Profiling Performance of Application Partitioning for Wearable Devices in Mobile Cloud and Fog Computing

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 12, 2018, accepted December 31, 2018, date of publication January 17, 2019, date of current version February 4, 2019.

*Digital Object Identifier 10.1109/ACCESS.2019.2892508*

# Profiling Performance of Application Partitioning for Wearable Devices in Mobile Cloud and Fog Computing

**CLAUDIO FIANDRINO[1], (Member, IEEE), NICHOLAS ALLIO[2], DZMITRY KLIAZOVICH[3], (Senior Member, IEEE), PAOLO GIACCONE[2], (Senior Member, IEEE), AND PASCAL BOUVRY[4], (Member, IEEE)**

[1]IMDEA Networks Institute, 28918 Madrid, Spain
[2]Department of Electronics and Telecommunications, Politecnico di Torino, 10129 Turin, Italy
[3]ExaMotive, 4362 Esch-sur-Alzette, Luxembourg
[4]Faculty of Science, Technology and Communication-Computer Science and Communications Research Unit and Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, 4365 Esch-sur-Alzette, Luxembourg

Corresponding author: Claudio Fiandrino (claudio.fiandrino@imdea.org)

**ABSTRACT** Wearable devices have become essential in our daily activities. Due to battery constraints, the use of computing, communication, and storage resources is limited. Mobile cloud computing (MCC) and the recently emerged fog computing (FC) paradigms unleash unprecedented opportunities to augment the capabilities of wearable devices. Partitioning mobile applications and offloading computationally heavy tasks for execution to the cloud or edge of the network is the key. Offloading prolongs the lifetime of the batteries and allows wearable devices to gain access to the rich and powerful set of computing and storage resources of the cloud/edge. In this paper, we experimentally evaluate and discuss the rationale of application partitioning for MCC and FC. To experiment, we develop an Android-based application and benchmark energy and execution time performance of multiple partitioning scenarios. The results unveil architectural tradeoffs that exist between the paradigms and devise guidelines for proper power management of the service-centric Internet-of-Things applications.

**INDEX TERMS** Mobile cloud computing, fog computing, energy efficiency, IoT, wearable devices.

## I. INTRODUCTION

Mobile devices have become essential for our daily activities such as business, health-care, social networking and entertainment [2]. Multiple types of devices, including watches, glasses, helmets, gloves and rings has contributed to raise, are available on the market, which has witnessed an increase of 16.7% in millions units (Mu) sold from 2016 to 2017 (from 265.88 Mu units to 310.37 Mu with a projection of 504.65 Mu in 2021) [3]. Wearable devices allow to perform advanced tasks such as monitoring and tracking of physiological functions or biofeedback being incorporated in clothing or worn on the body. Unlike generic Internet of Things (IoT) devices, wearable devices have several unique features, such as mobility.

Modern mobile devices have at disposal computing, communication, storage resources and sensing capabilities. However, being constrained by capacities of their batteries, the use of such resources is limited. Distributed computing paradigms, including MCC, FC and mobile edge computing (MEC), have emerged to overcome such limitations [4], [5]. MCC and FC provide the developers with the possibility to exploit the rich set of resources of the cloud and of the edge of the network in terms of computing capabilities and storage for their applications. To this end, MCC and FC outsource part of the computing tasks from weak mobile devices to the powerful cloud or fog. This process reduces battery consumption of the mobile devices, and enhances and augments performance of the mobile applications. Additionally, MCC and FC paradigms enable application execution in constrained environments, e.g., with limited connectivity.

For outsourcing, applications need to be first partitioned into a number of computing, communication or storage tasks of independent nature. Then, those tasks that are not tight

to the mobile device specifically and do not require specific hardware for execution can be offloaded to the cloud. Offloading involves either traffic or computing tasks. The former case involves steering traffic from cellular network towards wireless local area networks [6]. The mobile network operators are highly interested in traffic offloading to relieve the burden of the cellular core network. Instead, in this paper, we focus on offloading of computing tasks. In this case, network awareness, i.e., the capability of assessing whether is more convenient to offload tasks using costly cellular interface or intermittently-available WiFi interface [7], provides higher levels of effectiveness and better copes with mobility issues such as roaming, rate and channel quality variations [8]. The problem of computation offloading with multiple-users has been proven to be NP-hard [9]. The benefit of computation offloading can be quantified by analyzing the trade-off between the amount of energy saved by avoiding local processing at the wearable device and the increase of energy spent for communications with the edge/cloud. Analysis of energy consumption in distributed clouds shows that access networks and not datacenter networks are the most energy hungry components of the cloud ecosystem [10].

In this paper, we experimentally study performance of application partitioning for wearable devices in both MCC and FC environments. Most of the currently existing techniques for modeling application partitioning do not capture the characteristics of MCC/FC entirely. *Location* or *entity* where each task is executed (e.g., local device, fog or cloud) and *technology* employed for data transfer between the entities (e.g., WiFi, Bluetooth) are two essential parameters. To run the experiments, we developed an Android-based application for Google Glass, called *TreeGlass*, which performs recognition of images of tree leaves and by them identifies the name of the trees. TreeGlass is designed so that its tasks can be flexibly partitioned between the Google Glass, the smartphone, and the cloud. Each configuration evaluates the execution time and energy consumption of mobile devices. Note that TreeGlass exemplifies the general class of object detection and recognition applications, where the task graph is simple and sequential. Other applications such as those of online gaming, remote control or multimedia streaming have more complex task graph with cycles and task interdependence [11]. In some scenarios, these task graphs can be unfolded and become sequential as in our case. To maintain full control over the partitioning mechanism, we do not rely specifically on optimization mechanisms for offloading such as in [12]. The evaluation methodology permits to derive design considerations and implementation trade-offs, as well as to provide guidelines for improving efficiency of power management.

The rest of the paper is organized as follows. Section II overviews the concepts of MCC and FC and presents related works. Section III details TreeGlass's architecture and partitioning scenarios. Section IV provides performance evaluation highlighting experimental results. Section V discusses trade-offs and provides guidelines for power management of service-centric IoT applications and, finally, Section VI concludes the work.

## II. A PRIMER ON MOBILE CLOUD AND FOG COMPUTING

MCC extends the traditional cloud computing paradigm to the mobile environment: processing and data storage still occur outside the mobile devices. When referring to MCC, in this paper we assume that the wearable devices offload task execution to the cloud solely.

