This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2961511, IEEE Access

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier

Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison

PAOLA PIERLEONI[1], ROBERTO CONCETTI[1], ALBERTO BELLI[1], AND LORENZO PALMA[1]
[1]Dipartimento di Ingegneria dell'Informazione, Università Politecnica delle Marche, Ancona, Italy
Corresponding author: Roberto Concetti (e-mail: r.concetti@pm.univpm.it).

ABSTRACT Internet of Things (IoT) aims to connect the real world made up of devices, sensors and actuators to the virtual world of Internet in order to interconnect devices with each other generating information from the gathered data. Devices, in general, have limited computational power and limited storage capacity. Cloud Computing (CC) has virtually unlimited capacity in terms of storage and computing power, and is based on sharing resources. Therefore, the integration between IoT and CC seems to be one of the most promising solutions. In fact, many of the biggest companies that offer Cloud Services are focusing on the IoT world to offer services also in this direction to their users. In this paper we compare the three main Cloud Platforms (Amazon Web Services, Google Cloud Platform and Microsoft Azure) regarding to the services made available for the IoT. After describing the typical architecture of an IoT application, we map the Cloud-IoT Platforms services with this architecture analyzing the key points for each platform. At the same time, in order to conduct a comparative analysis of performance, we focus on a service made available by all platforms (MQTT middleware) building the reference scenarios and the metrics to be taken into account. Finally, we provide an overview of platform costs based on different loads. The aim is not to declare a winner, but to provide a useful tool to developers to make an informed choice of a platform depending on the use case.

INDEX TERMS AWS, Azure, Cloud Computing, Cloud-IoT, Google Cloud Platform, Internet of Things, MQTT.

I. INTRODUCTION

Internet of Things (IoT) is an Internet-based paradigm that includes several interconnected technologies for the information exchange between devices, generally small "things" of the real world, that can be identified and monitored through the Internet. IoT applications must take into account different factors depending on the application context. Data produced by things must be processed, interpreted, stored and each implementation choice is important for the success of an application, such as choosing the best Data Base Management Systems (DBMS) for storing the sensed data [1]. Things are widely distributed and they usually have limited storage capacity and processing, with problems concerning reliability, performance, security and privacy. Similar problems that have been found in Mobile Computing (e.g storage, bandwidth, scalability) [2]. These limits lead to an integration with Cloud Computing (CC) [3], which has virtually unlimited capacity in terms of storage and computing power, and is based on sharing resources and maximizing them. CC is a model that allows access to a set of shared and configurable computing resources (e.g. networks, servers, storage structures, applications) offered as services. These resources can be quickly requested, managed and used in a pay-as-you-go model, so the user pays for the amount of effective use of a resource. CC is also location independent, allowing the user's access to cloud services from any location and with any device through the internet connection.

In literature, the focus on CC and IoT and their integration is growing:

- Botta *et al.* [4], [5] have demonstrated the effective complementarity of IoT and CC in terms of communication, storage and computation [5].
- The need for integration of Cloud and IoT is presented in [6]. The authors assert that the Cloud will act as intermediate layer between the applications and the things, concealing all the functionality and complexities

required for processing.

- The study in [7] shows how Cloud-Assisted Remote Sensing (CARS) enables distributed sensory data collection, global resource and data sharing, remote and real-time data access, elastic resource provisioning and scaling, and pay-as-you-go pricing models, underlining potentials for enabling the so-called Internet of Everything (IoE).
- The paper [8] mainly focuses on a common approach to integrate the IoT and CC under the name of CloudThings architecture and examine an IoT-enabled smart home scenario to analyze the IoT application requirements.
- A solution for merging IoT and Cloud is proposed by Nastic *et al.* [9]. They argue that system designers and operations managers face numerous challenges to realize IoT Cloud systems in practice, due to the complexity and diversity of their requirements in terms of IoT resources consumption, customization and runtime governance.
- Liu *et al.* [10] propose a data-centric IoT framework that takes advantages of the Azure public Cloud to realize a centralized IoT management service, extending their previous work [11], where they present a remote monitoring and management solution, that is specific to a plant wall system, based on Cloud and IoT.
- As described in 2013 by Gubbi *et al.* [12] IoT anc CC can converge. They present a Cloud centric vision for worldwide implementation of IoT and a Cloud implementation using Aneka, which is based on interaction of private and public Clouds.

On the global market the first IoT platforms integrated with CC start being developed, in parallel with studies on IoT-Cloud native platforms.

- The paper [13] presents a survey that identifies several service domains IoT Cloud platforms should deal with.
- In [14] the authors propose a framework for evaluating the IoT platforms from the perspective of how widely they cover the potential needs of the application providers, and their results suggests how none of the platforms today offers comprehensive support.
- Pflanzner et al. in [15] introduce a taxonomy of IoT application properties and investigate 23 IoT Cloud use cases performing a detailed classification of them in a survey.

IoT devices produce a large amount of data that must be transmitted for processing. Traditional CC architectures may not meet requirements in terms of latency and real-time decision-making approach. Therefore, according to recent studies [16], Fog Computing could be a solution.

- Fog Computing provides computation, storage and networking at the edge of the IoT network, between devices and Cloud data centres. [17].
- Bonomi *et al.* [18] define characteristics of Fog Computing and demonstrate the role of Fog in the IoT, highlighting how Fog nodes provide localization, enabling low latency and context awareness, and how the Cloud provides global centralization.
- Al-khafajiy *et al.*in [19] focus on the resource management and monitoring and how to balance the Fog load in order to avoid delays for real-time systems and to not get the QoS levels worse.
- In [20] authors present the improved previous framework with optimal management of resources as far as job allocation in an Healthcare scenario.

It is possible to consider Fog Computing as a connection point between devices and the Cloud, in fact all the mentioned studies agree on the central role of the Cloud. Different Fog nodes flow the data to the Cloud. Therefore, studying the performance of a Cloud platform, it is also useful to make weighted decisions on when and how to offload the Fog nodes.

As reported in [21], with the increasing use of the Cloud, depicting the performance of Cloud services become a priority. Many different performance tests that could be performed.

- In [22] authors discuss the evaluation of different Infrastructure as a Service (IaaS) Cloud systems, using micro-benchmarks.
- Shukla *et al.* [23] propose RIoTBench, a real-time IoT benchmark suite used to evaluate Distributed Stream Processing System for IoT applications hosted in Cloud data centers.
- CloudHarmony [24] offers a large collection of benchmarks and users are able to build their own set of benchmarks and reports. This solution can compare Cloud Providers and services, e.g. compute engine, storage, DNS, CDN, but not IoT related services.

According to our best knowledge, there are no studies that compare a typical architecture of an IoT application with the related services that a selection of Cloud-IoT platforms makes available. There are also no studies comparing the performance of the selected platforms on the basis of the IoT services made available.

There are hundreds of IoT platforms available from a range of vendors. Some platforms are highly specialized, other are focused on providing only a subset of the functions that might be required for an IoT system. According to "The IoT Developer Survey 2018" [25] drawn up by the Eclipse IoT Working Group, in collaboration with the Agile-IoT H2020 Project, IEEE, and the Open Mobile Alliance (now OMA SpecWorks), the main general-purpose end-to-end Cloud-IoT platforms that currently have the leadership on the global market are Amazon Web Service (AWS), Microsoft Azure and the Google Cloud Platform. Our analysis will focus on the comparison of these Cloud platforms regarding the services they make available for the IoT.

In order to conduct a complete comparative performance analysis of the three selected platforms, it is necessary to construct a typical scenario of an IoT application. In the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2961511, IEEE Access

meantime, the breadth of the application fields (e.g. smart-home, smart-traffic, industrial applications, e-health) and consequent diversity of the involved services, would make the study limited to the concerned sector. Selected Cloud platforms are all based on a common architecture for IoT services. After a safe connection to the Cloud, devices send their sensed data (directly or through a gateway) to the Cloud. The same devices (as well as other applications or other devices) receive messages resulting from any elaboration on the Cloud. Selected platforms are all based on a publish/subscribe mechanism [26], a message-oriented middleware that allows a distributed, asynchronous, and loosely coupled communication between messages producers and consumers.

All three selected platforms support the MQTT protocol. There are several studies in literature that make a qualitative comparison between the MQTT protocol and other protocols applied in IoT, [12], [27]–[30], underlining how MQTT could be one of the most suitable protocol for IoT solutions. Other studies compare MQTT performance with other protocols such as HTTP [31], REST [32], AMQP [33], COAP [34] o DDS [35]. In all these studies MQTT obtains the best results in term of end-to-end delay and bandwidth consumption. The MQTT protocol is based on a message broker that has an intermediary role between publishing clients' messages and receiving subscribers' messages. The broker uses a topic based system (hierarchical strings) in order to send and receive messages. To receive messages on topic of interest, clients subscribe to that topic.

The main objectives of this study are:

- identify the key elements of a typical architecture of an IoT solution in the Cloud;
- map the services offered by the three platforms according to this architecture;
- identify comparison metrics;
- conduct a comparative analysis building same scenarios and implementing them on the three platforms, in such a way as to have results based on these metrics.

The rest of this paper is organized as follows: in Section II we present the typical architectures of an IoT solution and a cloud-based solution, and we identify the metrics for comparison, focusing on one of the services provided, discussing the creation of scenarios for analysis and implementation choices; in Section III we conduct the analysis of the three platforms according to a reference architecture; we focus on the performance results obtained from the simulations and on their analysis and finally we analyze the pricing tiers of the platforms; conclusions and future developments of our work are illustrated in Section IV.

