

Towards a Proper Service Placement in Combined Fog-to-Cloud (F2C) Architectures

V.B. Souza^{*§}, X. Masip-Bruin[§], E. Marín-Tordera[§], S. Sánchez-López[§]
J. García[§], G.J. Ren[†], A. Jukan^{††}, A. Juan^{**}

^{*}Universidade Federal de Viçosa, UFV, Brazil
vitorbs@dpi.ufv.br

[§]Universitat Politècnica de Catalunya, CRAAX-UPC, Spain
{xmasip, eva, Sergio, jordig}@ac.upc.edu

[†]IBM, Almaden Research Center, USA
gren@us.ibm.com

^{††}Technische Universität Braunschweig, Germany
a.jukan@tu-bs.de

^{**}ATOS, Barcelona, Spain
ana.juanf@atos.net

Abstract – The Internet of Things (IoT) has empowered the development of a plethora of new services, fueled by the deployment of devices located at the edge, providing multiple capabilities in terms of connectivity as well as in data collection and processing. With the inception of the Fog Computing paradigm, aimed at diminishing the distance between edge-devices and the IT premises running IoT services, the perceived service latency and even the security risks can be reduced, while simultaneously optimizing the network usage. When put together, Fog and Cloud computing (recently coined as fog-to-cloud, F2C) can be used to maximize the advantages of future computer systems, with the whole greater than the sum of individual parts. However, the specifics associated with cloud and fog resource models require new strategies to manage the mapping of novel IoT services into the suitable resources. Despite few proposals for service offloading between fog and cloud systems are slowly gaining momentum in the research community, many issues in service placement, both when the service is ready to be executed admitted as well as when the service is offloaded from Cloud to Fog, and vice-versa, are new and largely unsolved. In this paper, we provide some insights into the relevant features about service placement in F2C scenarios, highlighting main challenges in current systems towards the deployment of the next-generation IoT services.

Keywords- Service placement and execution; resource allocation; cloud & fog computing; distributed systems; Quality of Service

1. Introduction

The advances in mobile computing—including emerging wireless communication protocols, the development of low-cost sensors and the growth of smart data processing in Machine-to-Machine (M2M) communication—, have recently facilitated the deployment of the so-called Internet of Things (IoT) framework. This framework has inspired a plethora of distinct IoT service ideas, tailored to different scenarios and business sectors, including Personal Health Assistants (PHA), Intelligent Transportation Systems (ITS), Smart cities, Smart Home, or Environment Monitoring, among others. Indeed, the predictions in [1] report that by 2021, 1 million IoT devices will be purchased and installed every single hour, what undoubtedly fuel an unstoppable deployment of innovative smart services. At the same time, the IoT framework has been unquestionably endorsed by the technological evolution in the cloud arena as well as by the recently coined fog computing (closely related to mobile edge computing or cloudlets, see [2] for a technology review). Cloud computing, including their corresponding infrastructure in form of Data Centers (DCs), has with no doubt facilitated the development of IoT services offering the distributed data-on-demand provisioning to and from IoT devices deployed in smart environments [3]. On the other hand, fog computing came up to address some of the specific service demands not suitably addressed by cloud, such as latency, while also bringing in novel opportunities, for example in terms of traffic offloading, energy consumption or security. A typical example includes real-time component tracking in manufacturing plants or real-time navigation in traffic control systems, both known to be completely intolerant to any degradations of latency [4]. In fact, in a spiral form, the smart scenarios built by IoT and cloud computing, along with the increasing hardware capacities observed on devices at the edge—notably, mobile devices, such as phones or even vehicles—stimulated the rapid positioning of fog computing [5].

More concretely, in fog computing a user running a service may benefit from proximal resources, thus decreasing the demand for remote cloud resources. In fact, the use of available resources located at the edge (i.e., close to the user), organized as fog nodes – also known as micro-data centers (MDC), mini-

clouds or cloudlets, see [6] for insights on what a fog node may be—, will not only reduce service latency, but will also contribute to reduce the overall Service Response Time (SRT), the network congestion and the energy consumption, while also bringing beneficial aspects on the system security. As an illustrative example, Figure 1 illustrates the benefits brought by fog computing vs cloud computing, as far as communication delay is concerned, in a say conventional smart city scenario, putting together typical smart devices, including smartphones, traffic lights, and smart vehicles. Instead of establishing a high delay communication from edge devices —such as sensors (placed in a car, bus, traffic light, etc.) and end-user devices (e.g. a mobile phone, etc.)—, up to a cloud DC, as shown in Figure 1(a), fog computing provides a low delay fog node (represented as a MDC) offering enough proximal resources for a proper service execution, as shown in Figure 1(b).

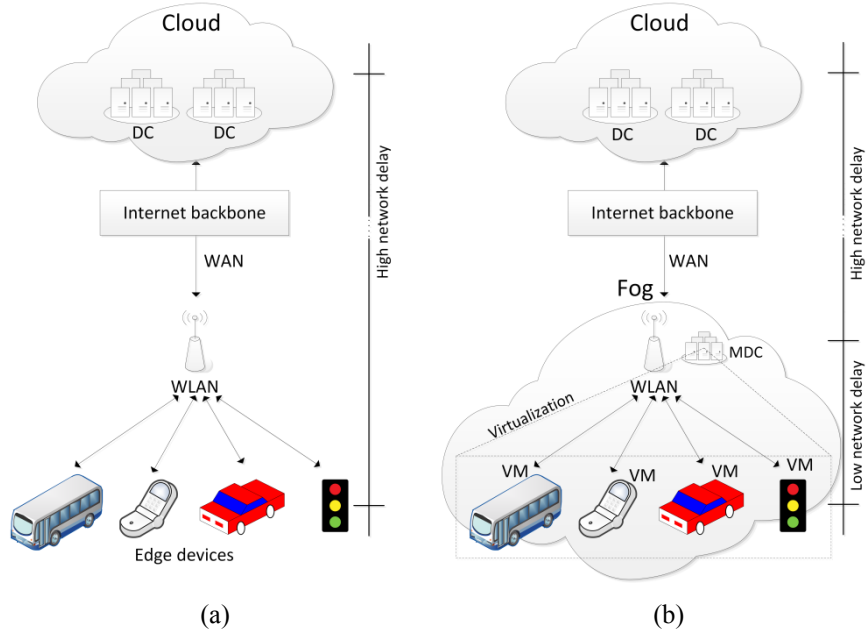


Figure 1. Distinct architectures applied to IoT: (a) Cloud Computing (b) Fog Computing.

Unfortunately, fog computing as a concept, introduces several challenges, which if unaddressed may hinder its real deployment and exploitation as well as limit its applicability. These challenges fall into three main categories, all related to innate characteristics of the edge devices, namely limited processing capacity, limited battery lifetime, and intermittent connectivity, the two latter strongly affecting fog resources availability. Indeed, it seems that the whole concept of fog computing defeats its purpose for the services requiring large processing capacities that can only be served by the cloud infrastructure. On the other hand, services sensitive to latency have no other choice but to resort to fog computing. Put this way, cloud and fog services are highly complementary to each other. Aligned to this complementarity, a new trend recently came up, based on combining fog and cloud skills, that is, making the most out of both, while simultaneously its combination will reduce individual weaknesses, turning into a sort of resource continuous scenario from the edge up to the cloud. Two ongoing initiatives are solidly working in this direction. The first, leverages the work done by the, industrially led, OpenFog Consortium, turning into the definition of a high-level architecture, referred to as the OpenFog RA (Reference Architecture) for cloud and fog management [7], intended to have a remarkable impact on the standards bodies. The second, so-called Fog-to-Cloud (F2C), was proposed in [8], evaluated in [9] and its design is the main purpose of the EU H2020 mF2C project [10]. The proposed management architecture is envisioned as a set of hierarchically distributed layers, putting together the complete set of resources from the edge up to the cloud. A Fog-to-Cloud service model can be highly collaborative, whereby resources required to run a service may be located either at cloud, fog or both. In such a specific scenario, assuming a distributed set of resources (including for example resources clustering in future collaborative models) and novel execution strategies (for example based on parallel execution), the way services are to be executed is definitely challenging. Thus, a key research avenue opens up, requiring the design of novel service placement and execution strategies involving clouds, fogs, and multiple edge devices, all in all in a coordinated way.

The main objective of this paper is to highlight the need for a suitable service placement strategy tailored to F2C scenarios, moving the research to the particular constraints required for a service to be deployed in

a combined F2C system. The paper does not focus on designing specific service placement strategies in either cloud or fog, rather on designing strategies to offloading services execution within the whole set of cloud and fog resources, according to the specific services needs and resources characteristics availability. After a careful analysis of the state-of-the-art, we end up proposing novel service placement strategies based on service atomization and parallel execution. The service atomization concept divides complex IoT services into smaller atomic services, hence easing their distributed execution and improving the overall service execution performance. The idea of parallel resource allocation, on the other hand, is to benefit not only from the parallel data obtained through distinct data sources, but also from the parallel processing of large raw data that may be split into nondependent data portions according to the service to be executed. Finally, we also include numerical results that demonstrate the effectiveness of the presented service placement strategies in a F2C architecture in terms of the overall service execution latency, for a service requested by an end-user device, which may offload the service execution to F2C resources located at distinct layers according to distinct network and processing demands.

The paper is structured as follows. In Section 2, we present the state-of-the-art for the services placement problem in Cloud and Fog computing, respectively, setting the stage to meeting specific service demands in a combined F2C scenario. In section 3, we provide insights in possible service placement and execution strategies and analyze the resource allocation problem in an F2C computing system. The newly proposed service placement and resource allocation strategies are numerically evaluated in Section 4. Finally, Section 5 concludes the paper.

