat, Konker, Linksmart, Orion+STH, and Sitewhere middleware.*

In a scenario with 100 concurrent users, the difference among response times is minimal, the only significant difference is at 100 parameters, where Linksmart response time is 12 times more than the others (demonstrating that even when few data are sent Linksmart has problems dealing with 100 parameters). Another noteworthy aspect is that Konker's median is higher than the average at 1 and 15 parameters. The order from best to worse in this scenario is Sitewhere, Orion, Linksmart, InatelPlat, and Konker.

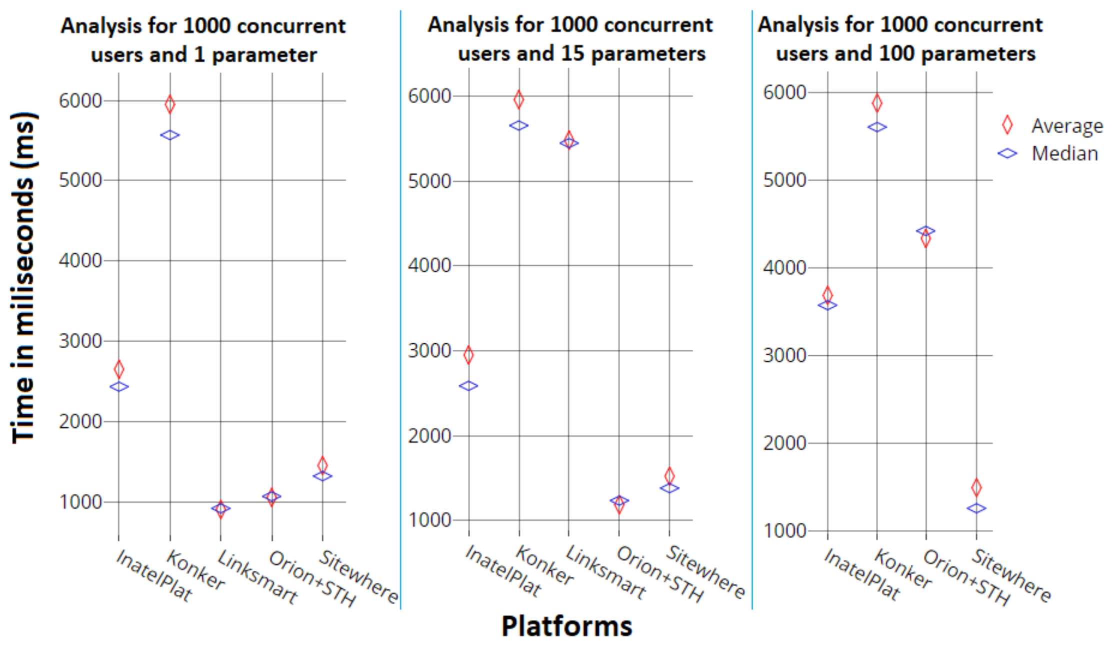


Figure 10 – *Analysis of the response time for 1000 concurrent users where 1, 15, and 100 parameters were sent considering the the InatelPlat, Konker, Linksmart, Orion+STH, and Sitewhere middleware*.

In a scenario with 1000 concurrent users the difference among middleware platforms starts to get noticeable. Orion and Sitewhere clearly dominate, but when sending 100 parameters, Sitewhere response time is far superior than the rest, and Orion is surpassed by InatelPlat. Linksmart was not included in the comparison, where 100 parameters were sent because it presented an error percentage above 15% (see 4.4.2). The ranking from best to worse in a scenario with 1000 concurrent users is Orion, Sitewhere, InatelPlat, Linksmart, and Konker.

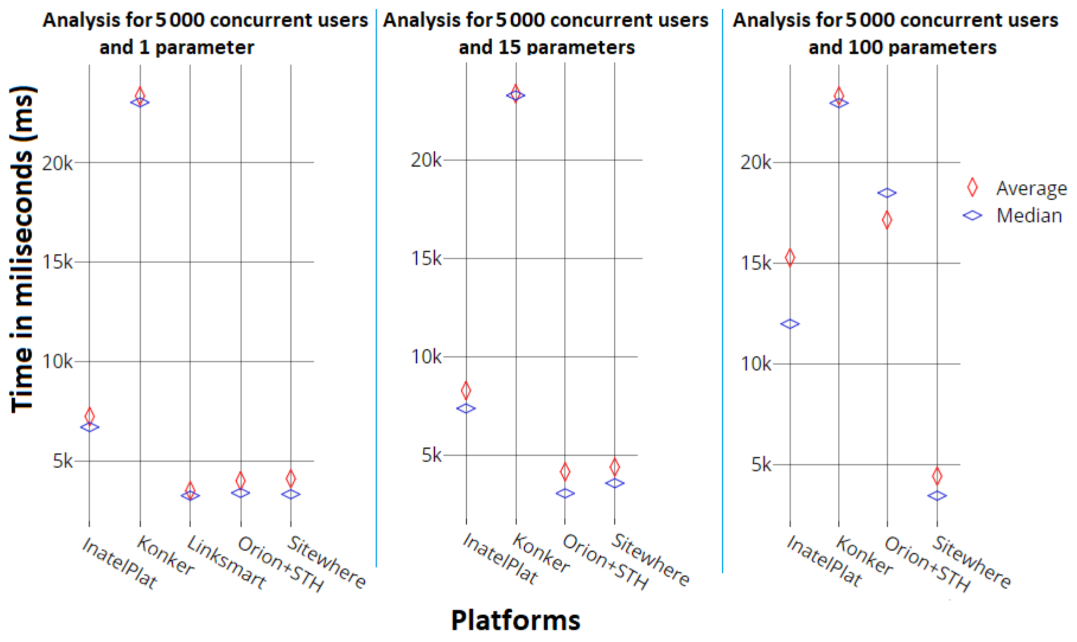


Figure 11 – *Analysis of the response time for 5000 concurrent users with 1, 15, and 100 sent parameters considering the InatelPlat, Konker, Linksmart, Orion+STH, and Sitewhere middleware*.