II. MATERIALS AND METHODS

The objective of the work is to conduct two parallel analysis on the services provided by Cloud Platforms for IoT. On the one hand an analysis on the key points of the platforms, on the other a performance analysis of platform service times and related costs.

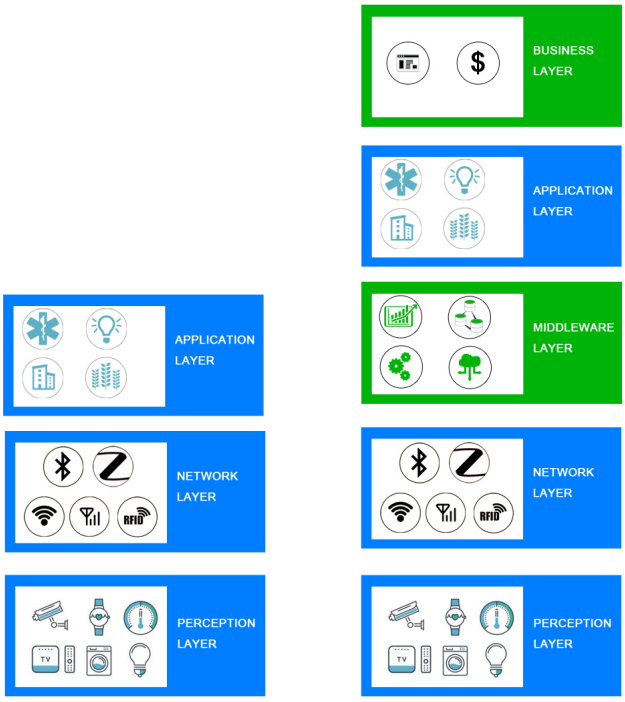

FIGURE 1: IoT Architectures: (a) 3 layers and (b) 5 layers

A. ARCHITECTURES

1) IoT Architecture

The increasing number [36] of heterogeneous connected and interconnected objects make the need for a flexible layered architecture. Even if there is no reference model, the basic models in literature [37] are a 3 or 5 layer architecture [38], [39]. The 3 layer architecture in Fig.1a consist of the Application, the Network and the Perception Layer. The 5 layer shown in Fig.1b add more abstraction introducing a Middleware layer between the Network and the Application, and a Business layer at the top.

The perception layer includes sensors and actuators to perform different functionalities. Data generated by this layer is sent through the network layer, that includes technologies such as RFID, 3G, GSM, UMTS, WiFi, Bluetooth Low Energy, infrared, ZigBee, etc, to the Middleware layer. This layer processes received data, makes decisions, and delivers the required services to the Application layer. Finally the business layer manages the overall IoT system activities and services, and supports decision-making processes based on data.

2) Cloud-IoT Architecture

The Cloud Standards Customer Council [40] created a standard reference architecture for large-scale IoT systems that utilize CC. The "Cloud Customer Architecture for IoT (CCAIoT)" [41] covers an end-to-end solution ranging from the users of IoT devices to the enterprises managing

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2961511, IEEE Access

the devices [42]. The core components of the architecture depicted in Fig.2 across five domains: *User Layer*, *Proximity Network*, *Public Network*, *Provider Cloud*, and *Enterprise Network*. *User Layer* contains IoT users and their end user applications. It is independent of any specific network domain. The *Proximity Network* domain has networking capabilities. Devices (including sensor/actuator, firmware and management agent) and the physical entity are part of the *Proximity Network* domain. The *Public Network* interconnects the devices of different proximity networks through a wide area network, typically the Internet. It also contains Edge Service, that allows the safe flow of data from the Internet into the *Provider Cloud*. The *Provider Cloud* captures data from devices or other data source and provides core IoT applications and associated services (storage, analytics, visualization). It includes components for device management (provisioning, remote administration, software updating, remote control). The insights generated by the *Provider Cloud* are used by user and enterprise application in the *Enterprise Network* domain. IoT governance and security subsystems span all elements of the architecture. The security system has to consider identity and access management (IAM), data protection, security monitoring, analysis and response.

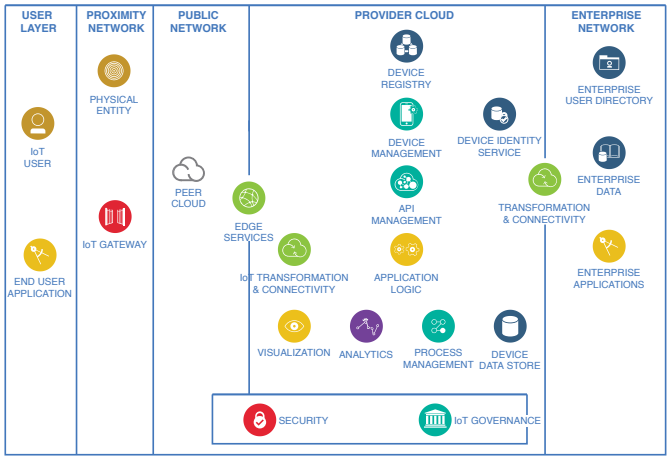

FIGURE 2: Elements of IoT solutions. (Source: Cloud Standards Customer Council, *Cloud Customer Architecture for IoT, 2016)*

The Cloud components of IoT architecture are in a three-tier pattern according to Industrial Internet Consortium Reference Architecture [43] as depicted in Fig.3. This architecture comprises *edge*, *platform* and *enterprise* layers. The *edge* tier includes Proximity and Public Networks of the reference architecture where data is collected from devices and transmitted to devices. A device can either communicate directly, or through an intermediate gateway, with the Cloud. The field-gateway has the role of protocol translation [44] and may be able to perform local storage, filtering and processing actions on received data before sending it to the Cloud [45]. The *platform* tier is the Provider Cloud. It receives, processes, and analyzes data flows both in flight and at rest from the edge tier and provides API management and visualization. It also provides the capability to initiate control commands from the Enterprise Network to the Public Network. The *enterprise* tier implements domain-specific applications, decision support systems and provides interfaces to end-users. It receives data flows from the *edge* and *platform* tier and also issues control commands to the *platform* tier and *edge* tier.

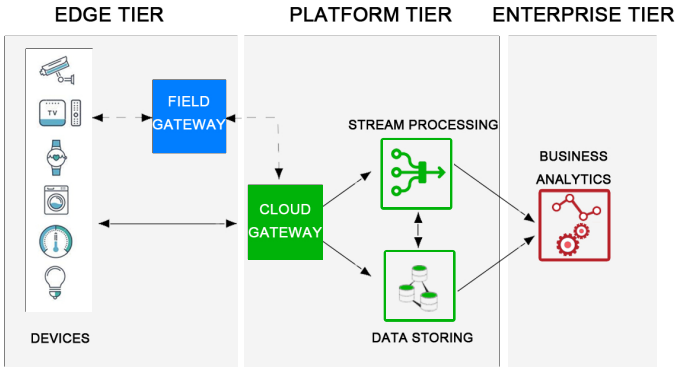

FIGURE 3: High level Cloud-IoT 3-tier architecture

3) Cloud-IoT Platforms

According to the previous depicted architecture, Cloud-IoT platforms provide middleware to connect and manage hardware devices and the data collected by them. IoT solutions in the Cloud require secure, bidirectional communication between devices and a solution back end. An IoT application involves many heterogeneous IoT devices, with sensors that produce and send data or events through a network. They can be used to generate insights (e.g. data processing and analytics). In the rest of this section we investigate the *platform* tier and the *enterprise* tier of the Cloud-IoT architecture and we map components and functionalities in each tier with the selected Cloud Platforms.

The keys that we consider to perform the analysis are:

- *Device management*. Managing IoT hardware devices is one of the core functions of an IoT platform. It includes provisioning of new devices, monitoring and maintenance of operating devices. IoT platforms have to provide features for managing device registration, configuration, over-the-air updates, and asset management, including the ability to list and search connected devices and to query and manage device metadata.
- *Data communication protocols*. In order to enable the remote management of devices and data transmission, a secure and reliable communication between IoT devices, gateways, and cloud-based apps and services is essential. In literature [46] these communi-

cation types for IoT environment are usually referred as "Device-to-Device (D2D)", "Device-to-Application (D2A)", "Device-to-Gateway (D2G)" and "Device-to-Cloud (D2C)". Some data can be stored and processed locally (e.g. field gateway in the WSN), as well as some of the data collected from sensors and other IoT devices need to be delivered to cloud services for further processing. IoT platforms incorporate message broker services to enable devices and gateways to send and receive messages with low latency and at scale. Message broker services use standard communication protocols like MQTT [47], CoAP [48], or XMPP [49], and support web sockets.

- *Rules*. Once data is ingested to the IoT back-end, the flow of data processing may vary, depending on scenarios and applications. Regardless, IoT platforms may offer a rules engine with conditions used to trigger actions.
- *Data storage*. Data generated by IoT devices need to be stored in order to be accessed by further processing. Data is usually split in hot and cold data stores [50]. Hot data stores need to be accessed with high frequency and low latency, whereas cold data stores are accessed infrequently generally with high latency and with lower storage costs still preserving historical data.
- *Integration*. IoT platforms provide integration with other platforms, devices, web services, tools, and applications through SDKs and APIs.
- *Security*. This includes securing devices and network communication in addition to implementing Cloud and application-level security.
- *Costs*. In a general Cloud environment all services are offered in a pay-as-you-go model, in such a way that users pay only the real use of a service/resource. We consider only the cost of the IoT core solution, without considering other connected services, such as storage or data analysis.

