2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)

DRIVE: Discovery seRvice for fully-Integrated 5G enVironmEnt in the IoT

Paolo Bellavista, Luca Foschini, Domenico Scotece
Computer Science and Engineering Dept. (DISI)
University of Bologna
Viale del Risorgimento, 2 – 40136 Bologna, Italy
{paolo.bellavista, luca.foschini, domenico.scotece}@unibo.it

Kyriaki Karypidou, Periklis Chatzimisios
Department of Informatics
Alexander TEI of Thessaloniki & CSSN Research Lab
Thessaloniki, Greece
kikikaryp@gmail.com, pchatzimisios@ieee.org

Abstract—**A lot of research is being carried out about Internet of Things (IoT) and in the following years it will emerge more and more in our lives. Furthermore, with the advent of future fully-integrated 5G networks, new constraints need to be satisfied such as ultra-reliability and low-latency. With the help of Fog computing and the Multi-access Edge Computing (MEC) framework, services can be offered to the end users in a fast and practical way. Our work presents DRIVE a framework for service discovery in a 5G environment. However, in order to guarantee dynamic distribution and best management of services, we plan to deploy those services as container (e.g. Docker container). Moreover, we propose distribution of edge services at three different layer of communication: Application, Service, and Communication Layer. Given the above considerations, we propose an edge node, placed at the edge of network, that acts as the "brain" and take over the computation. The main innovative elements of the proposed framework, compared to the existing literature, include the possibility to select the working layer, the dynamic reconfiguration of the edge node and the field experimental results about the performance achieved by our solution over rapidly deployable environments with resource-limited edge nodes such as Raspberry Pi devices.**

Keywords—IoT; Fog Computing; MEC; 5G; Service Discovery; Raspberry Pi

I. Introduction

During the past decade, the Internet of Things (IoT) has revolutionized the ubiquitous computing with multitude of applications built around various types of sensors. It has resulted in a massive increase in data traffic due to the frequent exchange of information between an enormous number of IoT devices. In fact, Cisco has predicted that about 500 billion devices will be connected to the Internet by 2030 [1]. Due to the resource-constrained nature of IoT devices, such as limited battery life and lower processing and storage capabilities, IoT devices cannot achieve same or similar performance compared to the desktop devices. Therefore, many computational offloading studies [2] have proposed which computational tasks should be offloaded to the public cloud, thereby prolonging battery life and reducing power consumption.

To overcome the aforementioned problems, the paradigm of Edge Computing, that implements the support the devices with a cloud closer to the edge of the network, appears as an appealing solution. In this area, two classes of proposals have recently emerged as important enablers, namely, the standardization effort under the name of Multi-access Edge Computing (MEC) and the platforms/solutions that start to be available under the name of Fog computing [3, 4].

MEC is an architectural model and specification proposal by European Telecommunications Standards Institute - ETSI that aims at evolving the traditional two-layers cloud- device integration model, where mobile nodes directly communicate with a central cloud through the Internet, with the introduction of a third intermediate middleware layer that is executed at the so-called network edges. Moreover, the new MEC model allows moving and hosting computing/storage resources at network edges close to the targeted mobile devices, thus, overcoming the typical limitations of direct cloud-device interactions.

In parallel, with a stronger emphasis on the (sensing) devices and communication-enabled things, hence stemming from the IoT scenario, Cisco has initially proposed the Fog computing paradigm. Moreover, we have recently witnessed some related standardization efforts inside the Open Fog Consortium [5]. The proposed Fog model shares with MEC the idea of interposing an intermediate layer, deployed at the edge of the network, that exists between final devices and the central cloud.

MEC and Fog focus on the quality and performance of several cloud-assisted device services, but actually these models still face some non-negligible incompleteness and weaknesses. In fact, these models lack the support for future 5G networks such as the support to heterogeneous devices and the support for dynamic network reconfiguration. Moreover, one of the main problems of MEC and Fog is that in most cases the software is embedded on the edge node and operates for a specific ecosystem. In addition, the network administrator rarely re-deploys software on edge nodes.