In a scenario with 5000 concurrent users, the difference between Orion and Sitewhere is minimal and they are virtually tied, but Sitewhere once again is much better with 100 parameters (InatelPlat is the closest middleware when 100 parameters are sent, it shows an average response time more than 3 times higher). Also, the average and median start distancing from each other in most scenarios, however, the average stills higher. The only exception to the median being lower than average is for Orion with 100 parameters. Linksmart was not included in the comparison when 15 and 100 parameters are sent because it presented an error percentage above 15% (see 4.4.2). The ranking from best to worse in this experiment is Orion, Sitewhere, InatelPlat, Linksmart, and Konker. Notice that Linksmart is not viable in this scenario when sending 15 and 100 parameters.

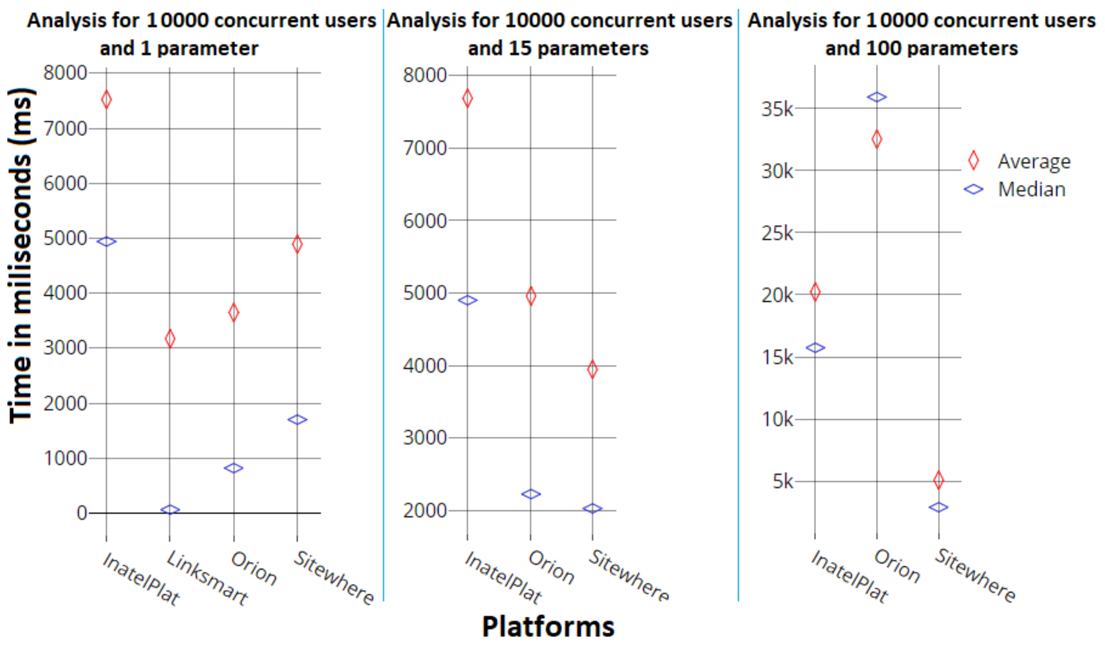
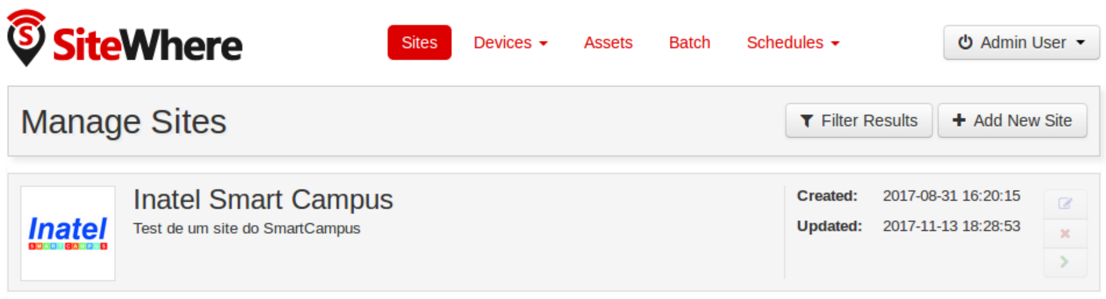


Figure 12 – *Analysis of the response time for 10000 concurrent users with 1, 15, and 100 sent parameters considering the InatelPlat, Linksmart, Orion+STH, and Sitewhere middleware.*

In a scenario with 10000 concurrent users, once again Orion and Sitewhere are virtually tied. When it gets to 100 parameters, once again Orion's performance moves from second, to third position, being supplanted by InatelPlat. Also, Orion's median is once again higher than the average at 100 parameters. The ranking from best to worse in this scenario is Orion, Sitewhere, InatelPlat, Linksmart, and Konker. Linksmart was not included in the comparison where 15 and 100 parameters were sent, because it presented an error percentage above 15% (see 4.4.2). Konker was not included in the comparison where 1, 15, or 100 parameters were sent, because it presented an error percentage above 15% (see 4.4.2). Figures 13 and 14 presents Sitewhere's Graphical User Interface to illustrate its operation.