2. State-of-the-Art: Learning from Current Efforts

In this section, we start revisiting the most commonly used cloud service management tools highlighting the need for further research to cover yet unsolved challenges. Then, we analyze state-of-art solutions for service placement in cloud and fog computing, discussing the viability of extending existing tools to operate in a F2C scenario, and ending up in the need for new solutions supporting the deployment of the expected F2C benefits.

2.1. Service management in cloud: Review and open issues

Today, cloud servers are composed of a set of virtualized data centers (DCs) offering distinct cloud service-on-demand models, such as Software as a Service (SaaS), Platform as a Service (PaaS) or Infrastructure as a Service (IaaS). Several tools are available for cloud DCs management, such as, Amazon EC2 [11], Microsoft Azure [12], or Aneka [3], as well as open source tools, such as CloudStack [13], Eucalyptus [14], OpenStack [15], and OpenNebula [16], to name a few. Table I surveys a few well-known solutions, including commonalities but also their distinct features.

Table I - Cloud Management Tools

	License	Model	Services	Virtualization	Compatibility	OS	Scalability
Open Nebula	Apache	IaaS	Compute, Storage	VMware, Xen, KVM	Multi-plataform	Linux	Dynamical, Scalable
Eucalyptus	GPL v3	IaaS	Compute, Storage	Xen, KVM	EC2, S3	Linux	Scalable
OpenStack	Apache	IaaS	Compute, Storage	Xen, KVM, LXC, QEMU, UML, etc.	Multi-plataform	Linux	Scalable
Nimbus	Apache	IaaS	Compute, Storage	Xen	EC2, S3	Linux	Scalable
CloudStack	Apache	IaaS	Compute, Storage	Xen, KVM, VMware	EC2, S3	Linux, Windows, BSD	Scalable
Cassandra	Apache	IaaS	Storage	-	Multi-platform	Linux, Mac, Windows	Scalable
IBM Bluemix	Proprietary	PaaS	Web Application	VMWare, Xen, OpenStack, etc.	Cloud Foundry	Linux, Mac, Windows	Scalable
Amazon	Proprietary	IaaS	Compute,	Xen	-	Linux	Scalable

EC2			Storage				
Google App Engine	Proprietary	PaaS	Web application	Application container	-	Linux, Mac, Windows	Scalable
MS Azure	Proprietary	PaaS	Web and non-web application	OS level through fabric controller	Through Web APIs (e.g., SOAP and REST)	Windows	Scalable
Aneka	Proprietary	PaaS	Compute	Resource manager and scheduler	MS Azure	Windows	Scalable

The OpenNebula platform supports highly heterogeneous platforms in terms of hardware and software in a DC, also centralizing the management of distributed resources, further guaranteeing their efficient use as well as a reduced energy consumption [17]. Moreover, it provides on-demand access to hybrid clouds, as well as the corresponding elastic resource allocation mechanisms [18]. Eucalyptus is an open-source implementation of Amazon EC2 and, in addition to being compatible with the Amazon implementation of EC2, it supports other clients [18]. It should be noted that Eucalyptus does not support VM migration among distinct resources [19]. In OpenStack, the resources management is handled through the implemented OpenStack RESTful APIs, and it also provides the EC2 and S3 compatibility. Moreover, OpenStack is compatible with several hypervisors, including Linux Containers (LXC), User Mode Linux (UML), Quick Emulator (QEMU), among others [20]. Cassandra is an open source key-value based database employed in cloud data centers by several companies, notably Microsoft, Netflix, eBay, among others. Cassandra offers high data availability by replicating data to all nodes in a cluster, eliminating single points of failure [21]. The replication strategy adopted by Cassandra shows high efficiency even for large data traffic. At the same time, it does not optimize resource management under constraints, which makes it suitable for employment in DCs with medium-to-high storage capacity per node. Consequently, Cassandra's utilization in IoT applications demanding storage on resource-constrained devices is not yet feasible. The Nimbus project [22], a cloud solution focused on science, offers the Nimbus Infrastructure, an open source IaaS implementation presenting EC2/S3 compatibility. It is focused on the interests of the scientific community implementing features such as support for proxy credentials, batch schedulers, best-effort allocations and others. CloudStack [13] is another open source IaaS solution. It is designed to deploy large networks of virtual machines, implementing an easy to use Web interface for users to manage their cloud resources. Important features provided by CloudStack include compute orchestration and Network-as-a-Service, among others.

Also proprietary solutions are worth noting. IBM Bluemix [23] is an open-standard, cloud based platform for building, managing and running applications of all types (web, mobile, big data and so on). Bluemix is also offered as Bluemix Local, which considers the DC at client premises jointly with the IBM resources thus allocating resources depending on services demands. Amazon Elastic Compute Cloud (EC2) provides compute infrastructure while Amazon Simple Storage Service (S3) provides storage to cloud services [20]. On the other hand, Google App Engine [24] uses a PaaS model allowing the user to run applications coded in distinct programming languages, such as Python and Java. It also offers integration to Google Cloud Services and APIs [20]. Another PaaS solution, namely Microsoft Azure, also supports Web APIs (e.g., SOAP and REST) in order to allow the communication with non-Microsoft tools and technologies [20]. Aneka is proposed as a service-oriented resource management platform where, instead of providing storage resources, it provides a PaaS that avails itself from the storage, as well as compute resources, from public and private clouds. One example of a symbiotic relation is proposed by [3], where Aneka is integrated to Microsoft Azure in a Cloud Centric architecture. As described in [3], Aneka creates and manages a hybrid cloud where public cloud extra capacity is requested when the private cloud is not able to execute the application in a predetermined deadline. Hence, Aneka may execute one application in several instances in the Azure Cloud as a collection of independent tasks.

Although Table I shows existing and commercial solutions, there are some open issues in cloud management yet unsolved. We should particularly highlight those strongly related to F2C computing, that are service quality delivery, energy savings and distributed resource management. This assessment does not attempt to exclude other relevant challenges –for example, security– highly relevant in the cloud/fog/F2C contexts, but it is just pointing out three key cloud inherited aspects we foresee strongly related to the service execution and placement challenges, the main objective of this paper.

As described in [25], a common concern for all DC providers is the delivery of data to consumers with the appropriate quality guarantees. Even though service quality provisioning is a key objective in the

portfolio for any cloud provider, it has significant impact on other important aspects of data center operation, such as energy savings, since the amount of resources strongly impact on the energy consumption. At the same time however, many DC platforms do not take into account the energy consumption when handling quality of service through their main resources management strategies, such as resource virtualization, service allocation or VM migration [26]. Despite the substantial focus on green DC implementations in the scientific community, this problem remains crucial in real systems, fueling new challenges in the development of cloud computing.

Several contributions consider the importance of reducing the energy consumption in cloud data centers, such as [26] and [27]. Managing energy consumption is strongly related to the amount of resources to be used, but although cloud providers may benefit from the static behavior of DC resources, the VMs allocation, deallocation and migration may vary in a highly dynamical fashion. Aligned to this resources dynamicity, authors in [28] use a load prediction algorithm as part of a strategy to distribute VMs, aiming at guaranteeing an efficient tradeoff between QoS and green computing.

Finally, geographically distributed DCs are adding complexity to the overall cloud management [29]. In this area, authors in [30] propose an integrated control plane for geographically distributed DCs. The paper introduces an enhanced network control plane architecture leveraging distinct strategies for network and IT resources aggregation in DCs. Furthermore, the paper also proposes an orchestration mechanism to enable the identification and selection of DC services, by extending the Path Computation Element (PCE) [31] to an IT-aware architecture. In [32], an extension to the Path Computation Element Communication Protocol (PCEP) was proposed –incorporating new *notify* message parameters embracing IT and storage information–, facilitating an IT-aware path selection. Such proposals are not suitable in scenarios with high resource dynamicity however, as they do not provide mechanism to keep the resources database updated in real time, thus being inefficient for the envisioned IoT scenario.

2.2. Service management in fog: Recent efforts

Service management in fog computing is currently more subject to research than deployment. Indeed, cloud and fog, even though conceptually similar, present crucial differences. Unlike cloud DCs, any fog computing system must deal with the particular constraints brought by the edge devices, such as high dynamicity, mobility, energy issues, reliability, security and heterogeneity, to name a few. In this section, we highlight main state-of-the-art contributions dealing with service placement for execution in fog computing.

Authors in [33] describe the FUSION framework and its architectural aspects regarding services orchestration and selection in fog (here called: edge clouds). The presented architecture aims at reducing the SRT by deploying both parallelism and service-chaining. To this end, the paper suggests the concept of a *session slot*, so that the service availability for any resource may be expressed in terms of the number of sessions it can handle. Moreover, the composed service is depicted by a graph, where nodes represent distinct instances of the provisioned services and edge weights reflect the cost metric of the data path. In [34], a run-time computational efficient heuristic is proposed, the so-called Real Time Thing Allocation algorithm (RTTA). The heuristics' accuracy is demonstrated by modeling the service allocation as an integer optimization problem with the objective of minimizing the energy consumption. Contribution in [35], on the other hand, presents a prediction strategy and pre-allocation of resources in edge MDCs. This strategy is based on customer's loyalty, measured by service relinquish probabilities. In short, historical customer's behavior data is utilized to grant both higher priority and better quality to customers with lower probability of requesting service reallocation. An extension of the PCE [31], so-called Service-oriented PCE (S-PCE), is presented in the position paper [36]. The presented S-PCE architecture is focused on service selection and allocation in mobile clouds and proclaims, albeit conceptually, to address important issues caused by the mobility, volatility and heterogeneity of mobile network elements –different from the work described in [30], where PCE is restricted to static clouds, as described in section 2.1. In a different application area, Fog Computing is applied in a healthcare scenario [37]. In this paper, the Fog nodes are constituted by fixed gateways deployed in the hospital receiving data collected from patients' wearables. The use of fixed gateways enables a simpler static IT resource allocation.