Multi-Access Edge Computing (MEC) was standardized by the European Telecommunications Standards Institute (ETSI) [13]. Formerly known as Mobile Edge Computing, MEC aims at providing computing service closer to the end user and is primarily a key enabler for the 5th generation mobile networks [14]. FC, that similarly to MEC brings computing services to the edge of the network, was proposed by Cisco [15] to support service-centric IoT characteristics like location awareness, low latency and geo-distribution. Typical examples of such IoT services are in the areas of health-care because the fog can process medical data with lower latency than the cloud, thus enabling real-time alerts or anomaly detection [16], and indoor localization [17], where devices in the vicinity collaborate to minimize the energy expenditure of performing computing tasks related to the fingerprinting process. With FC, in this paper we intend a scenario where a wearable device can offload task execution to both (i) nearby devices (e.g., smartphone), and (ii) to the cloud. This scenario is realistic as the majority of the applications for wearable devices is designed to work in pair with smartphones [18]. Resource management in FC is a major concern. According to Deng *et al.* [19], FC-based resource allocation strategies need to take into account the trade-off between power consumption and communication delays.

In the literature, several studies analyze task offloading experimentally. Miettinen and Nurminen [20] study the trade-off between local computation and offloading performing energy measurements on smartphones from various vendors running different applications. The authors conclude that the characteristics of the application workload, the technologies employed for communications affect the performance of the offloading process. Segata *et al.* [21] study the trade-off between the energy consumed for communications with 2G, 3G and WiFi versus local computation. The experimental results show that WiFi outperforms cellular technologies from an energy standpoint and that uploading is more demanding than downloading. Altamimi *et al.* [22] study a similar problem, but focus on energy models for WLAN, 3G and 4G technologies. The models assess the energy costs considering all the stack, from the application layer all the way down to the physical layer. Unlike previous studies, in our paper we profile performance of application partitioning for different MCC and FC scenarios. Similarly to the state-of-the-art, we also consider different communication technologies such as Bluetooth and WiFi as they uniquely define the implementation scenario and impact on energy consumption.

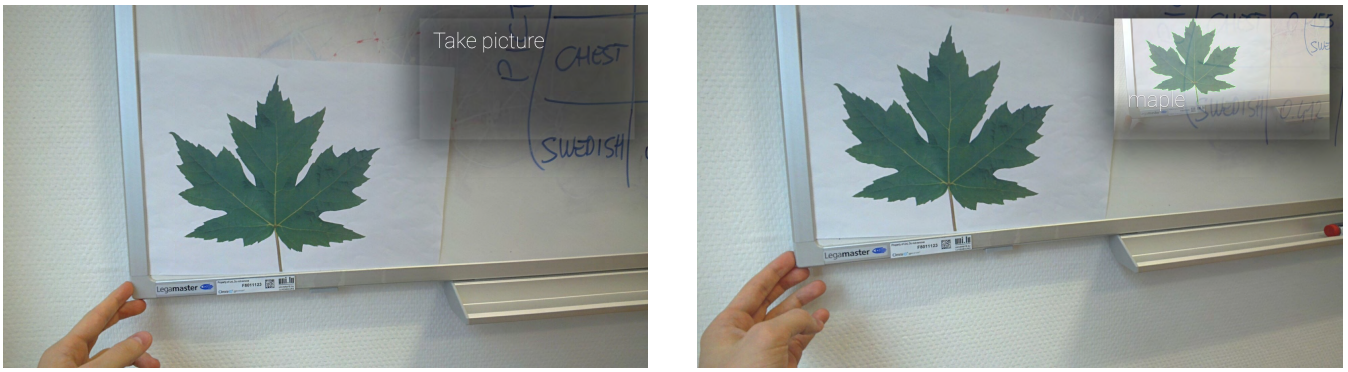


**FIGURE 1.** Execution of TreeGlass at Google Glass: (a) image acquisition and (b) display of the result.

## III. PRACTICAL CASE: THE TreeGlass APPLICATION

The TreeGlass application performs recognition of images with tree leaves and identifies the corresponding tree. Its ultimate goal is to test the performance of different partitioning configurations under realistic scenarios when applied to wearable devices in MCC and FC. Thus, the application runs either in MCC or FC mode by distributing the computation of tasks among the system components, i.e., the wearable device, the smartphone and the cloud.

In a nutshell, TreeGlass operates similarly to the workflow of object recognition applications [23]–[25]. Specifically, unique features are first extracted from a picture (detection) and are successively compared against a database (recognition). TreeGlass resorts on Google Glass to acquire pictures of leaves, while the edge and/or the cloud perform detection and recognition. Fig. 1 illustrates the workflow of TreeGlass. In more details, Fig. 1(a) shows the image acquisition phase and Fig. 1(b) displays the result from the perspective of the user wearing the Google Glass.

At a glance, *TreeGlass'* workflow is as follows. After image acquisition, the application detects the leaf from the image and extracts key features such as its color and contour. The latter is a closed curve shape and can be analyzed with similarity metrics [26]. Such features are then sent to the cloud for recognition, which practically translates into finding a match in a database. An answer to the user is then sent and displayed regardless of the outcome of the recognition phase. Fig. 1(b) exemplifies a positive match: the Google Glass highlights the contour of the leaf in bright green and displays the name of the tree in the bottom right part of the screen.

### A. THE ARCHITECTURE

In essence, TreeGlass runs over any Android-based wearable device featuring a camera, a smartphone as the fog/edge, and the cloud. Fig. 2 illustrates TreeGlass architecture with the three components. In the experiments, the reference wearable device are the Google Glass. The application is designed in such a way that its tasks can run simultaneously on both Google Glass and smartphone. This enables flexible and easy-to-customize partitioning. Fig. 2 also highlights the technologies that allow each of the application parts to communicate one with each other, and the programming languages employed for implementation.

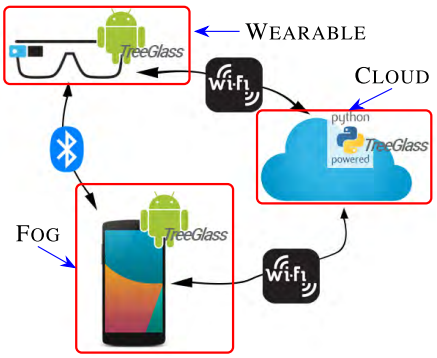


**FIGURE 2.** The components of the TreeGlass architecture.

The minimum operating system supporting TreeGlass is Android KitKat 4.4.4 (API Level 19). It is the native version running on the Google Glass and that was employed for all the experiments. The mobile application is written in Java and employs the standard APIs provided by Android SDK.

The recognition phase is always executed in the cloud, as it is not practicable to store large database in a distributed fashion on fog platforms. Each entry of the database contains a master picture of the leaf and its features like the color and the shape of the contour. Additionally, the cloud is also designed to run image processing tasks. This design choice enables high flexibility in application partitioning and allows to obtain an exhaustive comparison between the possible offloading scenarios. The component of TreeGlass that runs in the cloud is written in Python. An Android version of such component would run on the cloud only with an emulator. This is not an effective solution as it introduces overheads that would undermine the validity of the results.