Manage Sites

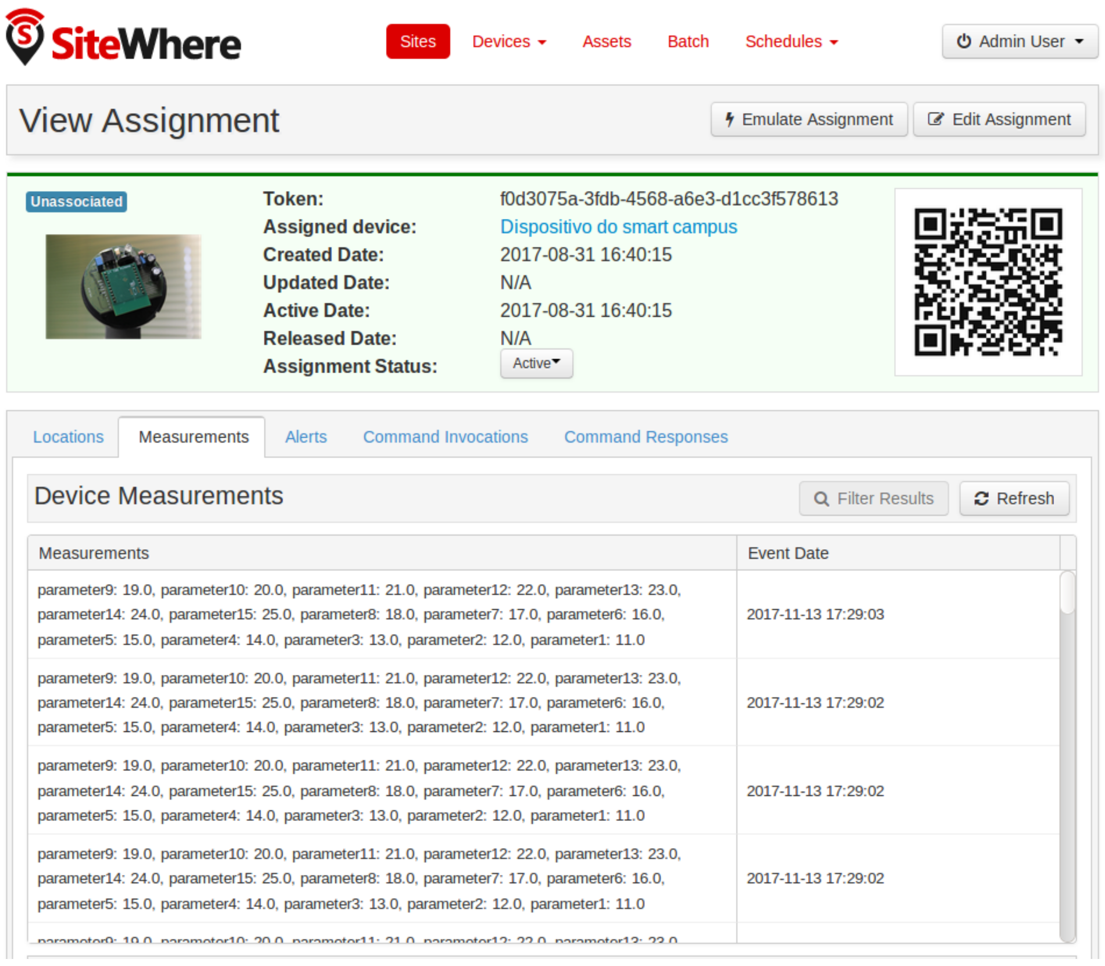
Inatel Smart Campus

Test de um site do SmartCampus

Created: 2017-08-31 16:20:15

Updated: 2017-11-13 18:28:53

Figure 13 – *Illustration of Sitewhere Graphical User Interface.*



View Assignment

Token: f0d3075a-3fdb-4568-a6e3-d1cc3f578613
Assigned device: Dispositivo do smart campus
Created Date: 2017-08-31 16:40:15
Updated Date: N/A
Active Date: 2017-08-31 16:40:15
Released Date: N/A
Assignment Status: Active