However, most existing contributions focus on a rather small set of issues of service allocation in Fog scenarios. Indeed, even though one of the key issues for a F2C deployment is the resource allocation on edge devices, most past research considers edge devices exclusively as distributed data sources, and do not explore their potential as IT service providers –albeit this is one important feature introduced by Fog Computing enforcing low delay in IoT services processing. On the other hand, although works such as [35] and [37] focus on the allocation of IT resources, the resource volatility is not considered and would

be critical for many applications, such as for smart cities. Authors in [33] aims at achieving low delay on service execution, however the focus is on data acquisition delay only, whereas [36] proposes a centralized control, which on the other hand can introduce additional delay until queries reply. Regarding the importance of consideration of heterogeneous resources, [33] focuses on the heterogeneity of the data type obtained by distinct data resources, whilst [34] considers energy availability only, while assuming that all other resources are unlimited. Moreover, for the multi-objective service selection and orchestration, several constraints beyond IT and network should be also taken into account, such as energy profile, mobility, and data quality, and this multi-objective selection has not been studied yet. Other works, such as [33] and [36], present use cases for service orchestration, albeit without the corresponding algorithms. It should also be noted that none of the current work focus on virtualized resource management in fog scenarios. Overall, the issue of scalability as well as dynamic service placement remains widely unexplored.

2.3. Applying state-of-art solutions to F2C

Recognized the fact that commercial cloud solutions are not designed to be applied to fog and F2C scenarios, issues such as load balancing, VM migration, and multi-objective resource selection may turn into complex challenges when moving from cloud to a fog computing scenario, and much more so in a resources continuous scenario as the one envisioned by F2C, where resource volatility is more the norm than the exception.

On the fog side, the service placement problem for dynamic IoT scenarios still faces many challenges. Although the inclusion of fog elements can reduce the delay, the network load and the energy consumption vs cloud DCs –delay and energy consumption are important concerns in the cloud arena–, unfortunately it brings dynamicity and volatility as two key constraining characteristics, thus further exacerbating the resource selection and allocation problems. Indeed, some works in the literature adopt IoT scenarios with static IT resources at the edge and consider these IoT devices only as source of data, hence ignoring its potential contribution on processing or even storage capacities. Other works, albeit considering both the allocation of these resources and the parallel execution inside one fog node, do not anticipate the allocation of resources in distinct fog and cloud nodes considering their capacities and service requirements.

Table II reviews existing research contributions in the cloud and fog domains against the basic requirements, as they would appear in a F2C system. The individual columns in Table II list the attributes relevant to F2C systems, while the rows give sample contributions as they address the attributes listed. As it can be seen, there are several aspects in resources management strategies in F2C systems that require the solutions currently proposed in the cloud and fog arenas to further evolve. Indeed, the tradeoff between energy efficiency and load balancing for service allocation in clouds has been addressed in [26] and [28]. This idea would require modification when applied in F2C architectures, where physical nodes are often organized in form of micro data centers (MDC), including energy and power constrained devices. In other words, the concentration of several VMs on devices with high volatility may lead to a short-term VM migration and data loss. Authors in [28] also propose a multidimensional resource utilization metric intended to detect possible overload on the individual resources of a physical machine. The resource utilization metric however, may not be enough for F2C –delay, CPU architecture, among others may need to be considered as well. Furthermore, the presented results focus on small scenarios with a low number of physical machines.

Put simply, several major F2C requirements –such as low communication delay, coordination of distributed and heterogeneous resources, scalability, or resources dynamicity management– cannot be met by merely combining a few current cloud and fog service management solutions, but require novel solutions with innovative resource allocation mechanisms and service placement strategies.

3. Service Placement in F2C

From what has been stated so far, it is clear that fog computing and cloud computing, when jointly deployed can bring remarkable benefits. In line with this, the F2C approach, as a theoretical concept, can become the right framework to optimize the execution of highly demanding IoT services, paving the way to a perfect matching between resources capacities and services demands, leveraging both the distributed and centralized resource allocation and service placement (see also [9]). As a matter of wording, we define that if a service to be executed is placed in several resources, a *distributed* resource allocation strategy is deployed. Otherwise, for services to be executed in a single resource, be it either cloud or fog, the resource allocation strategy is referred to as *centralized*.

Table II – Comparison of distinct research contributions in cloud and fog regarding F2C needs

	IT resource allocation	Load Balance	Energy Efficient	Scalability	Low delay	Resource heterogen.	VM management	Distributed allocation	Multi-objective selection	IT resource volatility	Data resource volatility
Heuristic [26]	-	Yes	Yes	Yes	No	Limited	Yes	No	No	No	No
Prediction [28]	-	Yes	Yes	Limited	No	No	Yes	No	Limited	No	No
NCP+ [30]	-	Limited	Limited	Limited	Limited	Yes	Limited	Yes	Yes	Limited	No
FUSION [33]	No	No	No	Limited	Limited	Limited	No	No	Limited	No	Yes
RTTA heuristic [34]	No	No	Yes	Limited	No	Limited	No	No	No	No	No
Prediction algorithm [35]	Yes	No	No	Limited	No	Yes	No	No	-	No	Yes
S-PCE [36]	Yes	Limited	No	Limited	Limited	Yes	No	Yes	Limited	Yes	Yes
Fixed gateway [37]	Yes	No	No	No	Yes	No	No	Yes	No	No	Yes

The strategy to be deployed for services placement must consider service specific demands as well as the set of available resources at fog and cloud premises. At this point it is important to plainly distinguish two main concepts. We define service placement as a service management decision about where a service (or its parts, i.e., subservices) is to be executed. Service offloading, on the other hand, refers to the fact that a service already placed in a type of resource (cloud or fog), must be moved (i.e., offloaded) to another set of resources to make performance much better. In fact, service offloading can be seen as a step further of service placement for optimization purposes. This paper is focused on the first challenge, namely service placement, although as said, service offloading can be an extension of any placement strategy. Put simply, in a perfect (i.e., conceptually, the one where resources and services mapping is optimal) placement scenario, offloading may not be required.

In F2C systems, fog resources can be located at distinct fog layers, created to reflect key resources attributes, such as overall capacity, energy limitations or volatility, just to name a few. The policies and strategies to define the appropriate number of layers is still an ongoing process in F2C systems, and certainly is out of the scope of this paper. However, for the sake of global understanding, an illustrative example of a layered F2C topology, as shown in [38], is depicted in Figure 2, presenting a simple architecture including two fog layers. The rationale behind the deployment of 2 fog layers lies on two main aspects: First, the main objective is to show the layered architecture, and two layers are enough to illustrate the concept. Second, although, as said before, the policy for setting the layers is not yet designed, we assume in the example that the layering leverages a potential categorization strategy envisioned by the F2C architecture, where distinct classes of resources may be grouped into distinct sets, presenting a tradeoff between processing capacity and network delay. It is also worth mentioning that the amount of layers may be different for distinct F2C implementations to accommodate both the distinct categories of the available resources and the set of requirements presented by services in different scenarios (smart city, industry, etc.). Thus, in Figure 2, Fog layer 1 is built by four distinct groups of resources. As such, fog layer 1 is characterized by its resource proximity to the end-users, generally guaranteeing low-latency access but also inheriting the higher volatility inherent to mobile IoT devices, such as sensors –located at cars, smartphones–, or wearables. Fog layer 2 is built by two enriched groups of devices, here assumed to be characterized by lower volatility and higher computational capacity than

fog layer 1. Fog resources in Fog layer 2 can also be seen as some sort of aggregation points, hence aggregating resources located in a neighborhood wide area, thus guaranteeing medium-latency access. For instance, fog layer 2 may be configured by clustering several cars in a parking lot.

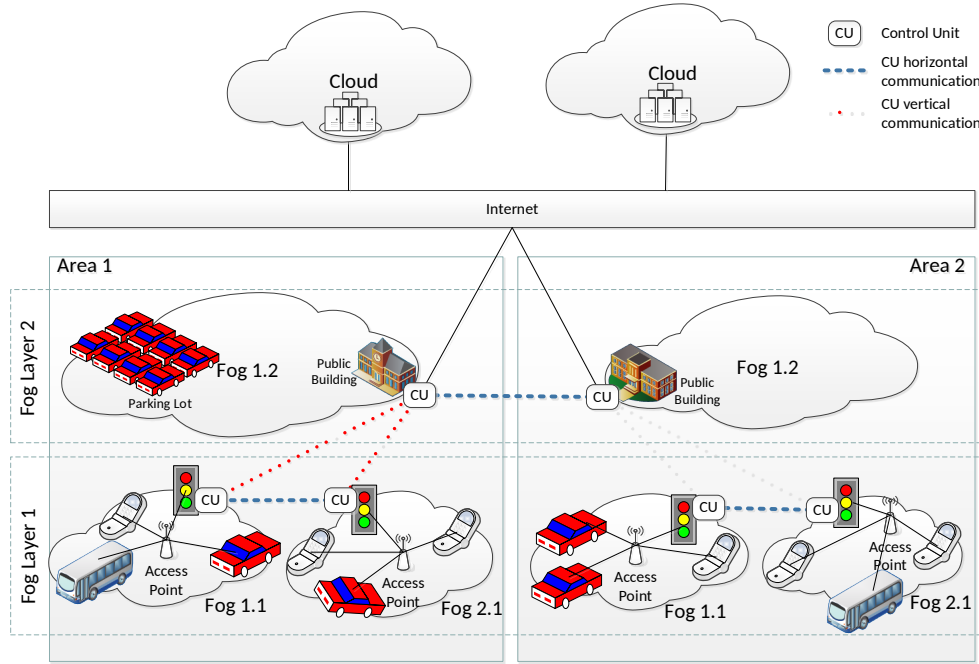


Figure 2. An illustrative F2C architecture based on [38].