In order to overcome the current limitations of MEC and Fog as well as to evaluate one solution for future 5G networks, we designed and implemented DRIVE, a framework for service discovery functionalities. Our edge node, deployed on a device with limited resources (e.g. Raspberry Pi), supports connection with heterogeneous devices and pluggable services. We evaluate our 5G DRIVE prototype in a real-world scenario by running IoT applications from sensors to edge nodes. The results show the significant responsiveness and scalability of our solution.

Our paper presents the design, implementation and experiments of an innovative service discovery functionality for edge computing based on two primary directions of gateway node improvement, i.e., i) dynamic reconfiguration of the edge node, and ii) Docker-based containerization [6] over resource-limited Raspberry Pi devices.

On the other hand, by utilizing DRIVE we significantly enrich the edge intermediate layer by giving the opportunity of exploiting container-based virtualization on top of IoT gateways, with full infrastructure support (download, update, and management of virtualized images based on industry-mature Docker). To the best of our knowledge, this is one of the first cases of implementation and experimentation of virtualization techniques over edge nodes, while working with IoT gateways with very limited resources, such as Raspberry Pi nodes.

The remainder of this paper is organized as follows. Section II provides the necessary background material as well as the overview of the most important related literature. Our prototype system overview with the design and implementation of DRIVE are described in Section III. Section IV provides the setup of our experiment environment and performance evaluation. Finally, we summarize and conclude our work in Section V.

II. Background & Related Work

In this Section we provide certain definitions for the involved technologies and paradigms and try to summarize the mainstream directions of the literature.

A. *Multi-access Edge Computing (MEC)*

MEC is considered an emerging technology and an important component of the emerging 5G networks. The main idea of MEC is to place storage and computation resources at the network edge, in the proximity of users. MEC can be seen as a relatively small datacenter running at the edge of a mobile network and performing tasks, such as hosting services on behalf of mobile nodes, that could not be achieved with the support of traditional network infrastructure. The European Telecommunications Standards Institute (ETSI) have published a well-knownwhite paper that emphasizes on how MEC is characterized by low latency, proximity, high bandwidth, real-time, radio network requirements, and location awareness. In addition to these standardization efforts and specifications, ETSI provides some examples of use cases that would benefit from the adoption of MEC solutions, for example, augmented reality, video analytics, gaming, IoT applications, etc. [3].

B. *FogComputing*

Fog computing starts from the primary idea of extending cloud computing and services to the edge of the network [4]. Similar to MEC, fog provides data, storage, computing, and application services to end-users, but with stronger emphasis on virtualization techniques on the one hand and IoT scenarios on the other hand.

The OpenFog Consortium was initiated by ARM, Cisco, Dell, Intel, Microsoft, and Princeton University in November 2015. Through the OpenFog Reference Architecture (OpenFog RA) [5], the consortium intends to help developers to implement and deploy fog-based solutions. The OpenFog RA describes a generic fog platform that is designed to be applicable to any scenario or application, from transportation and agriculture to smart cities and smart buildings, from healthcare to virtual sensor networks, by providing business value for IoT applications that require low latency, real-time decision making, and other constrained quality indicators. The OpenFog RA is based on a set of core principles, called pillars; these pillars represent the key attributes that a system needs to embody and include security, scalability, openness, autonomy, agility, hierarchy, and programmability. For our purposes, the most interesting pillar is hierarchy that will be detailed in the following sections. In particular, depending on the scalability and nature of the scenario, the hierarchy may be a network of smart and connected partitioned systems arranged in either physical or logical layers, or it may collapse into a single physical system.

C. *Related Work*

In edge/cloud computing, there are several works related to this research that include a survey by Dinh et al. [7] in the area of mobile cloud computing. Moreover, the evolution of IoT, due to its heterogeneous and dynamic characteristics, has created new emerging fields of research and several approaches and proposed have been presented both by research community and companies. For instance, the work presented by R.Muñoz et al [8] is moving on the direction to integrate heterogeneous wireless networks, IoT and cloud computing.