Device Measurements

Measurements	Event Date
parameter9: 19.0, parameter10: 20.0, parameter11: 21.0, parameter12: 22.0, parameter13: 23.0, parameter14: 24.0, parameter15: 25.0, parameter8: 18.0, parameter7: 17.0, parameter6: 16.0, parameter5: 15.0, parameter4: 14.0, parameter3: 13.0, parameter2: 12.0, parameter1: 11.0	2017-11-13 17:29:03
parameter9: 19.0, parameter10: 20.0, parameter11: 21.0, parameter12: 22.0, parameter13: 23.0, parameter14: 24.0, parameter15: 25.0, parameter8: 18.0, parameter7: 17.0, parameter6: 16.0, parameter5: 15.0, parameter4: 14.0, parameter3: 13.0, parameter2: 12.0, parameter1: 11.0	2017-11-13 17:29:02
parameter9: 19.0, parameter10: 20.0, parameter11: 21.0, parameter12: 22.0, parameter13: 23.0, parameter14: 24.0, parameter15: 25.0, parameter8: 18.0, parameter7: 17.0, parameter6: 16.0, parameter5: 15.0, parameter4: 14.0, parameter3: 13.0, parameter2: 12.0, parameter1: 11.0	2017-11-13 17:29:02
parameter9: 19.0, parameter10: 20.0, parameter11: 21.0, parameter12: 22.0, parameter13: 23.0, parameter14: 24.0, parameter15: 25.0, parameter8: 18.0, parameter7: 17.0, parameter6: 16.0, parameter5: 15.0, parameter4: 14.0, parameter3: 13.0, parameter2: 12.0, parameter1: 11.0	2017-11-13 17:29:02
parameter9: 19.0, parameter10: 20.0, parameter11: 21.0, parameter12: 22.0, parameter13: 23.0	

Figure 14 – *Sitewhere Graphical User Interface showing sent parameters to Sitewhere's Database.*

Chapter 5: Conclusion and future work

5.1. Learned lessons

Through this research study, valuable lessons were learned, that can aid future researchers interested in open-source software solutions, either as developers trying to create/improve solutions, or users that want to evaluate solutions.

Eliminating blank spaces before sending data to the middleware is necessary, and data precision should be evaluated: Most REST communications are sent without trimming white spaces, and every additional white space is counted towards the number of sent bytes. People who deploy IoT solutions should carefully manage the data their devices send, because as the number of concurrent users grows, the additional white spaces start adding up. The first step is trimming white spaces beforehand. The second is planning the precision of the sent data according to the scenario. For example, if a device is measuring temperature, the user should define if it matters to the scenario that the temperature sent to the middleware is 7.5 instead of 7.533323222.

Documentation should enable users to easily and quickly trial the solution: Ensuring that anyone can easily install and run the solution should be one of the developer's priority, because it increases the likelihood that interested parties will be able to successfully trial the solution. This not only makes it more popular, but ensures that bugs (especially documentation bugs) are discovered. A docker implementation or a virtual machine with every aspect configured ensure that installation is not a problem. Another aspect that is important is related to the post-installation, where users finish installing the solution, but there are no examples to follow. Regarding the virtual machine, it is essential that the developers configure it with LVM (Logical Volume Manager) during the OS installation, so the Disk can be expanded.

Support medium should be public: When trying middleware platforms for the first time, it is common to encounter bugs either in the documentation or the software itself. Another aspect that should be taken into account is human error. For this reason, middleware platforms (especially the open-source) should use a public forum, where users can expose issues, avoiding direct email contacts unless the subject is too sensitive to be exposed in public. Such action creates a knowledge base that is easily accessible by all users. Solutions can use their own private forum, but with so many public forums available on the web it seems unnecessary, especially for open-source solutions.

Developers should provide more information on the necessary software: Something that can be confusing for users that are interested in running the solution is the additional software that is necessary, and how to install it. For this reason, developers should provide details on how to install them, and not assume that the user already knows how to do it. Also, the versions of the additional software in which the installation tested, as well as the OS should be present in the documentation. Clarifying the version is important because in Linux distributions it is common to install packages using YUM (Yellowdog Updater Modified) or APT (Advanced Packaging Tool), and they download the latest version of the desired software, which can be incompatible with the middleware platform. So, imagine that the documentation tells users to install the database through APT, and it was written when the latest version of the database was 1.4. Five months later a user tries to follow the installation, and it fails, because the database version is now 2.0.0. The user will waste valuable time repeating the installation steps and debugging.

Documentation should not be written exclusively by the programmers: It is natural for someone that is so familiar with what was developed to neglect important aspects of documentation. It is not necessarily bad faith because in such cases, the person assumes that what was not described is intuitive. Also, they are so familiar with the problem that they forget to think outside of the box when using it.

Documentation should provide a performance tuning section: Every software is prone to performance issues, either due to programming mistakes, or configuration issues when it is deployed. In the case of mal-configuration, it can be the application itself that was not well configured, the supplementary software such as the database, or even the server in which the solution is hosted. For this reason, the middleware documentation should provide a section where performance tuning issues, and best operation methods are discussed, explained, and exemplified. The presence of such section can also improve security (because the server is well configured), and guarantees that solutions are deployed to their maximum potential. Without such section, users have to search the web for performance tuning of each auxiliary software, or even experiment with the configurations of the software itself through trial and error.

Studies should avoid mentioning discontinued solutions: There are many available IoT middleware platforms, either paid or free. It is very common for free solutions get discontinued. Some such as ThingSpeak even start as an open-source, then move to the model of PaaS (an old version of ThingSpeak can still be downloaded from the Github repository). When conducting research, authors should avoid mentioning discontinued solutions. It is difficult to evaluate whether a solution was discontinued or not, but authors should do their best to warn readers, that the mentioned solution does not receive updates in a significant amount of time. Interested parties can easily track solutions, as well as discover new ones if this is done. This recommendation is not only valid for IoT middleware platforms, but every other type of IoT platform, and software in general.

5.2. Main Conclusions

Throughout this dissertation, an up-to-date study regarding IoT middleware was presented, and the performance of open-source middleware solutions, as well as a proprietary solution from Inatel were studied and evaluated.