In order to facilitate the overall F2C management, Figure 2 considers a geographically distributed locality of resources divided into distinct Areas, all uniquely identified, and each including both fog layers as well as distributed Control Units (CU). The CUs are responsible for implementing the control functionalities required by a F2C system. It is important to mention that policies and rules for each Area definition are out of the scope of this paper. Hence, CUs are responsible for managing the communication among distinct fogs within a specific area, as well as for the communication among distinct areas and the Internet backbone. Notice that a hierarchical architecture is proposed, thus CUs can communicate horizontal only within the same layer in an Area, while vertical communication to higher layers is always permitted. It is also worth mentioning that if a service can be entirely executed in the lower fog layer (depending on the service requirements and the lower layer capacities) the communication with higher levels is not required. Thus, the strategy for the coordination of any service placement strategy will depend on the CU distribution deployed mechanism. In fact, the CU may either centralize all control tasks within an area or be partially deployed among the different fogs within that area, as shown in Figure 2 –CUs are deployed at traffic lights and public buildings.

In this section we first review the main concept of service atomization when applied to F2C systems, and then present two main distributed service execution approaches, sequential and parallel, along with their corresponding service placement strategies.

3.1. Service Atomization, Placement and Execution

Certainly, it is worth highlighting that service atomization is a widely known solution in the distributed computing arena. In short, a service atomization scheme divides complex services into smaller atomic services (subservices), hence easing a distributed execution. The outcome of the atomization –i.e., a set of tailored subservices, hereinafter called atomic services–, can be used flexibly and distributed along different type of resources, be it fogs or clouds, in the F2C topology. Indeed, the main benefit inferred from services atomization is the overall service execution performance.

Figure 3 illustrates the service atomization concept particularly applied to a layered F2C architecture. In the presented architecture, the cloud layer is the highest layer in the hierarchy and while moving down in the hierarchy, fog layers can become more and more distributed throughout the locality of the resources. In this scenario, the outcome of the service atomization process may be represented as a directed graph,

where atomic services may be later placed (i.e., allocated) to resources (physically located at devices) either sequentially or in parallel (parallel execution). Hence, the placement process (including all strategies to be developed for an optimal matching between the atomized service demands and the available resources characteristics), is supposed to have a complete knowledge of available resources, whereby it is assumed that some atomic services may run in parallel and some sequentially, always seeking for both an optimal service execution workflow and an optimal resource utilization. However, this assumption may be rather difficult or non-realistic. Indeed, in general, the information about resources availability is either incomplete, or inaccurate, resulting in suboptimal resource usage (the impact of the information inaccuracy in network control decision has been studied for long, in different network domains, see for example [39] or [40]).

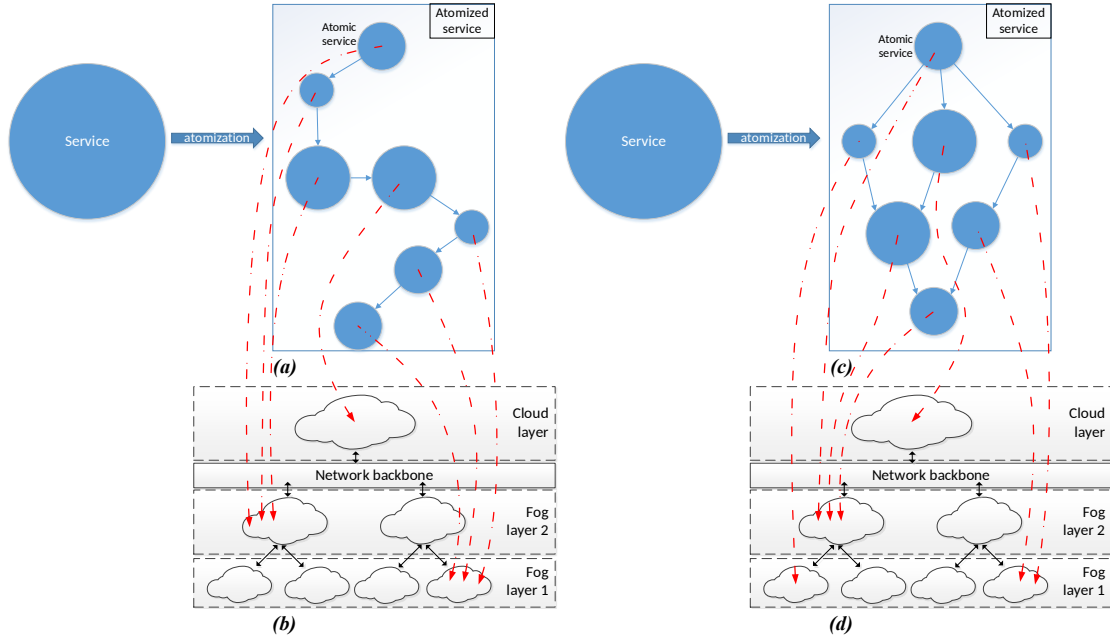


Figure 3. (a) Service atomization for sequential execution and (b) distributed allocation at cloud and fog resources; (c) service atomized for parallel execution and (d) distributed allocation at cloud and fog resources.

While still in its infancy from the research perspective, a salient feature of the F2C concept is its collaborative model for service placement and resource allocation. To stimulate the discussions and open up new research avenues for further studies in this area, we propose to consider two distinct approaches for distributed service execution and placement in F2C systems (as drawn in Figure 3). Figure 3 shows how a service is atomized for either sequential or parallel execution (see Figure 3(a) or Figure 3(c) respectively) and later executed in the available resources (see Figure 3(b) and Figure 3(d)). No matter how the service is atomized, atomic services may be executed at cloud, fog or at both of them, according to specific requirements of each atomic service. We elaborate on the sequential and parallel service execution approaches in the following lines:

- **Sequential service execution:** This approach leverages the distributed resource stack in F2C systems to split service execution into different resources allocated sequentially (see Figure 3(a)). To that end, a service is atomized into a workflow of atomic services, each one to be executed at the particular resource chosen to best match the atomic service needs and resources capacities, be it at either cloud, fog or a mix of both.
- **Parallel service execution:** This approach considers parallel services execution (see Figure 3(c)). The proposed parallel model is akin to the parallel concept brought by High Performance Computing (HPC) several decades ago, which led to the evolution of the current multicore platforms.

The above service execution strategies assume that fog and cloud resources are used collaboratively, in a distributed fashion. It should be noted however, that a centralized approach, which we refer to as *Exclusive*, might also be possible, where a service would be placed and hence executed entirely at either cloud, or one fog-related edge-device. In this paper we do not go much into the details for this centralized approach, indeed we only use it for comparison purposes in the performance results presented in Section 4.

Figure 4 illustrates a possible workflow for service atomization, placement and execution strategy, as previously discussed. Let us assume that the service is complex enough to be atomized, meaning that it implements the required primitives and signaling mechanisms. The workflow runs as follows. A service request is sent to an existing F2C management system for service placement. Reacting to the request, two sequential decision-making processes are issued. The first decision comes right after the service request is received and decides whether the service should be atomized. This analysis is done considering both the requested service –i.e., type and the minimum IT capacity required for its execution, among others–, and some information about the resources –i.e., type, process capacity, network load, the fog/cloud they are currently located, among others. If the decision goes for service atomization, the second decision must decide whether the service must be executed either sequentially or in parallel. This decision is handled by analyzing the atomization graph drawn by the atomization process.

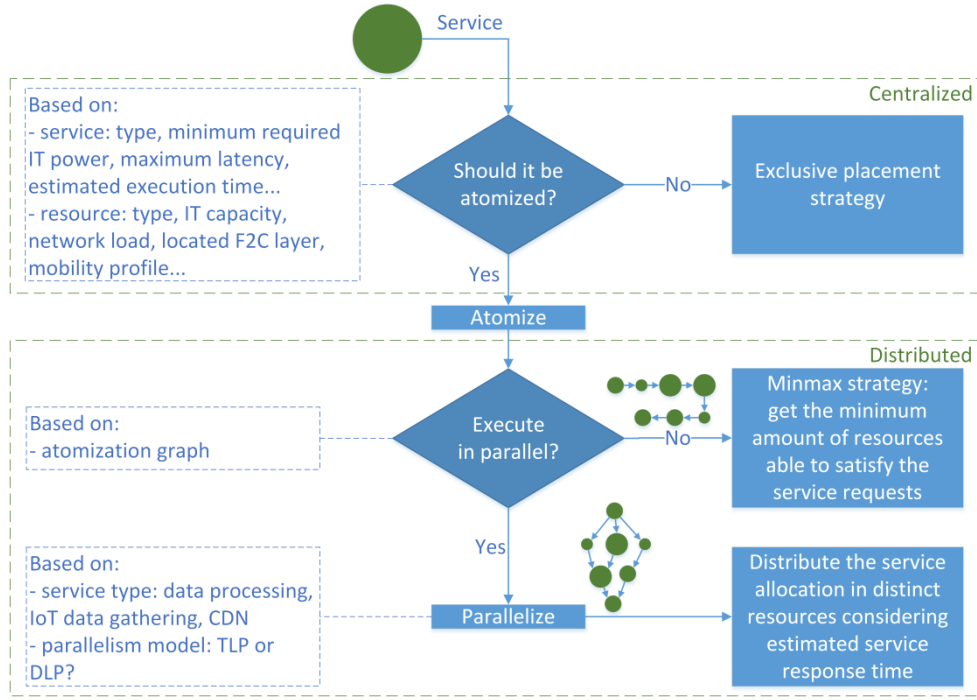


Figure 4. Service atomization, placement and execution overview.

It is worth mentioning that this paper does not go deep into the atomization process. In fact, earlier papers dealing with service slicing can be found in the literature, e.g. [41], [42] and [43]. Instead, our focus is on service placement strategies, particularly for parallel service execution in F2C systems, since this is the scenario where placement strategy may make the most out of simultaneously using fog and cloud resources. Therefore, we assume that a state-of-the-art atomizing strategy has been previously deployed in order to create atomic services that must be allocated at suitable resources.

