Hierarchical Scheduling Mechanisms in Multi-Level Fog Computing

Maycon Leone Maciel Peixoto, Thiago A. L. Genez, *Member, IEEE*, and Luiz F. Bittencourt, *Member, IEEE*

Abstract—Delivering cloud-like computing facilities at the network edge provides computing services with ultra-low-latency access, yielding highly responsive computing services to application requests. The concept of fog computing has emerged as a computing paradigm that adds layers of computing nodes between the edge and the cloud, also known as *micro data centers*, *cloudlets*, or *fog nodes*. Based on this premise, this article proposes a component-based service scheduler in a cloud-fog computing infrastructure comprising several layers of fog nodes between the edge and the cloud. The proposed scheduler aims to satisfy the application's latency requirements by deciding which services components should be moved upwards in the fog-cloud hierarchy to alleviate computing workloads at the network edge. One communication-aware policy is introduced for resource allocation to enforce resource access prioritization among applications. We evaluate the proposal using the well-known iFogSim simulator. Results suggest that the proposed component-based scheduling algorithm can reduce average delays for application services with stricter latency requirements while still reducing the total network usage when applications exchange data between the components. Results have shown that our policy was able to, on average, reduce the overload impact on the network usage by approximately 11 percent compared to the best allocation policy in the literature while maintaining acceptable delays for latency-sensitive applications.

Index Terms—Fog computing, cloud computing, Internet of Things, allocation

1 Introduction

Cloud computing is indeed a consolidated paradigm that leveraged the industry of the Internet of Things (IoT). Applications, which usually require a lot of computing capabilities to run, have been successfully performed "in the cloud" using *offloading* capabilities [1]. This practice is an effort that offloads (hence the name) most of the application's workload to a remote data center facility for intensive data processing. This action significantly improves the computing resources of IoT devices facing high computation demands [2]. Particularly, offloading has been useful for resource-hungry applications that span the *Big Data* concept [3], which habitually generate a large volume of data at different velocity and variety [4]. However, the reliance on computation capabilities outside of edge devices is not ideal for those applications that require tightly controlled delays (latency) on the data transfer time with remote resources. As cloud data centers are ordinarily localized far away from the devices situated at the edge of the network, a degradation in the application's quality of service is likely to occur, therefore putting the user's quality of experience at critical risk [5].

- *Maycon Leone Maciel Peixoto is with the Federal University of Bahia (UFBA), and Institute of Computing, University of Campinas (Unicamp), Sao Paulo 13083-970, Brazil. E-mail: maycon.leone@ufba.br.*
- *Thiago A. L. Genez is with the European Molecular Biology Laboratory, European Bioinformatics Institute, Wellcome Genome Campus, Hinxton, Cambridgeshire CB10 1SD, U.K. E-mail: thiagogenez@ebi.ac.uk.*
- *Luiz F. Bittencourt is with the Institute of Computing, University of Campinas (Unicamp), São Paulo 13083-970, Brazil. E-mail: bit@ic.unicamp.br.*

Manuscript received 30 Sept. 2020; revised 12 Apr. 2021; accepted 6 May 2021. Date of publication 11 May 2021; date of current version 7 Oct. 2022. (Corresponding author: Maycon Leone Maciel Peixoto.) Digital Object Identifier no. 10.1109/TSC.2021.3079110

When using cloud-based assistant services such as *Siri* or *Alexa*, for instance, the user can tolerate up to a few seconds of delay without experiencing a quality of service degradation [6]. However, applications with complex video and audio processing, such as online gaming [7] and other interactive services, a delay of a few tens of milliseconds can expose the application's performance to severe quality decay, making it unusable for activities that require prompt real-time responses because crucial actions can be lagged or frozen [8]. Indeed, the use of traditional cloud computing methodology for edge devices is undoubtedly a poor strategy to put latency-sensitive applications into the offloading practice [9]. Clearly, the Achilles' heel here is in terms of the network latency on the end-to-end communication channel between the IoT application (situated at the network edge) and the cloud data centers (confined at the network core) [10].

The fog computing concept emerged recently to address this issue. It attempts to mitigate the relatively high latency from using traditional cloud computing resources to perform the offloading procedure for delay-critical services [7], [11]. It introduces cloud-like computing services very close to end-devices, in an infrastructure that places small data centers (also called as *cloudlets* [10] or *fog nodes* [12]) in the network between the edge and the core. A generic model of a fog computing network considers the deployment of several layers of fog nodes from the edge to the core, composing a hierarchy of computing nodes [8] (Fig. 1). The higher a fog node is localized in the hierarchy, the larger its computing capacity is, since it should cover a broader set of users downwards the hierarchy. Analogously, the lower a fog node is established in the hierarchy, the closer to the edge it will be situated, thus presenting lower communication delays to edge devices. As shown in [1], [2], [3], [5], [7], [8],

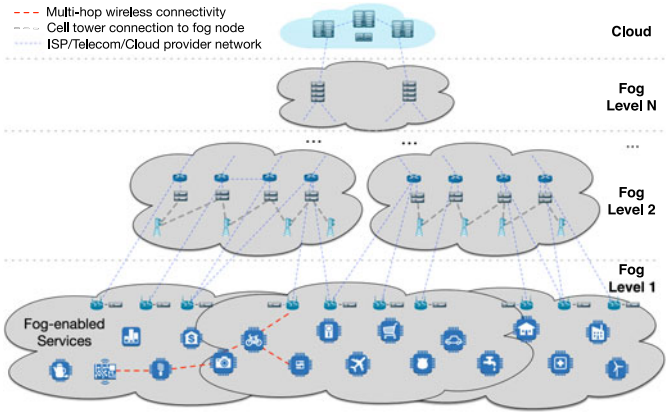


Fig. 1. Multi-level fog computing architecture: Fog-enabled services and their components can be run at any layer of the hierarchy. Upper layers offer more computing capacity, but also increased delay to the edge. Network characteristics (topology, bandwidth, delay, etc.) between devices depend on underlying network connectivity of different Internet Service/Telecom providers (adapted from [2]).

[12], fog computing has indeed great potential for delay-critical applications in performing offloading without having strong service performance degradation.

The deployment of cloud-like computing services closer to users introduces significant resource management issues that are still open [2]. For instance, due to the physical proximity of multi-tiered fog nodes to the network edge, this environment introduces challenges to control the latency for applications with different latency requirements. As the lower cloudlets in the hierarchy have lower computing capacity when compared to the higher ones, applications presenting tight-delay requirements should have resources allocated on these cloudlets to perform the offloading. In comparison, applications having loose-delays requirements can offload to higher cloudlets or even directly to the cloud. These aspects and trade-offs between the cloudlet location in the hierarchy and applications' latency requirements need to be considered by a new design of algorithms to (i) yield proper resource allocation in fog computing, (ii) balance the workloads among all the cloudlets, and (iii) satisfy the overall applications' latency requirements. This paper's contributions focus on addressing these three issues for a multi-layered fog computing environment and latency-sensitive applications.

This paper proposes a resource allocation scheduling algorithm for a fog-cloud hierarchy with multiple cloudlet layers, or tiers, considering applications composed of services that can have heterogeneous delay requirements. Our proposed algorithm examines each latency-sensitive application as a composition of modules service components that communicate with each other through data transfers. Modules can execute away from each other, which introduces delays in response times. When a cloudlet is overloaded, the proposed algorithm decides which application's module should be moved upwards in the fog-cloud hierarchy to satisfy latency requirements while reducing the delay experienced by application users. The migration of workloads carried out by our proposed approach is based on the characteristics of the applications' data dependencies.

The novelty of this paper is the proposal of a communication-aware scheduling policy in a multi-layered fog network. Based on a component allocation policy from the literature, *Edgewards* [2], [8], [13], we proposed one policy in this paper: *Communication Based – Edgewards* (CB-E). In comparison with our previous work [8] and the literature, the contributions of this paper are four-fold:

(1) We propose a communication-aware scheduling policy for a fog-cloud computing system, namely Communication Based & Edgewards (CB-E);
(2) We consider a fog-cloud hierarchy topology in our experiment comprising three levels: two layers of cloudlets (or fog nodes) and a cloud layer;
(3) We evaluate our proposed policy using two types of common component-based services for the IoT industry, such as a latency-sensitive online game and a delay-tolerant video surveillance network;
(4) We evaluate and demonstrate the effectiveness of our proposed policy using the well-established Fog Computing Simulator known as iFogSim [13].

This paper is organized as follows. Section 2 discusses related work. Sections 3 and 4 introduces definitions, concepts and models used in this paper, while Section 5 describes the proposed algorithm. Section 6 covers the evaluation setup and Section 7 presents the results of simulation carried out to assess the proposal's performance against other approaches. Finally, Section 8 concludes the paper.

2 Related Work

Fog-aware mechanisms to implement efficient resource management for applications with different levels of latency requirements is indeed a challenging task [5]. Several efforts have addressed this body of work to provide a latency-driven fog computing environment for IoT applications using one fog layer *only* between the edge and the cloud (as illustrated the fog-cloud topology in Fig. 1) [3], [4], [8], [9], [10], [14], [15]. Effective mechanisms addressing stacked fog layers (also known as *cloudlet*, *micro data centers*, or *fog nodes*) is still an open issue [2].