Focusing on service discovery functionality, there are many proposals in the literature for fog/edge scenarios. Most of them have been conceived for Local Area Networks (LANs), such as UPnP [9], and are focused to work with a specific mechanism or protocol. Moreover, existing protocols are not targeting for constrained devices, such as those used in IoT environments. A common approach is to have an intermediate layer (edge layer) that by supporting multiple networks technologies, it enables constrained devices to communicate with the edge node.

Efforts have been done to adapt these solutions to the world of heterogeneous constrained devices. In [10], the authors proposed an architecture for integrating the cloud and the IoT. They introduce the concept of “Smart Gateway”, which is intended to act as an intermediate layer between heterogeneous sensors network and the cloud. The role of “Smart Gateway” is similar to our edge node, in terms of supporting several heterogeneous networks. In [11], a Fog node is proposed, denoted IoT Hub, which enhances the networks capabilities by implementing the functions of border router, cross-proxy, cache and resource directory.

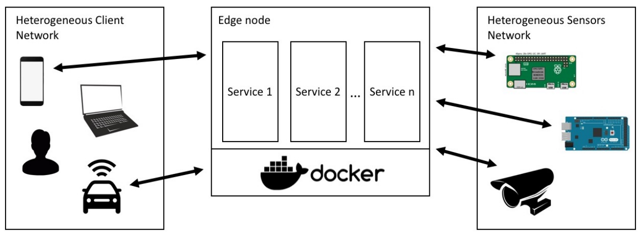

Fig. 1. General Architecture.

The proposed DRIVE framework extends the vision of a Smart Gateway handling only network functionalities and few support services. Our edge node adds enhanced heterogeneity by implementing discovery services in different layers. In addition, our edge node exploits the use of container-based virtualization on top of the IoT gateways.

III. DRIVE: Architecture and Implementation

The proposed DRIVE framework aims at supporting discovery services functionalities for fully-integrated 5G networks. With the goal of creating an edge node suitable for heterogeneous IoT devices and for dynamic 5G networks, the current Section presents the main architectural models and shows our implementation.

A. *Architecture*

DRIVE platform consists of three main components as shown in Fig. 1, clients network, edge nodes and sensors network. In the client domain, there are not only smartphones and laptop devices but also various types of devices that connect to the edge node via different communication technologies. On the other side, there are different types of IoT devices that are also connected to edge node via different network connections.

Fig. 1 illustrates the general architecture of DRIVE infrastructure. As presented in our previous work [12], we aim to merge the MEC and Fog architectures in a single infrastructure with the combination of the supported functionalities.

In general, the architectures mentioned in the previous Section operate by utilizing a specific hardware and software ecosystem. For instance, as described in the MEC white paper [3], the mobile nodes are connected to the MEC Server via LTE through a Radio – Access Network (RAN) and the services, installed in the MEC server, are accessible via IP communication. On the contrary, in the Fog architecture, the ecosystem is more heterogeneous than in the MEC architecture due to the affinity with the IoT. Moreover, as mentioned in [4], the IoT environment is composed by several heterogeneous “smart objects” each one with different network protocols and different features. The Smart Gateway is responsible of offering compute, storage, and networking resources to underlying devices. Thus, our proposed architecture targets to overcome the limitations imposed by each architecture. Actually, DRIVE aims to bring the best of the functionalities provided by MEC and Fog. In particular, work in [12] describes the list of the services and functionalities that the new

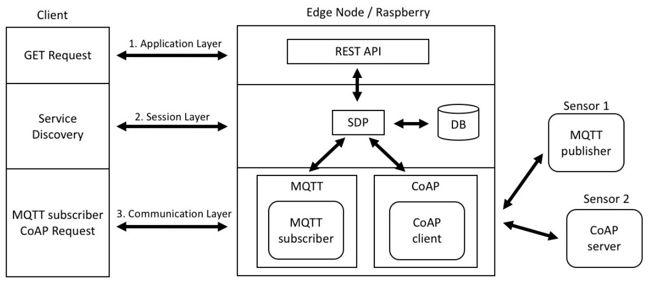

Fig. 2. Implementation of the DRIVE framework.

architecture supports. These services are deployed on top of containerization technology (such as Docker Container) in order to simplify the execution and distribution of services across the infrastructure.

