# A Smartphone Perspective on Computation Offloading – A Survey

Quang-Huy Nguyen[1], Falko Dressler[2*]

[1] *Heinz Nixdorf Institute and Dept. of Computer Science, Paderborn University, Germany*

[2] *School of Electrical Engineering and Computer Science, TU Berlin, Germany*

**Abstract**

Computation offloading has emerged as one of the promising approaches to address the issues of restricted resources, leading to poor user experiences on smartphones in terms of battery life and performance when executing CPU intensive tasks. Meanwhile, an entire research domain has been initiated around this topic and several concepts have been studied touching both the smartphone and the cloud side. In this paper, we develop a categorization of fundamental aspects regarding computation offloading in heterogeneous cloud computing from the perspective of smartphone applications. We refer to heterogeneity in terms of the multitude of smartphone applications, the various uplink channels, and the variety of cloud solutions. We also survey state-of-the-art solutions for the identified categories. Finally, we conclude with a summary of the most important research challenges in making computation offloading reality.

*Key words:* Computation offloading, mobile edge computing, smartphones, energy consumption

## 1. Introduction

Smartphones are still considered to be resource-bounded mobile computing devices despite a number of new developments and the evolution of their hardware. At the same time, user demands keep increasing rapidly with a lot of resource-intensive and power-hungry applications. Thus, prolonging the battery life of smartphones and improving the application performance faces many difficulties. Along with efforts in extending hardware resources of smartphones, several approaches on mobile applications deployment and execution have been proposed to solve above problems. Among these mechanisms is computation offloading, which outsources intensive parts of applications running on smartphones to the cloud for remote execution in order to save time and energy [1, 2, 3].

Different frameworks have been developed in order to demonstrate the feasibility of computation offloading [4, 5, 6, 7, 8, 9, 10]. In this context, many positive results in terms of energy saving and performance improvement have been shown for many common applications through experiments. However, the benefits of offloading tasks to remote servers are still unclear in some realistic scenarios [11, 12]. This is due to the complex parameterization of offloading solutions regarding mobile applications, network conditions, execution platform, and cloud management [13].

There have been a number of surveys on the various issues of computation offloading in mobile cloud computing so far (cf. Table 1). In many early works [32, 14, 33], the authors present brief overviews of the concept and summarize basic techniques, issues, and solutions regarding to computation offloading. Besides, several studies have been published, which only focused on single aspect of computation offloading problem.

For example, Hoque et al. [34] provide a taxonomy of energy consumption profilers and models for mobile devices. A number of common algorithms for application partitioning are presented in [18]. Khan et al. [35] summarize many proposed strategies in task offloading for improving application performance. The adaption characteristic and various techniques to make the offloading system adapt to the changing of environment parameters are discussed in [13]. In [19, 21], the authors survey different challenges and solutions to many problems of computation offloading in cloud computing in terms of security and authentication. Sanaei et al. [15] discuss heterogeneity in mobile cloud computing, in which a taxonomy of heterogeneity roots as hardware, platform, feature, API, and network is described. In the last few years, with the idea to bring computation resources closer to mobile users, the offloading problem has been actively considering in the context of edge computing [30], or more specifically MEC [23, 28], instead of mobile cloud computing.

In addition, some surveys have been conducted for other similar remote infrastructures, such as fog computing [24] and D2D [25]. Just recently, there have been some studies focusing on the decision-making problem [29] and the intelligent approaches of task offloading [31]. In contrast to aforementioned surveys, we aim to provide a comprehensive guide on computation offloading for smartphones. Golkarifard et al. [22] presented a work closely related to

*Corresponding author.

*Email addresses:* nguyen@ccs-labs.org (Quang-Huy Nguyen[1]), dressler@ccs-labs.org (Falko Dressler[2])

*Article published in Elsevier Computer Communications 159 (2016) 133-154. (10.1016/j.comcom.2020.05.001)*

| Work | Topic |
|---|---|
| Kumar et al. [14] (2013) | Survey of computation offloading for mobile devices |
| Sanaei et al. [15] (2014) | Taxonomy and challenges in Heterogeneity cloud computing |
| Shiraz et al. [16] (2015) | Analysis on computation offloading techniques |
| Khan et al. [17] (2015) | Context-aware in mobile cloud computing |
| Liu et al. [18] (2015) | Application partitioning algorithm in cloud computing |
| Ali et al. [19] (2015) | Security in cloud computing |
| Ahmed et al. [20] (2015) | Seamless application execution in mobile cloud computing |
| Alizadeh [21] (2016) | Authentication in cloud computing |
| Bhattacharya and De [13] (2017) | Survey of adaptation techniques used by offloading systems |
| Golkarifard et al. [22] (2017) | Guide on Computation offloading |
| Mach and Becvar [23] (2017) | Survey existing concepts and user-oriented use cases in Multi-access Edge Computing (MEC) |
| Hu et al. [24] (2017) | Survey of fog computing and open issues |
| Jameel et al. [25] (2018) | Review proposed solutions and open issues in Device-to-Device (D2D) |
| Mahmud et al. [26] (2018) | Analysis, taxonomy, and directions of fog computing |
| Xu et al. [27] (2018) | Literature of opportunistic offloading |
| Peng et al. [28] (2018) | Survey of the MEC from the perspective of service adoption and provision |
| Wu [29] (2018) | Explore methods of multi-objective decision making for task offloading |
| Jiang et al. [30] (2019) | Literature review on computation offloading in the edge computing |
| Cao et al. [31] (2019) | Review of approaches for intelligent offloading in MEC |
| **Our survey** | Guideline on computation offloading in heterogeneous cloud computing |

Table 1: A summary of related works

ours; however, given the magazine style of this paper, the authors were not able to go as broad and deep.

Moreover, most existing works only focused on singular aspects of computation offloading problem, whereas the efficiency of the whole system is influenced by the combined effect of multiple parameters. Early works summarizing the issues and solutions to the offloading problem focused particularly on migrating intensive computation to the cloud has just been proposed to solve the resource shortage of mobile devices [32, 14, 33].

Thus, we found that it is necessary to provide a comprehensive survey on computation offloading for smartphones, which studies not only current challenges from different viewpoints but also their relationship. We systematically assess fundamental issues regarding computation offloading and survey the state-of-the-art solutions to different challenges in making this approach practical. Besides, while most of the previous survey papers put much effort on studying various issues of cloud computing in order to support computation offloading, we mostly focus on the problems existing on the smartphone side in the context of heterogeneous cloud computing.

Our main contributions can be summarized as follows:

- We show and categorize the "big picture" of computation offloading concepts and systems;
- we describe the most important components of an offloading system and highlight the open research issues to be coped with in task offloading; and
- we discuss several up-to-date techniques and algorithms applied to offloading systems in details.

The rest of this paper is organized as follows. Section 2 presents an overview of computation offloading and its main problems. Sections 3 and 4 discuss two of the most indispensable components in an offloading system, namely energy models and profilers, which support collecting operating data and assessing the efficiency of the system in terms of energy usage. We also provide a literature review on power/energy models for different hardware components on smartphones. An overview of some potential remote infrastructures for offloading tasks and their characteristics are described in Section 5. Section 6 explains the working principles of an offloading system and Section 7 discusses relevant frameworks, which facilitate the process of computation offloading. Section 8 highlights selected research challenges in implementing and deploying offloading systems and outlines some future directions for improving the efficiency and maximizing the benefits of offloading systems. Section 9 concludes this paper.