An architecture for mobile computing to place computing and storage nodes at the Internet's edge close to users is proposed by Satyanarayanan *et al.* in [10]. The new vision of mobile computing in mid-2009 has endeavored to free mobile devices from a severe limitation of computing capabilities, allowing resource-hungry applications to use cloud-like computing service that would be free of delay, instability, congestion and WAN failures. Since then, Satyanarayanan *et al.* have been advancing this body of work to emphasize the importance of having cloudlets at the network edge to emerge the industry investment and research interest in edge computing [4]. Still, none of them has discussed the importance of deploying multi-layers of fog nodes between the edge and the cloud to manage the requirements of applications with heterogeneous tolerance levels of latency.

To emphasize the effectiveness of having one layer of cloudlets between the edge and the cloud, Bittencourt *et al.* have introduced in [3] three simple resources allocation policies in fog computing, namely *First-Come First-Served (FCFS)*, *Delay-priority* and *Concurrent*. According to the authors, even bringing the cloud closer to users, the potential overloads in cloudlets upon incorrect policy enforcement can negatively

impact applications' quality of service. The authors emphasized that robust allocation strategies in fog computing-based systems are also needed for mobility-sensitive applications on top of latency-driven resource management.

Charântola *et al.* proposed in [8] a scheduling algorithm for a fog-cloud environment considering only a single layer of cloudlets. The authors assume each application as a set of modules (components) that communicate with each other to complete a job. As expected, modules that are far away from each other introduce delays in the job completion time. The authors proposed an algorithm to allocate/schedule resources on the fog-cloud infrastructure according to the application delay requirement to mitigate this issue. In this paper, however, we introduce an efficient scheduling algorithm for real-time applications, such as online games and video tracking surveillance, for multi-tiered cloudlet infrastructures.

Shah-Mansouri *et al.* studied in [9] the problem of resource allocation for IoT application in a hierarchical computing paradigm, including one layer of fog nodes and the remote computing services from the cloud. Due to the fog nodes' limited computing capabilities to satisfy all the resource requests from the IoT applications, they addressed the problem using a competitive game approach to allocate resources efficiently. Munir *et al.* proposed in [16] a fog-cloud computing architecture for IoT applications. They proposed an energy-efficient reconfigurable architecture that can adapt to the workload performed at a given time. The authors emphasized the benefits of using their proposal architecture for future IoT industry sectors, such as the intelligent transport system, when merging the benefits of fog and cloud computing paradigms.

In [17], Ali *et al.* present an NGSA-II adaptation to fog-cloud environments, where a centralized decision-maker acts to optimize tasks allocation throughout the distributed system. Similarly, Aburukba [15] models a centralized optimization using Genetic Algorithms for IoT applications, focusing on minimizing latencies. In this work, we propose the decision-making to occur in a distributed fashion, with local decisions focused on the communication weights of tasks in an application loop, which improves scalability but, on the other hand, neglects global optimization objectives. Both approaches can be complementary in a real-world scenario if they are designed to cooperate in a large fog-cloud scenario.

Although the three works mentioned above advance the adoption of fog computing for latency-sensitive IoT applications, they only consider a scenario including one layer of cloudlet between the edge of the network and the cloud. Still, there is a gap of research in this area – which this paper attempts to cope with – to consider *several* layers of cloudlet nodes between the edge and the cloud, composing the fog-cloud hierarchy illustrated in Fig. 1. In doing so, resource allocation management becomes a task even harder to solve when handling resource allocation requests from applications with different latency requirements. In this paper, we proposed that workloads are smartly moved towards the cloud to satisfy the latency requirements as the delay measurements become higher.

Recently, Souza *et al.* attempted to embrace this research gap in [14] by proposing an architecture consisting of a dual-cloudlet layer in a fog-cloud computing environment for IoT applications. The authors aimed to diminish the necessity of demanding further cloud resources preventing the applications to face high delays on the resource access from the cloud. Still, their proposed mechanism cannot cope with latency-sensitive applications with different priority for delays, as we address in this paper. Moreover, the study proposed in this paper can handle as many levels of cloudlet as the fog-cloud hierarchy presents.

3 Fog Computing Key Concepts

3.1 Fog-Cloud Hierarchy Topology

The Fog Computing concept emerged to introduce small data centers, or cloudlets, closer to the user at the network edge [2], [18], [19]. It attempts to dodge the high latency in the communication channels between the edge and the cloud while still supplying cloud-like features close to edge devices. The Fog Computing architecture works as a "glue" that fastens the cloud closer to the edge by placing cloudlets (micro data centers) at the network providers' access points (which are utilized by edge devices to connect to the Internet). This model can be adapted to consider some levels of computing capacity nodes (cloudlets) placed between the edge and the cloud, composing a fog-cloud hierarchy of several levels [8]. Multi-level Fog computing is a complementary of the traditional architecture (which contains one cloudlet level) to enhance the performance services, reducing delay, energy consumption and network usage in IoT environment [20]. The higher a fog node (cloudlet) is in the network hierarchy, the larger its computing capacity is, once it should satisfy a broader set of users downwards the hierarchy. The lower a fog node is in this hierarchy, the closer it is to the network edge, hence presenting lower communication delays to devices [8]. Fig. 1 illustrated an example of a fog-cloud hierarchy. The key features here include heterogeneity support, geographical distribution, location awareness, ultra-low latency, support of real-time and large-scale IoT applications.

The computing hierarchy in the fog-cloud infrastructure can offer a broader range of service levels for IoT applications that cannot be supported only by systems comprising cloud computing. While fog computing provides reduced latencies for latency-sensitive applications running at the network edge, it also avoids pushing traffic (upwards in the hierarchy) for processing at the cloud, hence reducing traffic congestion in the network core [2]. However, the synergy between IoT, cloud, and fog comes at a price: more complex and sophisticated resource management and scheduling mechanisms are required to make the whole system work properly [3]. Hence, new challenges to be overcome arises, e.g., dynamically deciding in which level of the fog hierarchy a workload should be offloaded to the cloud for processed to meet its quality of service requirements.

This paper contributes to advance research in this topic by proposing resource allocation policies for this highly distributed computing hierarchy. The Fog hierarchy is organized according to the number of micro data centers levels (cloudlets or fog nodes) amongst the edge and the cloud. The hierarchies considered in this paper are as follows:

- *Single-Level fog-cloud topology*: A single-level topology comprises one cloudlet layer between the edge and

the cloud for offloading purposes. By using this traditional fog computing layout, if the cloudlet is not sufficient to cope a new workload due to the lack of resource availability, for instance, the workload is transferred for processing directly at the cloud layer – where is assumed to have infinite computing resources available.

- *Multi-Level fog-cloud topology*: In a multilevel topology, the fog-cloud infrastructure layout comprises several layers of cloudlets between the edge and the cloud. In this setup, if a cloudlet is almost overloaded or it is operating at maximum capacity, incoming workloads are assigned for processing to the next cloudlet layer above in the hierarchy. This process repeats upwards until the workloads achieve the last layer, i.e., the cloud layer.

In a multi-level hierarchy, the scheduler needs to be distributed in the hierarchy setup [21]. Therefore, in the scheduling algorithms studied in this paper, a scheduler's decision is local, based on the current load of the cloudlet. If overloaded, the services components can be forwarded to the next level in the hierarchy. In this way, load state synchronization among cloudlets at different levels is not necessary, improving scalability.

3.2 Resource Allocation Policies for Fog Computing

The applications running in a fog-cloud environment pose a challenge in coordinating the resources allocated and allocating the resources for incoming workloads. As data in IoT-Fog-Cloud infrastructures must traverse one or more cloudlet tiers, policies are introduced to describe rules and protocols on how this resource allocation problem must be addressed in this type of distributed system running highly heterogeneous and dynamic applications. To that end, a resource allocation policy must accommodate incoming applications considering busy cloudlets while satisfying each application request's requirements concerning networking delays. The *iFogSim* [13] – used later in the evaluation of this paper – is a well-known simulator for modeling Internet of things, Edge and Fog Computing environments. iFogSim is developed on the top of the CloudSim simulator [22], which is a largely adopted tool for modeling resources management approaches in Cloud Computing environments. Extending CloudSim, iFogSim allows the use of different scheduling policies in a fog-cloud scenario. In this paper, we consider the following policies: *mapping*, *cloud only*, and *edgewards*. These policies work as follows:

- *Mapping*: The mapping strategy allocates (maps) application workloads to a specific cloudlet in the hierarchy without measuring to see if sufficient computational resources will be available to meet the request's requirements. As a result, if the cloudlet becomes busy running previously allocated workloads, this policy does not re-schedule workloads for executions on upward cloudlets in the fog-cloud layout. In sequential dispatch, the mapped requests may have either to wait for a long time until sufficient computational power becomes available to satisfy the workload computation requirements. Or, in the case of concurrent dispatch, the mapped requests have their turnaround times profoundly impacted by concurrency as the overloaded computing resources are failing to provide enough computing power for every workload to run smoothly.
- *Cloud Only*: The cloud-only placement strategy is based on the traditional cloud-based implementation of applications. All workloads of an application run in remote cloud data centers – the higher layer in the fog-cloud hierarchy. This strategy is useful in a scenario where resources are not available in any cloudlet layer, and inbound applications workloads need to be offloaded to the cloud for further processing. As expected, this approach has the disadvantage of high latency in communication between the edge and the cloud since the cloud is – networking speaking – further away from the edge than the resources of intermediate cloudlets in the fog-cloud hierarchy.
- *Edgewards*: The edge-ward strategy attempts to allocate resources for incoming application requests on cloudlets as close as possible to the network's edge. However, if the cloudlet is not computationally powerful enough to satisfy the request's requirements, either because it is overloaded or does not have enough computing capacity at all, the policy iterates from the bottom towards the top in the fog-cloud hierarchy to find enough available resource in a cloudlet for allocation. In the worst-case scenario, all cloudlets are fully occupied, and this policy will have the same result as *cloud only*.