Finding the right combination of the aforementioned capabilities and the use case is the basis for designing an IoT solution based on CC. For example in a Smart City scenario information in terms of privacy and data loss are not as important as in a healthcare domain [51]. According to [5] Smart City is a fragmented scenario where common challenges are related to reliability, scale and timeliness, whereas in a healthcare IoT solution data security and privacy by users are the main challenges [52]. Device management and analytics capabilities would be much more important for an industrial and cultural heritage [53] IoT application. They can involve thousands of sensors to monitor different parameters inside the same environment. Next sections investigate services offered by the three platforms and finally we summarize them in Table 2.

B. PERFORMANCE METRICS AND SCENARIOS

In order to achieve messages reliability, MQTT supports three QoS levels [54]. In QoS level 0 a message is delivered at most once and no confirmation of reception is required. In QoS level 1, every message is delivered at least once and an acknowledgement of message reception is required. QoS level 2 uses a four-way handshake mechanism for the delivery of a message exactly once. The Cloud platforms included in this study ensure support only for QoS level 0 and 1, not for QoS level 2.

The aim of this study is to measure the performance of the access point to Cloud services for IoT: message broker in AWS IoT-Core, Microsoft Azure IoT Hub and the MQTT Bridge of the Google's Cloud IoT-Core. As reported in a recent survey [46], the messaging technology has the greatest usage in IoT applications, and the time is one of the most important evaluation factors. Therefore, in order to compare MQTT performance we first consider the end-to-end delay, that is the time elapsed between producing and sending a message by a publisher client and receiving that message by a subscriber client or application. Nevertheless end-to-end delay is affected by factors, such as transmission time, propagation time, procession and queuing time [55]. These conditions are not always stable during the tests run at different times. Therefore, what we consider is only the cloud-broker's service time in different conditions, which we illustrate in the construction of the scenarios in Sec.II-B2. We also analyze the cost of the involved services, extending a recent work by Kalmar *et al.* [56] where an investigation of different IoT Cloud service providers pricing models is conducted. They construct 2 scenarios and perform the costs estimation. In this work we conduct a comparative analysis with a variable number of devices (and consequently a changing number of messages) connected to the platforms.

The following subsection illustrates the method used to calculate the service time of the broker on Cloud platforms.

1) Cloud Service Time

Consider a simple case with only one publisher client, P1 and one subscriber client S1, that are MQTT clients of the same MQTT broker in a Cloud Platform. The experiment flow is shown in Fig.4: P1 publishes a message with QoS level 1 at time t_0 with the timestamp t_0. At t_1 the message arrives at the broker, that answers to P1 with an acknowledge (PUBACK); this PUBACK arrives at time t_4 to P1. In the meanwhile the broker sends the message at time t_2 to S1 and S1 receives the message at t_3. The broker service time is given by (1)

$$T_c = t_2 - t_1 \tag{1}$$

All three platforms provide to users a log system that can be analyzed to obtain this value. Our aim is to obtain T_c without the use of platforms logs. It would be possible to implement a rule in the Cloud that adds new fields to the incoming message with the timestamps t_1 and t_2, but this choice is not practicable because this operation would add significant and not easily quantifiable delays, negatively affecting the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2961511, IEEE Access

final result. Furthermore, it is not possible to compare client times with platform times unless using synchronization algorithms. In addition, in order to guarantee efficiency and scalability, the platforms provide a connection endpoint, but the actual location of the endpoint is obviously different over time, making synchronization highly expensive in terms of computation. Therefore we implement the simulated clients in the same physical machine, so as to avoid any synchronization problem between hosts. Starting from algorithms of the Network Time Protocol [57] it is possible to calculate the Round Trip Delay by (2)

$$RTD = (t_3 - t_0) - (t_2 - t_1) \tag{2}$$

The Round Trip Time on the publisher is calculated by using (3)

$$RTT_p = t_4 - t_0 \tag{3}$$

where t_0 is the time when P1 publish the QoS level 1 message and t_4 when the PUBACK sent by broker comes back to P1. Since the clients reside in the same machine we can hypothesize that the One Way Delays are symmetric, so the searched value T_c is simply given by (4)

$$T_c = t_2 - t_1 = t_3 - t_4 \tag{4}$$

In order to verify that these assumptions are not too simplified, with consequent inconsistency of subsequent results, a validation of the data is carried out by simulations to be compared with the logs obtained from a platform. We examine the AWS platform that provides the CloudWatch service [58] for log analysis. After creating a policy in the IoT-Core that allows communication between the two services, it is possible to analyze the generated log, a JSON file with the following format:

```
{ "timestamp": "2019-08-19 10:54:07.180",
"logLevel": "INFO",
"traceId": "xxxxxxxx-xxxx-xxxx-xxxx-xx",
"accountId": "0123456789",
"status": "Success",
"eventType": "Publish-In",
"protocol": "MQTT",
"topicName": "/path/to/topic",
"clientId": "cliendId",
"principalId": "principalIdentification",
"sourceIp": "xxx.xxx.xxx.xxx",
"sourcePort": 01234 }
```

The *eventType* field has the value "Publish-in" when a message arrives at the broker and the value "Publish-out" when the broker sends it to a subscriber client. The value we are looking for is the difference between the *timestamp* fields of the two JSON files. We write a python application that simulates a publisher and a subscriber client: a client publishes a message with an Identification number and writes the instant t_4 in a text file. The subscriber client does the same operation when it receives the message in the instant t_3.
We conduct 100 tests in 23 different simulations and for each simulation we compute the Mean and the Standard Deviation,

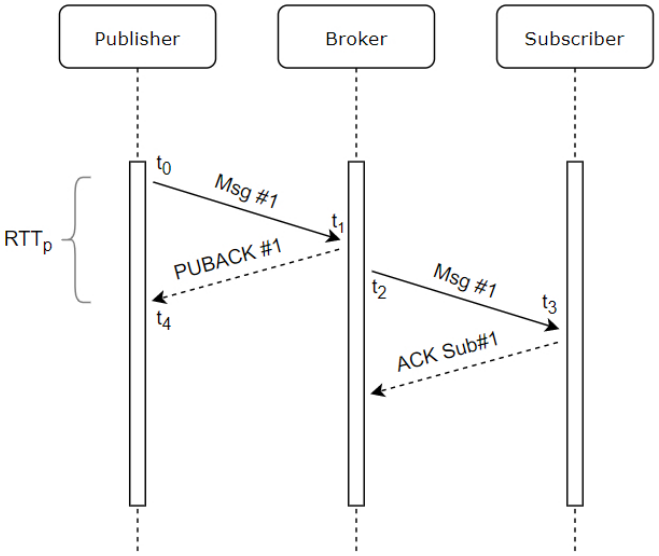

FIGURE 4: Experiment flow. Publisher sends a message to the broker and the broker forwards it to the subscriber.

both from Amazon's Log and from text files obtained from tests, that we analyze to verify the value of the assumptions.

Starting from a base case with one client publishing one message per second (mps) on a topic which has one subscriber client, we send 1000 messages and we track the obtained results. The sending frequency has increased from 1 mps to 1000 mps, and then the number of publisher and subscriber clients has changed to 5 and 10 (always maintaining 1:1 ratio), resulting in an increase of up to 10000 mps. The Mean values and Standard Deviations obtained, as shown in Table 1, are very close for each case and in all cases the probability distributions follow identical trends. Therefore, it is possible to state that the errors obtained by the simplifications adopted are negligible for the final evaluation.

2) Scenarios

In order to analyze the performance in terms of processing time of the platforms' Cloud Gateway , we consider different scenarios with different loads. Loads refer to different dimensions, such as number of publisher clients, number of subscriber clients, number of messages exchanged, size of messages, rate of messages published or consumed. We made the following assumptions for scenarios construction: fixed size of the message payload equal to 150 bytes and number of topics equal to the number of publishers (i.e. each client publishes messages on its own reserved topic). In addition, each client, both publisher and subscriber, works by creating its own connection to the Cloud gateway. In this way connections are not shared by multiple clients, in order to simulate different connected devices (or applications).
We consider 3 cases, increasing the loads for each case, in such a way as to be able to analyze both the service time of the broker for each platform, and the reliability of the limits imposed by the platforms themselves.
For all platforms we sign free accounts, that introduce some

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2961511, IEEE Access

TABLE 1: Mean and Standard Deviation of Cloud Service Time obtained from AWS Log and from simulations. Times are in ms.

Case Id	Client(pub/sub)	Mps	T-Cloud (Log)	St.Dev. (Log)	T-Cloud (Sim.)	St.Dev. (Sim)
1	1	1	26.1703	0.1974	26.1457	0.2445
2	1	10	29.042	3.5295	29.0924	3.6021
3	1	20	27.4792	1.2097	27.5122	1.2839
4	1	50	27.5687	4.2957	27.5172	4.3147
5	1	100	27.2277	6.1284	27.1758	6.0668
6	1	200	27.1459	5.7324	27.2542	5.7913
7	1	500	27.0698	5.0916	26.8597	5.138
8	1	1000	28.0938	4.4566	28.1551	4.6476
9	5	5	27.3336	2.1754	27.5341	2.0532
10	5	50	26.8552	3.4997	26.8982	3.6003
11	5	100	27.6465	3.5523	27.5893	3.3221
12	5	250	27.5393	3.1055	27.5828	3.0937
13	5	500	27.3057	3.1549	27.1894	3.2549
14	5	1000	27.001	3.1324	27.1843	3.2905
15	5	2500	26.7681	4.5647	26.6941	4.4938
16	10	10	25.7714	1.3541	25.7149	1.2945
17	10	100	25.7922	0.5585	25.8143	0.4947
18	10	200	25.847	5.0159	25.8023	5.0648
19	10	500	28.0939	1.4566	28.2154	1.5471
20	10	1000	27.9916	2.1064	27.8596	2.0754
21	10	2000	28.1483	2.7193	28.0868	2.6738
22	10	5000	28.5947	3.2648	28.4681	3.8844
23	10	10000	30.7249	4.8845	30.7684	4.9901

limitations, especially with regard to the Microsoft platform. Indeed, as reported in [59], the free-tier for Azure IoT Hub allows the creation of a single unit, resulting in a limit of 100 connections/sec accepted and a maximum of 8000 messages/day.