## 2. Computation Offloading Concepts

We first briefly outline the concept of computation offloading and present some main issues need to be solved to make it practical. Computation offloading is the process

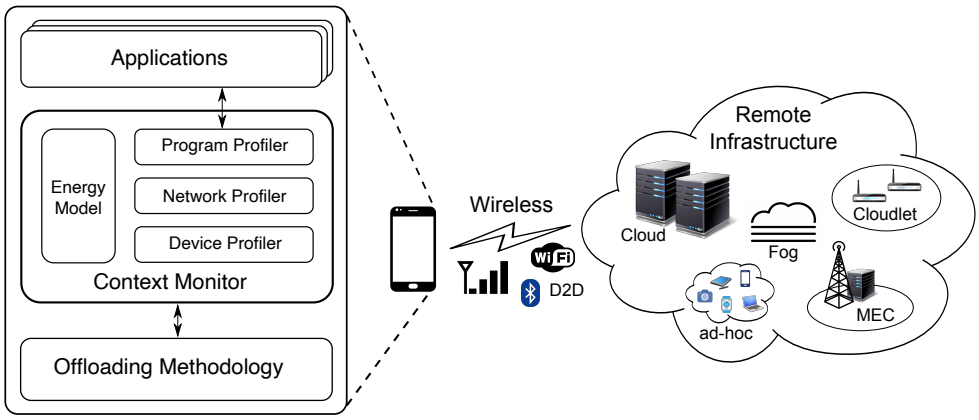


Figure 1: Architectural overview of computation offloading concepts.

of migrating intensive computation to remote servers in order to alleviate the resource limitations on local devices. Such systems, therefore, need certain setups in both local side and server side. Figure 1 depicts a general architecture for computation offloading. The computation offloading process covers the following main problems:

- *what to offload*: determine which methods, classes, threads, or applications to be offloaded;
- *where to offload*: select the infrastructures for offloading (e.g., to an Internet cloud, edge computing, a mobile ad hoc network, or cloudlets);
- *how to offload*: solve the problems related to offloading implementation and deployment; and
- *when to offload*: decide the suitable time for migrating intensive computation from smartphones to selected remote servers in order to maximize the offloading benefits.

To answer the *what* question, a software component called application partitioning is used to divide applications into offloadable and non-offloadable parts. The non-offloadable modules are then executed locally on the smartphone, while the offloadable ones represent candidates for the offloading process. The *where* question is related to Mobile Cloud Computing (MCC) and its services. Concerning the *how* question, developers need to solve many technical problems on how to install the software on remote infrastructures and how to migrate offloadable parts to the cloud for execution, e.g. which communication technologies are used or how to store/send the data and results before and after offloading. The *when* question is handled by the decision engine, which performs a certain logic (linear programming, Markov models, fuzzy logic, etc.) on profiling data received from the profilers to determine whether or not to offload candidates to the remote cloud. We discuss in details each component of a computation offloading system in the following.

## 3. Power/Energy Models

Power/energy models play an essential role in offloading systems, which are used to estimate the energy consumption of the end system. And further, they are also used to assess the energy efficiency of applications/offloading programs. A number of recent works developed power/energy models for different components on smartphones including the Central Processing Unit (CPU) as well as cellular and WiFi communication [36, 37, 38, 39, 40, 41]. Power models can be used by developers for better understanding and evaluating their applications in terms of energy efficiency [42, 43]. Despite the differences in power model construction, these works share the same underlying method for power/energy measurements. In the following, we first describe the methodology and various possible ways to measure the overall power consumption of a smartphone. We then present selected state-of-the-art approaches to model energy consumption of a smartphone and its components. Finally, we discuss some issues in building power/energy models of smartphones.

### *3.1. Power/Energy Measurement*

In practice, the energy consumption of a smartphone over a given period of time is calculated by doing the numerical integration of traced power during this interval. The power trace values are obtained through sampling voltage and current using specific hardware, which can be either external devices or self-equipped components on smartphones.

#### *3.1.1. External devices*

Several commercial and custom-made instruments are suggested to use as external devices to measure smartphone power consumption; we list selected examples in Table 2. Most of them have the capability of directly reporting both voltage and current drawn by the smartphone, which can be straightforwardly used to obtain the power consumption. For devices which only can measure voltage, the current value is computed through the voltage dropped over a

| Devices | Type | Features |
|---|---|---|
| Monsoon Power Monitor[1] [44, 4, 45] | Commercial | Current and voltage |
| National Instruments [46, 47] | Commercial | Voltage |
| Voltech PM1000+ [48] | Commercial | Voltage |
| Tektronix TDS1012B [49] | Commercial | Voltage |
| Battor [50] | Custom | Voltage |
| Arduino Duemilanove [51] | Custom | Voltage |

Table 2: External devices for energy measurement

sensing resistor connected in series with the power supply of the smartphone.

Commercial instruments are able to provide power data with high accuracy and precision but normally expensive, especially in such case that multiple ones are needed for simultaneous measurements. Moreover, they are also rather big and require a mains power supply. This makes them unsuitable for mobile measurements. Other self-designed devices have been proposed in order to surpass these drawbacks. They have a number of advantages: low cost and simple design so it can be rebuilt by everyone interested; flexibility to support a wide range of energy measurements; portability and mobility. However, external instruments, whether commercial or custom-made ones, are not feasible for large-scale deployment due to their inflexibility in measurement setup. They are usually configured with some specific settings for each operating mode and, therefore, are limited to laboratory usage.

#### *3.1.2. Self-Measurement*

Many modern smartphones have the capability to self-report their own battery status without the support of external devices [52, 41, 49]. This feature is supported by a self-equipped hardware component on smartphones, called the battery fuel gauge. The fuel gauge IC is associated with the smart battery interface of the Operating System (OS), such as the Android BatteryManager, which provides the battery information to applications running on smartphones. The updating rate of the battery status and available information depend on the fuel gauge type and the phone model (cf. Table 3). Some fuel gauges integrate a Coulomb counter, which enables the battery Application Program Interface (API) to estimate the current drawn by the smartphone, while the others only support voltage and battery State Of Charge (SOC) reporting. Besides, a number of battery-related properties could be also provided by the smart interface, such as the status, capacity, and temperature. Generally, the intervals for updating power data of smart battery interface are rather coarse, from hundreds of milliseconds to few seconds or even longer [49]. Therefore, the accuracy of the power estimation by means of self-metering is not as high as using external devices.

| Device | Fuel Gauge | Measure | Update Rate |
|---|---|---|---|
| Galaxy W | MAX17043 | SOC | - |
| Galaxy S3 | MAX17047 | Current | 175.8 ms |
| LG G3, G4 | MAX17048 | SOC | - |
| Nexus 6 | MAX17050 | Current | 175.8 ms |
| Nexus 10 | DS2784 | Current | 3.5 s |

Table 3: Fuel gauge chips of Android devices

However, using battery interface offers better flexibility and potential in performing the power consumption measurement for smartphones, especially for models with non-removable battery, where it is non-trivial to connect to external instruments. This approach is often proposed as the solution for online monitoring the battery usage for mobile systems.