The core of the application performing detection is based on the OpenCV[1] library. OpenCV features high level

[1] Available at: `http://opencv.org/`

functions that make easy to manipulate and extract features from images. In addition, the library provides methods to compare features of different images by returning metrics usable for searching matches. Custom Java and Python libraries operate on top to aggregate OpenCV methods and divide the application into tasks. These tasks will be executed in different configurations to verify the performance of MCC and FC offloading scenarios. The Java and Python libraries form a unique library called *TreeRecLib*. The version of *TreeRecLib* written in Java is exploited by Google Glass and the smartphone, while the Python version is employed in the cloud. They both contain the same function calls and perform the very same operations.

### B. APPLICATION PARTITIONING IN TreeGlass

TreeGlass operates in real-time and is modeled with a set of tasks that have specific precedence constraints. The application is partitioned into computing tasks and communication tasks, which depend on the offloading scenario (see Section III-C for the details). The computing tasks include the following:

- *Image acquisition* (*Ia*): The Google Glass acquires the images. This task is always executed locally at the wearable device and does not contribute to the analysis of energy consumption for the following two reasons. First, the task cannot be logically and physically offloaded to any other entity. Second, users spend arbitrary time to capture images and often perform this operation multiple times before obtaining an acceptable image.
- *Image processing* (*Ip*): The OpenCV methods prepare the picture for detection and recognition.
- *Feature Extraction* (*Fe*): The image is further elaborated for key features extraction. This task performs detection of the leaf.
- *Finding match* (*Fm*): With the help of the features extracted, this task searches for a match in the database and returns the outcome of the process (positive or negative).
- *Building and showing* (*Bs*): Once the wearable device receives the feedback, it presents the outcome to the user. The result is ''built and displayed'' according to the user interface guidelines of the Google Glass. Similar to *Ia*, the task is always performed by the Google Glass for obvious reasons.

To improve readability and understanding of the offloading scenarios and the results, a color uniquely defines a computing task. Table 1 details task description and its associated color.

In TreeGlass, communication tasks transfer information from one task to another among different entities. Internal communications, i.e., those happening within the same entity are not profiled. Within the application workflow, both Bluetooth and WiFi technologies can be employed for individual data transfer among tasks. These technologies impact on throughput and energy consumption differently [27].

**TABLE 1. Task description and color association.**

| COLOR | ACRONYM | DESCRIPTION |
|---|---|---|
| | *Ia* | Image acquired by Google Glass |
| | *Ip* | Input image processing |
| | *Fe* | Features extraction from the image |
| | *Fm* | Database query for match finding |
| | *Bs* | Preparation and display of the result |

Bluetooth was designed for personal area communications. Hence, it features short communication ranges and low bit rates (up to 3 Mbit/s). In comparison, WiFi technology provides higher data rates (up to 54 Mbit/s with the standard 802.11g) and a longer operating range. To graphically differentiate the technologies, Section III-C uses a dot-dashed and a double line for WiFi and Bluetooth respectively.

### C. OFFLOADING SCENARIOS

TreeGlass always involves the wearable device and the cloud in all scenarios. When the smartphone is present, then TreeGlass operates over a fog platform. Overall, four representative scenarios for both MCC and FC are identified. The rationale and the specifics of the design implementation are illustrated in the following paragraphs.

#### 1) SCENARIO Gg-Cl LOCAL

Fig. 3(a) is a MCC scenario where main computation is performed locally at the Google Glass (Gg). The communication between the cloud (Cl) and the wearable device is done via WiFi. Specifically, the wearable device performs tasks *Ia*, *Ip* and *Fe* locally and sends the resulting data to the cloud. Such data is a JSON string which contains the features extracted from the acquired image. Once the cloud receives such input, it looks for potential matches in the database and returns the result to the Google Glass again in form of a JSON string (task *Fm*). Finally, the wearable device builds and displays the received result (task *Bs*).

#### 2) SCENARIO Gg-Cl REMOTE

Fig. 3(b) shows a MCC scenario. Unlike the previous Gg-C Local case, the computation of core tasks is completely offloaded to the cloud while wearable device performs simple input/output operations represented by the tasks *Ia* and *Bs*. After having acquired the image at the Google Glass, the *whole picture* is sent via WiFi to the cloud for execution of tasks *Ip*, *Fe* and *Fm*. The result, in form of JSON string, is returned to the Google Glass using WiFi again.

#### 3) SCENARIO Gg-Sm-Cl FOG

Unlike Gg-Cl Local and Gg-Cl Remote, this FC scenario (see Fig. 3(c)) offloads computing tasks not to the cloud located in wide-area network, but to the nearby smartphone (Sm) in the edge/fog. Similarly to Gg-Cl Remote, the Google Glass is in charge of the input/output operations (tasks *Ia* and *Bs*), while the rest of the computation is completely offloaded. The motivations are as follows: (i) WiFi is less energy

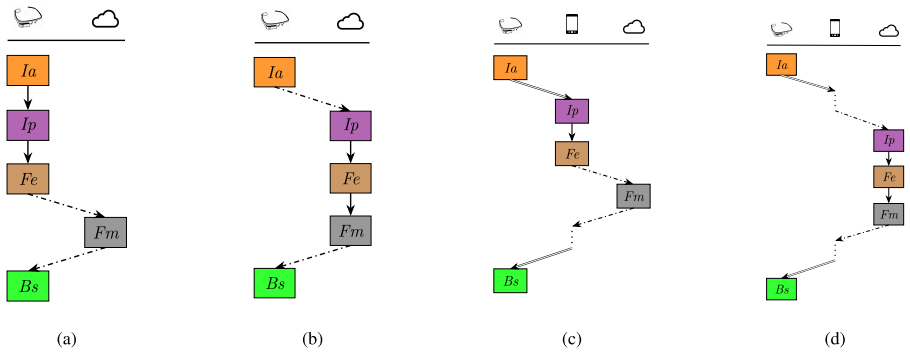


**FIGURE 3.** Offloading scenarios.

efficient than Bluetooth that is specifically designed for communications in small operative range [27]; (ii) Bluetooth is the most widely adopted technology by the majority of mobile OSs to pair smartphones and IoT devices. Once the information reaches the smartphone, it performs preliminary computation on the received image executing tasks *Ip* and *Fe*. Then the extracted features are sent to the cloud using WiFi. The cloud executes task *Fm* and returns the result to the smartphone using WiFi again. Then the smartphone relays back to the Google Glass the result with Bluetooth and finally performs task *Bs*.