- *Point to Point:* in this scenario the number of publisher clients is equal to the number of subscriber clients, each of which is listening on a single topic. The load is increased in two different ways: keeping the rate of messages sent by publishers fixed to 10 mps and increasing the number of clients (between 1 and 600), or keeping the number of clients fixed (100 publisher clients and 100 subscriber), varying the frequency of sending messages (between 1 mps and 10 mps per client).
- *Fan in:* in this scenario, more clients publish messages (each on their own topic) and a subscriber client is listening, using the wildcard #, on all topics. The loads are changed by acting on the number of publisher clients connected with a message sending rate equal to 10 mps per client.
- *Fan out:* in this scenario, also called broadcasting, a publisher client publishes messages on a topic, where multiple subscribers are subscribed. In this scenario, the cases are simulated keeping the sending message rate fixed at 10 mps and increasing the number of clients subscribed.

III. RESULTS

A. ARCHITECTURE

1) AWS IoT Core

The architecture of the AWS IoT Core is represented in Fig. 5. Devices report their state by publishing messages on MQTT topics, that have a hierarchical name in order to identify the device. The message is sent to the AWS IoT MQTT Message Broker, which sends it to all clients subscribed to that topic. Each device has a shadow object (a virtual representation of the device) that is used to store and retrieve state information in a JSON document divided into a last reported state and a desired state. An application can send a request with the current state of the device or with a change in its state. When the message arrives to the broker, the Rules Engine provides message processing and integration with other AWS services.

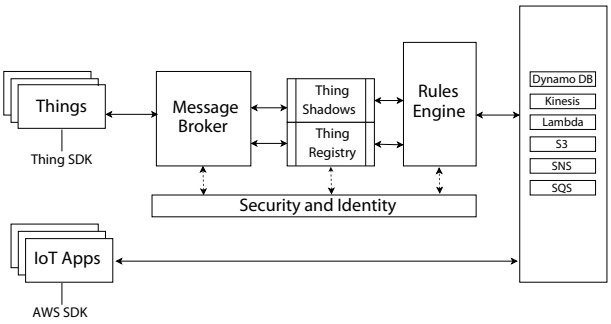

FIGURE 5: AWS IoT Core architecture and integration

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2961511, IEEE Access

- *Device management*. AWS IoT Device Management is the service of the AWS IoT platform that allows organization, monitoring, and management of IoT devices. It allows devices to register in bulk and to organize them into groups, attaching them to access policies. AWS IoT provides a registry to manage things, stored as JSON data. Interaction with registry is possible with the AWS IoT console or the AWS Command Line Interface. Furthermore, the platform provides Device SDKs for Android, iOS, Java, JavaScript, C++, Python, and Embedded C that include open-source libraries.
- *Data communication protocols*. Communication to and from AWS IoT Core is allowed by a publish/subscribe message broker service. The message broker supports MQTT to publish and subscribe, and HTTPS only to publish, both through IPv4 and IPv6. The implementation of the message broker is based on MQTT v.3.1.1, but it does not support QoS 2, and it does not allow the connection of two or more clients with the same client ID simultaneously. All topics that start with $ are reserved topics, utilized for device shadow operations (e.g. get or update state). The broker supports connections with the HTTP protocol by the use of REST API.
- *Rules and analytics*. The platform uses rules in order to interact with other AWS services. Rules are composed by a trigger written in a SQL-like syntax and one or more actions activated.
- *Data storage*. Data ingested to the AWS IoT Core is required to be stored. AWS IoT Core offers the direct connection with Amazon DynamoDB (NoSQL database) and AWS S3 (Simple Storage Service), a scalable storage in the AWS Cloud.
- *Integration*. In order to create and interact with devices AWS IoT provides a Command Line Interface (CLI), AWS IoT API to build applications using HTTP or HTTPS requests and Device SDKs. AWS offers services for the collection and the processing of data records: Amazon Kinesis Data Stream to real-time process of streaming data, AWS Lambda to execute serverless code, Amazon Simple Notification Service to send or receive notifications, and Amazon Simple Queue Service to store data in a queue.
- *Security*. Devices must have credentials to access the message broker and all traffic must be encrypted by Transport Layer Security (TLS). Platform support as identity principals for authentication X.509 certificates, typically used by AWS IoT devices, Identity Access Management (IAM) users, groups and role, Federated identities used by web and desktop applications, and Amazon Cognito identities, in general used by mobile applications, that allow the use of other identity providers.
- *Costs*. AWS bills separately for usage of Connectivity, Messaging, Device Shadow usage (device state storage), Registry usage (device metadata storage), and Rules Engine usage (message transformation and routing), all based on the region selected.

2) Microsoft Azure for IoT

Microsoft Azure for IoT provides two paths: a PaaS solution named Azure IoT solutions accelerator, and a SaaS solution named Azure IoT Central. Both solutions utilize Azure IoT Hub as Cloud gateway to securely accept data and provide device management capabilities. The Hub is natively integrated with other Azure cloud services and it allows secure bi-directional communication between devices and applications. The Azure for IoT architecture is depicted in Fig. 6, according to the 3 layer Cloud-IoT architecture. The message sent by an authenticated device arrives to the Hub, that has built-in message routing functionalities in order to send the message to one or more endpoint of other services. Devices have a virtual representation in the Cloud called twin device, stored as a JSON document which contains desired properties (set by an application and read by the device) and reported properties (set by the device and read by an application).

- *Device management*. Microsoft Azure IoT Hub Device Provisioning Service is a service that enables just-in-time provisioning of devices to an IoT hub, without requiring human intervention. Devices contact the provisioning service endpoint passing their identifying information. The service registers the device with an IoT Hub and populates the desired twin device state. Furthermore, Azure provides device SDKs that can be used on devices or gateways to simplify the connectivity to Azure IoT Hub. SDKs are available for .NET, C, Java, Node.js, Python and iOS.
- *Data communication protocols*. In addition to the IoT Hub, the platform offers the Event Hub Service to permit communication with the Cloud. Both services are designed for data ingestion on a massive scale, but the IoT Hub includes more specific features for IoT context, such as bi-directional communication to and from the Cloud and device-level identity. Azure IoT Hub provides support for the AMQP 1.0 with optional WebSocket support,MQTT 3.1.1, and native HTTP 1.1 over TLS protocols. QoS 2 delivery assurance of MQTT is supported, but not recommended due to the high impact on latency and availability of the entire system. The platform also offers the IoT Protocol Gateway to enable other protocol adaptation for the IoT Hub.

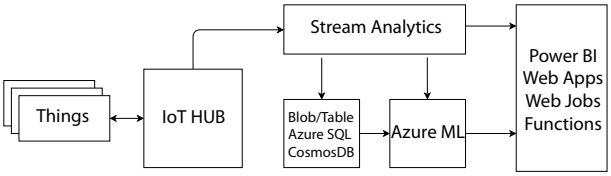

FIGURE 6: Azure solution architecture and integration

- *Rules*. Azure IoT Hub exposes its functionality by the use of endpoints. In order to route messages from device

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2961511, IEEE Access

to these endpoints, Azure uses rules written in an SQL-like syntax evaluated on the message headers and body.

- *Data storage*. Regarding data storage, according to the previous section, Azure offers services for hot and cold storage. In the hot path, data has to be available with lower latency than the data in a cold storage. Azure services for hot storage are Azure Cosmos DB as NoSQL database, and Azure SQL DB as relational SQL DBMS. Services for cold storage are Azure Blob Storage (a file storage database) and Azure Data Lake, a distributed data store. Azure offers also Time Series Insight that provides analytics, storage and aggregation services.
- *Integration*. IoT Hub is natively connected with other Azure services: Azure App Service, a managed platform to build web and mobile apps, Notifications Hub, in order to send push notifications, and PowerBI, to create dashboards.
- *Security*. There are three primary areas to be considered regarding security: device, connection and cloud security. The Azure Hub Identity Registry provides secure storage of device identities and each security key; all connections have to be initiated by the device to the Hub, not vice versa, and use TLS authentication with X.509 certificate; Azure Active Directory is used for user authentication and authorization for cloud access.
- *Costs*. Azure's costs management is based on two levels of service. In each level we find three tiers. Every tier has a daily message limit after which you will experience throttling. The consumption of IoT Hub units is measured on a daily basis, and the billing is generated on a monthly basis. Customers are billed based on the number of IoT Hub units that have been consumed during the month.

3) Google Cloud IoT Core

Google solution for IoT is Google IoT Core. The main components are the device manager and the protocol bridge. The device manager has the role of registering devices with the service, whereas the two protocols bridge (HTTP/MQTT) are used by devices in order to connect and send data to the Cloud. According to the Google Cloud Platform architecture depicted in Fig. 7, devices send data to the Google IoT Core that is directly connected with Google Cloud Pub/Sub; it is an enterprise message-oriented middleware to the Cloud and it provides a message ingestion service. Messages are then delivered to a pipeline service, the Google Cloud Data Flow, that processes data and sends it to other cloud services, depending on the IoT project use case. Devices are represented by an ID and identified by a full resource name. It is possible to define custom metadata for a device, a state which is sent to the Cloud and a configuration, which is sent from the Cloud to the device. As a difference from the previous platforms, in Google solution information can be an arbitrary user-defined blob of data.