All the measurement methods mentioned above only result in the total energy consumption of the whole smartphone, but lack of information about how much energy each hardware components of the device contributes to the overall value. Moreover, it is non-trivial to attach external devices to smartphones for energy measurements. This normally can only be done in laboratory environment [50, 48]. These limitations bring challenges to the analysis of mobile device operations from the perspective of energy efficiency. One of the most typical approaches to overcome these difficulties is to use a power model, which represents the power dissipation of a smartphone under a function of different subcomponent variables [38]. Each selected variable reflects the impact of a particular hardware component on the power consumption. A power model should cover most of the subcomponents on smartphones that are commonly used in different mobile applications, such as CPU, memory, Global Positioning System (GPS), Graphical Processing Unit (GPU), and network interfaces. Each component has its own power consumption characteristic, which varies according to the changes of relevant parameters. For example, while the CPU power consumption depends on the computation load and the clock frequency, the power consumption of network interfaces is affected by the many factors like protocols, traffic load, or the amount of data transfer. The developed power model can then be used in power estimation process to predict the power/energy consumption of the whole smartphone or a particular application running on it, without the support of power measurement.

There are several ways to characterize power and/or energy consumption of a component and there are also various strategies to construct the power model. We now describe the characteristics of some most important hardware components and examine related energy models.

### *3.2. CPU*

CPU is one of the most power-hungry components in a smartphone [36, 47, 53]. As almost every operation on the phone is carried out under the control of the CPU, it

is vitally important to examine and understand its power consumption. Most of the recent power models for the CPU are utilization-based approaches, which correlate the power draw of the CPU with measured resource usage.

In [54], the authors proposed a performance-counter-based power model for CPUs. The hardware performance counters are coordinated with real-time total power measurements from the external multimeter to provide per-subcomponent power estimation of the CPU. These counters are accessed using Linux Loadable Kernel Modules (LKM) and their values imply the number of CPU events triggered during the execution. The operation of a subcomponent can be associated with one or more different events, in order words, one or more performance counters. The higher the values of one component's counters, the more accesses to that component have been made, and the more power has been consumed as well. Generally, the power model for a single component is represented as

$$\begin{aligned} P_{C_i} = &\text{AccessRate}(C_i)\times \\ &\text{ArchitecturalScaling}(C_i)\times \\ &\text{MaxPower}(C_i)+ \\ &\text{NonGatedClockPower}(C_i)\ , \end{aligned} \tag{1}$$

where $C_i$ means the $i^{th}$ component; the access rate is used as a weighting factor, which is obtained either directly by a performance counter or indirectly by a combination of performance counters; the maximum power and clock power values are estimated empirically during the implementation; and the architectural scaling is depending on the micro-architecture of the CPU. This model can be applied to 22 different subunits of the Intel Pentium 4 processor, such as L1 and L2 caches, L1 and L2 branch prediction units, integer and floating point executions, and bus control. The total power consumption is the sum of all sub-component powers, along with a fixed power baseline when the processor is in the idle state.

Another very common utilization-based approach to model the power consumption of a CPU is to represent the dependence of its power on workloads or CPU usage. In this kind of model, developers normally assume a linear relationship between the power consumption and CPU loads; thus, a linear regression power model is constructed [40]. Generally, the power function for a single-core CPU can be given as

$$P_{\text{CPU}} = a \times U_{\text{CPU}} + b\ , \tag{2}$$

where $a$ and $b$ are the model parameters and $U_{\text{CPU}}$ is the CPU utilization. The development of the power model comprises the following steps: (1) a training program is used to generate different CPU loads for the smartphone; (2) the total power consumption of the smartphone is measured using one of the methods mentioned in Section 3.1; (3) a power model for the CPU is created using the linear regression. In order to improve the accuracy of the model, the CPU workloads should imitate real-life scenarios and all irrelevant components like display, network interfaces, and GPS, should be turned off during the training phase to avoid polluting the measured total power consumption.

However, this power model does not take into account the CPU frequency, which is also a very important factor that highly influences the CPU power. In [47], it was shown that a CPU operating at different frequencies has different power distributions. The power model can, therefore, be further improved by constructing a linear model for each available operating frequency [38, 55, 45, 41]. Another approach to reflect the impact of CPU frequency on power consumption is to consider this parameter as an additional predictor besides the workloads [56]. This method is particularly efficient for CPUs that have multiple cores and support a high number of operating frequencies, where it is too expensive to create a power model as in Equation (2) for each CPU core and frequency. The general power model can be represented as

$$P_{\text{CPU}} = \sum_i c_i \times f_i \times u_i\ , \tag{3}$$

where $c_i$, $f_i$, and $u_i$ are the coefficient, frequency value, and CPU utilization of the $i^{th}$ core, respectively.

In [47], based on the experimental results related to the CPU measured power consumption, the authors analyzed the effect of Dynamic Voltage and Frequency Scaling (DVFS) on the overall energy after the execution. DVFS is a technique that allows a CPU to automatically adjust its frequency on-the-fly depending on the computational demands. Even though it was shown that DVFS helps to significantly improve the power efficiency of the CPU, its actual benefits to the total energy are still debated. This is because there is a trade-off between operating frequency and the execution time. Lower CPU speed also means higher running time of the task and more time spent in the idle state. Thus, the final energy consumed by a CPU heavily depends on its idle power and the workload. This relationship has been introduced as

$$E = P \times t + P_{\text{idle}} \times (t_{\text{max}} - t)\ , \tag{4}$$

where $E$ is the overall energy, $P$ is the average power over the run-time of the workload, $t$ is the run-time, $P_{\text{idle}}$ is the CPU idle power, and $t_{\text{max}}$ is the maximum run-time of the workload over all frequencies.

The work presented in [57] went further on CPU power modeling. It was argued that existing models have several limitations in estimating the power consumption of multi-core CPUs. These models only consider the effects of frequency and utilization of the CPU, which were shown to be inappropriate and give high estimation errors for multi-core smartphones. The experimental results expose a high variation in power consumed by a CPU for different workload patterns even under the same frequency and utilization. Therefore, the authors proposed a new power model for multi-core CPUs with better estimation accuracy, which takes idle power states into account. A CPU may have many idle power states, which draw different power

consumptions. The developed power model takes the time spent in each idle state as a new predictor besides the utilization. The details of the power function for a single-core CPU working at frequency $f$ is represented as

$$P_{\text{core}} = \sum_i \beta_{C_i} \times WED_{C_i} + \beta_U \times U + c \ , \qquad (5)$$

where $WED_{C_i}$ is weighted average entry duration of idle state $C_i$, $\beta_{C_i}$ and $\beta_U$ are the coefficients of $WED_{C_i}$ and the utilization $U$, and $c$ is a constant. Then, the power function for a multi-core CPU is given as

$$P_{\text{CPU}} = P_{BL,N_c} + \sum_i^{N_c} P_{\Delta,\text{core},U_i,f_i} \ , \qquad (6)$$

where $N_c$ is the number of cores enabled, $P_{BL,N_c}$ is the baseline CPU power with $N_c$ enabled cores, and $P_{\Delta,\text{core},U_i,f_i}$ is power increment of core-i when it is working at frequency $f_i$ with utilization $U_i$. For each frequency $f_i$, $P_{\Delta,\text{core},U_i,f_i}$ can be predicted using the single-core power model as in Equation (5), while $P_{BL,N_c}$ is a constant value that can be measured beforehand. $P_{\Delta,\text{core},U_i,f_i}$ are modeled separately for the cases when only one and multiple cores are online because of their different sets of CPU idle states.