3.2. Open Discussion

In some scenarios, an exclusive service execution may be the best choice, whereby the whole service is placed on one resource, be it cloud or fog. Such centralized strategy in fact reduces the signaling and atomization effort, as well as the network load. An exclusive strategy might be useful for example, for services collecting all required data from one single data source. For more complex service scenarios, on the other hand, exclusive resource allocation might not be optimal since no single resource can be usually found to meet multiple constraints regarding service quality.

Should service execution is assumed to be distributed, the easiest next choice would be a sequential service execution over distributed resources, whereby multiple resources are involved in executing a service. For example, the resource allocation can follow distributed data sources, such as sensors or datasets, whereby the selection of each source depends on previous processing outcome. In some cases, the resource selection for sequential allocation may be accomplished in runtime by deploying MINMAX heuristics in order to diminish network communication overhead (as shown in Figure 4). The goal can be to minimize the set of resources able to provide the total required resources. As an example of possible challenges for service placement for this scenario, we can mention the deployment of distributed cameras in a smart city, used to trace, in real-time, the location of a car moving through an urban area. In such an

application, the upload of each picture to a dedicated server at cloud may be replaced by its processing at edge devices, which may also play an important role on the selection of the region where further data should be collected, for example according to the car moving direction. An important challenge of this strategy is the efficient allocation of processing resources able to process locally raw data close to each data source, reducing network load and delay.

Finally, atomic services can be executed in parallel (also shown in Figure 4, after being atomized), thus improving the overall service performance. The rationale behind considering a distributed and parallel execution is supported by the benefits brought not only from obtaining data simultaneously from distinct data sources, but also from the processing of large raw data that may be split into nondependent data portions according to the service to be executed. Indeed, delay sensitive services presenting data level parallelism (DLP) may rely on the processing capacity of upcoming devices at the edge of the network; however, the independent processing –i.e., no shared memory and minimum control messages among allocated resources– is an important requirement due to the unreliable and high latency transmission medium.

At the same time, parallel service execution and placement opens up new issues and challenges. For instance, the time expended for each atomic service executed in parallel may differ due to many factors, such as service type, amount of processed data, or allocated resources differences, among others. Therefore, if a service completion depends on the outcome generated by a set of atomic services possibly running in parallel with distinct execution times, and assuming the F2C Control Unit (CU) can estimate the execution time for each atomic service, then, it would be possible to allocate two or more lower time consuming services at the same resource, to be executed sequentially in the same time interval of another time consuming service executed in parallel. Strategies for message passing reduction employed in other distributed scenarios should also be studied and adapted to the F2C architecture, as well as the tradeoff between distributed parallel processing and network load.

4. Assessing Distributed Service Execution in F2C

In this section, we assess the effectiveness of the distributed F2C architecture taking into account the inherent characteristics of the discussed strategies. To that end, we analyze Quality of Service (QoS) performance in terms of delay for a service demanding data generated at the edge of the network, computing the overall service execution latency that includes both transmission and processing latencies. The service execution is triggered by an end-user device, which offloads the service execution to F2C resources located at distinct layers, according to varying network and processing demands –the amount of data that should be transmitted and processed.

It seems obvious that the selection of the most suitable set of resources for service allocation is highly depending on the service type and the knowledge of resources characteristics, such as the nature of input and output data, amount of data generated, demanded capacity of processing resources or dependability among distinct atomized services, among other. Aligned to that, in order to validate the proposed distributed F2C architecture we use a testbed deployed at in-lab premises. The particular service to be deployed consists in an automatic license plate recognition (ALPR), running along with specific strategies for service placement according to its inherent characteristics. The ALPR service may use images obtained from digital cameras placed on highways or roads in a city as input to recognition algorithms intended to detect the license plate number of cars in movement. One of the characteristics of this service is the need to obtain a set of sequential pictures, to facilitate accurate vehicles recognition, since vehicles' movement usually yields blur images, plate view obstruction due to overlapped objects or inadequate viewing angles [44].

In this section, we describe the main characteristics of the trials carried out, including the implemented service placement algorithms the proposed ALPR service. We end up showing main evaluation results and opening a discussion about them.

4.1. Proposed Strategies for Parallel Service Placement

The implemented service, ALPR, brings atomization as one of its main characteristics, since several images may be processed at distinct devices independently. Thus, let's assume a service atomization strategy in place maps each individual image processing task into an atomized service –in other words, each atomic service consists in the processing of a complete image. Our main objective is the provisioning of efficient strategies for parallel service placement –the last step in the workflow shown in Figure 4. Three different placement strategies are proposed for parallel service execution, assuming different resources at the F2C stack may be simultaneously used to execute the service. The presented

service placement algorithms, especially the first and the second, are tailored versions of strategies such as first-fit and best-fit, which are broadly studied in distinct areas of computer science ranging from memory allocation in operating systems to wavelength assigning in optical networks. The third presented algorithm is a tailored version of the best-fit-based algorithm.

In the first proposed placement strategy, First-Fit (FF), the CU always selects one of the available edge devices for allocation and processing of each image. When there are no available resources at the edge, new acquired images are sent for processing at cloud. The rationale behind this strategy is to make the maximum use of edge resources whilst avoiding queuing delay when all edge resources are busy.

The second strategy, Best-Fit (BF), sets a threshold to define the maximum input size available edge resources can handle. When the processing demanded by an atomized service surpasses the threshold, it is sent to cloud. Otherwise, the requested services are sorted in ascending order according to the demanded processing and allocated at the available edge resources. If the amount of edge resources is not enough to satisfy all service requests, the cloud is responsible for handling the remaining requests. It is worth mentioning that, for each evaluated strategy, we consider the set of devices at fog to be homogeneous, hence “best” in the BF algorithm refers to the service placement in either fog or cloud. In other words, we consider, for simplicity, that the available fog resources offer similar hardware capacity for service execution and similar network characteristics –e.g., bandwidth or one-hop communication topology. The BF strategy is described by Algorithm 1.

Algorithm 1: Best-fit strategy (BF)

Input:

T = fog/cloud placement threshold

L = set of images to be processed

F = set of homogeneous fog resources (similar processing capacity and network delay)

Start:

1. **while** true:
 2. **while** L is empty:
 3. update L #wait until receiving new image
 4. Sort L by resolution (ascending)
 5. **while** L is not empty:
 6. $resource$ = look for one free resource in F
 7. **if** $resource$ is not null **and** $L.first.resolution < T$:
 8. #remove the smaller image from L and assign to $picture$
 9. $picture$ = pop first image from L
 10. Offload $picture$ to $resource$
 11. **else:**
 12. #remove the larger image from L and assign to $picture$
 13. $picture$ = pop last image from L
 14. Offload $picture$ to cloud
-

The third strategy, Best-fit with Queue (BQ), is similar to the BF strategy. However, when the amount of resources at the edge devices is not available for service allocation, instead of simply offloading the service execution to cloud premises, an analysis of the service request data along with the expected time for edge resources to be free is performed. The objective is to estimate both the transmission and processing delay (including the uploading and downloading) of the current service at cloud or at fog resources. Taking this into account, this strategy can determine the best option between offloading the service execution to cloud or queuing it to one of the fog resources. This strategy is described by Algorithm 2. It is worth mentioning that fogTime (in line 16) does not depend on the devices since we consider all devices at fog to be homogeneous. Accordingly, we assume that the time to process the image in any device at fog will be the same (as we consider in cloud). This assumption is justified by the fact that our aim is to distinguish the service placement in either cloud or fog, while the insights within fog are left for future studies.

Notice that some efforts available in the recent literature already propose strategies enabling the split of services according to their characteristics [45][46]. However, albeit the set of services requiring atomization and thus a distributed execution into IoT nodes is yet limited, it is with no doubt that the development of such strategies may substantially foster the deployment of novel services in future IoT scenarios requiring the utilization of distributed resources. As an illustrative example we may consider a

“look & find” app that collects pictures from users’ smart phones to find objects. The app will connect to the smart phones where access (by the owner) is previously granted (and the app is also installed), to collect a subset of pictures, determined by a specific pruning and selection policy running in the smartphone, that will go through a matching processing to help find the missing object. This app may be applied for example, to find people, pets, or even to locate cars.

Algorithm 2: Best-fit with Queue strategy (BQ)

Input:

L = set of images to be processed

F =set of homogeneous fog resources (similar processing capacity and network delay)

Start:

```

1. for all resource in  $F$  do:
2.   resource.busy=False      #start with all resources free
3.   resource.queue=[]         #start each queue as an empty list
4.   resource.freetime=now    #estimated time for resource  $F$  to be free
5.
6. while True:
7.   while  $L$  is empty:
8.     update  $L$ 
9.   Sort  $L$  by resolution (ascending)
10.  while  $L$  is not empty:
11.    #remove the smaller image from  $L$  and assign to picture
12.    picture = pop first image from  $L$ 
13.    #estimate SRT (including added latency for uploading and downloading picture)
14.    cloudTime=estimated SRT if placing picture at cloud
15.    #considers homogeneous resources at fog (similar SRT)
16.    fogTime=estimated SRT if placing picture at any fog resource in  $F$ 
17.    if fogTime > cloudTime:
18.      Offload picture to cloud
19.    else:
20.      resource=get one fog resource where resource.busy is False
21.      if resource is null:  #all fog resources are busy
22.        #get the fog resource that is expected to be free sooner
23.        resource=get fog resource with lowest freetime
24.      if resource.busy is False:
25.        #set busy to True. It is set to False after processing all images in the queue
26.        resource.busy = True
27.        resource.freetime=now + fogTime
28.        Offload picture to resource
29.      else:
30.        #adjust freetime in case it is in the past
31.        if resource.freetime < now:
32.          resource.freetime = now
33.        if resource.freetime + fogTime > cloudTime:
34.          Offload picture to cloud
35.        else:
36.          resource.freetime= resource.freetime + fogTime
37.          resource.queue.add(picture)