- *Device management*. The IoT Core Device Manager provides the service for managing devices. It includes registration, authentication and authorization processes. With the device manager it is possible to create and configure registries and devices within them. The device registry is configured with one or more Cloud Pub/Sub topics to which telemetry events are published for all devices in that registry. A device is defined with metadata, it sends telemetry messages and receives configurations, user-defined blob of data sent from the Cloud.
- *Data communication protocols*. The platform supports MQTT and HTTP for managing devices and communications. By the use of MQTT, devices send publish requests to a specific topic, whereas using HTTP, devices do not maintain a connection to the platform, and they send requests and receive responses. Regarding MQTT's Quality of Service, the MQTT Bridge supports QoS 0 and 1.
- *Rules*. In order to manage data arriving at the Cloud, the platform uses the concept of pipelines offered in the Google Cloud Data Flow: it allows to transform, aggregate, enrich and move data to other services. It is also possible to operate on each published event individually by the Google Cloud Functions, that can be used to filter invalid data, trigger alarms, or invoke other APIs.
- *Data storage*. Devices send different types of data, from their state (normally in a structured way), to telemetry data, and unstructured blobs of data (e.g. video streams). In the case of structured data that identify the status of a device, the storage takes place directly in the service provided by the IoT Core. Telemetry data arrives with high frequency and it has to be available in a low latency and high performance way: the platform offers Cloud Datastore and Cloud BigQuery as NoSQL databases, and Cloud BigTable as a fully managed data warehouse with SQL interface. The Cloud Storage is used to archive data used infrequently and for unstructured data.
- *Integration*. The platform provides the Google Cloud SDK which contains a command line tool called *gcloud*. Operations are also possible by the Console and by the use of APIs client library for C#, Java, NodeJS, GO, PHP, Python and Ruby. The Iot Core is natively integrated with Google's big data and machine learning analysis services such as Cloud ML, Data Studio and DataLab.
- *Security*. The IoT Core offers per-device public/private key authentication using JSON Web Tokens [60] and supports for RSA or Elliptic Curve algorithms to verify signatures. Concerning communications security, the TLS 1.2 protocol, using root certificate authorities, is required for MQTT connections. Google Cloud Identity and Access Management (IAM) allows to control, authenticate and authorize the Cloud IoT Core API access.
- *Costs*. The IoT Core prices are calculated on the volume of data used in a month, over 250 MB, in three different levels and considering a minimum size of 1024 bytes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2961511, IEEE Access

for each message (for messages less than 1024 bytes, a cost of 1024 bytes will be applied). All other services utilized in a solution will be billed separately.

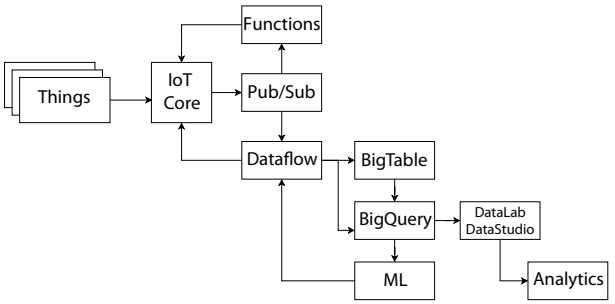

FIGURE 7: Google Cloud IoT Core architecture and integration

B. PERFORMANCE ANALYSIS

We have simulated all clients as static on a machine with the following features: Intel Xeon X5650 (x2) CPU, 12 MB cache, 2.66 GHz, 16 GB RAM with Ubuntu 18.04.1 LTS. The clients have been implemented in GoLang, a language developed by Google, whose approach to concurrency differs from the classic use of threads and shared memory. Concurrency is an intrinsic part of the Go programming language and is managed using *Goroutine* and *Channels*. *Goroutine* are functions or methods that are performed concurrently with other functions or methods in the same address space, so access to shared memory must be synchronized. They are not real threads, but light-weight threads managed entirely by the Go runtime. *Channels* are the pipes that connect concurrent *Goroutine*. A tool has been developed that takes the following parameters from the command line: the MQTT broker endpoint (scheme://endpoint-url:port), the scenario to be simulated, the number of clients, the number of messages, the interval in ms between messages, the size of each message, the publisher and subscriber QoS, eventual username and password for the connection. In the case of connection to the Google Cloud Platform's IoT Core, each client must provide a JSON Web Token (JWT) for authentication. We use the Paho Go Client Library [61], which allows connection to an MQTT broker via TCP, TLS or WebSocket.

The tool creates as many *Goroutines* as the *clients* parameter depending on the scenario considered (e.g. in scenario A #clients publisher and #clients subscriber, in the case B #clients publisher and only one subscriber). Routines wait for each client to be safely connected and authenticate to the broker, and consequently start to publish messages in their own topic. At the same time subscribers are listening to that topic. Publisher clients keep track of the PUBACK timestamp received by the broker (thanks to QoS level 1) in an array which index is the message identifier contained in the payload. In the same way, subscribers perform the same operation, keeping track of the reception timestamp. At the end of the *Goroutine*, the values are sent to the main routine, which saves the values of each publisher and subscriber client in a bi-dimensional array. The obtained matrices are passed to a function that writes in a MySql database every single performed measurement and the parameters of the simulation. Cloud services performance may vary over time. Therefore, we conduct 42 different measurements for each simulation (i.e. 2/day in different times of day for 3 weeks). Subsequently a function computes the Mean Value of the Cloud Service Time for each simulation and its standard deviation and write results to the database. Finally a MatLab function connects to the database and plots the obtained results.

1) Point-to-point

As reported in Sec. II-B2, Azure imposes limitations on the number of accepted connections per second in the free-tier and maximum number of messages per day accepted, as well as a maximum number of device to cloud messages equal to 100 per second. Therefore the choice was to separate the results obtained from this platform from the others. Simulations will be repeated in the future using a pay tier.

In this scenario, the Amazon and Google platforms have a similar and uniform behavior in terms of broker service time in relation to the number of messages published per second. In Fig. 8a the number of messages per second on the abscissa is obtained by increasing the sending messages rate with only one client connected up to the value of 100 mps, and subsequently keeping the sending frequency of each client fixed at 10 mps and increasing the number of connected clients between 1 and 600. Google IoT-Core responds faster than the other platforms for almost all simulated mps values, except for values between 150 and 750 mps, range during which Amazon provides better performance. Azure, even for load conditions that can be compared to the others, has an average service time much higher than its competitors, equal to $180ms \pm 20ms$. Platforms do not seem to be particularly affected by increases in load. The limits declared by the platforms are respected, in particular the number of accepted connections per second per account and the number of messages per second accepted by the broker for connection.

In Fig. 8b the load to the broker has been increased only acting on the sending rate and fixing the number of clients to 100. In this case the platform brokers behavior is even more uniform compared to the previous case, showing no significant changes in terms of service time. In this case, however, it is Amazon's platform to achieve slightly better average performance, remaining at $27ms \pm 0.3ms$.

In order to make a description of the distributions of performed simulations, box plots are shown in Fig.9 - 10 - 11. It is interesting to note how the results of Azure (Fig. 11) follow more symmetrical distributions than the other platforms, and how the values are more concentrated around the median. The distributions of the simulations obtained on AWS and GCP are positive asymmetries, therefore with a greater dispersion for higher values. Finally, note the total absence of outliers for the distributions of AWS and GCP, minimally present in Azure distributions.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2961511, IEEE Access

TABLE 2: Cloud Platforms Services

	AWS	Azure	GCP
Device management	IoT Device Management Console Command Line Interface	Device Provisioning Service Console Command Line Interface	IoT Core Device Manager Console Command Line Interface
Data communication protocols	MQTT 3.1.1 HTTP(S)	MQTT 3.1.1 HTTP(S) AMQP 1.0 (WebSocket support)	MQTT 3.1.1 HTTP(S)
Rules	Rules Engine (SQL-like rules)	Endpoints exposed for SQL-like rules	Google Cloud Data Flow Google Cloud Functions
Cold Storage	AWS S3	Azure Blob Storage Azure Data Lake	Cloud Storage
Hot Storage	Amazon DynamoDB	Azure CosmosDB Azure SQL DB	Cloud Datastore Cloud BigQuery Cloud BigTable
Integration	APIs, SDK Amazon Kinesis Data Stream AWS Lambda AWS SNS AWS SQS	APIs, SDK Azure App Service Notifications Hub PowerBI	APIs, SDK Cloud ML Cloud Data Studio Cloud DataLab
Languages	Java JS C++ Python Embedded C	.NET C Java NodeJS Python	C# Java NodeJS GO PHP Python Ruby
Security	TLS X.509 certificate AWS IAM Federated Identities Amazon Cognito	Azure Hub Identity Registry TLS X.509 certificates Azure Active Directory	JWT TLS X.509 certificates Google Cloud IAM