#### 4) SCENARIO Gg-Sm-Cl RELAY

Fig. 3(d) shows FC-based scenario whose setup is similar to Gg-Sm-Cl Fog. However, in Gg-Sm-Cl Relay the smartphone is exclusively used to relay communications between the wearable device and the cloud. The computation is instead completely offloaded to the cloud similarly to Gg-Cl Remote. Thus, while the Gg-Sm-Cl Fog scenario takes advantage from the computing capabilities of the edge/fog, the Gg-Sm-Cl Relay exploits its communication potential. Similar to the previous configuration, the Google Glass executes input/output related tasks (*Ia* and *Bs*). Then, it sends the raw image to the smartphone via Bluetooth (output of task *Ia*). After having received the data from the Google Glass, the smartphone immediately forwards the raw image to the cloud employing WiFi. In this phase, no additional operations take place, with the exception of internal stream manipulation from Bluetooth communication stack to WiFi's one. The same operation will take place in the reverse direction when the smartphone relays the result back to the Google Glass. Upon reception of the image, the cloud performs the tasks *Ip*, *Fe* and *Fm*. After having successfully executed *Fm*, the cloud returns the result to the Google Glass, using again the smartphone as a relay.

## IV. PERFORMANCE EVALUATION

To assess performance of the various partitioning scenarios illustrated in Section III-C, we utilized a PowerMonitor similarly to previous research [28], [29]. Fig. 4 shows the setup for the measurements.

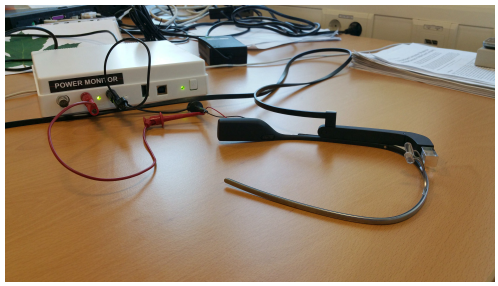


**FIGURE 4.** Setup for power management study.

### A. EVALUATION SETTINGS

The smartphone employed for experimentation is the Samsung Galaxy Note 4 and runs Android OS. In principle, the Google Glass can also be paired with an iOS device. However, the use of an Android-based phone guarantees more control during implementation and course of the experiments. Indeed, different operating systems would introduce an overhead due to translation of the application code to the language necessary for its execution. In the experiments, the smartphone runs Android Lollipop version 5.1.1 (API Level 23). It is equipped with a quad-core 2.7 GHz Krait 450 processor and has 3 GB of RAM. It also features a 16 MP camera and 32 GB of flash storage. The smartphone provides WiFi and Bluetooth connectivity, supporting 802.11 a/b/g/n/ac and Bluetooth v4.1 standards respectively. The smartphone is powered by a 3220 mA, 4.4 V battery.

The Google Glass operating system is based on Android (release KitKat 4.4.4, API Level 19). The device is equipped with a dual-core OMAP 4430 system-on-a-chip processor and features 2 GB of RAM. The display is a Prism projector supporting 640 × 360 pixels that is the equivalent of a 25 in/64 cm screen from 8 ft/2.4 m away. Additionally, the Google Glass are equipped with a 5 MP camera and have 16 GB of flash storage. The Google Glass support WiFi connectivity (802.11 b/g at 2.4 GHz) and Bluetooth connectivity. Finally, the wearable device is powered by a 570 mA, 3.7 V battery.

The cloud is emulated with a personal laptop. As the computing capabilities of the laptop are significantly superior of those of both mobile devices, the hypothesis is consistent.

The laptop is a MacBook Pro (Retina, 13-inch of 2014). At the time of the experiments, the laptop run OS X El Capitan version 10.11. The laptop features a dual-core 3.0 GHz Intel Core i7 and 16 GB of RAM; it has a 256 GB Apple SSD as storage and an AirPort Extreme card for WiFi 802.11 a/b/g/n/ac connectivity.

The Power Monitor hardware by Monsoon[2] is employed for power measurements. Previous research collected power consumption measurements via software by means of system calls [30], [31]. Although such methodology is valid, the power monitor directly acquires voltage, current and power measures, thus providing higher level of accuracy. For data retrieval, the power monitor needs to power the wearable/mobile device directly, hence in the equivalent circuit it substitutes the internal battery. The measurements are recorded in real time with a sampling rate of 5000 samples/s. A specific software displays a real time chart of the measures and provides the user with the capability to export at the end of the measurement campaign the readings of the session in `csv` format.

### B. EXPERIMENTAL RESULTS

This section illustrates the results obtained from profiling performance of the MCC and FC partitioning scenarios illustrated in Section III-C. First, execution time results are commented, then those pertaining to energy consumption.

#### 1) EXECUTION TIME

The execution time of applications depends on many factors such as current level of the battery and eventual active energy-saving mechanisms that limit computing and transmission power, the environmental conditions such as network load or contention in accessing the network. Other environmental aspects like fading and shadowing influence channel conditions and negatively affect performance of communications. Such issues do not impact on performance of computing tasks as much as communication tasks. To obtain insightful measurements, we limit the influence of external factors, i.e., *TreeGlass* is the only application running on Google Glass and the smartphone in static position. For WiFi we resort on eduroam network, which is a public network, to emulate a realistic scenario.

##### *a: EXECUTION TIME OF COMPUTING TASKS*

Only the scenarios Gg-Cl Local and Gg-Sm-Cl Fog are considered here because all the computing tasks but task *Fm*, are executed by the mobile devices (see Table 2). Task *Fm* corresponds to the database search. As the cloud always performs this task, its duration is negligible with respect to those of the other tasks. Task *Ia* is also not considered as it corresponds to the image acquisition and its duration is highly user dependent. Table 2 presents the results for the execution of the remaining tasks. Task *Bs* is always executed locally at the Google Glass and it is the task that takes longer to complete. In addition to display the name of the leaf, task *Bs* is also overlays a green contour around the leaf. Both operations are time and energy expensive. Tasks *Ip* and *Fe* are executed faster if performed by the smartphone, hence supporting the claim that offloading to more powerful devices is convenient.

**TABLE 2.** Execution time of computing tasks.

| ACRONYM | GOOGLE GLASS | SMARTPHONE |
|---|---|---|
| *Ip* | 0.711 s | 0.594 s |
| *Fe* | 0.125 s | 0.114 s |
| *Bs* | 2.853 s | – |

**TABLE 3.** Execution time of communication technologies.

| TECHNOLOGY & DATA | GOOGLE GLASS | | SMARTPHONE | |
|---|---|---|---|---|
| | SEND | RECEIVE | SEND | RECEIVE |
| WiFi - JSON String | 0.209 s | 0.370 s | 0.017 s | – |
| WiFi - Raw Image | 12.079 s | 2.906 s | 1.809 s | 0.416 s |
| Bluetooth - Raw Image | 15.402 s | 4.035 s | – | 8.017 s |