4 System and Application Models

In order to demonstrate how scheduling latency-critical services can be performed in a multi-layer fog considering the location and different application classes, we have used two types of services from [13]: a *delay-tolerant video surveillance network* and a *latency-sensitive online game*. We briefly present how these two services are modeled and their importance in the context of our experimentation.

4.1 Video Surveillance Object Tracking – VSOT

Resource availability is a stringent requirement to process a massive amount of data stream from multi-cameras over a long period of time. These resources are fundamental to store and process all data to, for example, track objects. Tracking analysis includes finding irregular events and notifying users when such activities are detected.

Data generated by a smart video surveillance environment software often comes from multiple cameras, which produce large quantities of a data stream in short intervals. These large amounts of data are sent to a fog-cloud based system for detecting and tracking objects. The process of tracing a moving object is initiated by calculating the proper horizontal and vertical position of the camera, including image details regarding the zoom and brightness intensities for more effective coverage of the tracked objects.

Fig. 2 depicts the application model used in this paper in our VSOT environment. Table 1 shows the VSOT software comprising five components, while Table 2 presents the five types of VSOT state transitions between them. First, for instance, the camera provides the raw video data to the

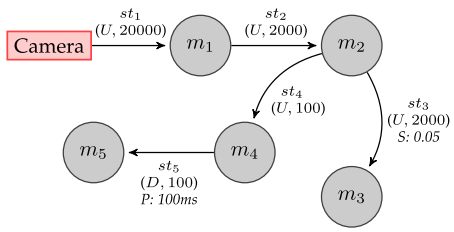


Fig. 2. Diagram of the components and edges representing the communication of the VSOT app – Figure adapted from [13].

motion detection component m_1, which is embedded within the camera device. When a motion is detected, the motion detection component m_1 sends the video data stream to the object detection component m_2. Next, m_2 compares the current object with data previously captured. If this object is not already being traced previously, m_2 triggers a trace activation for the current object, providing an initial calculation of that object's positioning coordinates in the image. Then, the object is sent to the object tracking component m_4 that uses the created coordinates and adjusts the camera's pan control parameters at the component m_5, which periodically keeps updating the parameters of the object tracing data is updated by m_4. Finally, the detected objects are sent to the user interface m_3 using fractions of the video where the object is present.

Each arc in the Fig. 2 is represented by a direction (U: Upload or D: Download) and an amount of data (bytes) transmitted throughout components. Arc labels may also include two compulsory additional parameters: *selectivity* (S) and *periodicity* (P). For example, the arc st_3 shows an action of uploading (U) from m_2 to m_3 with 2000 bytes. It also presents a selectivity of 0.05, indicating the proportion of st_3 generation concerning the total of edges st_2 is inputting in the component $m2$; therefore, the amount of data arriving at m_3 is 100. Moreover, in this example, st_5 is the only arc periodically generated (exist) with a $100ms$ interval, representing the periodic download act from m_4 to m_5. Arcs containing the parameter P are emitted at regular intervals. For more details of the VSOT app, please see [8].

4.2 Electroencephalography Tractor Beam Game

Interactive gaming presents latency requirements that can be fulfilled by computing at the edge. For example, Deutsche Telekom is preparing an Edge-native gaming experiences [23], showing the applicability of edge in such scenarios.

Electroencephalography Tractor Beam Game (EEGTBG) is a latency-critical application where the player wears a special headset connected to her/his smartphone. The objective of this game is to gather some items by concentrating on them. The application running on a user's smartphone processes the EEG signals produced by the headset. The player who has a better concentration score on an item can grab it to herself/himself. The more concentrated the players are, the more things they will attract to themselves. Real-time processing and ultra-low delays are the desired specifications for this game to work accurately, and therefore they are the crucial application requirements.

The five components of EEGTBG are shown in Fig. 3. Table 3 shows all the EEGTBG components, while Table 4 presents the EEGTBG state transitions between them. The first EEGTBG's component is the EEG Sensor m_1. This component is responsible for sending the concentration data from the user headset to the Client component m_2, which is running on the user phone. After the m_2 has received the concentration data, it checks for any inconsistencies. It sends the data to the Concentration Calculator component m_4 (running outside the user's phone) to obtain the player concentration level score. The concentration level score is calculated based on signals collected from the client headset. After that, m_2 gets the concentration level results from other players by requesting the data from the Coordinator component m_5. All these data

TABLE 2
VSOT State Transitions

Arc	Action Name
st_1	Raw Video Stream
st_2	Motion Video Stream
st_3	Detected Object
st_4	Object Location
st_5	PTZ Parameters

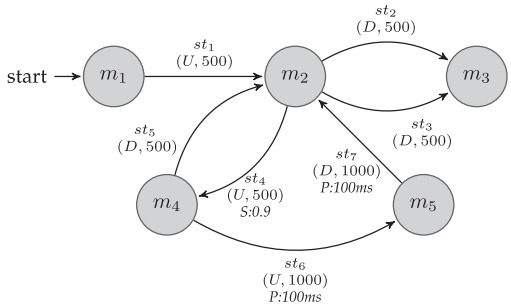


Fig. 3. Diagram of the components and edges representing the communication of the EEGTBG app – Figure adapted from [8].

TABLE 1
VSOT Components

Index	Component Name
m_1	Motion Detector
m_2	Object Detector
m_3	User Interface
m_4	Object Tracker
m_5	PTZ Control

TABLE 3
EEGTBG Components

Index	component Name
m_1	EEG Sensor
m_2	Client
m_3	Display Actuator
m_4	Concentration Calculator
m_5	Coordinator

TABLE 4
EEGTBG State Transitions

Arc	Action
st_1	EEG
st_2	Self state update
st_3	Global state update
st_4	Sensor
st_5	Concentration
st_6	Player game state
st_7	Global game state

(local and global players) are transmitted to Display Actuator component m_3 that displays the game's current state.

The arcs with their respective directions (Up or Down) and the number of transmitted bytes are represented in Fig. 3. For instance, the arc st_4 has a selectivity of 0.9, while arcs st_6 and st_7 are periodically generated with a time interval of $100\ ms$. For more details of the EEGTBG app, please see [13].

5 Scheduling Mechanisms

Let $D = \{d_1, d_2, \ldots\}$ be a set of edge devices connected to access points (e.g., WiFi or LTE/5G) of the fog-cloud hierarchy infrastructure. Also, let $A = \{a_1, a_2, \ldots\}$ be a set of applications running in the fog system by those devices. As discussed in Section 4, each application of A is represented by a directed graph $G_{a_i} = (V_{a_i}, E_{a_i})$, where V_{a_i} comprises the set of the app a_i's components, and E_{a_i} depicts the set of directed arcs connecting them. The fog-cloud hierarchy is represented by the overall set of computing resources $R = \{r_1^1, r_2^1, \ldots, r_i^j, \ldots, r_{n-1}^k, r_n^k\}$, with $1 \leq i \leq n$ and $1 \leq j \leq k$, where n and k are the indices representing, respectively, the unique identification number for resources and cloudlets. Each resource r_i^j of R has a connectivity vector B representing the available bandwidth with every resources in the fog scenario; i.e., $r_a^j.B[r_b^k] \geq 0$ and $r_a^j B.[r_a^j] = \infty, \forall r_a^j, r_b^k \in R$. If r_a^j and r_b^k are unreachable, then $r_a^j.B[r_b^k] = r_b^k.B[r_a^j] = 0$. All model parameters are shown in Table 5.

5.1 Previous Work

We consider two decision-making processes from our previous work [3], namely *prioritization policy* and *component allocation policy*. The former focuses on choosing which application(s) have the highest priority to stay closer to the edge. This policy is modeled using a simple ranking mechanism, as detailed in [3]. On the other hand, the latter concerns choosing which application components should be moved upwards in the fog-cloud hierarchy when a resource cannot handle all incoming requests. These two decision-making rules are then individually combined to compose the scheduling approaches (discussed later in this section), which is used to distribute the tasks over the fog-cloud infrastructure. We briefly describe the details of the prioritization & component allocation policies before diving into the scheduling approaches.