```

4.2. Evaluation Results

This section presents the results obtained after running distinct experiments on a scenario set by considering cloud and fog layers, formed by robust computers and devices at the edge of the network with constrained processing capacity, respectively. Indeed, the deployed testbed puts together distinct infrastructure. A computer with Intel® Xeon® CPU E5-2620 v2 @ 2.10GHz (6 cores) and 24GB RAM plays as the cloud provider. Edge devices are served by three Raspberry Pi (RPI) with similar processing capabilities, ARMv7 Processor rev 4 (v7l) @ 1.20GHz and 1GB RAM. The ALPR service request is issued by a laptop where a set of vehicles’ pictures, consisting of files with distinct sizes and image

resolution, is located. We consider images are organized according to the file size, turning into different sets of images (represented in terms of either pixels or kB). All requests are sent through a WLAN and the communication delay with cloud provider is added through netem [47], a network emulator for WAN. With the aim of easing the deployment of the testing scenario and the performed experiments, we assume that the cloud provider and the edge devices are all located within the same network. Therefore, the communication between edge and cloud is performed by a link configured with 100ms delay (RTT), bandwidth uplink/downlink respectively 8Mbps/54Mbps and 0.03% chance of packet loss, as shown in Figure 5. The parameters employed for that configuration were inferred from the existing literature, notably [41] and [48], as well as from traditional network analyzer tools, such as *traceroute* and *Zenmap* [49]. The operating system used in all devices is Ubuntu 16.04. Moreover, it is also worth mentioning that the cloud provider does not consider virtual machines –the ALPR algorithms are logically executed at the same OS deployment.

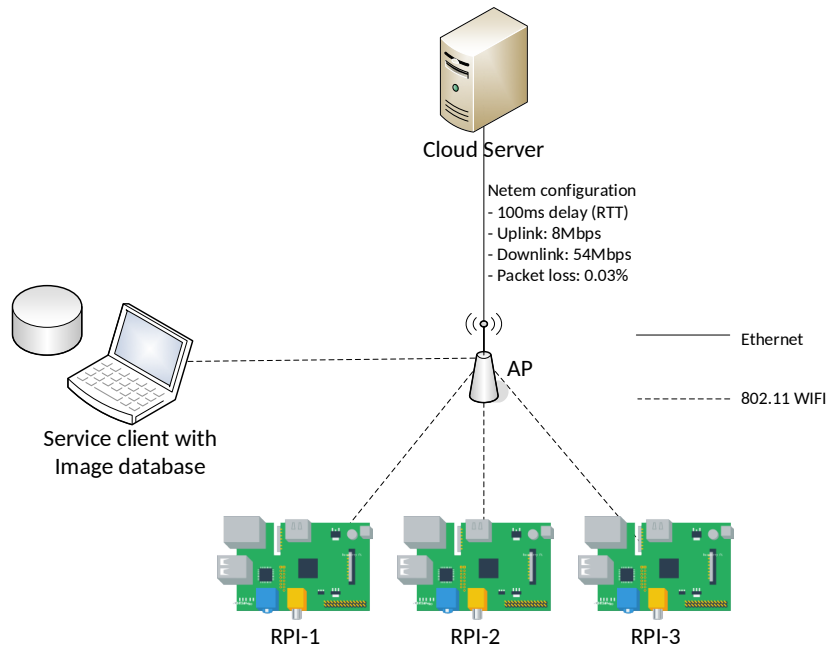


Figure 5. Communication among distinct elements of the deployed testbed.

As described at the beginning of this section 4, the evaluation is performed deploying the ALPR service, that is, a service responsible for acquiring and processing in real-time a huge amount of pictures, in order to locate specific vehicles in a smart city. The presented results aim at showing how the Service Response Time (SRT) is affected by distinct service placement strategies in F2C architectures. To that end, we run two distinct experiments. The first experiment is focused on the assessment of sequential service execution at distinct F2C resources. The main aim of the first experiment is to get the pictures size threshold to be used for the second experiment. Indeed, the second experiment aims at making the most out of the distributed resources through the parallel execution of atomic services. In summary, the set of performance metrics analyzed in the presented evaluations are:

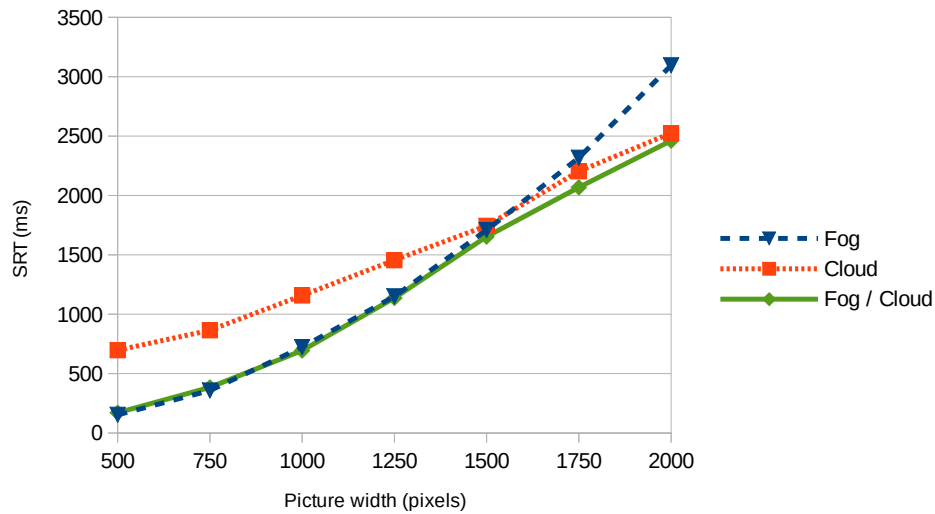
- Network delay: is the total transmission and propagation latency offered by the network for both uploading and downloading service related data.
- Processing delay: is the latency observed at either fog or cloud employed resources for service execution.
- Service response time (total delay): is the sum of both network and processing delay.
- Core network load: is the total amount of data transmitted for service processing in cloud resources.

Our first experiment consists in analyzing the requirements of the deployed service in terms of the SRT, (see Figure 6(a)), computed by adding the network delay (Figure 6(c)) and the processing delay (Figure 6(b)). Results show the evolution of each measured delay for each set of images. This is done through the individual evaluation of the overall latency on the atomic services execution when using cloud or edge

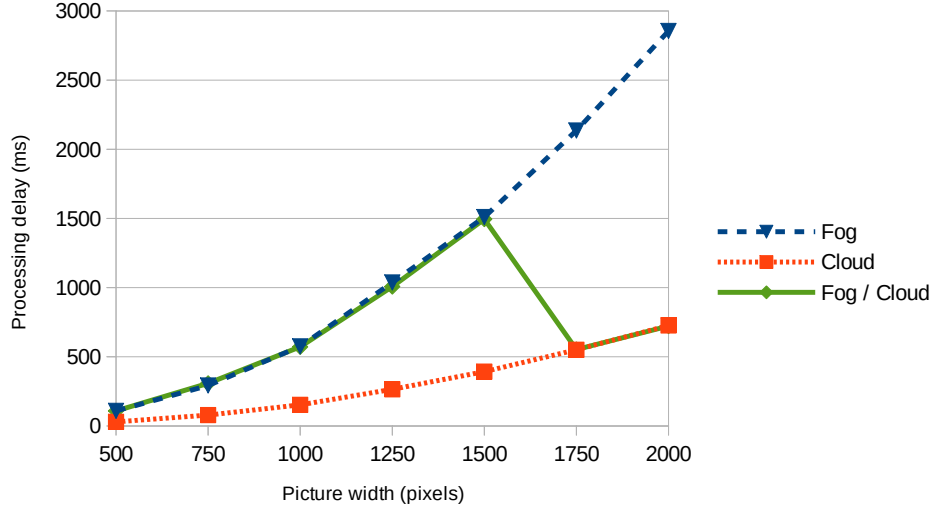
devices for distinct image resolutions. For this purpose, in this first experiment, requests are sequentially offloaded to fog or cloud resources.

The results presented in Figure 6(a) show that when images' width is higher than 1500 pixels, the total SRT exclusively at edge devices (fog) is higher than at cloud. Indeed, by setting 1500 as threshold for offloading the service execution to cloud, rather than fog, the SRT, considering both fog and cloud resources, is always the lowest (see line Fog/Cloud in Figure 6(a)). It is worth noting that, as illustrated by Figure 6(b) the complexity of the employed ALPR service makes fog resources processing latency to increase exponentially as the picture resolution increases, whilst network latency shows a linear behavior (see Figure 6(c)). It is also worth emphasizing the fact that the main aim for the first set of experiments shown in Figure 6 is to detect the threshold value that will be applied to Algorithm 1 and Algorithm 2 as represented in Figure 7. In order to make this threshold value more visible, the fog/cloud line in Figure 6, represents what the behavior would be for the three metrics (SRT, processing and network delay) in a scenario combining fog and cloud, once the threshold is already determined. The curve is simply used to graphically show in a single line, how the system would behave considering cloud or fog resources, assuming the threshold is determined. Once we have the threshold computed, the algorithms can be applied, as shown in the results in Figure 7.