There were still some other works that also studied other power or energy consumption aspects of multi-core smartphones [58, 59]. In [60], by analyzing the behavior of a broad range of applications and measuring the Thread Level Parallelism (TLP), the authors examined the potential of smartphones applications in exploiting parallelism and their energy efficiency in multi-core architecture. It was shown that the increment in CPU cores is not proportional to the gain in power consumption of applications. Similarly, the disproportionate energy consumption by different numbers of active CPU cores was also exhibited in [61]. The paper agreed with the statement that multi-core CPU architecture can achieve high energy efficiency when dealing with high level of parallelism; however, most of the smartphone applications fail to take full advantage of all available CPU cores because of the lack of parallelism support in their implementation [62, 60]. Another interesting observation is that the energy consumption for activating the first core of a processor is typically more than double the energy for activating each additional core of that processor. This is due to the hardware sharing between cores of the processor, which was also exposed to have great influence on the idle states of CPU.

### *3.3. Network Interfaces*

In order to perform computation offloading, network communication is a mandatory part; its energy consumption plays one of the key roles in making offloading decisions. WiFi and Cellular (3G and 4G) are currently the most dominant types considered in computation offloading. Understanding the power behavior and characterizing the power/energy consumptions while performing network operations have attracted much attention of many early works: WiFi [63], cellular [64, 65], or both of network technologies [66, 44, 67, 68]. In these papers, based on the power trace collected during data transmissions, the influences of different parameters on network communication have been examined. However, similar to other components, directly measuring the power/energy consumed by network interfaces is normally infeasible. Models for estimating their power/energy consumption, therefore, are needed.

#### *3.3.1. General Models*

There are several approaches that can be applied to both WiFi and cellular networks. The work presented in [44] suggested modeling the energy consumption of a network interface as a simple linear function of the data transfer size. Similarly, the model presented in [48] is a simple linear function with transfer size as a predictor and a coefficient obtained by using both linear and quantile regression. In [69], the authors carried out a study on the impact of wireless signal strength on battery drain. The experimental results reveal that smartphones consume much extra energy consumption for data transferring in the poor wireless network conditions. New power models for WiFi and 3G have been developed taking the wireless signal strength into consideration.

The above-mentioned modeling methods can be referred to as statistical approaches, in which the power consumption is estimated based on its relationship with one or more network variables. Another very common approach for modeling the power/energy of a network interface is based on its power Finite State Machine (FSM) [39, 70, 65, 71]. This approach assumes that each power state has a constant power drain. The total energy consumption $E_t$ of a network interface over time $t$ is the sum of the energy costs that it spends in each power state and the overheads spent on switching between the states [53]. It can be calculated as

$$E(t) = \sum_j E_j(t_j) + \sum_j \sum_k E_{j,k} \times C_{j,k}(t) \ , \qquad (7)$$

where $t_j$ is the duration spent in power state $j$ and $t = \sum_j t_j$, $E_j(t_j)$ is the energy spent during $t_j$, $E_{j,k}$ is the overhead caused by the transition from power state $j$ to $k$, and $C_{j,k}(t)$ is the number of times this transition has occurred during $t$. None of these models, however, takes network operation or changing topologies into account.

The combination of these two above approaches has also been used in many works [38, 68]. This hybrid technique uses a linear power function for each state in the power FSM. Although using similar approaches, the power model for each network interface has its own characteristics due to the differences in architecture and operation. We discuss in details power models for each network interface in the following.

#### *3.3.2. WiFi*

First, several works have studied the relationship between the power consumption of WiFi interface and different network variables using statistical methods. It has been

shown that the network contention existing in multiple Access Points (APs) context has a high impact on the power drawn by the WiFi interface [72]: The higher density of APs, the longer client devices have to wait for their desired AP to send packets and, therefore, the more energy they consume. Experimental results presented in [73] generally reveal a linearity between the throughput and power consumption of the WiFi interface on smartphones. Similarly, Sun et al. [74] studied the properties of WiFi based on parameters that are readily available to smartphone applications, such as packet loss rate, signal strength, transfer size, and throughput. Based on these insights, many authors use linear regression approach for modeling the WiFi energy. The models estimate the power consumption based on different network variables like transfer size [44], data rate [38], signal strength [69], throughput [41, 74], transmission time [75].

Second, a lot of recent works suggest modeling power/energy consumption of WiFi interface based on its power state machine during data transmissions [38, 39, 40]. These models can offer satisfactory accuracy for the WiFi communication in a stable condition, in which most of the parameters are not highly varying. However, this is also a major drawback for modeling the WiFi power/energy consumption in real scenarios, where different factors such as fading, interference, collisions, can affect the characteristics of the dynamic wireless channel. Depending on the division of power states, the WiFi power model may have different complexity.

According to PowerTutor [38], the power consumed by WiFi interface could also be modeled to simply be either in high or low level. The power consumption of low power state only depends on the baseline power corresponding to the low power state of the WiFi interface, and the one of high power state is calculated based on the high state baseline power, the uplink channel rate (ranging from 1 Mbit/s to 72 Mbit/s), and the packet rate. While the baseline powers are constant and hardware-specific, the uplink channel rate and the packet rate are collected and calculated using the logging information from the phones during runtime.

Li et al. [76] proposed another WiFi power model, which considers the WiFi hardware operation patterns in details to improve the accuracy of the developing model. Generally, most of the works only determine two main WiFi power states, namely Continuously Active Mode (CAM) and Power Saving Mode (PSM). The authors, however, suggest a more fine-grained power state division including the following states: IDLE, Deep Sleep (DS), Packet Reception (PR), Packet Transmission (PT), and Light Sleep (LS), which is due to the asymmetry between the uplink and downlink operation. A transition between two states is triggered by both the packet rate and the packet inter-arrival time. A similar idea is presented in [70], however, mainly focusing on the PSM of an IEEE 802.11g WLAN. The transmission process in this mode can be divided into four states: IDLE, SLEEP, TRANSMIT, and RECEIVE. The model takes different parameters into account that determine the Internet traffic, including burstiness, size, duration, and average data rate.

Finally, many WiFi power models have been developed based on MAC and PHY layer features [77, 78, 79, 80, 81, 82]. The accuracy of such approaches is rather high; however, the lack of support for these features at driver/firmware level of smartphones limits the ability to construct the WiFi power models using this approach [83, 73].

### *3.3.3. Cellular 3G*

Like WiFi, the power/energy consumption of 3G interface can also be modeled using statistical methods such as linear regression on different network variables [44, 48]. However, it is more popular to rely on the power FSM to construct the energy model. The state machine for 3G transmission is defined by the Radio Resource Controller (RRC) protocol, which controls the operation of the 3G network [84]. Each state has a particular specification of radio resources associated with it and consumes a certain amount of energy. The RRC state machine consists of four states: CELL_DCH, CELL_FACH, CELL_PCH, and IDLE. In CELL_DCH state, the 3G interface uses a Dedicated Channel (DCH), which offers the best network throughput, to transfer data in high traffic conditions. Switching the state to CELL_FACH and to CELL_PCH are performed when data traffic is low, and when there is no active data transmission, respectively. And if no connection exists, the RRC switches the state to IDLE for energy saving. In terms of power consumption, CELL_DCH obviously draws the highest power, CELL_FACH is roughly 50 % of that in CELL_DCH, and CELL_PCH and IDLE use almost the same amount of energy, which is about 1–2 % of the power consumed in CELL_DCH [49]. All state transitions from a high-power to a low-power state are controlled by their own inactivity timer.

In [38, 68, 56], a power model for a 3G network interface was presented that is based on the transmission power FSM. Generally, 3G power is estimated as

$$P_{3G} = \sum_i \beta_i \times p_i \ , \tag{8}$$

where $\beta_i$ and $p_i$ are the coefficient and parameter of the $i^{th}$ power state. The power parameters are derived by using the linear regression on the data collecting from the transmission of packets with different sizes and intervals between a smartphone and servers via the 3G interface.