TABLE 5
Table of the Model Parameters

Devices Parameters	Description
$D = \{d_1, d_2, \ldots\}$	Set of devices connected to access points
Application Parameters	**Description**
$A = \{a_1, a_2, \ldots, a_k\}$	Set of applications deployed by the devices
$G_{a_i} = (V_{a_i}, E_{a_i})$	Graph representing the application a_i
V_{a_i}	Set of the app a_i's components
E_{a_i}	Set of directed arcs connecting a_i's components
Fog-cloud parameters	**Description**
$R = \{r_1^1, \ldots, r_i^j, \ldots, r_n^k\}$	Set of computing resources in the hierarchy
n	Resource id
k	Cloudlet id
$r_i^j.B$	Associated Bandwidth Vector

5.1.1 Prioritization Policy

The prioritization concentrates on determining which application(s) have the highest priority to stay closer to the edge, taking advantage of having low latency with the user and offer a quality of service accordingly. Three prioritization flavors are considered for this policy. They are as follows:

- *Concurrent*: This policy is actually a no-prioritization scheme. Application requests that arrive at a cloudlet are simply allocated to such a cloudlet without checking its capacity or current usage upon application computational demand request.
- *First Come-First Served (FCFS)*: Until there are no more computing resources available in the cloudlet, application requests are served in the order of their arrival. When the cloudlet becomes full, i.e., the remaining CPU capacity is smaller than application requirements), applications requests are moved upwards in the cloud-fog hierarchy. The upwards movement follows another policy, as we describe in Section 5.1.2.
- *Delay Priority*: An application requiring lower delay are prioritized and scheduled first at the (lowest) cloudlet level if it is not full. The next class of application requests is scheduled in the cloudlet until there is a resource available. Otherwise, the application is moved upwards according to a policy chosen for this action, as we described in Section 5.1.2.

5.1.2 Component Allocation Policy

The component allocation policy decides how the application components are moved up in the fog-cloud hierarchy. Derived from [13], two allocation approaches are considered for this policy in this paper:

- *Edgewards*: This policy then iterates on the hierarchy towards the cloud and tries to place the application component on alternative cloudlets, reaching the cloud if none cloudlet is available. The application components that have data dependency from the

component that is supposed to be migrated are also moved upwards in the hierarchy toward the cloud. It means that all the components are grouped and moved up to reduce the delay communication among them. Previously allocated application instances are also migrated towards the cloud. This action collaborates to free up space in the cloudlets for future allocation requests.

- *Individual*: Unlike Edgewards, which moves all application components together upwards in the hierarchy, this policy focuses on choosing individual components to move upwards. Thus, without having the overhead of migrating all application components, this policy can offer low delays for real-time applications that seek a better quality of service.

5.1.3 Scheduling Mechanisms

In our previous works [3], [8], we combined the prioritization and allocation policies to yield the scheduling mechanisms. Four were developed and are described below:

- *FCFS & Edgewards (FCFS-E)*: This mechanism combines the FCFS prioritization policy with the edgewards component allocation policy. To this end, the first application to arrive at the cloudlet has priority to remain at the cloudlet, and when incoming application arrives and not enough free resources are available, all components of previously allocated application instances are grouped and moved upwards.
- *FCFS & Individual (FCFS-I)*: Similarly to the previous mechanism, but in this case, application components are moved upwards individually.
- *Delay-Priority & Edgewards (DP-E)*: In this scheduling mechanism, when there are not enough resources available for the incoming application, the application components with the lowest delay priority are grouped and moved up together in the hierarchy.
- *Delay-Priority & Individual (DP-I)*: Similar to the above mechanism; but, components of the lowest delay-priority application are moved individually.

In the next section, we advance in the discussion of scheduling strategies. We describe the proposed scheduling mechanism to perform the decision-making on which application components run on which computing resources, considering the application computing requirements, the resources capacity, and the data transfer between application components.

5.2 Proposed Communication-Based Policy

In this paper, we are concerned with allocating resources from the set of resources R to applications' components within the applications A. Our previous policies work well in many scenarios, albeit real-time applications requiring low delays may produce a smaller or unacceptable quality of service for users of that application. The Delay Priority Strategies (DP-E and DP-I) we created in [8] have successfully reduced delays. Still, these strategies have increased the total network for applications that are network demanding. For instance, in our previous experiments, the VSOT components are always moved to the cloud to prioritize the EEGBTG's components.

To contour this issue, we propose in this paper a different policy to move components upwards in the fog-cloud infrastructure, focusing on the amount of data present in the component's dependencies. Aiming to reduce the network impact on the application's delay, the proposed communication-aware scheduling policy, called *Communication Based & Edgewards (CB-E)* (Algorithm 1), generates all possible sets of components that could be moved upwards in the hierarchy (Algorithm 2) before triggering the migration of the lowest impact set (Algorithm 3).

Algorithm 1. Communication Based & Edgewards (CB-E)

Require: Cloudlet c
Ensure: Set of modules M moved upwards
1: $S =$ Component sets in c (Algorithm 2).
2: **for all** component set $M \in S$ **do**
3: $Cost_M = 0$
4: **for all** component $m \in M$ **do**
5: $Cost_M = Cost_M +$ impact of M (Algorithm 3).
6: **end for**
7: **end for**
8: Move M upwards whose $Cost_M$ is minimum.

Algorithm 2. Component Sets Generation

Require: Cloudlet c
Ensure: Component sets S
1: $S = \emptyset$
2: **for all** component $m_s \in C$ **do**
3: **for all** component $m_t \in c$ **do**
4: **if** $\notin$ Up arc from m_s to m_t **then**
5: $M = \{m_s\}$
6: $S = S \cup \{M\}$
7: **end if**
8: **end for**
9: **end for**
10: $S_{old} = \emptyset$
11: **while** $S_{old} != S$ **do**
12: $S_{old} = S$
13: **for all** set $M_s \in S$ **do**
14: **for all** component $m_s \in M_s$ **do**
15: **for all** m_s incoming edge e **do**
16: $M_{m_e} = \{m_s\} \cup \{source(e)\}$
17: $S = S \cup M_{m_e}$
18: **end for**
19: **end for**
20: **end for**
21: **end while**

The proposed algorithm starts when a new application request arrives, which is initially assigned to the cloudlet at its access point. Note that, at this point, other existing applications are already running at the cloudlet and with components potentially running at other levels of the fog-cloud hierarchy. The proposed CB-E scheduling policy starts by generating all component sets that could be moved upwards (Algorithm 2). Starting from the cloudlet c at the lowest level, it creates a collection of sets S containing a singleton for each component that is not the source of an arc to

another component in the same cloudlet c (lines 2-10 in Algorithm 2). Next, for each set added to S, the algorithm searches for source components of edges which target is a component in that set (lines 13-19). Then, for each component that satisfies this condition, a new set is created with this new component and the component from that set. This process is iterated until no new set is created.

With the component set created by Algorithm 2, it is necessary to compute the data transfers each component set generates, which is used to compare the impact each set could have on the overall network. The communication impact I_e of an edge e is defined in Equation (1) as follows:

$$I_e = b \times s \times \frac{T_a}{p}, \tag{1}$$

where b is the amount of bytes transferred from the edge source to the target component, s is the edge selectivity (how much data is sent out through the edge based on the data from input edges), T_a is the periodicity at which the application transmits data, and p is the periodicity of edge e. The communication impact of a component m is computed by Algorithm 3, which utilizes the sets computed by Algorithm 1 in conjunction with Equation (1) to determine the communication impact of each component in a component set.

Algorithm 3. Communication Impact for a Component Set m

Require: Component set M; $m \in M$; application arcs E.
Ensure: $total\_impact_m$
1: $total\_impact_m = 0$
2: **for all** application edge $e \in E$ **do**
3: $I_e = b \times s \times \frac{T_a}{p}$ – Equation (1)
4: **if** m is the source of e **then**
5: **if** No upward device has the target component of e **then**
6: **if** target component of $e \notin M$ **then**
7: $total\_impact_m = total\_impact + I_e$
8: **end if**
9: **else**
10: $total\_impact_m = total\_impact - I_e$
11: **end if**
12: **end if**
13: **if** m is the target of e **then**
14: **if** No upward device has the source component of e **then**
15: **if** source component of $e \notin M$ **then**
16: $total\_impact_m = total\_impact + I_e$
17: **end if**
18: **else**
19: $total\_impact_m = total\_impact - I_e$
20: **end if**
21: **end if**
22: **end for**

5.3 Toy Example

To demonstrate the entire process described for the proposed communication-based & Edgewards (CB-E) scheduling policy, we illustrate a toy application example in Fig. 4 comprised of 4 application components, m_1, m_2, m_3, m_4, running in three different locations (device, cloudlet, and cloud), as shown in Table 6. Assume the cloudlet where the components m_2 and m_3 are running is overloaded, the proposed CB-E policy tries to relocate them upwards in the hierarchy. Edge labels describe (i) the amount of data to be transferred between components, (ii) data transfer indicator (U for up and D for down in the hierarchy), (iii) respective edge selectivity, and (iv) a standard periodicity of 10ms for application edges.

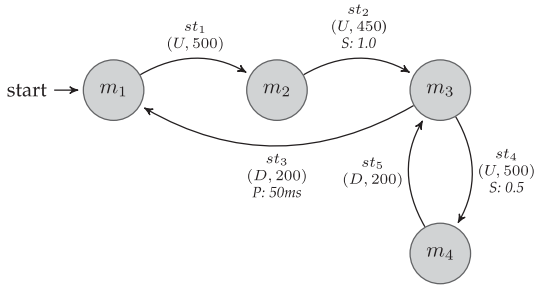


Fig. 4. Toy example application with four component nodes and five communication edges.