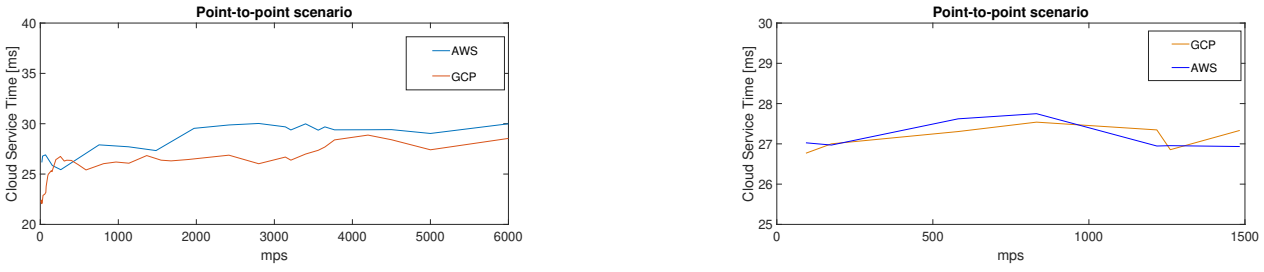

FIGURE 8: Point-to-point scenario - Cloud Service Times in relation to mps sent by clients. In (a) each client sends 10 mps and we change the number of clients from 1 to 600; in (b) a fixed number of clients (100) sends messages from 1 mps to 15 mps

2) Fan-in

In this type of scenario, where each client publishes on its own topic while a single subscriber is subscribed by wildcard to all the topics, it is interesting to evaluate the message loss, i.e. the number of messages that have not been delivered to subscriber. Note that the Google and Microsoft platforms do not allow direct subscription via wildcard, but they allow forwarding of all messages to an additional service: Google allows registration of all virtual devices in a registry, which has a topic associated in the PubSub service [62]. Each device sends its telemetry messages to its MQTT topic and the platform forwards them to PubSub. A client application then simulates the connection to this service and to the topic of the registry, thus allowing to perfectly simulate the considered

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2961511, IEEE Access

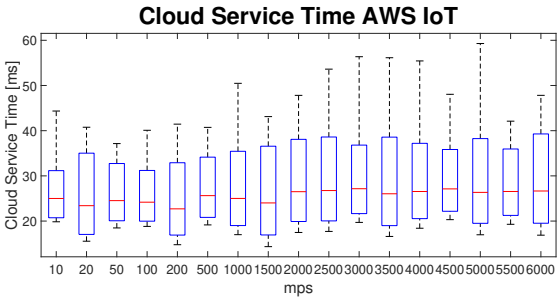

FIGURE 9: Boxplot of AWS results

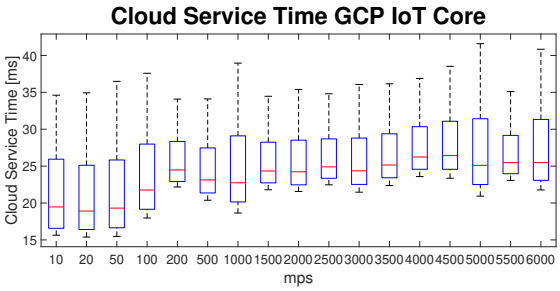

FIGURE 10: Boxplot of GCP results

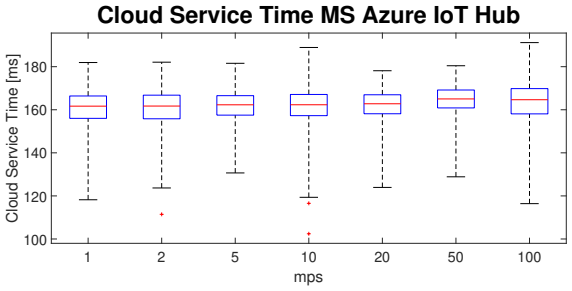

FIGURE 11: Boxplot of Azure results

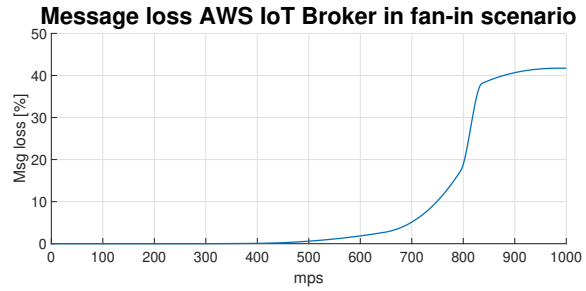

FIGURE 12: Message loss vs mps obtained with an increasing number of clients and fixed 10 mps/client in fan-in scenario

scenario. Azure allows native integration of the IoT Hub with one or more default endpoints, and also in this case a client is connected to that endpoint to receive forwarded messages. Therefore results depicted in Fig. 12 concern the message loss only for Amazon by connecting directly to the MQTT broker and subscribing to all topics. The broker is able to forward all messages to the subscriber up to the limit of 400 mps (40 connected clients each sending 10 mps); significant losses begin to occur with 70 connected clients (5%) with an exponential trend up to about 810 mps, and then stabilize on 42% for a load of 1000 mps. These values can be useful for developers and solution architects to make a possible partition of the topic and connected subscriber clients, in order to avoid significant losses of published messages. In order to make the same analysis on this scenario, we use a further service on AWS that allows forwarding all messages from a topic to AWS SNS [63], by creating a rule in the Rules Engine. Also with regard to Azure the reasoning was the same: creation of a rule that allows the forwarding of messages addressed to all topics of the devices towards the message bus [64]. The subscriber client, for all platforms, listens to the channel on which these messages are forwarded and counts the actual delivery. In this case, for all platforms, all messages were delivered in each load condition considered. Service Cloud Times were then calculated using the procedure illustrated in Sec. II-B1, obtaining the service time of the broker added to the service time of the following message bus. In this case too, these results can be useful to the specialists of Cloud-IoT solutions for the most correct choice based on the application needs. Simulations in Fig. 13 shows that the service time trend is similar to that found in the point-to-point scenario, up to 3000 mps. After this value, Google and Amazon platforms have an important growth in recorded time. However, analyzing the logs made available by the platforms, the time to forward messages from the broker to the following service does not have such significant variations; these delays are attributable to the subscriber client who fails to send the PUBACK messages to the broker, thus reaching the so-called value of "Maximum inbound/outbound unacknowledged QoS 1 publish requests" (due to the limitations imposed by the platforms themselves), thus delaying the forwarding. As regards Azure, the results are the same for the point-to-point scenario shown in Fig. 11, showing that they are in no way influenced by the different type of scenario.

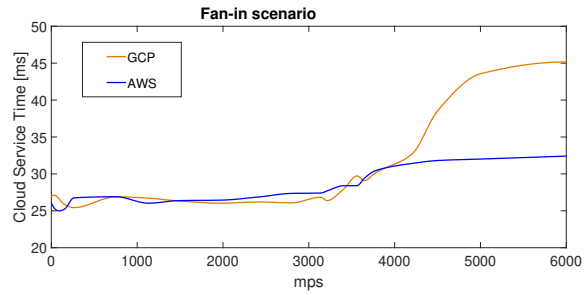

FIGURE 13: Fan-in scenario - Cloud service time vs mps sent by clients with 10 mps/client and from 1 to 600 connected clients

3) Fan-out

The workload in this scenario is generated by a single producer publishing 10 messages per second in a single topic. An increasing number of clients is subscribed to that topic.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2961511, IEEE Access

We analyze the Cloud Service Time as depicted in Sec. II-B1 in function of the fan-out factor (i.e. number of subscriber clients). In this scenario we have the result shown in Fig.14 where we can see how GCP has lower Cloud Service Time than both AWS and Azure for a fan-out factor over 15. Before this value AWS has the lowest values. Azure's Hub IoT forwards messages 15X slower than the other platforms, yet having (as in previous scenarios) a lower gap between outliers. Indeed, for a fan-out factor equal to one the Cloud Service Time is 26.479 ms, 24.991 ms and 160.567 ms, and for 300 subscribers the difference is 26.7%, 68.1% and 7.1% respectively for GCP, AWS and Azure.

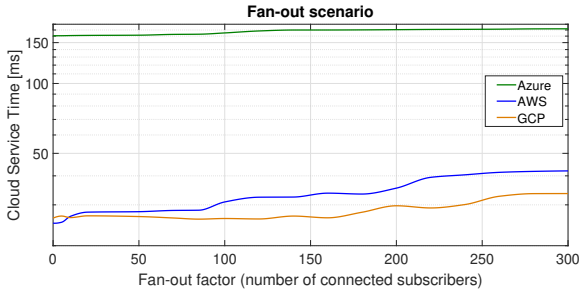

FIGURE 14: Fan-out scenario - Cloud service times vs fan-out factor (from 1 to 300 subscribers client)

C. COSTS

In this Section, according to each platform's documentation, we analyze costs in function of different work loads. Each platform has a different approach in terms of billing:

1) In Amazon Web Services IoT Core connectivity, messaging and shadow devices (device status storage), registry (device metadata storage), and rules engine (message transformation and routing) are billed separately. Different costs are applied according to region/zone. Messages are measured in increments of 5 kB. For EU London zone:
   - Connectivity is metered in 1 minute increments and is based on the total time devices are connected to AWS IoT Core: $0.096 (per million minutes of connection);
   - Messaging is metered by the number of messages transmitted between devices and AWS IoT Core: $1.20 when the monthly message volume is up to 1 billion messages, $0.96 for the next 4 billion messages, $0.84 over 5 billion messages.
2) Azure IoT Hub offers two tiers, basic and standard, that differ in the number of features they support. The standard tier of IoT Hub enables all features for any IoT solutions that want to make use of the bi-directional communication capabilities, whereas the basic tier enables a subset of the features and is intended for IoT solutions that only need uni-directional communication from devices to the Cloud. Prizes for each level in the two tiers are summarized in Table 3.
3) The Google IoT Core prices are calculated on the volume of data used in a month and there is no charge for creating, reading, updating and deleting devices. Messages below 1024 bytes are counted as 1024 bytes, whereas for larger messages the actual size is billed. Prizes are reported in Table 4.