##### *b: EXECUTION TIME OF COMMUNICATION TASKS*

Recall that all the devices are connected under the same WLAN (eduroam), which is not under our control. The setting guarantees that the result presented next are in line with the performance that an application would obtain in real scenarios. Table 3 shows the results obtained for the tasks *Ip*, *Fe* and *Bs* both in Gg-Cl Local and Gg-Sm-Cl Fog scenarios. These are the two scenarios that make use of all possible communication technologies. As expected, the smartphone outperforms the Google Glass as it is equipped with more recent hardware and supports updated firmware versions. Upon the scenario under consideration, the Google Glass transfer different types of data. The MCC scenarios require the Google Glass to send via WiFi a JSON string for Gg-Cl Local and a raw image for Gg-Cl Remote. FC scenarios require the Google Glass to transmit the raw image to the smartphone via Bluetooth. On the one hande, in Gg-Sm-Cl Fog the smartphone locally performs tasks *Ip* and *Fe*, which then require to transmit to the cloud only the JSON string via WiFi. On the other hand, in Gg-Sm-Cl Relay, the smartphone forwards to the cloud the entire raw image via WiFi. Interestingly, for the scenario Gg-Sm-Cl Fog the smartphone takes long time in receiving the raw image via Bluetooth. The data transfer from the wearable device is slow and the smartphone receives data with low data rates.

#### 2) ENERGY CONSUMPTION

This section presents results obtained with the Monsoon power monitor and the precise knowledge of the task duration estimated in Section IV-B.1. Specifically, the power monitor only provides the power consumption profile of the *entire* application duration. To obtain precise estimate of power consumption in each task, we observe that computing tasks

[2]Available at: `http://www.msoon.com/LabEquipment/PowerMonitor/`

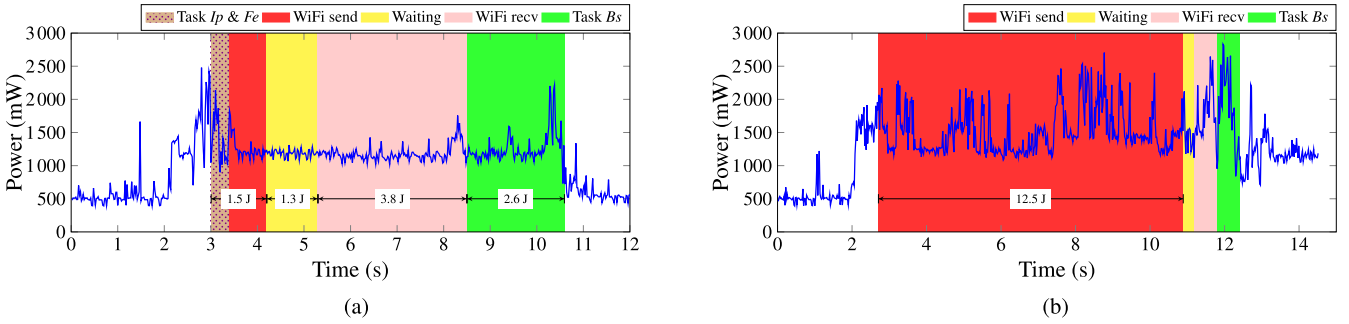


**FIGURE 5.** Power consumption profiles of scenarios (a) Gg-Cl Local and (b) Gg-Cl Remote measured by Google Glass.

**TABLE 4.** Color code definition in charts.

| COLOR | DESCRIPTION |
|---|---|
| | Execution of tasks *Ip* and *Fe* |
| | Execution of task *Bs* |
| | Data transmission via WiFi |
| | Data reception via WiFi |
| | Data transmission via Bluetooth |
| | Data reception via Bluetooth |
| | Waiting time: the device is in idle mode |

are interleaved by either internal or external (with WiFi and Bluetooth) communications. We therefore mark the start and end time of data exchange operations from time execution experiments and synchronize power measurement traces accordingly.

Table 4 shows the color code employed to illustrate the results in the next graph and differentiate between computing and communication tasks. The color code for computing tasks is inspired by the one of Table 1 (for task *Bs* it is the same). Tasks *Ip* and *Fe* are combined with a dotted-pattern and the background is a combination of the correspondent individual backgrounds shown in Table 1. The color of the communication tasks depends on the operation and technology used, WiFi - Bluetooth and Send - Receive. For each of those, a different shade of color is employed. The waiting time indicates the time that a component waits for the result from a task before starting the execution of the next task.

### *a: ANALYSIS OF* Gg-Cl LOCAL

Fig. 5(a) shows the power consumption profiles of the Google Glass. Initially, the power profile exhibits increasing consumption with peaks of at most 2500 mW. This are attributed to the start of the application and the acquisition of the image. We merge tasks *Ip* and *Fe* for the sake of easy representation as their execution is extremely fast. When *Fe* completes, the features are extracted and sent via WiFi to the cloud in form of a JSON string. While a local search would be faster at the cost of high energy consumption, keeping a large database on a resource constrained device is unfeasible. Hence, Google Glass waits that the cloud performs the search in the database and returns the outcome. During the waiting time, the device consumes an amount of energy that is similar to the one spent while transmitting data via the WiFi interface. The motivation is that the device stays in listening mode and keeps running all the functionalities of TreeGlass in the background. The energy cost attributed to the reception of the result from the cloud is in the range of the waiting and sending phases (the average instantaneous power consumption is 1300 mW). However, and unlike the mentioned phases, at the end of this phase it is possible to notice an increase of power consumption. This is due to the processing of the received data stream from lower layers of the protocol stack to the application layer. The final sector shows the power spent by the Google Glass to built and display the obtained result. The task *Bs* is the highest energy consuming tasks because it updates the user interface of the application. Specifically, the screen consumes energy to be refreshed. Note that the peeks of power consumption are almost at the same height of the ones in the initial phase, tasks *Ip* and *Fe* and before.

### *b: ANALYSIS OF* Gg-Cl REMOTE

Fig. 5(b) shows the performance of the Gg-Cl Remote scenario from the Google Glass' perspective. Gg-Cl Remote offloads all the computing tasks. Thus, the large red part of the power consumption profile represents the *WiFi send* operation where the Google Glass sends to the cloud the raw image. Unlike the previous Gg-Cl Local case, the entire picture and not a string is compressed and sent as a data stream. Consequently, Gg-Cl Remote creates a significant burden to the communication phase, that takes longer to complete at the expense of higher energy consumption. The power profile reaches peeks higher than 2500 mW (while for Gg-Cl Local the highest peak is around 1850 mW) and the total amount of energy spent for this operation is 12.5 J. Upon reception of the image, the cloud executes extremely fast the tasks *Ip*, *Fe* and *Fm* (see Fig. 3(b)) and the Google Glass remains only for a little amount of time waiting mode. The *WiFi recv* operation is short although the average power consumption is higher than in Gg-Cl Local. Similarly, also the duration of the *Bs* task is faster and reaches higher peak of power consumption.