According to Algorithm 2, the proposed CB-E policy attempts to move the set of components that generate less impact in the application loop (i.e., $(\text{raw data})\, m_1 \rightarrow m_2 \rightarrow m_3 \rightarrow m_4 \rightarrow m_3 \rightarrow m_1\, (\text{processed data})$) and in the overall network bandwidth. As the cloud is traditionally further away from the cloudlet, the proposed policy should avoid migrating components to the cloud. Hence, a singleton (M) comprising m_3 is initially created, since m_3 is not a source of an arc to another component in the same location (cloudlet, and m_4 is in the cloud – see Table 6), and meanwhile component m_2 has an upward edge to m_3 and both are running in the same location (cloudlet), thus m_2 has no singleton created for itself. Next, in the loop iteration between lines 13-19), a new set $\{m_3, m_2\}$ is created, since m_2 is the source of an edge targeting m_3. Finally, in the third iteration, no set is created, resulting in the final set $S = S \cup \{M\} = \{\{m_3\}, \{m_3, m_2\}\}$. After Algorithm 2 finalizes, Algorithm 3 is called. For the set $M_1 = \{m_3\}$:

- e_3 will not transfer data to the cloud, thus its impact $(-500)(0.5) = -250$.
- e_4 will not transfer data from the cloud, thus $(-200)(1) = -200$.
- e_2 will start transfer data to the cloud, thus $(450)(1) = 450$.
- e_5 will transfer data from the cloud, thus $(200)(2)(\frac{10}{50}) = 40$.

This results in a communication impact of $(-250) + (-200) + 450 + 40 = 40$ if the set M_1 is moved to the cloud. For the set $M_2 = \{m_3, m_2\}$, the impact is calculated as follows:

TABLE 6
Toy Example Component

Index	Component Name	Location
m_1	Component 1	Device
m_2	Component 2	Cloudlet
m_3	Component 3	Cloudlet
m_4	Component 4	Cloud

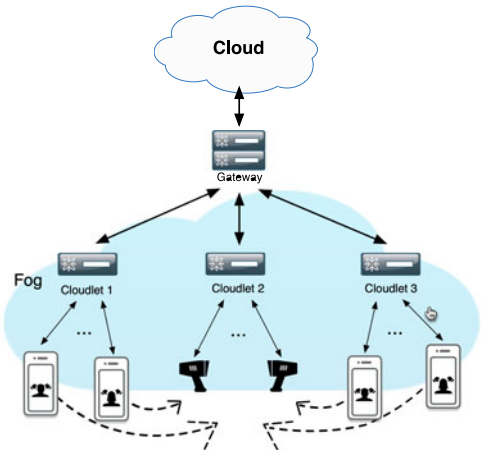


Fig. 5. Mobility scenario: A total of 12 EEGTBG devices with 2 game instances each move towards Cloudlet 2, where 8 cameras (VSOT) are running.

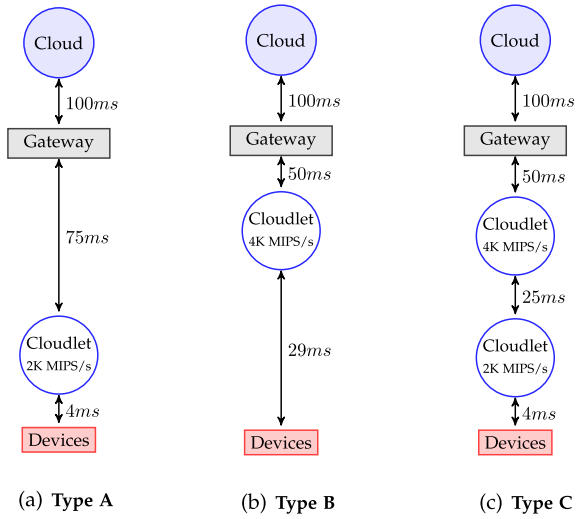


Fig. 6. Fog-cloud hierarchical types.

- e_3 will not transfer data to the cloud, thus $(-500)(0.5) = -250$.
- e_4 will not transfer data from the cloud, thus $(-200)(1) = -200$.
- e_1 will transfer data to the cloud, thus $(500)(1) = 500$.
- e_5 will transfer data from the cloud, thus $(200)(2)(\frac{10}{50}) = 40$.

Thus, the communication impact for $M_2 = 90$. In this toy example, the set which results in the lowest impact amount of transmitting data to/from the cloud is $M_1 = \{m_3\}$.

6 Evaluation Setup

Different module allocation policies were discussed in this paper, considering the execution of applications in different fog computing topologies. We have analyzed the impact on the network consumption and runtime of the proposed scheduling procedures. The key to the experimental evaluation is the use of many hierarchical levels of cloudlets to run the VSOT and EEGBTG applications. We use the iFogSim simulator [13] to carry out this evaluation. iFogSim provides only one level of cloudlets by applying the default configuration. To analyze the impact of introducing more hierarchical levels of cloudlets, we have created extra levels of cloudlets in iFogSim to measure the application's network usage and delays using the proposed scheduling mechanisms. This evaluation aims to take full advantage of cloudlets capacity and keep more applications' components closer to the user instead of pushing them to the cloud. The network and computing resources were modeled according to the CloudSim and iFogSim primitives, reflecting applications with data dependencies and computing resources that can become overloaded when concurrency occurs in the discrete event simulator. Even though iFogSim does not natively introduce mobility simulation in real-world maps, its functionalities allowed us to evaluate the allocation policies when users leave one cloudlet and connect to another. More complex mobility patterns can be studied using an extension called MobFogSim [24].

It is also important to highlight that our proposed scheduler is *ready* to embrace multiple levels of cloudlets in the cloud-fog hierarchy. In the simulation carried out in this paper, we have used a scenario comprising three levels (two layers of cloudlets and one cloud layer) to expose the efficacy and robustness of our work. Further studies comprising more cloudlet layers are left as future work.

The evaluated mobility scenario is illustrated in Fig. 5. There are 4 VSOT users with 2 VSOT applications each, consisting of 8 VSOT applications running in only one cloudlet. Moreover, the scenario also contains 12 users (devices) running the EEGTBG application. Each EEGTBG user has 2 instances of the EEGBTG game application running in his/her device, totalizing 24 EEGTBG instances. The EEGTBG users are initialized at the lowest level of cloudlets, and then they are moved to Cloudlet 2. Fig. 5 illustrates only one cloudlet level (i.e., Type A or B), but the experimental evaluation was conducted using three different fog topologies, as illustrated in Fig. 6.

The hierarchical level type A shown in Fig. 6a presents a single cloudlet with low processing capacity and close to the user. It has a processing capacity of 2,000 MIPS (millions of instructions per second) and was connected to a proxy (service function) through a network link with 10,000 Kbps of bandwidth and 75 milliseconds of latency. The devices were connected to the cloudlet through a wireless network link with 10,000 Kbps of bandwidth and four milliseconds of latency. The hierarchical level type B depicted in Fig. 6b has a cloudlet connected to proxy service function through a network link with 10,000 Kbps of bandwidth and 50 milliseconds of latency. It has twice the processing power of A, but it is located farther from the end-users, with 29 milliseconds of latency in communication with the devices. Lastly, the hierarchical level type C illustrated in Fig. 6c brings together the benefits of the previous A and B scenarios; that is, one cloudlet with low processing capacity and close to users, and another cloudlet slight farther from the network edge (mobile devices) with more processing power and closer to the cloud.

The proposed algorithms manage the VSOT and EEGTBG applications' component placement at the different fog-cloud levels. It worthwhile to highlight that the VSOT application is not delay-sensitive, but it is network demanding

TABLE 7
Algorithm's Acronym

Acronym	Definition
Concurrent	Concurrent execution [3]
FCFS-E	First-Come First Served & Edgewards from [3]
DP-E	Delay-Priority & Edgewards from [3]
FCFS-I	First-Come First Served & Individual from [3]
DP-I	Delay-Priority & Individual from [3]
CB-E	**Proposed Communication Based & Edgewards**

(i.e., a bandwidth-hungry application). Meanwhile, the EEGTBG application is a real-time game, which means that it is delay-sensitive. For both applications, the *loop delay* consists of the round-trip time for the execution of all components until displaying the results on the devices' screen.

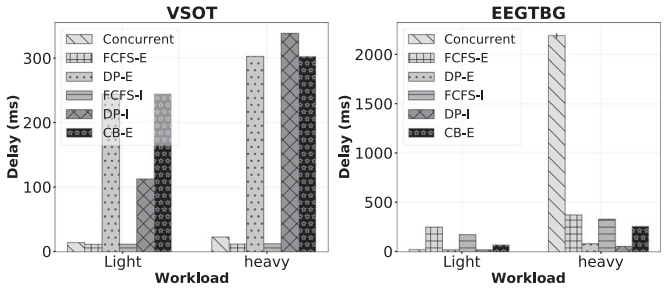
Based on the scenario shown in Fig. 5, we have deployed the simulation setups shown in Figs. 6a, 6b and 6c with eight instances of the VSOT application and twenty four instances of the EEGTBG application. The number of instances was selected to create a bottleneck due to users' mobility in terms of light and heavy workloads in the cloudlet. After the simulation has started, we move the EEGTBG players, one by one, from cloudlets 1 and 3 to cloudlet 2 as an act of emulating the user mobility behavior. By doing so, we attempted to assess any quality of service degradation resulting from poor resource allocation. Since EEGTBG has low-latency requirements, we hypothesize that a player having its instance in a cloudlet will only play against players whose EEGTBG instance would be allocated in the same cloudlet.