TABLE 3: Azure IoT Hub prizes (in $)

EDITION TYPE	MONTHLY COST/UNIT	MSG/DAY/UNIT
Free	Free	8.000
S1	25	400.000
S2	250	6.000.000
S3	2.500	300.000.000
B1	10	400.000
B2	50	6.000.000
B3	500	300.000.000

TABLE 4: Google IoT Core prizes (in $)

PRIZE	MONTHLY DATA VOLUME
0,00	<250 MB
0,0045 per MB	From 250 MB to 250 GB
0,0020 per MB	From 250 GB to 5 TB
0,00045 per MB	>5 TB

In order to conduct an analysis of changes in cost, we consider a scenario where devices send one message per minute of 1kB. Devices are continuously connected to the platforms. Fig. 15 shows the number of devices connected on the abscissa and on the ordinate the monthly cost in dollars (the volume of monthly traffic exchanged with each platform is easily obtained by multiplying the number of devices per 1440 messages/day per 30 days per kB). We use a logarithmic scale for a better visualization.

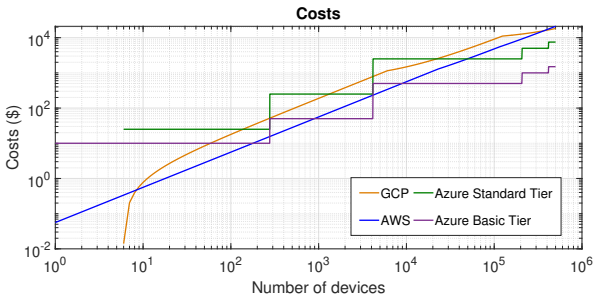

FIGURE 15: Costs in function of increasing number of devices

We analyze costs evolution varying the number of connected devices between 1 and 500000 (from 1440 to 720 millions of messages per day, from 1.40 MB to 687 GB per day in terms of data size). Table 5 shows the costs of the platforms in reference to some ranges of devices obtained from the analysis of the graph. We highlight for each range the lowest cost.

IV. CONCLUSIONS

The growing attention of the main Cloud platforms to the IoT world is nowadays evident. With this study we wanted

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2961511, IEEE Access

TABLE 5: Platform prizes (in $) according to number of connected devices

Devices	Azure (Basic Edition)	Azure (Standard Edition)	AWS	GCP
1-6	10	**Free**	<15	**Free**
7-70	10	25	**<3**	<10
70-250	10	25	**3-15**	10-45
250-1000	50	250	**15-56**	45-185
1000-4100	50	250	**56-230**	185-810
4100-10000	500	2500	**230-560**	810-1440
10000-50000	500	2500	**560-2500**	1440-4640
50000-100000	500	**2500**	2500-4800	4640-8640
100000-420000	500/1000	**2500/5000**	4800-17700	8640-16300
420000-500000	1500	**7500**	17700-21058	16300-17815

to highlight the characteristics of each platform, through a mapping of services with a reference architecture, so as to have a broader view of all these services that are made available. Each platform provides a Cloud access point through which physical devices can connect in a secure and private way. After an authorized connection, devices can start to send their data to the Cloud: the most used protocol is MQTT. The tests we conducted did not try to reach the maximum message throughput. In fact, regarding the performance we measured the service time of the message broker, in different load conditions and in three different scenarios. At the same time, by increasing the load in each scenario and keeping in mind the use of free tiers for each platform, we were also able to verify the limitations imposed by the providers. Results show how the platforms, although with different service times, answer in a uniform way, thus guaranteeing predictable performance levels that comply with the specifications indicated in the documentation. The final analysis made on the costs of the platforms according to different types of load, allows the reader to have a further element of comparison to make the choice. In fact, the work does not want one platform to be preferred over another, but tries to provide the developer with elements to make an informed choice. In the future the same study will be carried out on the different paid levels that the platforms make available, in order to have no limitations on the number of devices or number of messages. In this study we used a fixed message size and we tested the MQTT protocol. We will study the behavior of platforms with different packets size and under different communication protocols (e.g. HTTP or AMQP). The focal point of each platform is the full integration with other services available (storage, AI, Machine Learning, etc); the performance analysis carried out had the purpose of measuring the time of the message broker. It would be interesting to carry out the same analysis on all the services involved in a complete IoT application, creating a common scenario to be tested on the three platforms. In fact, in applications where response time is a critical element, analyzing the time of each service can be interesting to understand where and how recover even milliseconds that can be fundamental. At the same time exploring the solutions that the platforms make available for Fog Computing (e.g. AWS Greengrass) could be an interesting extension of this work.

REFERENCES

[1] Lorenzo Incipini, Alberto Belli, Lorenzo Palma, Roberto Concetti, and Paola Pierleoni. Databases performance evaluation for iot systems: the scrovegni chapel use case. In 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pages 463–468. IEEE, 2019.

[2] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. Wireless communications and mobile computing, 13(18):1587–1611, 2013.

[3] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems, 25(6):599–616, 2009.

[4] Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. On the integration of cloud computing and internet of things. In Future internet of things and cloud (FiCloud), 2014 international conference on, pages 23–30. IEEE, 2014.

[5] Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. Integration of cloud computing and internet of things: a survey. Future Generation Computer Systems, 56:684–700, 2016.

[6] Shaik Masthan Babu, A Jaya Lakshmi, and B Thirumala Rao. A study on cloud based internet of things: Cloudiot. In Communication Technologies (GCCT), 2015 Global Conference on, pages 60–65. IEEE, 2015.

[7] Sherif Abdelwahab, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Enabling smart cloud services through remote sensing: An internet of everything enabler. IEEE Internet of Things Journal, 1(3):276–288, 2014.

[8] Jiehan Zhou, Teemu Leppanen, Erkki Harjula, Mika Ylianttila, Timo Ojala, Chen Yu, Hai Jin, and Laurence Tianruo Yang. Cloudthings: A common architecture for integrating the internet of things with cloud computing. In Computer Supported Cooperative Work in Design (CSCWD), 2013 IEEE 17th International Conference on, pages 651–657. IEEE, 2013.

[9] Stefan Nastic, Sanjin Sehic, Duc-Hung Le, Hong-Linh Truong, and Schahram Dustdar. Provisioning software-defined iot cloud systems. In Future Internet of Things and Cloud (FiCloud), 2014 International Conference on, pages 288–295. IEEE, 2014.

[10] Yu Liu, Kahin Akram Hassan, Magnus Karlsson, Zhibo Pang, and Shaofang Gong. A data-centric internet of things framework based on azure cloud. IEEE Access, 7:53839–53858, 2019.

[11] Yu Liu, Kahin Akram Hassan, Magnus Karlsson, Ola Weister, and Shaofang Gong. Active plant wall for green indoor climate based on cloud and internet of things. IEEE Access, 6:33631–33644, 2018.

[12] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. Future generation computer systems, 29(7):1645–1660, 2013.

[13] Partha Pratim Ray. A survey of iot cloud platforms. Future Computing and Informatics Journal, 1(1-2):35–46, 2016.

[14] Oleksiy Mazhelis and Pasi Tyrvainen. A framework for evaluating internet-of-things platforms: Application provider viewpoint. In Internet of Things (WF-IoT), 2014 IEEE World Forum on, pages 147–152. IEEE, 2014.

[15] Tamás Pflanzner and Attila Kertész. A taxonomy and survey of iot cloud applications. EAI ENDORSED TRANSACTIONS ON INTERNET OF THINGS, 3(12):Terjedelem–14, 2018.

[16] Rabindra K Barik, Harishchandra Dubey, Chinmaya Misra, Debanjan Borthakur, Nicholas Constant, Sapana Ashok Sasane, Rakesh K Lenka, Bhabani Shankar Prasad Mishra, Himansu Das, and Kunal Mankodiya. Fog assisted cloud computing in era of big data and internet-of-things: systems, architectures, and applications. In Cloud computing for optimization: foundations, applications, and challenges, pages 367–394. Springer, 2018.

[17] Amir Vahid Dastjerdi and Rajkumar Buyya. Fog computing: Helping the internet of things realize its potential. Computer, 49(8):112–116, 2016.

[18] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In Proceedings of the first edition of the MCC workshop on Mobile cloud computing, pages 13–16. ACM, 2012.

[19] Mohammed Al-Khafajiy, Thar Baker, Atif Waraich, Dhiya Al-Jumeily, and Abir Hussain. Iot-fog optimal workload via fog offloading. In 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), pages 359–364. IEEE, 2018.

[20] Mohammed Al-Khafajiy, Lee Webster, Thar Baker, and Atif Waraich. Towards fog driven iot healthcare: challenges and framework of fog computing in healthcare. In Proceedings of the 2nd International Conference on Future Networks and Distributed Systems, page 9. ACM, 2018.

[21] Sanjay P Ahuja. On the use of system-level benchmarks for comparing public cloud environments. In Handbook of Research on Cloud Computing and Big Data Applications in IoT, pages 24–38. IGI Global, 2019.

[22] Lee Gillam, Bin Li, John O'Loughlin, and Anuz Pratap Singh Tomar. Fair benchmarking for cloud computing systems. Journal of Cloud Computing: Advances, Systems and Applications, 2(1):6, 2013.

[23] Anshu Shukla, Shilpa Chaturvedi, and Yogesh Simmhan. Riotbench: An iot benchmark for distributed stream processing systems. Concurrency and Computation: Practice and Experience, 29(21):e4257, 2017.

[24] Cloudharmony, transparency for the cloud. http://www.cloudharmony.com/. [Online] Accessed 29-November-2019].

[25] Iot developer survey 2018. https://www.eclipse.org/org/press-release/iotdevsurvey2018.php. [Online] Accessed 28-June-2018].