As shown in Fig. 1, our solution architecture is a vertical multi-layers architecture, composed of a heterogeneous client network, an edge computing layer, and a heterogeneous sensor layer. All of them are described as follows:

1) Heterogeneous Client Network Layer: It consists of a set of heterogeneous clients that wants communicate with edge node via different network protocols. The clients must be able to search which services are available in edge nodes and to use those services in the best possible way.

2) Edge node Layer: It consists of multiple distributed nodes to provide functionalities for both heterogeneous client network layer and heterogeneous sensor network layer. From the application perspective, edge nodes provide, to client layer, a containers-based self-intelligent application able to perform calculations and statistical analysis on the information sent. In fact, it retrieves data from the sensor layer, stores it into a database, performs analysis and communicates the results back to the client devices layer. The edge layer manages the mobile devices layer providing them the functionalities needed for IoT endpoints to analyze environmental information and to act on the environment accordingly to the goal of the application scenario usecase considered.

3) Heterogeneous Sensor Network Layer: It consists of IoT endpoints immerse into the environment. IoT devices are all the sensors that sense the information from the surrounding environment and the actuators that modify it. We consider IoT devices as the smallest possible entity, with no internal computational power and only network ability, to send the data towards the nearest edge node. Like the clients, the sensors are able to communicate with edge node via different network protocols.

B. *Implementation*

In order to guarantee the maximum interoperability between components, the DRIVE framework has been implemented using three different layers; Application, Session and Communication layer. Each layer is able to provide service discovery functionalities to a node in the network. We also provide an implementation of client side together with an

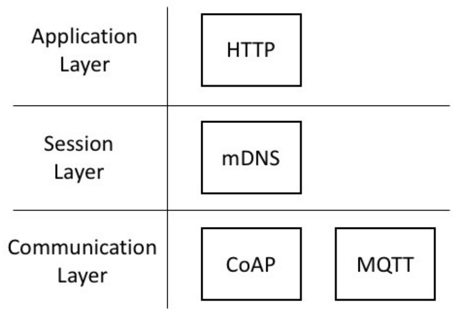

Fig. 3. The protocol stack.

implementation of the sensors layer. Fig. 2 illustrates the overall implementation of the DRIVE framework and a more detailed description of all layers can be found below:

- **Application layer.** This layer contains the implementation of REST API for client requests. The APIs contain invocation of functionalities provided by sensors. This layer was implemented in Java using Retrofit API [13] and allow clients to bypass the Service Discovery protocol. We have realized several API, such as *get_All_Services*, for retrieving the list of al services deployed in the edge node, *get_Humidity*, *get_Aggregate_Humidity*, *get_Temperature*, *get_Aggregate_ Temperature* and so on, for invoking a specific service.
- **Session layer.** This layer includes the Service Discovery Protocol and the Database. The JmDNS [14] library that was used to develop this layer is an implementation of multi-cast DNS in Java. The multicast Domain Name System (mDNS) resolves host names–in our case it is service names- to IP addresses within small networks. It is based on zero-configuration [15], [16], this means that we don't have to configure any network setting programmatically to make it work. It used the same programming interfaces, packet format and operating semantics as the unicast DNS. The job of mDNS is to send IP multicast User Datagram Protocol (UDP) packets that include the services to a certain multicast address. Then all mDNS capable hosts-the android client is the host we are interested in- listen to this address. The client will discover the services by specifying the service type in the code. In more detail, the first step in this procedure is to find the name of the proxy that provides the services. This will give us the mDNS name of the proxy. Then the second step is to resolve this name of the host using mDNS. By multicasting the name, the proxy will listen to the packet that looks for him and responds via broadcast with its IP address, port number and return the services.
- **Communication layer.** By utilizing this layer the clients are able to communicate directly with the participant sensors in the network. This is possible because there is block of software that allow client to communicate with sensor. For instance, in our implementation there is a CoAP server and a MQTT publisher inside the sensors.