The dissertation first introduced the motivation and delimited the research topic, describing the objectives and displaying its main contributions. In Chapter 2, an up-to-date study regarding IoT middleware was presented. This chapter began with a description of IoT technologies, giving attention to the fact that it is common for the same tech company to support competing standards, which is a clear sign that they are not sure which standard will prevail. Also, it is shown how connectivity is different in IoT (in comparison to the current Internet), where the primary concern is low energy consumption on end-devices. Then, the chapter discusses IoT platforms, showing their functional and non-functional requirements, dividing IoT platforms into 3 categories while also revealing the priorities of each one. A table displaying a list of IoT platforms and which categories each one respectively targets is also presented. The dissertation then focuses on IoT middleware platforms, detailing their purpose in IoT and how they accomplish their goals, while also providing more information on some of the existing middleware solutions. The chapter ends by proposing a reference architecture for middleware solutions that details the best operation method of each module.

Chapter 3 presented the importance of performance metrics to objectively compare middleware solutions and highlights the fact that no metrics or guidelines are available in the literature to objectively compare this type of software. The chapter then proposes qualitative and quantitative metrics to compare middleware solutions. The qualitative metrics are an excellent way to filter solutions for a final quantitative test. In both qualitative and quantitative metrics, this chapter also displays honorable mentions, which are metrics that although exceptional, do not deserve top honors.

In chapter 4, the performance of open-source middleware solutions, as well as a proprietary solution from Inatel were evaluated using the qualitative and quantitative metrics previously proposed in Chapter 3. This chapter began with a description of the experimentation scenario, which is Inatel Smart Campus. It then proceeds on showcasing a qualitative comparison among 11 middleware solutions. After finishing the qualitative comparison, 5 middleware were compliant with the proposed scenario; they are InatelPlat, Konker, Linksmart, Orion+STH, and Sitewhere. It is hard to verify the fact that of the 11 solutions, only Konker uses individual authentication for each device.

The 5 middleware that were compliant with the proposed scenario proceeded to a performance assessment using the quantitative metrics where it was verified that when there are few concurrent users (up to 100), the difference between solutions is minimal, and it does not matter much which middleware is deployed. The only exception to this rule is with 100 parameters, in that case, the response time of Linksmart is almost 12 times more than the rest. Overall, Orion+STH and Sitewhere were more stable through all experiments. However, there is no such thing as a best middleware, and when deploying an IoT solution, users should take their scenario into account. In the case of low throughput where packet size is the most crucial aspect, Konker and Linksmart are the best. If the number of parameters sent by the device is the most critical aspect, Linksmart can either be the best in most scenarios with 1 parameter, or the worse with 100 parameters. If error percentage is the top priority, Orion and Sitewhere are the best with less than 1% error rate (Orion is slightly better). If the number of concurrent users is not more than 5000, InatelPlat and Konker are also viable.

The metrics related to Packet size should not be underestimated, because it is a valuable tool when dimensioning the solution. Imagine that devices are directly connected to a gateway, which then forwards the requests to the middleware, and Konker (the most efficient regarding packet size) is the middleware being used, also, the goal is to send 15 parameters. The gateway transfer rate is 1 Mbps. Now consider that packets are sent to the gateway without any compression meaning that 535 Bytes are sent, and 580 received. The maximum number of devices per gateway will be 215 (because more Bytes are received than sent in this case). In a real IoT environment, the data sent to the gateway would be compressed. However, in a real IoT environment, it is common for devices to be connected through a mesh network which is usually slower, and very crowded, meaning that fewer devices would connect directly to the gateway. Knowing the packet size that is transmitted in each scenario, helps users in the distribution of network load and allow a planning of their IoT solution.

Perhaps the biggest challenge in IoT is related to security. Many tech experts do not advise consumers to purchase IoT devices, such as, door locks or children toys that are connected to the Internet. They mention such advises because IoT is insecure, mainly, because developers neglect important security aspects to deliver products faster. If IoT image does not change soon, regaining public (users) trust will be difficult.

5.3. Future work

As future work, it would be interesting to verify if the performance of the middleware platforms would the same in a scenario with unreliable data transmission. However, in such scenarios, many challenges occur, especially regarding load balancing. The number of gateways, as well as the devices per gateway, should be well planned. In case of a mesh network, further planning is necessary because some devices may become overloaded. Even if the work abstracts itself from such constraints by sending data through a wireless IoT network, in which the requests are sent through Apache Jmeter, the variation of the of latency would be significant. Take Inatel Smart Campus as an example, where devices transmit at regular time intervals, it takes around 6 minutes for a group of 20 devices to transmit data to the server. For a similar experiment to be made in an unreliable environment, the number of repetitions should be extremely high due to the unstable network conditions, and the experiment could last years.

Another study that would be relevant is to repeat the experimentations on this dissertation using clustered servers, mainly to observe if the middleware platforms that were unviable at 10.000 parameters improved their performance. Also, repeating the experimentations present in this dissertation using a different application protocol such as CoAP and MQTT.

Finally, developing or modifying a middleware to comply with the proposed security features as well as the proposed security aspects would be interesting.

References

[1] D. Evans, “The Internet of Things - How the Next Evolution of the Internet is Changing Everything,” *Cisco white paper*, pp. 1–11, April 2011.

[2] J. Manyika, M. Chui, J. Bughin, R. Dobbs, P. Bisson, and Marrs, “Disruptive technologies: Advances that will transform life, business, and the global economy,” *McKinsey Global Insitute*, pp. 1–162, May 2013.