Besides that, some additional aspects of 3G communication have been studied. In [85], it has been shown that the energy consumption of cellular network is highly affected by the data traffic. Balasubramanian et al. [44] focus on the impact of tail energy on the overall energy consumption of smartphones when performing network operation. All of these parameters should be also taken into account in the constructed model.

*3.3.4. Cellular 4G*

Currently, 4G networks are considered state of the art with broad coverage. Even though there is a significant improvement in transmission speed and bandwidth of 4G technologies, its potential for being used in computation offloading is still a challenging question. Most previous works have been done in order to investigate the features of a Long-Term Evolution (LTE) network as well as the parameters that influence its performance and power or energy consumption [86, 87, 71]. Similar to 3G networks, the operation of 4G LTE is controlled by an RRC, however, there are only two RRC states: RRC_CONNTECTED and RRC_IDLE [88].

In [71], the authors studied the energy usage of a commercial LTE network and developed a high accuracy model (6 % error rate) for estimating its power consumption. The 3G and WiFi power data collected from real users was embedded into LTE model simulation framework for comparison. The experimental results showed that the LTE network has higher power efficiency compared to 3G, however, it still lacks behind WiFi for bulk data transfers. In case of small data, LTE has been shown to consume more power than 3G and WiFi. Other parameters that influence the LTE energy consumption have been examined: the tail time or the power of the tail state is considered the key factor that determines the energy usage and performance of LTE.

The power consumption for data transfer via LTE can be estimated using a linear model [71, 89] as

$$P_{\text{LTE}} = \alpha_u t_u + \alpha_d t_d + \beta \ , \tag{9}$$

where $t_u$ and $t_d$ are uplink and downlink throughputs, which are calculated within a time window $W$ seconds; $\alpha_u$, $\alpha_d$, and $\beta$ are coefficients of the power model.

Besides, in [90], the output power levels of 4G User Equipment (UE) are estimated for different environments including rural, suburban, urban, and indoor. The results show that the output power values are significantly below the maximum possible power of the UE, i.e., 23 dBm (200 mW), with the time-averaged values being less than 2.2 % of the maximum for rural environments and less than 1 % of the maximum for other areas.

Approaching from service-based perspective, an investigation on energy consumption characteristics of LTE communication is presented in [91]. The authors measured the average 4G energy consumption for different services on smartphones, such as web browsing, video play, cloud upload/download, or Virtual Reality (VR) video, which confirms LTE communication is the main consumer for heavy data application over the network. The results also show a potential reduction of over 80 % in energy consumption when performing offloading to edge computing.

*3.3.5. Cellular 5G*

Despite the evolution of 4G LTE systems, which provide a higher capacity compared to 3G networks, next-generation 5G networks are already at the horizon [92]. With several advantages such as higher capacity, lower latency, and lower power consumption when compared to 4G, 5G systems are expected to offer more benefits and take full potential of computation offloading. In the early stages, the research was mainly focused on understanding the energy efficiency of 5G networks [93, 94]. The application and energy modeling for computation offloading have been actively studied in the past few years [95, 96, 97, 98]

Comparing to 4G network, 5G technology also makes use of the radio spectrum between 30 GHz and 300 GHz (mm-waves). The energy consumption of 5G mm-wave cellular network has been studied in many previous works [99, 100]. D2D communication, another aspect of 5G systems, is also an essential part beside Device-to-Infrastructure (D2I). The power consumption of different application scenarios for D2D was investigated in [101]. More details, based on the recent models of 3GPP standardization, the author analyzed the power consumption from both the infrastructure and user device perspective.

*3.4. Discussion*

Constructing power/energy models is one of the most challenging phases in an offloading system. The accuracy of these models depends on many factors. First, different smartphones have different power characteristics, and therefore different power coefficients. A specific model is normally used for only one or a few certain smartphones. Moreover, due to the limitations on system accessibility, some information only can accessed with root access, which is normally not available to users. Models developed based on such information can only be used by developers and are not applicable in practice.

Second, the values and scopes of training data also determine the accuracy of power/energy models. Generally, the training data set is constructed to be as close to reality as possible. This task, however, is non-trivial and labor intensive to cover a sufficient range of operating conditions. As a result, many power/energy models suffer from errors in assessment values.

## 4. Profilers

Offloading tasks to the cloud may improve performance and save energy on the smartphones. However, the actual benefits of an offloading process depend on many parameters, which involve smartphone hardware, running applications, network conditions, and cloud infrastructures. In order to make accurate offloading decisions, it is essential to collect all necessary information for offloading process in advance and at runtime as well. There are three main types of profilers, including device profiler, program profiler, and network profiler. These software programs do not only monitor the parameters of the operating environment (the surrogate and the network) but also the available resources of the mobile device. Besides the accuracy of the profiling data, the profilers need to be lightweight to minimize the overhead incurred.

| Data source | Information |
|---|---|
| /proc/meminfo | Allocation of main memory |
| /proc/stat | CPU usage |
| /sys/devices/system/cpu | Frequency and statistics for CPU cores |
| /sys/class/net/ | Wireless interface use |
| /sys/class/leds/lcd-backlight/brightness | Backlight status of screen |
| /sys/class/power_supply/battery | Battery state |

Table 4: Android virtual files providing hardware statistics; this is only a selection and by no means an exhaustive list.

### *4.1. Device Profiler*

The device profiler is used to collect the information related to smartphone hardware at runtime and feeds them into the pre-built energy models. Applications running on the phone may utilize various hardware components and, as mentioned in Section 3, each of these components may have its own power/energy model. Depending on the properties of the model, different parameters can be used for the energy estimation, such as the brightness of the Liquid Crystal Display (LCD), the pixel colors of Organic Light-Emitting Diode (OLED), the frequency and utilization of the CPU, the power state and signal strength of WiFi or cellular network interfaces. All necessary state information of relevant components has to be monitored by the device profiler.

On Android smartphones, the Linux kernel is normally orchestrated to make hardware information available via a virtual file system [38, 7, 102]. In particular, Android provides information of several hardware components under /proc and /sys directory. We summarize some of the most relevant virtual files in Table 4. Other parameters, such as signal strength of wireless networks and network types, can be retrieved by using appropriate Android API calls.

Generally, the update intervals of a component's parameters have a strong influence on the accuracy of the profiling data. Therefore, it also affects the accuracy of the power estimation for that component.

### *4.2. Program Profiler*

The program profiler records information concerning the execution of software modules. While the device profiler collects the runtime values of parameters, this type of profiler, on the other hand, mainly logs the statistical information after the given methods, threads, or processes are completed. For example, the logging information can be the overall execution time, the per-thread CPU time of a method, the number of instructions executed, the number of method calls, or the size of allocated memory [7]. Some information, such as duration and CPU cycles, is used for estimating the energy consumption of the running method [4, 10]. These calculated values combined with other profiling data that related to the performance of the running application will be used as part of the input data set to improve application partitioning and offloading decision for future invocations.

In general, instrumental data of each method, thread, or process may highly vary among different execution times, which is due to the non-deterministic behavior of applications on smartphones. For example, different execution paths of program code and the changing context at runtime can lead to high variation in profiling information. For instance, the execution time duration can differ from ones of the previous invocations.