Fig. 3 clearly reports the whole protocol stack showing all protocols specifically used on each stack layer.

Message Queuing Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP) [17,18] are the two main protocols used in this implementation to able to send and receive values from different devices. Two of the most promising protocols for small devices that are very well suited to constrained environments [19]. They provide mechanisms for asynchronous communication and are open standards. As mentioned previously, CoAP is a specialised web protocol based on the REST model [20]. More extensively, REST model makes possible for servers to create resources and let the clients access these through a URL using methods like GET, PUT, POST, DELETE. From the other side, MQTT follows a different logic, it is a lightweight publish/subscribe messaging transport protocol that is ideal for mobile applications due to its small size, low power usage, minimised data packets and efficient distribution of information to one or many receivers [11]. A broker is required to exist in the middle of the two ends (the subscriber and the publisher).

The client was realized with an Android smartphone. The implementation presented in this paper is based on independent layers so one can simply implement only one layer of the three or only the layers that he/she needs. The Android application includes an integration of the MQTT and CoAP communication protocol, the JmDNS library on the session layer and the Retrofit API which makes possible to the user to connect with the REST API by sending HTTP requests and receiving JSON objects as java objects.

Finally, we realized sensors using Raspberry Pi Zero and installed on them the right software for communicating with the edge node and the client as well. For instance, we provided the implementation of JmDNS in order to register the services on edge node as well as the implementation of MQTT and CoAP protocols.

IV. Experimental Results

We widely assessed and validate the feasibility of our framework for service discovery. The current Section presents a significant selection of experimental results obtained in our lab deployment scenario. We carried out several different sets of experiments.

A. Experimental Environment

The prototype of our experimental environment is set up from clients/sensors to edge node. The edge node is a Raspberry Pi 3 Model B equipped with 64-bit quad-core ARM Cortex-A53 processor, 1 GB of RAM and 16 GB of storage space, Wi-Fi and Bluetooth connection. Instead, the two sensors are achieved by Raspberry Pi Zero equipped with 1GHz single-core CPU and 512MB RAM. Finally, the client is a smartphone (Android 6.0.1 Marshmallow version and Android API level 23).

B. Experiments on the Session layer

In this series of experiments, we stressed out the edge node by sending service requests via session layer. In the case of

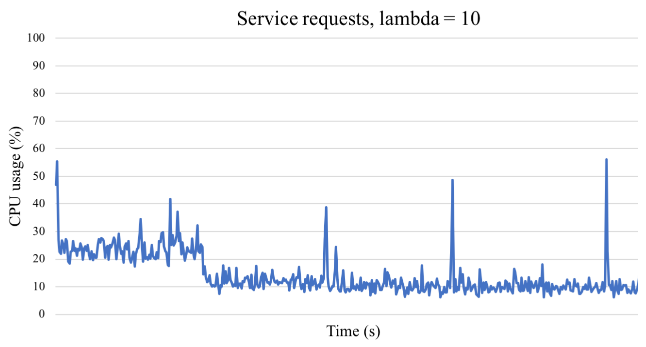

Fig. 4. Performance evaluation for service discovery: CPU usage.

CoAP services requests, the edge node asks to sensor for a new value. Instead, for MQTT services requests the edge node uses MQTT subscriber and the database for saving and retrieving new values. To be able to stress the edge node in a more physical scenario the Poisson distribution was used, it was included in the experiment application that makes services requests. The algorithm that represents the Poisson distribution was embedded in the implementation to mimic realistic arrival patterns. The Poisson algorithm receives as a configuration parameter the lambda value to change the frequency of incoming requests.