[3] Lihong Jiang, Li Da Xu, Hongming Cai, Zuhai Jiang, Fenglin Bu, and Boyi Xu, “An IoT-Oriented Data Storage Framework in Cloud Computing Platform,” *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1443–1451, May 2014.

[4] Ofcom, “The communications market report,” pp. 1-431, August 2015.

[5] G. Fortino, A. Guerrieri, W. Russo, and C. Savaglio, *Internet Things Based on Smart Objects*. Springer International Publishing, Switzerland, 2014.

[6] Guangyi Xiao, Jingzhi Guo, Li Da Xu, and Zhiguo Gong, “User Interoperability With Heterogeneous IoT Devices Through Transformation,” *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1486–1496, May 2014.

[7] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Cla, “Middleware for internet of things: A survey,” *IEEE Internet Things Journal*, vol. 3, no. 1, pp. 70–95, February 2016.

[8] A. H. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and M. Z. Sheng, “IoT Middleware: A Survey on Issues and Enabling technologies,” *IEEE Internet Things Journal*, vol. 4, no. 1, pp. 1–20, February 2017.

[9] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Future Generations Computer Systems*, vol. 29, no. 7, pp. 1645–1660, September 2013.

[10] A. Farahzadi, P. Shams, J. Rezazadeh, and R. Farahbakhsh, “Middleware technologies for cloud of things-a survey,” *Digital Communications and*

Networks, pp. 1-13, April 2017.

[11] S. Huckle, R. Bhattacharya, M. White, and N. Beloff, “Internet of Things, Blockchain and Shared Economy Applications,” *Procedia Computer Science*, vol. 98, pp. 461–466, 2016.

[12] I. Sommerville, *Software Engineering*. 10th ed., Addison-Wesley, Boston, 2015.

[13] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, October 2010.

[14] International Telecommunication Union, “Recommendation ITU-T Y.2060: Overview of the Internet of things,” pp. 1–14, June 2012.

[15] A. M. Alberti, “A conceptual-driven survey on future internet requirements, technologies, and challenges,” *Journal of the Brazilian Computer Society,* vol. 19, no. 3, pp. 291–311, September 2013.

[16] International Telecommunication Union, “Measuring the Information Society Report 2016,” pp. 1-256, November 2016.

[17] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,” *IEEE Communications Surveys and Tutorials*, vol. 17, no. 4, pp. 2347–2376, June 2015.

[18] J.-C. Lee, J.-H. Jeon, and S.-H. Kim, “Design and implementation of healthcare resource model on IoTivity platform,” *2016 International Conference on Information and Communication Technology Convergence (ICTC),* Jeju, South Korea, October 19-21, 2016, pp. 887–891.

[19] O. Tomanek and L. Kencl, “Security and privacy of using AllJoyn IoT framework at home and beyond,” *2016 2nd International Conference on Intelligent Green Building and Smart Grid (IGBSG),* Prague, Czech Republic, June 27-29, 2016, pp. 1-6.

[20] J. Zhou, Z. Cao, X. Dong, and A. V Vasilakos, “Security and Privacy for Cloud-Based IoT: Challenges,” *IEEE Communications Magazine*, vol. 55, no. 1, pp. 26–33, January 2017.

[21] R. K. Ghosh, *Wireless Networking and Mobile Data Management*. Springer, Singapore, 2017.

[22] B. Reynders, W. Meert, and S. Pollin, "Range and coexistence analysis of long range unlicensed communication," *23rd International Conference on Telecommunications (ICT) 2016,* Thessaloniki, Greece, May 16-18, 2016, pp. 1-6.

[23] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, "Long-range communications in unlicensed bands: the rising stars in the IoT and smart city scenarios," *IEEE Wireless Communications*, vol. 23, no. 5, pp. 60–67, October 2016.

[24] B. Badihi, L. F. Del Carpio, P. Amin, A. Larmo, M. Lopez, and D. Denteneer, "Performance Evaluation of IEEE 802.11ah Actuators," *IEEE 83rd Vehicular Technology Conference (VTC Spring) 2016*, Nanjing, China, May 15-18, 2016, pp. 1-5.

[25] J. G. Andrews *et al.*, "What Will 5G Be?," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 6, pp. 1065–1082, June 2014.

[26] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi, "Toward better horizontal integration among IoT services," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 72–79, September 2015.

[27] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight Internet protocols for web enablement of sensors using constrained gateway devices," *2013 International Conference on Computing, Networking and Communications (ICNC)*, San Diego, CA, USA, January 28-31, 2013, pp. 334-340.

[28] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Singapore, Singapore, April 21-24, 2014, pp. 1-6.

[29] E. P. Frigieri, D. Mazzer, and L. F. C. G. Parreira, "M2M Protocols for Constrained Environments in the Context of IoT : A Comparison of Approaches,"

XXXIII Brazilian Telecommunications Symposium, Juiz de Fora - MG, Brazil, September 1-4, 2015, pp. 1-5.

[30] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of JSON and XML Data Interchange Formats: A Case Study," *2009 ISCA 22nd International Conference on Computer Applications in Industry and Engineering (CAINE),* San Francisco, USA, November 4-6, 2009, pp. 1-6.

[31] H. Lampesberger, "Technologies for Web and cloud service interaction: a survey," *Service Oriented Computing and Applications*, vol. 10, no. 2, pp. 71–110, June 2016.