### *4.3. Network Profiler*

The network profiler is responsible for characterizing the network conditions during the operation of the system. A number of parameters have to be monitored such as available network interfaces, network Round Trip Time (RTT), amount of data sent/received in a period of time, received signal strength [69], and uplink channel rate and uplink data rate [38]. These information reflect the current network performance. Already very simple techniques help improving the estimation of the network capabilities. In [4], the authors propose a method to estimate the average throughput of a wireless link by sending 10 kB of data over TCP to the server and measuring the transfer duration. This helps to take into account both the latency and bandwidth characteristics.

### *4.4. Discussion*

Profiling builds directly on top of power/energy models. As mentioned above, the application behavior is non-deterministic at runtime. This brings difficulties to estimate the execution data of offloading candidates, such as running time, available memory, and communication cost. Furthermore, profiling is typically not non-invasive. The more accurate the profiler gets, the more sophisticated its program is and the more computation and energy costs are spent on collecting profiling data. Therefore, it is essential to take into account the accuracy-cost trade-off when building the profilers.

## 5. Remote Infrastructure

Cloud computing, with high computation power and storage capacity, has been used as a centralized processing infrastructure for computation offloading [103, 7]. However, there is a bottleneck of cloud computing that hinders the application of computation offloading, which is the network bandwidth. This is especially critical issues in many applications require real-time response and mobility support. Different cloud architectures, namely cloudlet [104, 105, 106], MEC [107, 108], ad-hoc clouds [109, 110], and fog-computing [111, 112], have been proposed as complements to the cloud computing for computation offloading. The

main objectives of these architectures are: (1) to reduce the amount of data sent to the cloud computing; (2) to decrease network latency; and (3) to improve system response time in real-time applications. We discuss each architecture in the following.

### *5.1. Multi-access Edge Computing*

One of the early approaches to bring computation resources closer to the end-user devices is to use Multi-access Edge Computing (MEC) [113, 108]. MEC provides cloud computing capabilities within the radio access network that enables mobile devices to offload their heavy tasks directly to the servers on the edge of the mobile network rather than in the core network that are far from the mobile devices. The introduction of this concept has great potential to liberate the mobile devices from intensive workloads and cope with the delay problem, thus enabling to run applications that require a huge amount of resources but still ensure time constraints at UEs.

Technically, edge computing offers only limited computational power and storage capability compared to cloud computing. On the other hand, it can be deployed in distributed manner, which helps to reduce the load on a certain centralized server as in the case of cloud computing. One of the notable characteristics of MEC is the exploitation of Small-Cell Network (SCN) for computation offloading with the coordination between fonthaul and backhaul networks [96, 96]. The combination of several tiers of small base stations like Micro Base Stations (MBSs) and Femto Relay Base Stations (FRSs) makes MEC a more flexible infrastructure for offloading, but along with it is an increase in the complexity of the optimization problem.

Similar to conventional mobile cloud computing, decision on computation offloading from a UE to MEC is primarily based on two factors: the energy efficiency and delay constraint. Several existing works have been showed a significant reduction in latency of MEC-based systems with respect to cloud computing [114, 115, 116], which makes MEC more suitable for low-latency applications like gaming or virtual reality. The energy consumption issue for MEC has been also studied vigorously [117, 118, 96, 119]. Besides, the allocation of computing resource and mobility management are also important aspects of MEC [120, 23].

Today, computation offloading has also been studied for application to Intelligent Tranportation System (ITS) as well, in which processing on smartphones of Vulnerable Road Users (VRUs) can be offloaded to the cloud or MEC [121, 122]. This concept could be extended to micro-cloud solutions [123, 124].

### *5.2. Cloudlet*

According to the original idea, cloudlet is defined as a trusted, resource-rich mobile computing, well-connected to the Internet and can be accessed by nearby mobile devices via Wireless LAN (WLAN) network [104]. Thus, cloudlet can offer good local services including data processing and storage, and lower the latency by reducing the amount of network transmission [125]. It was shown in [105] that cloudlet can be beneficial for many real-time constraint applications such as augmented reality. However, only accessing through WiFi makes cloudlets less suitable for ubiquitous computing and scalable services. An extended architecture of cloudlets integrated with MEC was proposed in [126], in which the gap of cloudlet systems can be filled by the potentiality of MEC that could offer better coverage area for the mobile users and help to reduce computation costs in terms of power and delay.

### *5.3. Mobile Ad-Hoc Clouds*

Another approach that has also attracted the attention of the research community is to offload intensive computation to neighbor devices. A number of mobile end-user devices could collaborate to form ad-hoc clouds, in order to perform a substantial amount of computation tasks [127, 25]. Different wireless technology could be used for D2D communication, such as Bluetooth, Zigbee or RFID [25], WiFi Direct [128], Near Field Communication (NFC) [110], or Dedicated Short Range Communication (DSRC) [129]. D2D in cellular network was introduced in the fourth generation but the application is still limited. With the emergence of 5G technology, computation offloading through D2D communication has become more realistic due to its ability to support high data rate and low latency [130]. The idea of combining D2D communication with MEC is also a promising direction in order to bring more computation capacity to the offloading system, especially with the support of 5G wireless networks [131, 132, 133]. Several existing works have been focused on different aspects of D2D-MEC-based computation offloading like the energy efficiency [134, 135] or security issue [136].

### *5.4. Fog Computing*

Fog computing has also been proposed as a complement to mobile cloud computing. However, unlike MEC which offers computation and storage in the base stations of cellular networks, fog computing presents a computing layer, which is a collection of ubiquitous devices. This paradigm was supposed to enable a processing of the applications on a large amount of connected devices at the edge of network [137]. Technically, fog computing is based on decentralized model consisting of Fog Computing Nodes (FCNs), which support computing and storing data from end devices and forwarding to the cloud.

One of the distinguishing characteristics of fog computing when compared to other architectures is that it offers more flexible choice of devices used as intermediate nodes, i.e., FCNs, such as routers, switches, access points, Internet of Things (IoT) gateways as well as set-top boxes [138]. This heterogeneity of FCNs opens up the possibility of supporting devices at different protocol layers as well as non-IP based access technologies [24]. The connection from UEs

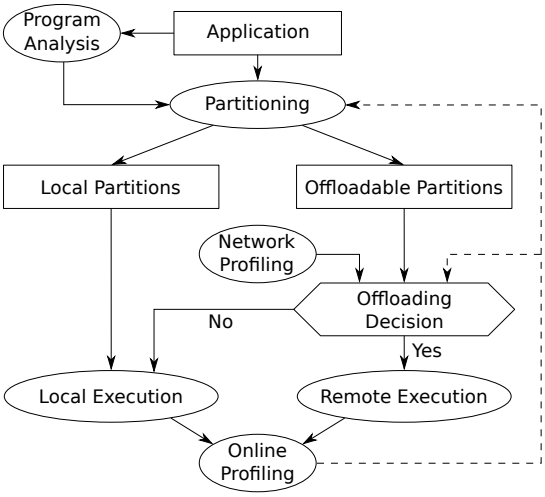


Figure 2: Computation offloading process.

to FCNs can be established over different mediums like WiFi, Bluetooth, Zigbee, Mobile Radio Networks. Many of these technologies, e.g., Bluetooth and Zigbee, are energy and memory efficient communication protocols, which are suitable for constrained devices with limited battery and storage capacity to connect and offload their computation.