We measured the CPU consumption and bandwidth usage at Raspberry Pis during service requests. For in-the-field measurements, we have used the RPi-Monitor tool, i.e., a monitoring application designed to run on Raspberry Pi nodes. RPi-Monitor provides an interactive Web interface to display status and graphs (for further information, see http://rpi-experiences.blogspot.fr/). As clearly shown in the following figures, the maximum CPU consumption is around 50% which averages that up to 2 cores are used over the 4 total cores available (100% means that all 4 cores are used). The tests considered the execution of the service requests from the client to edge node, we carried out the experiments using several lambda values between 1000 and 10. In the paper we report the experiments with 10 as lambda value for the Poisson algorithm (that mimics a realistic scenario with 10-100 requests-per-second), over a 10-minutes observation period. We described the behavior of our work over time to show the stability of the framework. The results are shown in Fig. 4 and Fig. 5 respectively for CPU usage and bandwidth usage.

It is possible to see that the amount of bandwidth used is extremely low and does not exceed 30 KB due to the good characteristics of communication protocols. As for the CPU usage, the results show that the processing load of the edge node is around 15%.

C. *Experiments on the Application layer*

In this series of experiments, we stressed out our edge node by sending continuous HTTP requests. The edge node is able to accept HTTP requests via the Application Layer. Moreover, we simulate multiple requests done by clients. For this purpose, we have used Apache ab (for further information, see https://httpd.apache.org/docs/2.4/programs/ab.html), is a tool for benchmarking a HTTP server. As done for the previous

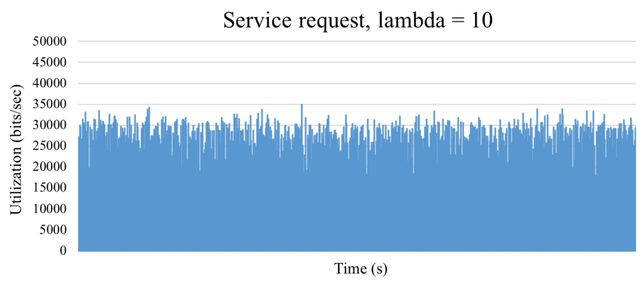

Fig. 5. Performance evaluation for service discovery: bandwidth usage.

experiment, we measured the CPU consumption and bandwidth usage during HTTP requests at edge node.

As shown in Fig. 6 and Fig. 7, the average use of CPU is around 15%. While, the amount of bandwidth usage has increased slightly due to the size of objects exchanged between edge node and device.

V. Conclusion and Ongoing Work

In this paper, we have introduced DRIVE, a new framework for service discovery functionalities for fully-integrated 5G environment in the IoT. The proposed framework operates at three different layers of the protocol stack: i) at the Application Layer, it is able to give fast access for smart clients such as smartphones and provides several functions that are used to enable resource discovery and interoperability among applications, ii) at the Session Layer, it provides a fully-access to service discovery functionalities using DNS facilities, and iii) at the Communication Layer, it provides a direct communication with sensors in the network, without using any service discovery facilities. An implementation of DRIVE has been realized in order to evaluate its practical feasibility and performance. The experiments have been conducted in a real-world scenario comprising the deployment on a resource-constrained device such as Raspberry Pi. DRIVE introduces significant benefits and has proved to be an enabler for resource discovery for fully-integrated 5G environment in the IoT. Encouraged by these positive results, we are already working on two new significant directions. On the one hand, we are exploring the impact of heterogeneous storage and overall performance on the basis of diverse services. On the other hand, we are studying the distribution and coordination of DRIVE framework on multiple devices to grant scalability in widely distributed large deployments.