[32] A. Clements, *Principles of computer hardware,* 4th ed. Oxford University Press, Oxford, 2006.

[33] A. Tiwana, *Platform Ecosystems Aligning Architecture, Governance, and Strategy*. Elsevier, 2013.

[34] M. Glinz, "On Non-Functional Requirements," *15th IEEE International Requirements Engineering Conference (RE 2007),* Delhi, India, October 15-19, 2007, pp. 21-26.

[35] P. Dempsey, "The Teardown: Google Home personal assistant," *Engineering and Technology,* vol. 12, no. 3, pp. 80–81, April 2017.

[36] M. A. Chaqfeh and N. Mohamed, "Challenges in middleware solutions for the internet of things," *2012 International Conference on Collaboration Technologies and Systems (CTS),* Denver, CO, USA, May 21-25, 2012, pp. 21-26.

[37] Open Mobile Alliance, "Device Management Requirements: Approved Version 2.0," pp. 1–14, 2016.

[38] Slashdot Media, "SourceForge - Download, Develop and Publish Free Open Source Software." [Online]. Available: https://sourceforge.net/. [Accessed: 22-Oct-2017].

[39] Amazon Web Services, "AWS IoT - Amazon Web Services." [Online]. Available: https://aws.amazon.com/iot/. [Accessed: 22-Oct-2017].

[40] SAMSUNG, “IoT Cloud Platform — Samsung ARTIK Cloud.” [Online]. Available: https://artik.cloud/.

[41] Autodesk Inc., “Autodesk Fusion Connect. Enterprise IoT Software Platform.” [Online]. Available: https://autodeskfusionconnect.com/. [Accessed: 22-Oct-2017].

[42] Carriots, “Carriots - Internet of Things Platform | Home.” [Online]. Available: https://www.carriots.com/. [Accessed: 22-Oct-2017].

[43] M. Autili, P. Inverardi, and M. Tivoli, “Choreography Realizability Enforcement through the Automatic Synthesis of Distributed Coordination Delegates,” *Science of Computer Programming*, October 2017.

[44] Chorevolution, “chorevolution.eu: (Main.WebHome).” [Online]. Available: http://www.chorevolution.eu/bin/view/Main/.

[45] CloudPlugs Inc., “CloudPlugs :: Internet Of Things Platform, IoT, public cloud ::” [Online]. Available: https://cloudplugs.com/. [Accessed: 22-Oct-2017].

[46] DataArt Solutions, “DeviceHive - Open Source IoT Data Platform with the wide range of integration options.” [Online]. Available: https://devicehive.com/. [Accessed: 22-Oct-2017].

[47] EVRYTHNG, “EVRYTHNG IoT Smart Products Platform |.” [Online]. Available: https://evrythng.com/. [Accessed: 22-Oct-2017].

[48] Fiware, “Fiware-Orion.” [Online]. Available: https://fiware-orion.readthedocs.io/en/develop/. [Accessed: 22-Oct-2017].

[49] Fiware, “Fiware-STH-Comet.” [Online]. Available: https://fiware-sth-comet.readthedocs.io/en/latest/. [Accessed: 22-Oct-2017].

[50] Grovestreams, “Welcome - Storage and Analytics for the Internet of Things.” [Online]. Available: https://grovestreams.com/. [Accessed: 22-Oct-2017].

[51] Linux Foundation, “Home | IoTivity.” [Online]. Available: https://www.iotivity.org/. [Accessed: 22-Oct-2017].

[52] KaaIoT Technologies, “Kaa Open-Source IoT Platform 2017 — IoT cloud

platform the Internet of Things solutions and applications that set the standard." [Online]. Available: https://www.kaaproject.org/. [Accessed: 22-Oct-2017].

[53] Konker Labs, "Konker - Your solutions connected in a fast and simple way!" [Online]. Available: http://www.konkerlabs.com/. [Accessed: 22-Oct-2017].

[54] Linksmart, "LinkSmart® Documentation Home - Home - LinkSmart® Docs." [Online]. Available: https://docs.linksmart.eu/. [Accessed: 22-Oct-2017].

[55] Losant IoT, "Losant | Losant." [Online]. Available: https://www.losant.com/. [Accessed: 22-Oct-2017].

[56] M2MLabs, "Home | m2mlabs.com." [Online]. Available: http://www.m2mlabs.com/. [Accessed: 22-Oct-2017].

[57] Microsoft IoT, "Azure IoT Suite—IoT Cloud Solution | Microsoft." [Online]. Available: https://www.microsoft.com/en-us/internet-of-things/azure-iot-suite. [Accessed: 22-Oct-2017].

[58] Nimbits Inc., "Nimbits Platform." [Online]. Available: https://www.nimbits.com/. [Accessed: 22-Oct-2017].

[59] Nitrogen, "nitrogenjs · GitHub." [Online]. Available: https://github.com/nitrogenjs. [Accessed: 22-Oct-2017].

[60] OpenIoT Consortium, "OpenIoT – Open Source cloud solution for the Internet of Things." [Online]. Available: http://www.openiot.eu/. [Accessed: 22-Oct-2017].

[61] SiteWhere, "SiteWhere | The Open Platform for the Internet of Things." [Online]. Available: http://www.sitewhere.org/. [Accessed: 22-Oct-2017].