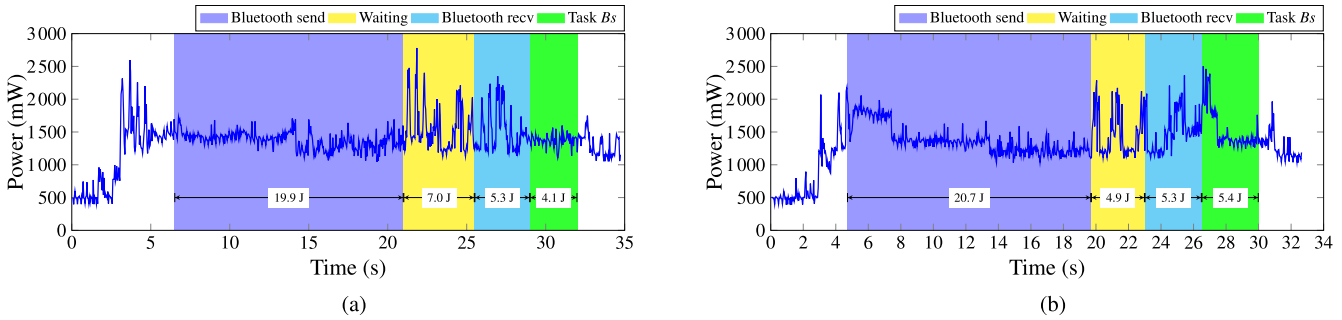


**FIGURE 6.** Power consumption profiles of scenarios (a) Gg-Sm-Cl Fog and (b) Gg-Sm-Cl Relay measured by Google Glass.

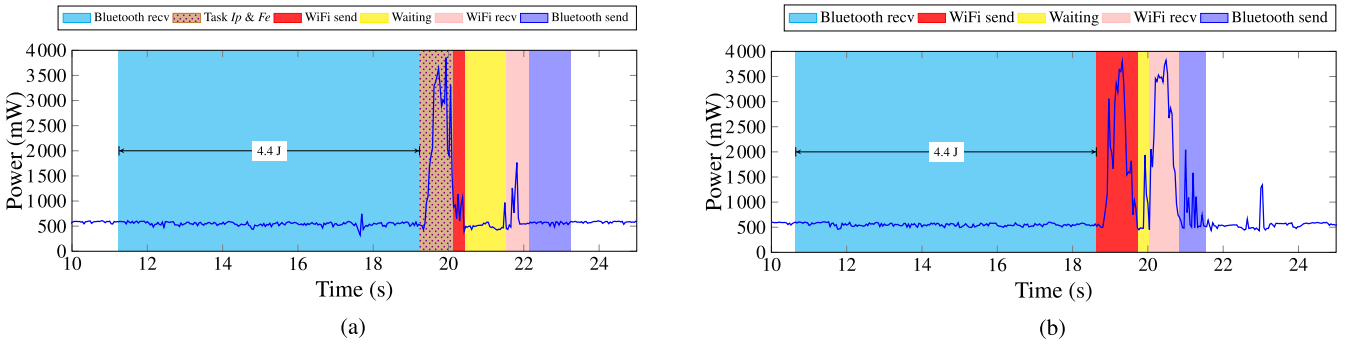


**FIGURE 7.** Power consumption profiles of scenarios (a) Gg-Sm-Cl Fog and (b) Gg-Sm-Cl Relay measured by the smartphone.

### *c: SUMMARY* Gg-Cl *CASES*

To summarize, the comparison of Gg-Cl Local and Gg-Cl Remote scenarios allows to understand pros and cons of MCC. The same entities are involved: the difference is in the type of tasks offloaded and, consequently, the type of data transmitted in the communication phases. As the captured image contains the leaf over a white background, it is easy to perform tasks *Ip* and *Fe* locally and transmit to the cloud a JSON string. Hence, offloading is not convenient as communication are the bottleneck because the transmission of the entire raw image is extremely expensive. This observation is not true anymore if the image captured is more complex, i.e., it contains realistic backgrounds. As Gg-Cl Local scenario requires the Google Glass to wait longer, this opens the possibility to further optimization and energy saving.

### *d: ANALYSIS OF* Gg-Sm-Cl FOG

This case is a FC scenario as all three entities are simultaneously involved in the execution of computing tasks. Fig. 6(a) shows the power consumption from the Google Glass standpoint, while Fig. 7(a) illustrates the power consumption from the smartphone standpoint. Note that Bluetooth provides connectivity between the Google Glass and the smartphone while WiFi is employed to interconnect the smartphone with the cloud. Similarly to Gg-Cl Remote scenario, the initial data transmission is the most energy expensive operation when performed by the wearable device. Altogether, this corresponds to an instantaneous power consumption values on average in the range of 1500 mW. Then, the wearable device waits for the completion of the data transmission. During this time (see yellow sector of Fig. 6(a)), the power consumption profile varies significantly because the device maintains active the Bluetooth interface waiting for the reply. The cyan sector identifies the reception of the result by the Google Glass. It is a energy expensive operation (overall 5.3 J) because involves decompression of the image. The green sector denotes the power profile of *Bs*, the final task.

Fig. 7(a) shows the power profile from the smartphone's perspective. The smartphone remains in receiving mode for a prolonged period of time (see the cyan sector) because the wearable device takes a long time to send the raw image. Unlike the Google Glass, the smartphone in this phase consumes a lower amount of energy because features a more performing Bluetooth antenna. Specifically, the power consumption profile remains low and stable around 500 mW. Upon receiving the image, the smartphone performs the tasks *Ip* and *Fe*. The execution of such tasks is clearly identifiable as the peeks ramp up and are as high as 3500 mW (see the brown-dotted-pattern of Fig. 7(a)). Once the features are extracted, the smartphone sends them to the cloud in form of a JSON string (red period) and waits (yellow period) the reception of the results (pink period). These, are finally returned to the Google Glass (dark blue period). The device efficiently stabilizes the power consumption between the data uploading and downloading performed via WiFi. It is also interesting to note that *Bluetooth send* and *recv* operations lead to similar energy consumption profiles.

#### e: ANALYSIS OF Gg-Sm-Cl RELAY

Fig. 6(b) and Fig. 7(b) respectively show the power consumption profile from the Google Glass and smartphone standpoint. In the former case, the scenario Gg-Sm-Cl Relay is almost identical to Gg-Sm-Cl Fog. However, from the smartphone standpoint, there is a significant change. The Gg-Sm-Cl Relay employs the communication and not computing resources of the fog. Thus, the smartphone simply acts as a relay. Specifically, it forwards the raw image to the cloud without performing *Ia*, *Fe* as in Gg-Sm-Cl Fog. Although the initial reception of the image (performed with Bluetooth) is similar to the Gg-Sm-Cl Fog scenario from a time duration and energy consumption perspective, the upload of the image and the download of the results from the cloud with WiFi lead to a higher energy cost. Between the two operations, the smartphone waits for the execution of the tasks *Ia*, *Fe* and *Fm* (performed by the cloud). This time is highlighted in yellow: it is short and almost imperceptible as the cloud performs all the task quick. The final part of the power profile (in dark blue) denotes the reception of the result by the Google Glass with Bluetooth.