Note that the actual CPU time consumed by each application component is not precisely known at the scheduling stage because the scheduling decision-making process occurs before runtime. When scheduling, strategies must verify that the Cloudlet (at each Cloudlet level is the case with Fig. 6c) has sufficient free CPU capacity to handle each application module. At runtime phase, each application uses at time t_i the CPU capacity C_i, which depends on the interaction between modules – for example, how much data it receives or sends to other modules. These estimations come from the application description (see Section 4), which models the application as a directed graph and its attributes, as commonly found in the scheduling literature [13], [25].

7 Results

We shall discuss in this section a comprehensive set of experiments to validate our proposed communication-aware scheduling policies for a fog-cloud computing environment: *CommunicationBased & Edgewards (CB-E)*. The experiment scenarios were set up using the parameters depicted in Figs. 6a, 6b, and 6c. Each simulation was repeated ten times to calculate the average time for both metrics delay (in ms) and total network usage (in KB), guaranteeing a statistically accurate result. Data generated by the applications at the edge are transferred in the network following a normal distribution, and the results presented in the graphs show a 95 percent confidence interval.

We have set up the scenarios using two distinct workload types (one light and one heavy) to represent the user demand in terms of the number of applications per cloudlet.



(a) VSOT: Hierarchical Type A (b) EEGTBG: Hierarchical Type A

Fig. 7. Application loop delays in Hierarchy A according to the scheduling algorithms and the workload type.

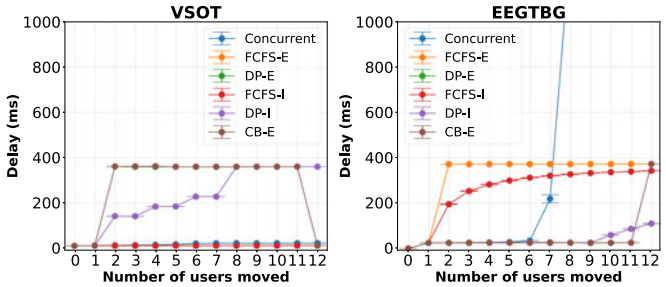
The light workload configuration consists of requests generated by four instances of the VSOT applications and up to six instances of the EEGTBG applications. The heavy workload configuration consists of requests generated by four instances of VSOT and up to twelve instances of EEGTBG.

As a reminder, the Table 7 shows the acronym and the meaning of each algorithm used in this set of experiments.

7.1 Hierarchical Level Type A

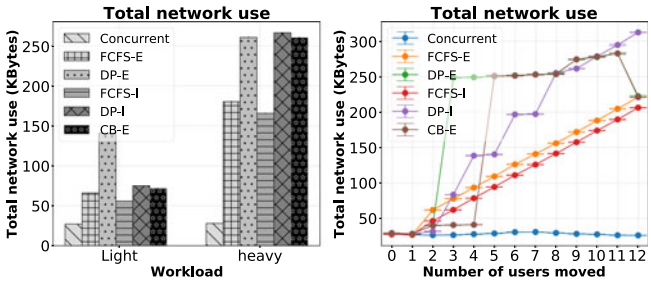
Fig. 7 shows an analysis of the loop delays on the VSOT and EEGTBG applications for the six different scheduling mechanisms stated in Table 7. According to results using both light and heavy workload shown in Fig. 7a, the scheduling approaches DP-E, DP-I, and CB-E have higher average delays than other scheduling mechanisms. Although this slight increase in delay does not impact the VSOT application, they have reduced the delay for the EEGTBG applications, as Fig. 7b illustrates. The reduction on the delay caused by DP-E, DP-I, and CB-E occur due to the prioritization-aware scheme for EEGTBG application's components, which is a delay-sensitive application. Another aspect about EEGTBG is related to the naïve concurrent policy, which does not contain any prioritization design. It negatively impacts the use of resources at the network edge, causing massive delays into the EEGTBG application loop, especially when EEGTBG faces a heavy workload.

Fig. 8 shows the loop delays on the VSOT and EEGBTG application using all the six scheduling mechanisms according to the number of users that have moved in the fog-cloud hierarchy. When only one EEGTBG player is moved to the cloudlet, all six scheduling mechanisms have the same delay results for both applications. However, when the second EEGTBG player is moved to the cloudlet, the scheduling mechanisms start to act on the application's instances



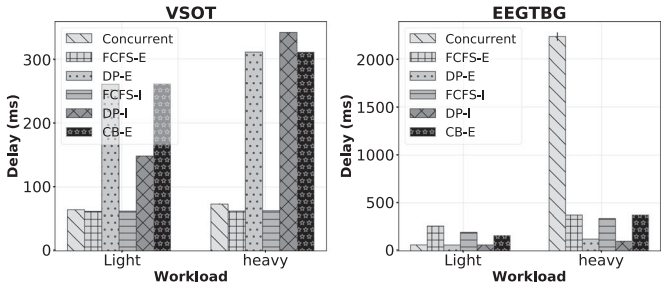
(a) VSOT: Hierarchical Type A (b) EEGTBG: Hierarchical Type A

Fig. 8. Application loop delays according to the scheduling algorithms.



(a) Network Usage per Workload (b) Network Usage per Number of users moved

Fig. 9. Network usage according to the scheduling algorithm in hierarchical type A.



(a) VSOT - DELAY - B (b) EEGTBG - DELAY - B

Fig. 10. Application loop delays in Hierarchy B according to the scheduling algorithms and the workload type.

differently. In order to maintain the delays as lower as possible for EEGTBG, DP-E, DP-I, and CB-E prioritize the EEGTBG application request arrival, (Fig. 8b), moving the VSOT components to the cloud. As a result, it increases the VSOT loop delay (Figs. 8a) to an acceptable level of about 360 milliseconds for DP-E and CB-E, and about 150 milliseconds for DP-I. However, CB-E reduces the VSOT delay after 11 users are moved. Although EEGTBG application has the highest priority to be closer to the edge, cloudlet #1 enters an overloaded state (after 11 users), which leads CB-E to move the EEGTBG modules upwards in the fog-cloud hierarchy (cloudlet #2 and cloud). At this point, VSOT modules have more resources available in cloudlet #1, which reduces the delay for VSOT, but increases the delay for EEGTBG.

According to the scheduling mechanism used in the evaluation, the decision to move the application's components to the cloud affects the total use of the network differently, as detailed in Fig. 9. The impact of the scheduling mechanisms on the total network usage is given by the total amount of data transferred within all network links. As can be seen in Fig. 9, among all scheduling mechanisms, the concurrent scheduling mechanism has a lower total network usage, because it maintains all components closer to the users in the cloudlet, avoiding the communication between the cloudlet and the cloud. However, the naive decision imposed by the concurrent approach affects negatively the loop delay results of the applications.

Although DP-E, DP-I, and CB-E offer the lower loop delays for the EEGTBG application, these scheduling mechanisms use the network (bandwidth) more intensively. This occurs as a result of the migration of the VSOT application components (a network-intensive application) to the cloud when the cloudlet is overloaded. Therefore, when the VSOT components are moved upwards in the fog-cloud hierarchy, the total network usage is increased because of a large amount of data stream is then transmitted from the devices to the cloud.

We notice in Fig. 9b that CB-E maintains the data transmitted in the network as lower as possible, besides promoting the EEGTBG components prioritization. A critical case in point is when the 5th instance of EEGTBG is moved to the cloud. In this particular case, the number of EEGTBG application components in the cloudlet is increased, which causes all VSOT application components to run currently on the cloudlet. As a consequence, new component requests of incoming VSOT application is sent straightaway to the cloud, which leads to an increase in the use of the network.

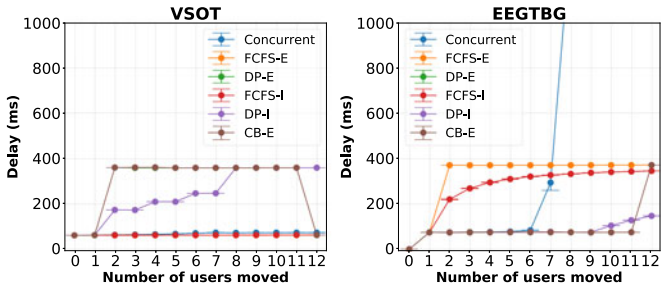
Nevertheless, considering only the three delay-aware scheduling mechanisms evaluated in this paper, CB-E transmits on average 2.7 percent less data than DP-I and 17.2 percent less data than DP-E.

7.2 Hierarchical Level Type B

The hierarchical level type B scenario has only one cloudlet farther from the users' devices when comparing to A, impacting negatively delay-sensitive applications. In this scenario, DP-E, DP-I, and CB-E repeat the same behavior of prioritizing the delay EEGTBG (Fig. 10b) over VSOT (Fig. 10a). Notably, these scheduling approaches reduce the delay by more than 50 percent compared with the other approaches for the EEGTBG scenario.