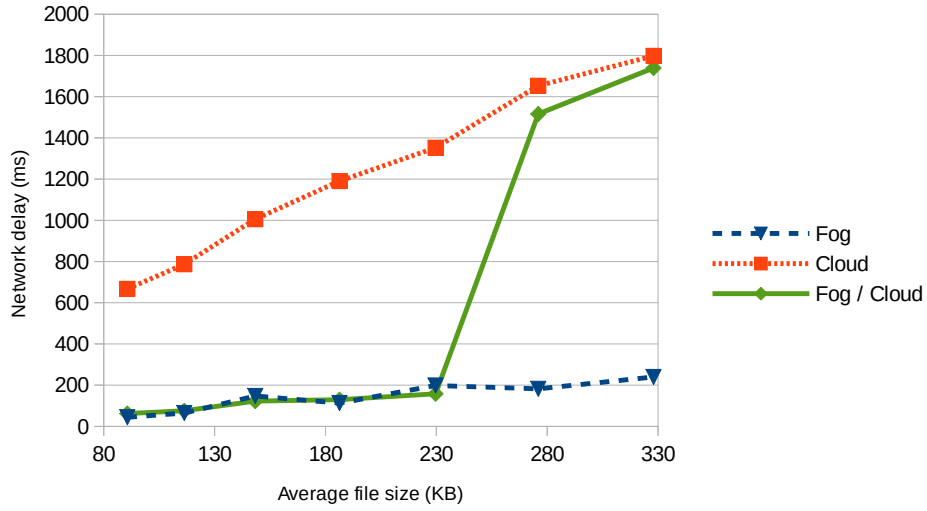
As a summary for the first set of experiments, we show three delays, processing, network, and the accumulative for both of them, SRT, in Figure 6 b,c and a respectively. The interesting value of these figures is to show the key point (threshold) where the requested service placement is moved from fog to cloud. Interestingly, results confirm what may be evident, that is the fact that large images can be better managed at cloud in terms of the required processing, but better managed at fog in terms of the required bandwidth to send the image to the resource. Accordingly, a threshold point is identified, as summarized in Figure 6.a, clearly showing what the image size is, and demanding services execution to switch from fog to cloud.



(a)



(b)



(c)

Figure 6. Delay for service placement at Fog, Cloud and both Fog and Cloud: (a) SRT, (b) processing, (c) network.

The second performed experiment aims at showing the performance for each one of the proposed strategies for parallel service placement and execution in a dynamic scenario. As previously discussed, the deployed vehicle identification service, ALPR, can be atomized into a set of atomic services, each one consisting in the processing of one acquired image, and unlike the settings considered in the first experiment, now atomic services are distributed for parallel execution into distinct fog and cloud resources. It is worth mentioning that, in the proposed service placement strategies, we consider, for simplicity, a static threshold value. Thus, the first experiment may be seen as a preliminary step for the service deployment, seeking for the proper configuration of the parameters to be used on the offloading decisions, such as threshold, according to resources capacity and atomic service demands. For the deployment of the described strategies, the following implementation decisions were considered.

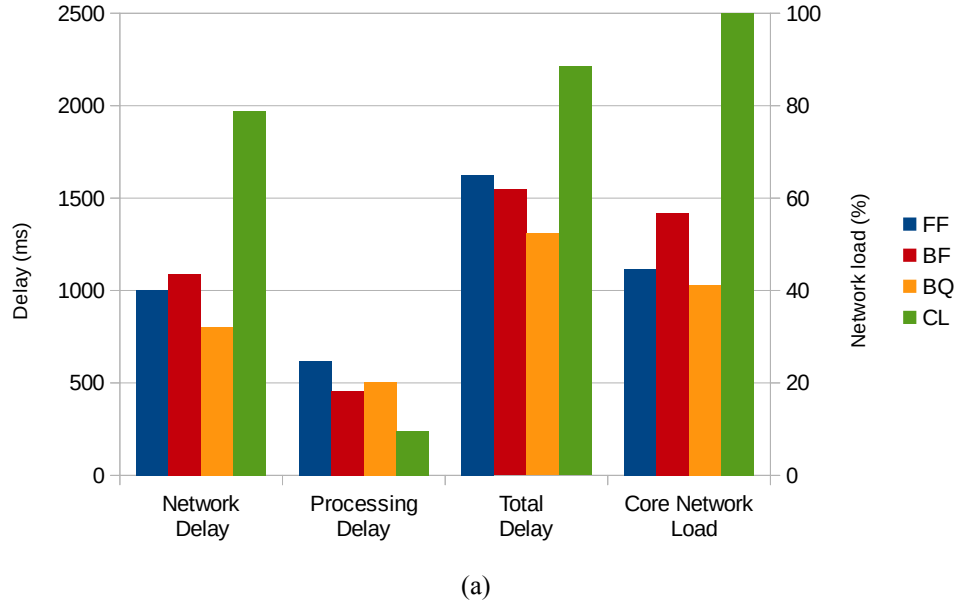
- The time interval between the arrival of each image and the following one is given, in milliseconds, by a gamma distribution with $\alpha=0.5$ and $\beta=1000$.
- The resolution of each image is randomly selected so that image's width is between 250 and 2000 pixels, with steps of 250 pixels. Moreover, the ratio height / width equals 0.667 for all images. In this scenario, the average size of resulting files is between 91 KB and 328 KB. Whilst file size is used for network usage measurement, the picture resolution is enough for determining

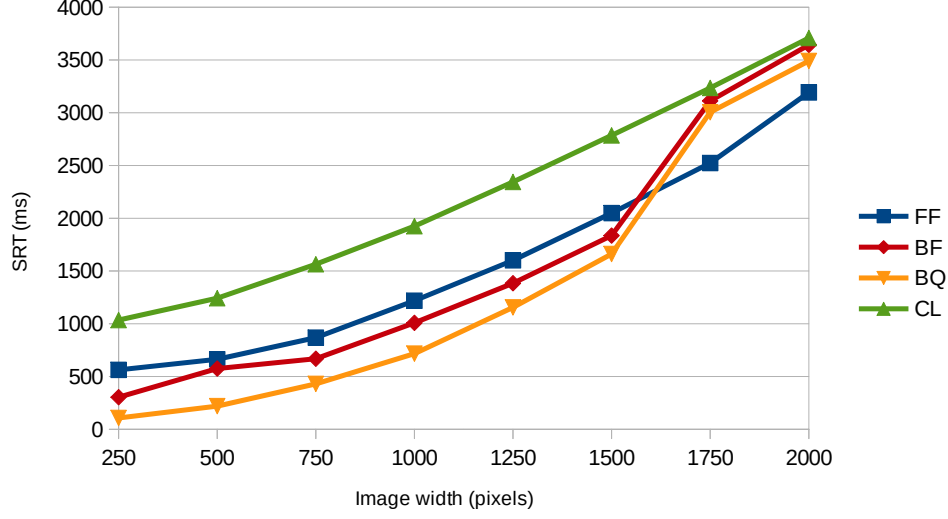
the most suitable computing resource for the execution of the employed ALPR algorithm.

- Although the employed Raspberry PI (RPI) present 4 processing cores, we consider that a prudent strategy to meet the sharing policy developed at the fog devices would be to consider the allocation restricted to one processor core per edge device –usually edge devices are continuously used by owners.
- In the implemented testbed, the amount of edge devices sharing resources for the service execution is limited to 3 RPIs. Moreover, these devices present hardware with the same processing capabilities. Although real IoT scenarios will surely be heterogeneous, in the proposed testbed scenario we only consider homogeneous devices, assuming it should be enough to show the distinct cloud/fog service placement strategies. Moreover, the whole scenario is scaled up to be aligned to the three considered RPIs, in terms of the size of the files to be processed. Certainly, the purpose of this experiment is to show the positive impact of the employment of edge resources on distinct placement strategies in a simplified scenario with the deployment of one single service type.

For the sake of comparison, in addition to the three described strategies for distributed and parallel placement, we also evaluate the traditionally centralized cloud model (aligned to the exclusive approach defined earlier in this paper), where each service execution is offloaded to cloud, hence requiring all the data to be transmitted to its DCs. Albeit this approach is theoretically reasonable considering the higher processing capacity perceived at cloud resources, we evaluate the impact of the added network delay on the overall SRT.

In Figure 7, results for the three presented placement strategies (FF, BF and BQ) as well as the exclusive allocation in cloud resources (CL) are compared. In Figure 7(a), a comparison of measurements regarding both latency and normalized network load is presented. The analysis of the presented results shows that BQ strategy presents the best performance both in terms of total execution delay (obtained by the sum of both network and processing delay) and core network load. Moreover, BF strategy performs better than FF in terms of total execution delay, albeit FF results in a reduced load in the network core in comparison to BF. Indeed, the employment of the FF strategy results in eventual allocation of large files for processing at fog resources. As expected, although the exclusive employment of cloud resources yields the lowest service processing delay, the added network delay highly impacts on the overall SRT, in addition to the high network traffic stemming from the transmission of each service request to cloud premises.





(b)

Figure 7. (a) Average delay for distinct strategies; (b) Total execution delay per image width.

In Figure 7(b), a comparison of the average SRT for distinct images' width is presented. The analysis of this figure shows that pictures presenting width equal or higher than 1750 pixels yield a lower service execution delay in FF than in other strategies. In fact, as FF strategies do not analyze the input data in order to select the most suitable resource according to each request, it has the same probabilities of allocating images with distinct width in fog resources, rendering at a nearly uniform delay increase. On the other hand, BF and BQ strategies are more likely to offload the processing of larger images to cloud resources. It is worth mentioning that, albeit the BQ strategy may add delay to the service execution due the queuing process, it is still lower than the estimated delay for executing at cloud. However, if the estimated delay for executing a service in fog –including the estimated queuing delay– were higher than in cloud, the service would be offloaded to cloud.

By analyzing the results presented for the first and second experiments, we may assess the following conclusions. The deployment of edge resources for processing tasks, extending their exclusive utilization as data providers, can provide a notable QoS improvement in terms of delay as well as a remarkable reduction in the traffic load at the network. However, offloading strategies may take into consideration that total service execution time (SRT) may suffer from a high impact due to the processing time at edge resources as showed by the exponential increase on the processing time depicted in Figure 6(b). Therefore, the benefits of edge computing are expected to be even more pronounced for services demanding less complex algorithms –compared to ALPR. ALPR requires relative large processing capacity even for small files requiring low networking time. Indeed, services employing algorithms with lower complexity and, at the same time, requiring larger data inputs shall take more advantages of the low network delay provided by resources at the edge of the network.