#### f: SUMMARY Gg-Sm-Cl CASES

To summarize, both FC cases exhibit similar execution time and power consumption performance. From the standpoint of the Google Glass (see Fig. 6), the overall energy consumption is 36.3 J in Gg-Sm-Cl Fog case and 36.2 J in Gg-Sm-Cl Relay case. From the smartphone perspective (see Fig. 7), although the overall executing time performance are similar, the highest peak of power consumption are due to processing and communication for Gg-Sm-Cl Fog and Gg-Sm-Cl Relay respectively. As a result, from an energy perspective, it is convenient to exploit the smartphone for computing purposes (8.2 J in Gg-Sm-Cl Fog case and 9.2 J in Gg-Sm-Cl Relay case).

## V. DISCUSSION

### A. ENERGY BUDGET AND OFFLOADING

Fig. 8 compares the energy costs of the computing and communications tasks for all the offloading scenarios measured at the Google Glass. This comparison allows to derive considerations of the convenience of offloading for the resource-constrained wearable devices. The results are obtained averaging 10 runs. Between each run, we made sure to switch off and on all the devices for proper initialization of all the components.

The graph shows clearly that FC offloading scenarios negatively impact the energy consumption of the Google Glasses. The reason is that the wearable device employs Bluetooth, a low rate technology if compared with WiFi, to transfer large size images. In addition, it is worth nothing the considerable amount of energy that the Google Glass spend during the waiting phase. In comparison, the smartphone handles in a much more efficient way the waiting times. We conclude that: i) the Bluetooth technology should be employed to transfer other type of data with lower size such as text to be effective in offloading scenarios, and ii) the waiting times of the wearable devices should be reduced as much as possible. For what concerns the scenarios with local computation versus full offload, from an energy perspective there is a slight advantage in performing local computation. The reason is that we capture images with a white background (see Fig. 1), hence the operations for feature extraction are not so computationally expensive. In the offloading scenario, the more energy expensive component is the transmission of the raw image. We remark that we did experiments with a realistic network environment using a publicly available network (eduroam), hence this component also includes the energy spent during contention for the radio access.

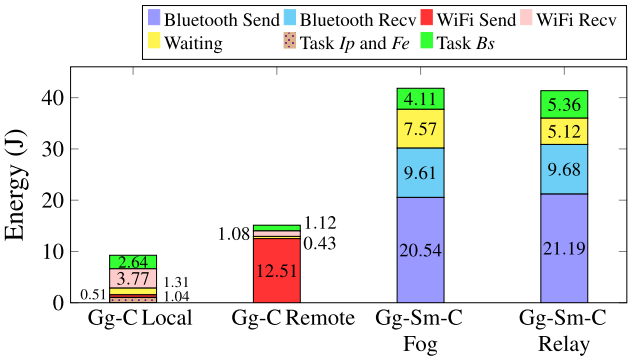


**FIGURE 8. Comparison of power consumption for all the offloading scenarios measured by Google Glass.**

### B. SERVICE-CENTRIC IoT APPLICATIONS

In TreeGlass, the task graph defining the temporal sequence of task execution is sequential (see Section III-B and Fig. 3). TreeGlass is one instance of a more generic class of applications, i.e., those performing object detection/recognition. Hence, the considerations given in Subsection V-A hold for this entire class of applications.

Other real-time applications, such as remote control or robots online gaming, include loops in the task graphs [12]. In such a context, communication tasks become even more important and the critical factor defining the overall performance of the application is latency and not only CPU and available data rates. Hence, such component should be better characterized and fog computing scenarios may become more appealing than in the case of object detection and recognition, where latency is not as critical as in real-time applications.

## VI. CONCLUSION

In this paper, we studied performance of application partitioning in MCC and FC focusing on the execution time and energy consumption analysis. For this, we designed an Android application, called TreeGlass, which can be flexibly partitioned between Google Glass, smartphone and the cloud. The methodology demonstrated that different partitioning

scenarios impact on the design of the application and on the choice of the communication media between the entities. The results highlight that in FC it is beneficial to employ nearby devices for computing purposes and to execute some of the tasks locally if communication overhead is small. In MCC, where only the wearable device and the cloud are involved, local computation is beneficial only if the cost of offloading becomes prohibitive because of the specific implementation, i.e, the type of data to be transmitted. We remark that our results have a broader scope than the specific insights our object recognition application provides. Specifically, the adopted methodology can be extended to other applications for MCC and FC to evaluate the optimal way to offload the tasks.

## ACKNOWLEDGMENT

This work is based on the doctoral dissertation that Dr. Claudio Fiandrino developed as a Ph.D. student at the University of Luxembourg [1].

## REFERENCES

[1] C. Fiandrino, "Energy-efficient communications in cloud, mobile cloud and fog computing," Ph.D. dissertation, Fac. Sci., Technol. Commun., Comput. Sci. Commun. Res. Unit, Esch-sur-Alzette, Luxembourg, 2016.

[2] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, "A first look at traffic on smartphones," in *Proc. ACM IMC*, 2010, pp. 281–287.

[3] Gartner. (2017). *Worldwide Wearable Device Sales*. [Online]. Available: https://www.gartner.com/newsroom/id/3790965

[4] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, "Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 337–368, 2nd Quart., 2014

[5] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[6] C. Fiandrino, D. Kliazovich, P. Bouvry, and A. Y. Zomaya, "Network-assisted offloading for mobile cloud applications," in *Proc. IEEE ICC*, Jun. 2015, pp. 5833–5838.

[7] H. Wu and K. Wolter, "Stochastic analysis of delayed mobile offloading in heterogeneous networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 2, pp. 461–474, Feb. 2018.

[8] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3317–3329, Dec. 2015.

[9] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[10] F. Jalali, K. Hinton, R. Ayre, T. Alpcan, and R. S. Tucker, "Fog computing may help to save energy in cloud computing," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1728–1739, May 2016.

[11] K. Kanoun, N. Mastronarde, D. Atienza, and M. van der Schaar, "Online energy-efficient task-graph scheduling for multicore platforms," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 8, pp. 1194–1207, Aug. 2014.

[12] T. Zhang, C. F. Chiasserini, and P. Giaccone, "TAME: An efficient task allocation algorithm for integrated mobile gaming," *IEEE Syst. J.*, to be published. [Online]. Available: [Online]. Available: https://ieeexplore.ieee.org/document/8360019

[13] F. Giust *et al.*, "MEC deployments in 4G and evolution towards 5G," ETSI White Paper, Feb. 2018. [Online]. Available: https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp24_MEC_deployment_in_4G_5G_FINAL.pdf

[14] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.