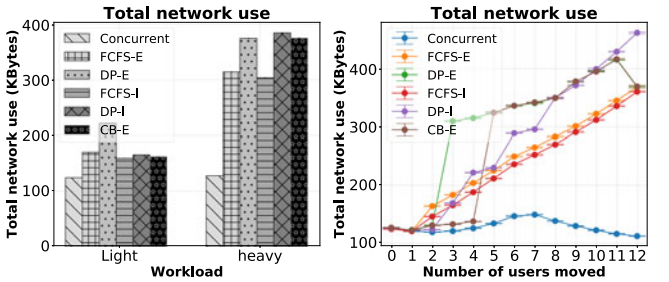
Fig. 11 shows the loop delays for the VSOT and EEGTBG application using all scheduling mechanisms. When only one EEGTBG player has been moved to the cloudlet, all strategies have the same results for both types of applications. However, when the 2nd EEGTBG player is moved to the cloudlet (see Fig. 11a), different scheduling mechanisms start to impact the application performance. DP-E, DP-I, and CB-E have increased the VSOT loop delays, while the other approaches have maintained a low delay for this application. In other words, DP-E, DP-I, and CB-E increased the VSOT loop delay to an acceptable level of about 360 milliseconds while maintaining the EEGTBG delays as lower as possible (Fig. 11b).

From the migration of the 7th user (see Fig. 11b), the concurrent scheduling mechanism brings resource contention to the cloudlet, increasing the loop delay of EEGTBG application. When the 8th user is moved (Fig. 11a), DP-E, DP-I and, CB-E start offering the same delay result for VSOT applications. These results keep occurring until 12th user is moved. When the 12th user is moved to the Cloudlet, the overloaded



(a) VSOT - DELAY - B (b) EEGTBG - DELAY - B

Fig. 11. Application loop delays according to the scheduling algorithms.



(a) Network Usage per Workload (b) Network Usage per Number of users moved

Fig. 12. Network usage according to the scheduling algorithm in hierarchical type B.

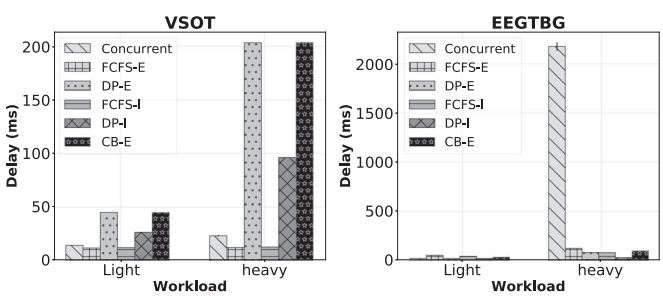
scenario leads the CB-E approach to balance the network usage with the EEGTBG applications, but keeping the VSOT components in the cloudlet, which decreases the delay for VSOT applications. On the other hand, it causes a slight increase in the delay for EEGTBG applications.

The scheduling decision also impacts the total network usage. Fig. 12 shows the total amount of data transmitted in the network for all different strategies. The concurrent approach results in lower network usage, since it keeps all modules in the cloudlet, making the communication to the cloud unnecessary. However, as all applications are in the same location, the concurrent strategy increases the average delay due to concurrency in the cloudlet resources. When we analyze only the three delay-aware strategies (DP-E, DP-I, and CB-E), we can notice that they increased the network usage as EEGTBG players arrived at cloudlet and VSOT modules are moved upwards to the cloud.

Considering the average for light and heavy workloads in Fig. 12, DP-E is the strategy that uses more the network overall. Otherwise, the CB-E strategy is able to reduce the application delay and transmit less data (2.3 percent less than DP-E and 9.8 percent less than DP-I) on the network.

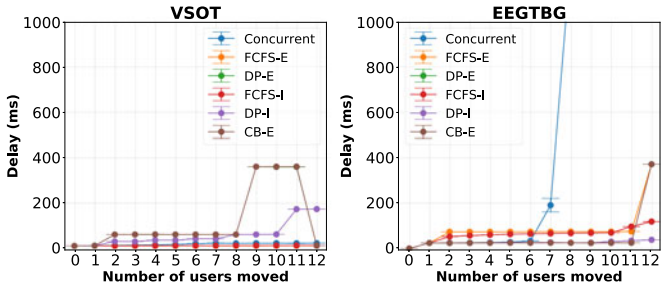
7.3 Hierarchical Level Type C

Unlike the previous topologies presented, the hierarchical level type C has two cloudlet levels besides the cloud layer. In this new case, as the concurrent scheduling mechanism would use only the first level cloudlet in the hierarchical type C, which has the same configuration as hierarchical type A. Then, we removed from the set of experiments the simulations using the concurrent approach when the hierarchical level type C scenario is employed.



(a) VSOT - DELAY - C (b) EEGTBG - DELAY - C

Fig. 13. Application loop delays in Hierarchy C according to the scheduling algorithms and the workload type.



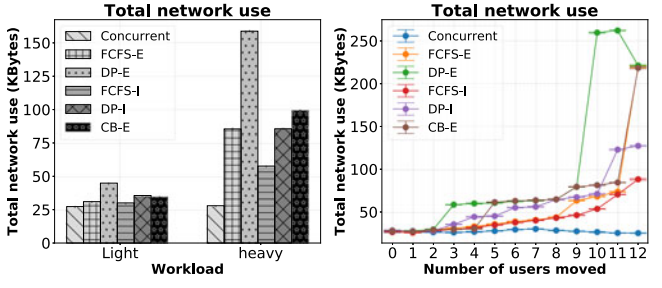
(a) VSOT - DELAY - C (b) EEGTBG- DELAY - C

Fig. 14. Application loop delays according to the scheduling algorithms.

Fig. 13b shows that, as the number of cloudlets (and resources closer to the users) increases, the EEGTBG applications delay is not significantly affected by DP-E, DP-I, and CB-E scheduling mechanisms. Likewise, the hierarchical level type C represents a positive impact on the VSOT application, offering more resources to run its modules closer to the users. Fig. 13a shows that DP-E, DP-I, and CB-E were able to slightly reduce delays for both light and heavy workloads when compared to the Figs. 7a and 10a.

DP-I offers the lowest average delay for EEGTBG applications, as illustrated in Figs. 13b and 14b. Due to the increase in the amount of resources closer to the users' devices in this scenario (Fig. 14a), DP-I maintains the EEGTBG components in the cloudlet #1 and moves upwards to cloudlet #2 the VSOT components. However, when the 9th user is moved, there is no resource available in the cloudlet #1 for more EEGTBG components applications, then, a few EEGTBG components are moved upwards to cloudlet #2, and all VSOT components moved upwards to the cloud. On the other hand, in the 9th, 10th, and 11th EEGTBG user arrivals, we notice an increasing in the network usage when using CB-E as the VSOT components are moved to the cloud (Fig. 13a). In fact, CB-E decides to compensate by maintaining all VSOT components in cloudlet #1 when the 12th EEGTBG user arrives. Next, CB-E distributes the EEGBTG components in cloudlet #2 and cloud, resulting in an increased delay for EEGTBG applications (Fig. 13b).

Analyzing the total network usage in Fig. 15, CB-E presents a reduction of 34.12 percent on data transmitted when compared to DP-E. However, as a simple prioritization technique looks more efficient in a more resourceful scenario, DP-I is 10.8 percent better than CB-E in this aspect.



(a) Network Usage per Workload (b) Network Usage per Number of users moved

Fig. 15. Network usage according to the scheduling algorithm in hierarchical type C.

Regarding the network usage shown in Fig. 15, the topology configuration C lessens the amount of data transmitted on the network in the presence of more resources in terms of the combination of two cloudlet levels, which avoids the migration of components upwards to the cloud. For example, CB-E in scenario type C represents 60 percent of network usage when compared to scenario type A, and 27 percent, when compared to scenario type B. This behavior occurs because CB-E makes heavy use of the network only when strictly necessary, that is, when all cloudlets are overloaded and it moving VSOT modules upwards is indeed unavoidable. The CB-E policy analyzes the communication impact generated by moving each application module's set and concludes that moving only one type of module in these circumstances is better because it causes less impact on the use of the network.

7.4 Discussion

We presented in this paper the combinations of policies (prioritization with component allocation) using three hierarchical layout types (Fig. 6) within the context of having a fog architecture comprising two-levels of fog nodes between the edge and the cloud. As the results show, it becomes clear that these policies have a direct impact on applications' quality of service, resulting in different behaviors according to the application requirements and computing infrastructure characteristics.

A multi-level fog hierarchy can achieve several benefits in terms of network usage and delays. For instance, by using the multi-level hierarchical type C, the network usage is 84.07 percent lower than the single-level hierarchical type A and B. Moreover, the network delay in the multi-level hierarchical type C was mitigated by 41.14 percent. These numbers emphasize the importance of addressing resource management and analyses of multi-layered fog infrastructures wisely. As we show, the employment of two types of applications (VSOT and EEGTBG) with different network requirements has given us insights on how the prioritization policy flavor can impact the network consumption and average delays differently. This impact is very likely to influence (negatively) the quality of experience of applications running on devices connected in a fog-based computing scenario.