Regarding use cases, fog computing has been considered a key infrastructure for IoT and big data applications [139, 140] as it enables the interconnection and intercommunication of a very large number of nodes. Despite lower computing and storage power than that of MEC, fog computing offers low latency and location awareness due to the widespread distribution of computing devices. Besides, fog computing could be the basics for many other applications in ITS, smart grids, or Augmented Reality (AR), which require real-time responses along with context-awareness and mobility [141, 142, 143, 144]

The feasibility of computation offloading scheme based on fog computing, like the aforementioned infrastructure, also depends mainly on the energy efficiency [111] and latency [145, 146]. Typically, these two parameters are taken into consideration when formulating the optimization problem for computation offloading decision and resource allocation [112, 147, 148, 149]

Generally, depending on the types of application, flexibility, computation and storage capabilities needed, an appropriate infrastructure or a combination of different architectures could be used for computation offloading. Comparison on these infrastructures is discussed in more details in [138, 24].

## 6. Computation Offloading Methodology

Figure 2 depicts the big picture of the computation offloading process, which consists of these following main phases: application partitioning, making offloading decisions, and local/remote execution. In this section, we discuss each phase in more detail. Besides, we also describe the roles of some other software components that are involved in the offloading process, such as program analysis, network profiling, and online profiling.

### *6.1. Program Analysis*

Program analysis is the initial step in the whole offloading system. This process provides preliminary information of offloading candidate applications in terms of resource usage and the relationships between components of the applications as well. These information could be the amount of energy consumption, running time, data exchange, memory cost, code size, etc., that is used as the knowledge base for the later optimization process (partitioning). Normally, instrumented applications are executed on one or multiple platforms and in different operating conditions. Then, profiling is carried out on every component (module/bundle/method/service) composing those applications to measure necessary data.

One of the most typical methods to analyze and model an application running on smartphones is to use a call graph, which is a Directed Acyclic Graph (DAG) that represents the relationship between the computational components of that program [150, 151, 152, 153, 154, 155, 156, 157, 158, 159]. Normally, the nodes of the graph represent the procedures/functions of a program and the edges represent the communication/invocations between them.

Depending on the characteristics of applications and offloading objectives, the call graphs may have different parameters. In [152], the authors present an approach to solve the optimization problem of energy consumption in computation offloading using parametric program analysis. The values of parameters including workload and communication cost at runtime decide the partitioning decisions. Task Control Flow Graph (TCFG) is used to represent the program execution and the issue of finding optimal partitioning can be reduced to the optimization problem for this graph.

Giurgiu et al. [153] presented the method of abstracting mobile application behavior as a consumption graph. Each node of the graph represents a software module of the given application, which is attached with two parameters: consumed memory and code size. Each edge of the graph represents the control dependencies and data paths between two modules. The graph is then used as the input of partitioning process. In [4], the execution behavior of the program at high level is modeled as an annotated call graph, which represents the call stack as the program executes. Each vertex represents a method in the call stack, and each edge represents an invocation of a method from another method. In [158], the authors construct the Weighted Object Relation Graph (WORG) of the mobile application using both the data from static analysis and dynamic profiling. Then the partitioning models that take the bandwidth variation into consideration are built with the goal of optimizing the execution time and energy consumption.

*6.2. Application Partitioning*

In general, it is impossible to offload the entire application to the cloud due to the frequent user interaction of mobile applications, which can only be performed locally at the smartphone. Additionally, the communication cost for outsourcing the whole application to servers is significantly high. Therefore, it is necessary to identify which parts of program code could be offloaded to the cloud and which parts can only be locally executed. Depending on the runtime conditions and offloading goals, offloadable partitions can be migrated to the clouds for distributed processing or retained on the local device for execution.

There are several approaches to categorize Application Partitioning Algorithms (APAs). Khan et al. [35] proposed a taxonomy of APAs based on their strategies to break out application execution components into non-offloadable and offloadable modules. A partitioning algorithm could be classified into either of two categories: (1) *static* [160, 161, 162, 158, 163]: when it is fixed at development phase based on the static analyzer and dynamic profiling; and (2) *dynamic* [5, 103, 156, 164, 165]: when it includes runtime information from different profilers, log files, and the interaction among components of the running application. While the resulting partitions of static algorithms do not change in all executions, those of dynamic algorithms could be updated before each execution to adapt to environmental conditions and optimize offloading objectives. Because of that, dynamic algorithms may have high overhead during the offloading process. In [35], these two categorizations are correlated with the goal of performance enhancement.

A more general study on APAs was provided in [18]. The criteria used for classifying algorithms include partitioning granularity, objectives, model and programming language support, profiler used, allocation decision, analysis technique, and annotation. Examples of static APAs that based on pre-specified annotation were presented in [7, 162]. Developers annotate the candidate methods for offloading as *remoteable* and leave the remaining to the framework. The server-executable versions of methods with annotation are automatically generated during the employment. From the perspective of partitioning models, there are also different strategies to address the partitioning problem: (1) *Linear Programming (LP) model* [163, 166]; (2) *Graph model* [153, 158]; and (3) *Hybrid* [5, 160, 167, 168, 169].

The LP for application partitioning can be modeled as follows [4]:

$$\begin{aligned} \text{maximize} \quad & \sum_{v \in V} I_v \times E_v^l - \sum_{(u,v) \in E} |I_u - I_v| \times C_{u,v} \\ \text{subject to} \quad & \sum_{v \in V} ((1 - I_v) \times T_v^l + (I_v \times T_v^r)) \\ & + \sum_{(u,v) \in E} (|I_u - I_v| \times B_{u,v}) \leq L \\ & I_v \leq r_v, \forall v \in V. \end{aligned}$$

- $u, v$: methods
- $I_v, I_u$: indicator variables, $I_v = 0$ if method $v$ is executed locally and 1 if remotely
- $r_v$: parameter that indicates annotation for method $v$, 0 if local, 1 if *remoteable*
- $E_v^l$: energy to execute method $v$ locally
- $T_v^l$: time to execute method $v$ locally
- $T_v^r$: time to execute method $v$ remotely
- $B_{u,v}$: time to transfer necessary program state (when $u$ calls $v$ - edge $e = (u, v)$)
- $C_{u,v}$: energy cost for transferring the state when $u$ calls $v$

With applications that can be represented as call graphs, partitioning is to seek for an optimal cut on the graph built in the analysis phase, which optimizes (maximizes or minimizes) a given objective function (consumed memory, latency, and data traffic). Call graphs have, for example, been used in [153]. The authors developed partitioning algorithms that work on the consumption graph of the running applications. Both static and dynamic partitioning strategies were considered. For the static case, the problem of identifying the local and offloading modules is solved in the offline phase. Different parameters like the mobile platforms and network conditions are taken into account and the `ALL` algorithm is proposed to find the optimal solution. For the dynamic case, the partitioning is performed on-the-fly and the `K`-step algorithm is applied. While the `ALL` algorithm examines all possible configurations, the `K`-step works on a reduced search space, therefore only finds a local optimum, but offers higher speed.

In the other hand, there are many applications, functions, or methods that are short-lived and not sensitive to the state of computation. Basically, it is much easier to parallelize stateless computation since there is no modification in data structure and data races between parts running in parallel. Besides, due to short-lived time, stateless jobs do not require a full life-cycle management and the persistent allocation of resources. In [170, 171], the authors propose solutions to enable the offloading of stateless tasks to remote servers. To identify which tasks should be offloaded to server, a simple stateless programming model is used [172].