5. Conclusion

In this paper, we analyze service placement strategies in dynamic IoT scenarios taking into account distinct computing paradigms, such as cloud computing, fog computing, and their combination, referred to as F2C Computing. Our goal was to study the benefits of the recently proposed fog and F2C computing, in terms of latency-constrained support in fog, extended with sharing opportunities and novel collaborative models in F2C. After revisiting the state-of-the-art contributions, we elaborated on the service execution and placement problem, by reusing the service atomization concept, as well as conceptually presenting parallel and sequential service execution strategies. Furthermore, we introduce *parallel* and *distributed* execution as the design principle guaranteeing higher quality in service execution, in terms of perceived delay, which we believe carries significant potential for an improved performance in F2C systems. For the assessment of distributed service placement in F2C, distinct strategies were presented and implemented. The presented results illustrate the benefits of distributed execution in F2C resources in terms of both service response time and network core load. Recognized the novelty of the fog and F2C paradigms, the paper revealed a number of open issues and challenges, which can help shape future research in the combined area of fog and cloud computing. Future work includes the enhancement of placement strategies in order to take heterogeneous fog resources into consideration as well as the

improvement of the employed testbed in order to add extra resources and enable the deployment of extra fog layers.

ACKNOWLEDGMENTS

This work was partially supported by the Spanish Ministry of Economy under contract TEC2015-66220-R (MINECO/FEDER) and by the H2020 EU mF2C Project ref. 730929. V. B. Souza is supported by CAPES Foundation, Ministry of Education of Brazil, Proc. No. 11888/13-0.

REFERENCES

- [1] D.C.Plummer et al., "Top Strategic Predictions for 2016 and Beyond: The Future is a Digital Thing", Gartner, https://www.gartner.com/binaries/content/assets/events/keywords/symposium/sym26/gartner_top_strategic_predictions_2016.pdf.
- [2] Y.Ai, M.Peng, K.Zhang, "Edge Computing Technologies for Internet of Things: A Primer", Digital Communications and Networks, Elsevier, open access, 10.1016/j.dcan.2017.07.001
- [3] J.Gubbi et al., "Internet of Things (IoT): A vision, architectural elements, and future directions. " Future Generation Computer Systems, v. 29, n. 7, p. 1645-1660, 2013.
- [4] F.Bonomi et al., "Fog computing: A platform for internet of things and analytics", In: Big Data and Internet of Things: A Roadmap for Smart Environments. Springer International Publishing, 2014. p. 169-186.
- [5] F.Bonomi et al., "Fog computing and its role in the internet of things", In: Proceedings of the first edition of the MCC workshop on Mobile cloud computing. ACM, 2012. p. 13-16.
- [6] E.Marin-Tordera, X.Masip-Bruin, J.Garcia, A.Jukan, GJ.Ren, J.Zhu, "Do we all know what a fog node is? Current trends towards an open definition", Computer Communications, Vol. 109, pp. 117.130, September 2017
- [7] OpenFog Consortium Reference Architecture at <https://www.openfogconsortium.org/#architecture>
- [8] X. Masip-Bruin et al., "Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud (F2C) computing systems", IEEE Wireless Communications Magazine, Vol. 23, issue 5, October 2016.
- [9] W.Ramírez, X.Masip-Bruin, E.Marin-Tordera, V.B. Souza, A.Jukan, GJ.Ren, O.González de Dios, "Evaluating the Benefits of Combined and Continuous Fog-to-Cloud Architectures", Computer Communications, Vol. 113, pp.43-52, November 2017
- [10] mF2C project at <http://www.mf2c-project.eu>
- [11] Amazon EC2. <http://aws.amazon.com/ec2> , Jan 2016.
- [12] Microsoft Azure. <https://azure.microsoft.com/en-us> , Jan 2016.
- [13] Apache Cloudstack Web Page. <https://cloudstack.apache.org> , Jan 2016.
- [14] Eucalyptus Web Page. <http://open.eucalyptus.com> , Jan 2016.
- [15] O.Sefraoui, M.Aissaoui, M.Eleuldi, "OpenStack: toward an open-source solution for cloud computing. International Journal of Computer Applications", v. 55, n. 3, p. 38-42, 2012.
- [16] OpenNebula Web Page. <http://www.opennebula.org> , Jan 2016.
- [17] S.Jain, R.Kumar, S.K.Jangir, "A Comparative Study for Cloud Computing Platform on Open Source Software", ABHIYANTRIKI: An International Journal of Engineering & Technology (AIJET), v. 1, n. 2, p. 28-35.
- [18] J.Peng et al., "Comparison of several cloud computing platforms", . Information Science and Engineering (ISISE), 2009 Second International Symposium on. IEEE, 2009. p. 23-27.
- [19] R.Buyya et al., "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility", Future Generation computer systems, v. 25, n. 6, p. 599-616, 2009.
- [20] S.Yadav, "Comparative Study on Open Source Software for Cloud Computing Platform: Eucalyptus, Openstack and Opennebula", International Journal Of Engineering And Science, v. 3, n. 10, p. 51-54, 2013.
- [21] Cassandra Web Page. <http://cassandra.apache.org> , Jan 2016.
- [22] Nimbus Project Web Page. <http://www.nimbusproject.org> , Jan 2016.
- [23] IBM Bluemix. <https://console.ng.bluemix.net/>, Jul, 2016.
- [24] Google App Engine. <https://appengine.google.com> , Jan 2016.
- [25] Y.Sun, et al., "ROAR: A QoS-oriented modeling framework for automated cloud resource allocation and optimization", Journal of Systems and Software, 2015. <http://dx.doi.org/10.1016/j.jss.2015.08.006>. In Press.
- [26] A.Beloglazov, J.Abawajy, R.Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing", Future generation computer systems, v. 28, n. 5, p. 755-768, 2012.
- [27] A.Wolke et al., "More than bin packing: On dynamic resource allocation strategies in cloud computing", Information Systems, 51(C):83-95, 2015.
- [28] Z.Xiao, W.Song, Q.Chen, "Dynamic resource allocation using virtual machines for cloud computing environment", Parallel and Distributed Systems, IEEE Transactions on, v. 24, n. 6, p. 1107-1117, 2013.
- [29] P. Takako et al., "Resource allocation for distributed cloud: concepts and research challenges". Network, IEEE, v. 25, n. 4, p. 42-46, 2011.
- [30] J.Buyse et al., "NCP+: An integrated network and IT control plane for cloud computing", Optical Switching and Networking, v. 11, p. 137-152, 2014.
- [31] A.Farrel, JP.Vasseur, J.Ash, "A path computation element (PCE)-based architecture", RFC 4655, 2006.
- [32] JP.Vasseur, JL.Le Roux, "Path computation element (PCE) communication protocol (PCEP)", RFC 5440, 2009.
- [33] P.Simoens et al., "Challenges for orchestration and instance selection of composite services in distributed edge clouds", Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on. IEEE, 2015. p. 1196-1201.
- [34] G.Tanganelli, C.Vallati, E.Mingozzi, "E. Energy-efficient QoS-aware service allocation for the cloud of things", Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on. IEEE, 2014. p. 787-792.
- [35] M.Aazam, E.Huh, "Dynamic resource provisioning through Fog micro datacenter", Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on. IEEE, 2015. p. 105-110.
- [36] W.Ramírez, V.Souza, E.Marin-Tordera, S.Sanchez-Lopez, "Exploring potential implementations of PCE in IoT world", Optical Switching and Networking, Vol. 26, pp.48-59. November 2017.
- [37] T.Nguyen et al., "Fog Computing in Healthcare Internet-of-Things: A Case Study on ECG Feature Extraction", IEEE International Conference on Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), Liverpool, UK, 2015
- [38] V.Souza, et al., "Handling Service Allocation in Combined Fog-Cloud Scenarios", in IEEE International Conference on

- Communications (ICC), Malaysia, May 2016.
- [39] X.Masip-Bruin, R.Muñoz, S.Sánchez-López, J.Solé-Pareta, J.Domingo-Pascual, G.Junyent, “An adaptive routing mechanism for reducing the routing inaccuracy effects in an ASON”, IFIP Conference on Optical Network Design and Modeling, ONDM 2003, Budapest, 2003
 - [40] X.Masip-Bruin, E.Marín-Tordera, M.Yannuzzi, R.Serral-Gracia, S.Sánchez-López, “Reducing the effects of routing inaccuracy by means of prediction and an innovative link-state cost”, IEEE Communications Letters, Vol. 14, issue 5, 2010.
 - [41] S.Kosta et al., “Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading”, IEEE Infocom 2012, p. 945-953.
 - [42] H.Li, S.Thompson, “Safe concurrency introduction through slicing”, proceedings of the 2015 Workshop on Partial Evaluation and Program Manipulation. ACM, 2015. p. 103-113.
 - [43] M.Alpuente et al., “Using conditional trace slicing for improving Maude programs”, Science of Computer Programming, v. 80, p. 385-415, 2014.
 - [44] S.Du et al., “Automatic License Plate Recognition (ALPR): A State-of-the-Art Review”, IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, no. 2, pp. 311-325, Feb. 2013.
 - [45] A.Ferrari, S.Giordano, D.Puccinelli, “Reducing your local footprint with anyrun computing”, in Computer Communications, v. 81, p. 1-11, 2016.
 - [46] S.Sahhaf, et al. “Network service chaining with optimized network function embedding supporting service decompositions”, Computer Networks, v. 93, p. 492-505, 2015.
 - [47] Linux Foundation Wiki: netem. <https://wiki.linuxfoundation.org/networking/netem> , Aug 2017.
 - [48] T.Soyata et al., “Cloud-Vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture”, 2012 IEEE Symposium on Computers and Communications (ISCC), Cappadocia, 2012, pp. 59-66.
 - [49] Zenmap webpage, <https://nmap.org/zenmap/>, Aug 2017.