The proposed communication-based scheduling policy was able to show a good trade-off between the network usage and delays when compared to a delay-priority allocation policy that does not take into account the communication between application components, especially when heavy workloads come into play. The use of solely delay-priority policy can, howsoever, achieve significant usability in occasions when the fog infrastructure is facing lighter workloads, which may not be always the case. As we emphasize in this paper, the combination of prioritization and communication-based policies along with state of the network workload is auspicious. Scheduling mechanisms aware of this can provide a good trade-off between network usage and delays in a scenario where dynamic and heterogeneous workloads are present. The results shown in this paper corroborate this statement. For instance, the hierarchical type C, which is more resourceful due to the presence of two cloudlet levels, enables the delay-priority only approach to achieve better results for both metrics (delay and network usage). On the other hand, the proposed CB-E mechanism outperformed the delay-priority in the hierarchical types A and B, where resources are more scarce, and making the system more capable of facing a heavier workload on average. Also, in this particular case, the proposed CB-E was able to maintain a low delay for latency-sensitive applications, while reducing the total network usage (data transfer) between the communication of application components. Results of which would not be able to be achieved by using non-sophisticated policies in the literature highlighted in Table 7.

If a multi-tier fog-cloud networking architecture would be a reality for end-users today, the proposed CB-E scheduling policy could assist the current massive increase of computing workload at the network edge driven by the coronavirus outbreak from the COVID-19 disease. In this particular case, as the moment the world entered in lockdown and people were forced to apply social distance measures and work from home, there was a significant increase in the use of cloud-based, bandwidth-hungry and latency-sensitive applications at the network edge. In a fog computing scenario comprising multiple cloudlet layers, the outcome of this problem could be slightly different if the edge users could appreciate an optimized fog-computing network managed by communication-aware approaches. For instance, our proposed CB-E allocation policy could be used by telco service providers/operators to mitigate this overload issue during this time. Our proposed CB-E offered low delay and low network usage by deciding which application component should be moved from a cloudlet layer upwards to the cloud to lessen potential network bottlenecks and keep overall delays reduced.

8 Conclusion

The unprecedented number of applications running at the edge of the network with vastly different requirements and characteristics makes the resource allocation problem challenging. Sophisticated bandwidth-hungry applications nowadays demand ultra-low latency requirements to operate correctly and satisfy the user quality of experience. Fog computing has emerged to satisfy these network demands and latency requirements by bringing cloud-like computing capabilities directly to the network's edge.

This paper has proposed a component allocation policy to consider the data transfers between application components in a fog computing architecture comprising several layers of fog nodes between the edge and the cloud. A communication-Based (CB-E) policy is proposed to determine the best locations for the application components to run (at the edge/fog node or the cloud) based on the amount of data transferred between components. For instance, the proposed CB-policy decides which application component (already deployed) should be moved upwards in the cloud or stay at the edge instead of dispatching the arriving application component directly to the cloud.

We pointed out in this paper that the common Delay-Priority (DP-I) resource allocation method from the literature has been successful in reducing network delay between application components, albeit this method only slightly reduces the network usage by the application components

when data need to be transferred. To overcome this limitation, our proposed Communication-based (CB-E) scheme computes the best component(s) to be moved upwards in the fog-cloud hierarchy based on data dependencies' characteristics. Because the proposed CB-E allocation method focuses on the amount of data transferred between each application component, our proposed approach overcomes the existing DP-I technique.

We aim as future work to develop a combined, tunable algorithm that combines aspects from both delay-priority and communication-based policies.

Acknowledgments

This work was supported in part by the Sao Paulo Research Foundation (FAPESP) under Grants 2018/23126-3 and 2015/24494-8 and in part by the Conselho Nacional de Desenvolvimento Cientfico e Tecnolgico (CNPq), Brazil, under Grants 432943/2018-8 and 309562/2019-8.

References

[1] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?," *Computer*, vol. 43, no. 4, pp. 51–56, 2010.
[2] L. Bittencourt *et al.*, "The Internet of Things, fog and cloud continuum: Integration and challenges," *Internet Things*, vol. 3-4, pp. 134–155, 2018.
[3] L. Bittencourt, J. Diaz-Montes , R. Buyya, O. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Comput.*, vol. 4, no. 2, pp. 26–35, Mar. 2017.
[4] L. Peterson *et al.*, "Democratizing the network edge," *SIGCOMM Comput. Commun. Rev.*, vol. 49, no. 2, p. 31–36, May 2019.
[5] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Trans. Internet Technol.*, vol. 19, no. 1, pp. 1–21, Nov. 2018.
[6] M. Satyanarayanan, R. Schuster, M. Ebling, G. Fettweis, H. Flinck, K. Joshi, and K. Sabnani, "An open ecosystem for mobile-cloud convergence," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 63–70, Mar. 2015.
[7] Y. Lin and H. Shen, "Cloud fog: Towards high quality of experience in cloud gaming," in *Proc. 44th Int. Conf. Parallel Process.*, 2015, pp. 500–509.
[8] D. Charântola, A. C. Mestre, R. Zane, and L. F. Bittencourt, "Component-based scheduling for fog computing," in *Proc. 12th IEEE/ACM Int. Conf. Utility Cloud Comput. Companion*, 2019, pp. 3–8.
[9] H. Shah-Mansouri and V. W. S. Wong, "Hierarchical fog-cloud computing for IoT systems: A computation offloading game," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 3246–3257, Aug. 2018.
[10] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct.–Dec. 2009.
[11] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-aware fog service placement," in *Proc. IEEE 1st Int. Conf. Fog Edge Comput.*, May 2017, pp. 89–96.
[12] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Edition MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.
[13] H. Gupta, A. Vahid Dastjerdi , S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments," *Softw. Pract. Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.
[14] V. B. C. Souza, W. RamA-rez, X. Masip-Bruin , E. Marn-Tordera, G. Ren, and G. Tashakor, "Handling service allocation in combined Fog-cloud scenarios," in *Proc. IEEE Int. Conf. Commun.*, 2016, pp. 1–5.
[15] R. O. Aburukba, M. AliKarrar, T. Landolsi, and K. El-Fakih, "Scheduling Internet of Things requests to minimize latency in hybrid fog-cloud computing," *Future Gener. Comput. Syst.*, vol. 111, pp. 539–551, 2020.
[16] A. Munir, P. Kansakar, and S. U. Khan, "IFCIoT: Integrated fog cloud IoT: A novel architectural paradigm for the future Internet of Things," *IEEE Consum. Electron. Mag.*, vol. 6, no. 3, pp. 74–82, Jul. 2017.
[17] I. M. Ali, K. M. Sallam, N. Moustafa, R. Chakraborty, M. J. Ryan, and K. K. R. Choo, "An automated task scheduling model using non-dominated sorting genetic algorithm ii for fog-cloud systems," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2020.3032386.
[18] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Gener. Comput. Syst.*, vol. 97, pp. 219–235, 2019.
[19] H. Elazhary, "Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions," *J. Netw. Comput. Appl.*, vol. 128, pp. 105–140, 2019.
[20] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on fog computing: architecture, key technologies, applications and open issues," *J. Netw. Comput. Appl.*, vol. 98, pp. 27–42, 2017.
[21] L. F. Bittencourt, A. Goldman, E. R. Madeira, N. L. da Fonseca, and R. Sakellariou, "Scheduling in distributed systems: A cloud computing perspective," *Comput. Sci. Rev.*, vol. 30, pp. 31–54, 2018.
[22] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Experience*, vol. 41, no. 1, pp. 23–50, 2011.
[23] A. Geelen, "Deutsche telekom prepares edge-native gaming experiences." Accessed: Sep. 25, 2020. [Online]. Available: https://www.telekom.com/en/company/details/deutsche-telekom-prepares-edge-native-gaming-experiences-606842
[24] C. Puliafito *et al.*, "MobFogSim: Simulation of mobility and migration for fog computing," *Simul. Modelling Pract. Theory*, vol. 101, p. 102062, 2020. .
[25] L. F. Bittencourt, E. R. M. Madeira, and N. L. S. Da Fonseca, "Scheduling in hybrid clouds," *IEEE Commun. Mag.*, vol. 50, no. 9, pp. 42–47, Sep. 2012.

Maycon Leone Maciel Peixoto received the master's and the PhD degrees in computer science from the University of Sao Paulo, Brazil, in 2008 and 2012, respectively. He conducted postdoctoral research at the University of Campinas, Brazil, in 2020. He is also an associate professor with the Department of Computer Science of the Federal University of Bahia. His research interests mainly include urban computing, smart grids, vehicular ad hoc networks, performance evaluation, cloud, edge, and fog computing.

Thiago A. L. Genez (Member, IEEE) received the PhD degree in computer science from the University of Campinas, São Paulo, Brazil. He is currently a software engineer with the European Bioinformatics Institute, European Molecular Biology Laboratory, Hinxton, U.K. He was a research associate with the University of Cambridge, U.K., a postdoctoral research fellow with the University of Bern, Switzerland, and research assistant with Loughborough University, U.K. His research interests mainly include distributed systems and computer networks, more specifically running workflow-like applications atop these environments.

Luiz F. Bittencourt (Member, IEEE) is currently an associate professor with the University of Campinas, Brazil. His research interests mainly include resource management, and scheduling in cloud, edge, and fog computing. He acts on the organization of several conferences in the cloud computing and edge computing subjects, and in several technical program committees. He is also an associate editor for the *IEEE Cloud Computing Magazine*, *Computers and Electrical Engineering Journal*, and *Internet of Things Journal*. He was the recipient of the IEEE Communications Society Latin America Young Professional Award in 2013.