[15] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. ACM MCC*, 2012, pp. 13–16.

[16] F. A. Kraemer, A. E. Braten, N. Tamkittikhun, and D. Palma, "Fog computing in healthcare—A review and discussion," *IEEE Access*, vol. 5, pp. 9206–9222, 2017.

[17] A. Sciarrone, C. Fiandrino, I. Bisio, F. Lavagetto, D. Kliazovich, and P. Bouvry, "Smart probabilistic fingerprinting for indoor localization over fog computing platforms," in *Proc. IEEE CloudNet*, Oct. 2016, pp. 39–44.

[18] Y. Lee, W. Yang, and T. Kwon, "Data transfusion: Pairing wearable devices and its implication on security for Internet of Things," *IEEE Access*, vol. 6, pp. 48994–49006, 2018.

[19] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1171–1181, Dec. 2016.

[20] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proc. USENIX Hot Topics Cloud Comput.*, 2010, pp. 4–11.

[21] M. Segata, B. Bloessl, C. Sommer, and F. Dressler, "Towards energy efficient smart phone applications: Energy models for offloading tasks into the cloud," in *Proc. IEEE ICC*, Jun. 2014, pp. 2394–2399.

[22] M. Altamimi, A. Abdrabou, K. Naik, and A. Nayak, "Energy cost models of smartphones for task offloading to the cloud," *IEEE Trans. Emerg. Topics Comput.*, vol. 3, no. 3, pp. 384–398, Sep. 2015.

[23] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 399–458, 2003.

[24] N. Powers *et al.*, "The cloudlet accelerator: Bringing mobile-cloud face recognition into real-time," in *Proc. IEEE Globecom Workshops*, Dec. 2015, pp. 1–7.

[25] N. H. Motlagh, M. Bagaa, and T. Taleb, "UAV-based IoT platform: A crowd surveillance use case," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 128–134, Feb. 2017.

[26] G. G. Demisse, D. Aouada, and B. Ottersten, "Deformation based curved shape representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 6, pp. 1338–1351, Jun. 2018.

[27] R. Friedman, A. Kogan, and Y. Krivolapov, "On power and throughput tradeoffs of WiFi and Bluetooth in smartphones," *IEEE Trans. Mobile Comput.*, vol. 12, no. 7, pp. 1363–1376, Jul. 2013.

[28] J.-W. Sung and S.-J. Han, "Data bundling for energy efficient communication of wearable devices," in *Proc. IEEE LCN*, Oct. 2015, pp. 321–328.

[29] R. LiKamWa, Z. Wang, A. Carroll, F. X. Lin, and L. Zhong, "Draining our glass: An energy and heat characterization of Google glass," in *Proc. ACM APSys*, 2014, pp. 1–7.

[30] J. C. V. Bedregal, A. Perú R. E. M. Arisaca, and E. G. C. Gutierrez, "Optimizing energy consumption per application in mobile devices," in *Proc. Int. Conf. Inf. Soc.*, Jun. 2013, pp. 106–110.

[31] G. P. Perrucci, F. H. P. Fitzek, and J. Widmer, "Survey on energy consumption entities on the smartphone platform," in *Proc. IEEE VTC Spring*, May 2011, pp. 1–6.

**CLAUDIO FIANDRINO** (S'14–M'17) received the bachelor's degree in ingegneria telematica and the master's degree in computer and communication networks engineering from the Politecnico di Torino in 2010 and 2012, respectively, and the Ph.D. degree from the University of Luxembourg, in 2016. He joined the Wireless Networking Group, in 2016, after receiving the Ph.D. degree. He has received the 2016 SmartICT Certificate on standardization for business innovation through the joint program from the University of Luxembourg and ILNAS, and the National Standardization Agency. He is currently a Postdoctoral Researcher with the IMDEA Networks Institute, Madrid, Spain. His primary research interests include mobile cloud and fog computing, mobile crowdsensing, and data center communication systems. He was a TPC member of several IEEE and ACM conferences, and workshops. He has served as the Publication and Web Chair of the IEEE CloudNet 2014 and the Publicity Chair of ACM/IEEE ANCS. He has received the Spanish Juan de la Cierva Grant and the Best Paper Award from the IEEE Cloudnet 2016 and ACM WiNTECH 2018.

**NICHOLAS ALLIO** is currently with Ulmon GmbH, Vienna. He received the master's degree from the Politecnico di Torino, University of Luxembourg, in 2016, where he performed the Final Master Thesis Project as an Intern.

**DZMITRY KLIAZOVICH** (M'03–SM'12) received the award winning Ph.D. degree in information and telecommunication technologies from the University of Trento, Italy. He is currently the Head of innovation with ExaMotive. He was a Senior Scientist with the Faculty of Science, Technology, and Communication, University of Luxembourg. He has co-ordinated organizations and chaired a number of highly ranked international conferences and symposia, including the IEEE International Conference on Cloud Networking in 2014. His main research interests include intelligent transportation systems, telecommunications, cloud computing, and the Internet of Things (IoT). His works on cloud computing, energy-efficiency, indoor localization, and mobile networks have received the IEEE/ACM Best Paper Awards. He currently holds several scientific awards from the IEEE Communications Society and the European Research Consortium for Informatics and Mathematics. He is currently an Associate Editor of the IEEE Communications Surveys and Tutorials and the IEEE Transactions of Cloud Computing Journal.

**PAOLO GIACCONE** (M'02–SM'16) received the Dr.Ing. and Ph.D. degrees in telecommunications engineering from the Politecnico di Torino, Italy, in 1998 and 2001, respectively, where he is currently an Associate Professor with the Department of Electronics and Telecommunications. During the summer of 1998, he was with the High Speed Networks Research Group, Lucent Technology Bell Labs, Holmdel, NJ, USA. From 2000 to 2001 and in 2002, he was with Information Systems Networking Lab, Electrical Engineering Department, Stanford University, CA, USA. His main areas of interests include the design of network algorithms, the theory of interconnection networks, and the performance evaluation of telecommunication networks through simulative and theoretical methods.

**PASCAL BOUVRY** received the Ph.D. degree in computer science from the University of Grenoble, France. He is currently a Full Professor with the University of Luxembourg and the Head of the ILIAS Research Unit, DS-CSCE Doctoral School. He is also a Faculty Member of the Interdisciplinary Center for Security, Reliability, and Trust. His research interests include cloud & parallel computing, optimization, security, and reliability. He is also acting as the Communication Vice Chair of the IEEE STC on Sustainable Computing and the Co-Founder of the IEEE TC on Cybernetics for Cyber-Physical Systems. He is on the Editorial Boards of the IEEE Cloud Computing and *Swarm and Evolutionary Computation* (Elsevier).