[26] Daniel Happ, Niels Karowski, Thomas Menzel, Vlado Handziski, and Adam Wolisz. Meeting iot platform requirements with open pub/sub solutions. Annals of Telecommunications, 72(1):41–52, Feb 2017.

[27] I Hedji, I Špeh, and A Šarabok. Iot network protocols comparison for the purpose of iot constrained networks. In Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2017 40th International Convention on, pages 501–505. IEEE, 2017.

[28] Ronak Sutaria and Raghunath Govindachari. Making sense of interoperability: Protocols and standardization initiatives in iot. In 2nd International Workshop on Computing and Networking for Internet of Things, 2013.

[29] Ángel Asensio, Álvaro Marco, Rubén Blasco, and Roberto Casas. Protocol and architecture to bring things into internet of things. International Journal of Distributed Sensor Networks, 10(4):158252, 2014.

[30] Lu Hou, Shaohang Zhao, Xiong Xiong, Kan Zheng, Periklis Chatzimisios, M Shamim Hossain, and Wei Xiang. Internet of things cloud: architecture and implementation. IEEE Communications Magazine, 54(12):32–39, 2016.

[31] Tetsuya Yokotani and Yuya Sasaki. Comparison with http and mqtt on required network resources for iot. In Control, Electronics, Renewable Energy and Communications (ICCEREC), 2016 International Conference on, pages 1–6. IEEE, 2016.

[32] Anderson Augusto Simiscuka, Tejas Moreshwar Markande, and Gabriel-Miro Muntean. Real-virtual world device synchronization in a cloud-enabled social virtual reality iot network. IEEE Access, 7:106588–106599, 2019.

[33] Jorge E Luzuriaga, Miguel Perez, Pablo Boronat, Juan Carlos Cano, Carlos Calafate, and Pietro Manzoni. A comparative evaluation of amqp and mqtt protocols over unstable and mobile networks. In Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE, pages 931–936. IEEE, 2015.

[34] Dinesh Thangavel, Xiaoping Ma, Alvin Valera, Hwee-Xian Tan, and Colin Keng-Yan Tan. Performance evaluation of mqtt and coap via a common middleware. In Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on, pages 1–6. IEEE, 2014.

[35] Yuang Chen and Thomas Kunz. Performance evaluation of iot protocols under a constrained wireless access network. In Selected Topics in Mobile & Wireless Networking (MoWNeT), 2016 International Conference on, pages 1–7. IEEE, 2016.

[36] Denise Lund, Carrie MacGillivray, Vernon Turner, and Mario Morales. Worldwide and regional internet of things (iot) 2014–2020 forecast: A virtuous circle of proven value and demand. International Data Corporation (IDC), Tech. Rep, 1, 2014.

[37] Miao Wu, Ting-Jie Lu, Fei-Yang Ling, Jing Sun, and Hui-Ying Du. Research on the architecture of internet of things. In Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on, volume 5, pages V5–484. IEEE, 2010.

[38] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer, and Shahid Khan. Future internet: the internet of things architecture, possible applications and key challenges. In Frontiers of Information Technology (FIT), 2012 10th International Conference on, pages 257–260. IEEE, 2012.

[39] Pallavi Sethi and Smruti R Sarangi. Internet of things: architectures, protocols, and applications. Journal of Electrical and Computer Engineering, 2017, 2017.

[40] Cloud standards customer council. http://www.cloud-council.org. [Online] Accessed 18-June-2018].

[41] Cloud customer architecture for iot. http://www.cloud-council.org/deliverables/CSCC-Cloud-Customer-Architecture-for-IoT.pdf. [Online] Accessed 18-June-2018].

[42] Attila Klenik and Andras Pataricza. Performance analysis of critical services. In Future IoT Technologies (Future IoT), 2018 IEEE International Conference on, pages 1–6. IEEE, 2018.

[43] Industrial internet consortium reference architecture. https://www.iiconsortium.org/IIC_PUB_G1_V1.80_2017-01-31.pdf. [Online] Accessed 18-June-2018].

[44] Paola Pierleoni, Alberto Belli, Lorenzo Palma, Lorenzo Incipini, Sara Raggiunto, Marco Mercuri, Roberto Concetti, and Luisiana Sabbatini. A cross-protocol proxy for sensor networks based on coap. In 2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT), pages 251–255. IEEE, 2019.

[45] Jelena Misic, M Zulfiker Ali, and Vojislav B Misic. Protocol architectures for iot domains. IEEE Network, 32(4):81–87, 2018.

[46] Alireza Souri, Aseel Hussien, Mahdi Hoseyninezhad, and Monire Norouzi. A systematic review of iot communication strategies for an efficient smart environment. Transactions on Emerging Telecommunications Technologies, page e3736, 2019.

[47] Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1. Standard, International Organization for Standardization, Geneva, CH, June 2016.

[48] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (coap). RFC 7252, RFC Editor, June 2014. http://www.rfc-editor.org/rfc/rfc7252.txt.

[49] P. Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. RFC 6120, RFC Editor, March 2011. http://www.rfc-editor.org/rfc/rfc6120.txt.

[50] Justin J Levandoski, Per-Åke Larson, and Radu Stoica. Identifying hot and cold data in main-memory databases. In Data Engineering (ICDE), 2013 IEEE 29th International Conference on, pages 26–37. IEEE, 2013.

[51] Paola Pierleoni, Luca Pernini, Lorenzo Palma, Alberto Belli, Simone Valenti, Lorenzo Maurizi, Loris Sabbatini, and Alessandro Marroni. An innovative webrtc solution for e-health services. In 2016 IEEE 18th international conference on e-health networking, applications and services (Healthcom), pages 1–6. IEEE, 2016.

[52] Alex Mu-Hsing Kuo. Opportunities and challenges of cloud computing to improve health care services. Journal of medical Internet research, 13(3), 2011.

[53] Paola Pierleoni, Alberto Belli, Lorenzo Palma, Simone Valenti, Sara Raggiunto, Lorenzo Incipini, and Piergiovanni Ceregioli. The scrovegni chapel moves into the future: An innovative internet of things solution brings new light to giotto's masterpiece. IEEE Sensors Journal, 18(18):7681–7696, 2018.

[54] Stefan Behnel, Ludger Fiege, and Gero Muhl. On quality-of-service and publish-subscribe. In Distributed Computing Systems Workshops, 2006. ICDCS Workshops 2006. 26th IEEE International Conference on, pages 20–20. IEEE, 2006.

[55] CJ Bovy, HT Mertodimedjo, G Hooghiemstra, H Uijterwaal, and Piet Van Mieghem. Analysis of end-to-end delay measurements in internet. In Proc. of the Passive and Active Measurement Workshop-PAM, volume 2002. sn, 2002.

[56] Edua Eszter Kalmar and Attila Kertesz. What does i (o) t cost? In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, pages 19–24. ACM, 2017.
[57] D. Mills, J. Martin, J. Burbank, and W. Kasch. Network time protocol version 4: Protocol and algorithms specification. RFC 5905, RFC Editor, June 2010. http://www.rfc-editor.org/rfc/rfc5905.txt.
[58] Amazon cloudwatch. http://aws.amazon.com/cloudwatch. [Online] Accessed 20-June-2018].
[59] Choose the right iot hub tier for your solution. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-scaling. [Online] Accessed 20-June-2018].
[60] M. Jones, J. Bradley, and N. Sakimura. Json web token (jwt). RFC 7519, RFC Editor, May 2015. http://www.rfc-editor.org/rfc/rfc7519.txt.
[61] Eclipse paho go client. https://www.eclipse.org/paho/clients/golang/. [Online] Accessed 10-May-2018].
[62] Google cloud pub/sub. https://cloud.google.com/pubsub/. [Online] Accessed 15-May-2018].
[63] Amazon simple notification service. https://aws.amazon.com/sns. [Online] Accessed 15-May-2018].
[64] Microsoft azure message bus. https://azure.microsoft.com/services/service-bus/. [Online] Accessed 15-May-2018].

PAOLA PIERLEONI received the Master's degree in Electronic Engineering in 1991 and the Ph.D. degree in Electrical Engineering in 1995 from the Polytechnic University of Marche. Since 1991 she joined the Department of Information Engineering of the same university where she is currently Assistant Professor in Telecommunications. Her main research topics include network protocols, wireless sensor networks, Internet of Things, signal processing and embedded devices development.

ROBERTO CONCETTI received the Master's degree in Computer Engineering from the Polytechnic University of Marche, Italy, in 2016. He is currently a Ph.D. student at the Department of Information Engineering of the Polytechnic University of Marche. His current research interests include computer networks and cloud computing for Internet of Things.

ALBERTO BELLI is Staff Scientist at the Department of Engineering of the Università Politecnica delle Marche. His main research interests are in Wireless Body Sensor Networks for Internet of Things solution, data fusion algorithms for array sensors, and applications for biomedical devices. He received the Ph.D. degree in Biomedical, Electronic, and Telecommunications Engineering in 2016 and the Master Degree in Telecommunications Engineering in 2012 from the Università Politecnica delle Marche, Italy.

LORENZO PALMA received the Master's degree (cum laude) in Electronic Engineering in 2012 and the Ph.D. degree in Information Engineering in 2017 from the Polytechnic University of Marche, Italy. In 2017 he had a research grant from Consortium GARR to develop a new system for seismic and structural monitoring based on IPv6 sensors able to provide data for the creation of an Earthquake Early Warning system. He is currently a research fellow at Polytechnic University of Marche. His research interests are IoT, Wireless Sensors Networks, IMU sensors, embedded systems, communication network.