References

[1] Cisco, “Internet of things.” [Online]. Available: https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf

[2] H. Wu, "Multi-Objective Decision-Making for Mobile Cloud Offloading: A Survey," in IEEE Access, vol. 6, pp. 3962-3976, 2018

[3] White Paper: ETSI’s Mobile Edge Computing initiative explained https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf

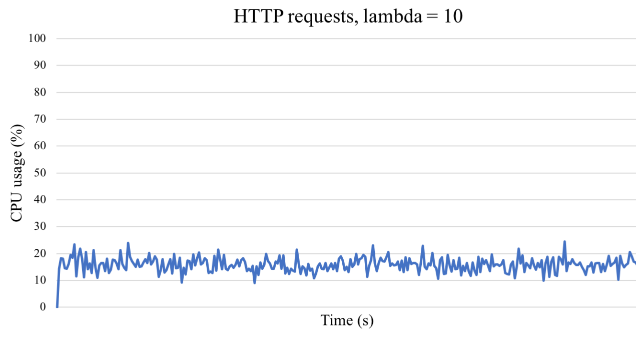

Fig. 6. Performance evaluation for application layer: CPU usage.

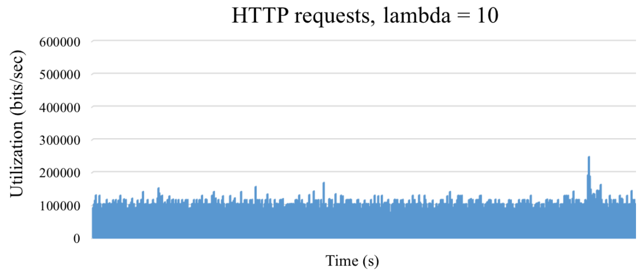

Fig. 7. Performance evaluation for application layer: bandwidth usage.

[4] F. Bonomi, R. Milito, J. Zhu and S. Addepalli, "Fog Computing and Its Role in the Internet of Things", Proc. of the first edition of the workshop on Mobile Cloud Computing (MCC 2012), ISBN: 978- 1-4503-1519-7, doi:10.1145/2342509.2342513, ACM, New York, NY, USA, August, pp. 13-16, 2012

[5] OpenFog Consortium, 2017, http://www.openfogconsortium.org

[6] Docker, [Online]. Available: https://www.docker.com/

[7] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587– 1611, 2013

[8] R. Munoz *et al.*, "The CTTC 5G End-to-End Experimental Platform : Integrating Heterogeneous Wireless/Optical Networks, Distributed Cloud, and IoT Devices," in *IEEE Vehicular Technology Magazine*, vol. 11, no. 1, pp. 50-63, March 2016

[9] UPnP Forums. (1999) [Online]. Available: https://openconnectivity.org/

[10] M. Aazam and E. N. Huh, "Fog Computing and Smart Gateway Based Communication for Cloud of Things," 2014 International Conference on Future Internet of Things and Cloud, Barcelona, pp. 464-470, 2014

[11] S. Cirani, G. Ferrari, N. Iotti and M. Picone, "The IoT hub: a fog node for seamless management of heterogeneous connected smart objects," 2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops), Seattle, WA, pp. 1-6, 2015

[12] P. Bellavista, L. Foschini and D. Scotece, "Converging Mobile Edge Computing, Fog Computing, and IoT Quality Requirements," 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud), Prague, pp. 313-320, 2017

[13] Retrofit REST API, [Online]. Available: http://square.github.io/retrofit/

[14] JmDNS, [Online]. Available: https://github.com/jmdns/jmdns

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

[16] S. Cirani et al., "A Scalable and Self-Configuring Architecture for Service Discovery in the Internet of Things," in IEEE Internet of Things Journal, vol. 1, no. 5, pp. 508-521, 2014

[17] MQTT, OASIS Standard, [Online]. Available: http://mqtt.org/

[18] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252 (Proposed Standard), Internet Engineering Task Force, Jun. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7252.txt

[19] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," 2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT), Namur, pp. 1-6, 2013

[20] L. Rodrigues, J. Guerreiro and N. Correia, "RELOAD/CoAP architecture with resource aggregation/disaggregation service," 2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Valencia, pp. 1-6, 2016