[62] F. Longo, D. Bruneo, S. Distefano, G. Merlino, and A. Puliafito, "Stack4Things: a sensing-and-actuation-as-a-service framework for IoT and cloud integration," *Annals of Telecommunications*, vol. 72, no. 1-2, pp. 53-70, February 2017.

[63] Stack4Things, "Stack4Things | An OpenStack-based Internet of Things Framework." [Online]. Available: http://stack4things.unime.it/. [Accessed: 22-Oct-2017].

[64] Tago LLC, “Tago - Home.” [Online]. Available: https://tago.io/. [Accessed: 22-Oct-2017].

[65] Telit, “IoT Platform Overview – Telit.” [Online]. Available: https://www.telit.com/products/iot-platforms/iot-platform-overview/. [Accessed: 22-Oct-2017].

[66] Temboo Inc, “Temboo.” [Online]. Available: https://temboo.com/. [Accessed: 22-Oct-2017].

[67] The MathWorks Inc, “IoT Analytics - ThingSpeak Internet of Things.” [Online]. Available: https://thingspeak.com/. [Accessed: 22-Oct-2017].

[68] PTC, “ThingWorx IoT Platform | PTC.” [Online]. Available: https://www.ptc.com/en/products/iot/technology-platform-thingworx. [Accessed: 22-Oct-2017].

[69] Ubidots, “IoT platform | Internet of Things | Ubidots.” [Online]. Available: https://ubidots.com/. [Accessed: 22-Oct-2017].

[70] WSO2, “WSO2 IoT Server - Flexible Open Source IoT Platform.” [Online]. Available: https://wso2.com/iot. [Accessed: 22-Oct-2017].

[71] Webinos Foundation, “webinos | The webinos Foundation.” [Online]. Available: http://webinos.org/. [Accessed: 22-Aug-2017].

[72] Xively, “IoT Platform for Connected Devices| Xively by LogMeIn.” [Online]. Available: https://www.xively.com/. [Accessed: 22-Oct-2017].

[73] A. R. S. Hammergren, Thomas C., *Data warehousing for dummies*, 2nd ed. Hoboken, N.J.: Wiley, 2009.

[74] L. Miller, *Public PaaS for dummies*, 2nd ed. Hoboken, New Jersey No: John Wiley & Sons, Inc., 2016.

[75] KaaIoT, “Kaa IoT Product Development Platform — IoT Application Enablement.” [Online]. Available: https://www.kaaiot.io/. [Accessed: 22-Oct-2017].

[76] Pivotal Software Inc, “Tools.” [Online]. Available: https://spring.io/tools.

[Accessed: 22-Oct-2017].

[77] M. Eisenhauer, P. Rosengren, and P. Antolin, “A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems,” *2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops*, Rome, Italy, June 22-26, 2009, pp. 1-3.

[78] X. Su, H. Zhang, J. Riekki, A. Keränen, J. K. Nurminen, and L. Du, “Connecting IoT Sensors to Knowledge-based Systems by Transforming SenML to RDF,” *Procedia Computer Science*, vol. 32, pp. 215–222, 2014.

[79] “nitrogen.io.” [Online]. Available: http://domain.hacker.sh/parked.html?domain=nitrogen.io. [Accessed: 22-Oct-2017].

[80] WP3, “D3.7: Final webinos specification,” [Online]. Available: http://webinos.org/files/2012/09/webinos-phase_II_architecture_and_components.pdf. [Accessed: 20-Oct-2017].

[81] G. Fersi, “Middleware for Internet of Things: A Study,” *2015 International Conference on Distributed Computing in Sensor Systems*, Fortaleza, Brazil, June 10-12, 2015, pp. 230-235.

[82] N. Leavitt, “Will NoSQL Databases Live Up to Their Promise?,” *Computer*, vol. 43, no. 2, pp. 12–14, February 2010.

[83] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context aware computing for the internet of things: A survey,” *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[84] F. C. Delicato, P. F. Pires, and T. Batista, *Middleware Solutions for the Internet of Things*. Springer, London, 2013.

[85] “freeboard - Dashboards For the Internet Of Things.” [Online]. Available: https://freeboard.io/. [Accessed: 27-Sep-2017].

[86] “Grafana - The open platform for analytics and monitoring.” [Online]. Available: https://grafana.com/. [Accessed: 27-Sep-2017].

[87] S. Singh, P. K. Sharma, S. Y. Moon, and J. H. Park, “Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions,” *Journal of Ambient Intelligence and Humanized Computing*, pp. 1-18, May 2017.

[88] Search Engine Journal, “Google PageRank Officially Shuts its Doors to the Public - Search Engine Journal.” [Online]. Available: https://www.searchenginejournal.com/google-pagerank-official-shuts-doors-public/161874/. [Accessed: 28-Oct-2017].

[89] “Moz | SEO Software, Tools & Resources for Smarter Marketing.” [Online]. Available: https://moz.com/.

[90] “Website Traffic, Statistics and Analytics - Alexa.” [Online]. Available: http://www.alexa.com/siteinfo.

[91] A. J. A. M. van Deursen and J. A. G. M. van Dijk, “Using the Internet: Skill related problems in users’ online behavior,” *Interacting with Computers*, vol. 21, no. 5–6, pp. 393–402, Dec. 2009.

[92] Apache Software Foundation, “Apache JMeter - Apache JMeter^{TM}.” [Online]. Available: http://jmeter.apache.org/. [Accessed: 25-Oct-2017].