In Table 5, we show a comparison of selected partitioning algorithms that fulfill the target of extending the battery life or increasing the energy efficiency of smartphones. The attributes to be compared include the under-test applications, the approaches for modeling and partitioning applications, and the parameters that are taken into account when considering the partitioning problems.

| Name/ Author | Year | Granul. | Core Component | Partitioning Approach | Parameters | Candidate Applications |
|---|---|---|---|---|---|---|
| Goraczko et al. [160] | 2008 | Task | DAG | Integer LP | Execution time for each task, relationship between processor power modes, dependency among tasks, performance constraints | Real-time applications (sound source localization) |
| MAUI [4] | 2010 | Method | Application call graph | Manual annotation and guides for determining non-offloadable methods | Computational and energy costs of methods, communication energy and data size exchange between methods | Face recognition, video game, chess game, and real-time voice-based language translator |
| Zhang et al. [161] | 2010 | Weblet | Cost model of applications | Naïve Bayesian learning techniques | CPU load, memory, and network conditions | Image processing application |
| Giurgiu et al. [103] | 2012 | Module | Resource consumption graph | Dynamic: finding an optimal cut in the graph that minimizing objective function value | CPU load, network conditions, and power profiling | Indoor localization, text-to-speech synthesizer, and ticket machine |
| Abebe and Ryan [156] | 2012 | Class | Distributed local application graph | Multilevel graph-partitioning | Memory, network, performance, and power costs | Java based n-body simulator, hospital system simulator, and the NASA world wind demo application |
| Kovachev and Klamma [163] | 2012 | Service/ Module | Modules structure with meta-information | Integer LP | CPU load, available memory, code size, transfer size, remaining battery, uplink bandwidth | Face detection and recognition, N-queens |
| Niu et al. [158] | 2014 | Object | Weighted Object Relation Graph | Static, branch-and-bound, min-cut, greedy | Bandwidth, execution cost, and data transfer | A set of Dacapo benchmarks |
| Wu et al. [165] | 2016 | Task | Weighted consumption graph | Min-cost offloading partitioning (MCOP) algorithm | Memory, code size, task type (offloadable or un-offloadable), and task communication | Face recognition |

Table 5: A comparison of partitioning algorithms for energy efficiency improvements.

### 6.3. Decision Engine

Generally, partitioning application and making offloading decision are sometimes not so easy to separate. In many previous papers [160, 153], these two operations are included in a single process, collectively called partitioning, which identifies the local and remote partitions of an application. All the relevant parameters including environmental conditions are taken into account during this process. The remote partitions are then migrated to servers for the execution without further examining any other parameters.

However, making offloading decision can also be considered as a discrete step in the offloading process, which takes the output of the partitioning phase as input [4, 7]. That means that the results of application partitioning are not the final decisions on execution locations, but only suggestions for which computational components have the possibilities to be offloaded to remote infrastructures. The actual execution site (local or remote) for offloadable partitions is then decided (offline or online) based on the execution conditions. For example, in the CloneCloud framework [5], during the program analysis phase and offline partitioning, a database of partitions for different execution conditions and objective functions is automatically generated. At runtime, depending on the environmental conditions, decision engine selects a partition from the database that yields the most benefits for the system and implements it.

The decision-making algorithms can also be classified into two groups: (1) static offloading [173, 174, 175], in which the locations for the execution of applications are determined before the invocation; and (2) dynamic offloading [176, 177, 178, 179], in which the decision on whether and what to offload is taken at runtime based on current conditions. Dynamic offloading is generally more efficient than static offloading, however, it produces more overhead on the system relating to latency, profiling, and runtime decision making.

The decision engine takes the data collected by profilers as input to a global optimization problem that determines whether the offloadable candidates should be remotely executed at the server or not. The goal of decision engine is to find a program execution strategy that minimizes the objective function, subject to a set of particular constraints. Deciding where to execute each method is challenging because it requires a global view of the program's behavior. Then an optimal execution strategy that minimizes the energy consumption of the smartphone can be found by solving the optimization problem.

### 6.4. Online Profiling

Online profiling is performed on both the local and the server side during the execution of applications by profilers (cf. Section 4). This process does not only collect the information related to the performance and cost of the running programs, such as execution time, amount of resources used, communication cost, and energy cost, but also traces the environmental conditions at runtime [4, 7, 10]. All collected information reflects the influence of operating parameters on the performance and energy consumption of the smartphone during the offloading execution, or in other words, the benefits of the offloading process are evaluated. This information is then utilized to improve the accuracy and efficiency of the partitioning algorithm as well as the offloading decision for future invocations.

### 6.5. Offloading Algorithms

In this section, we review various algorithms used in computation offloading systems. As described above, one of the most common approaches to solve the partitioning problem and making offloading decision is to use LP [160, 4, 5]. Many other strategies are also considered for the offloading problem. A summary is provided in Table 6.

#### 6.5.1. Fuzzy control

In [176], the authors applied Fuzzy Control model to the offloading inference engine, which is responsible for making decisions on whether to trigger offloading. The offloading decision for the presented system is based on two parameters: the current available memory and wireless bandwidth. It was argued in the paper that the solution to the decision problem using the simple threshold-based approach has many limitations in terms of adaptability, reconfigurability, and stability. The Fuzzy Control model, on the other hand, was proved to be effective for such system. Following a similar idea, the system presented in [180] employs fuzzy decision engine for code offloading that takes into account the variables in both mobile and cloud side. In addition, a learning approach based on the code offloading traces with neuronal network algorithm is also applied to the decision process. By using fuzzy logic, offloading decisions can get different degrees of truth, rather than only yes or no answer. As a result, the system could offer more fine-grained decisions by analyzing the offloading traces and operating variables.

#### 6.5.2. Bayesian networks

In [181], computation offloading is formulated as a Bayesian statistical decision problem, in which the bandwidth between local and remote server is modeled as a random variable. This parameter is used to predict the time costs for remote execution and input/output data transfer to/from the server. It was shown in the paper that the Bayes model makes the decision problem more general and uniform when incorporating new prediction data. With traditional systems, network predictors are assumed to be always correct. A Bayesian approach allows to take into account the belief (the expected risk taken) on network prediction data. Similarly, Zhang et al. [161] applied Naïve Bayesian Learning techniques to find the optimal execution locations (local or in the cloud) for weblets given several parameters including device status (CPU memory and network consumption), user settings on processing speed, and history data of the application.

*6.5.3. Learning agents*

More general, in [182], the authors formulated the decision-making problem for service-oriented recommender systems using learning agents. Based on a set of context states (battery state, connection, devices location, cloud availability, etc.), attributes used to describe percepts, training data, and learned knowledge, the agent selects the best location to perform each of service/task in the given set with the objective of optimizing the running time. The execution results including the energy cost, time cost, and user's satisfaction are then stored in the training data set. After a certain number of running steps (user-defined), the learning process is repeated on the updated training data set to generate new knowledge, which is then utilized for future invocation. Different learning algorithms, such as the Naïve Bayes and random forest, can be employed for learning module.

*6.5.4. Genetic programming*

Another approach that is promising in modeling offloading decision problem is to use Genetic Programming (GP). In [183], the GP module plays as the core of the offloading inference part of the proposed framework. This module is responsible for developing a population of models and deciding the possible solution to the offloading decision of a mobile application. Each model in the population represents a decision tree, which determines a strategy for the offloading process. The simulator with a fitness function evaluates all the models generated by the GP module. Only models that pass a predefined threshold are kept for the future use. Besides, the paper provides a taxonomy of the parameters including application data, user requirement, device status, and network conditions, which are used by the GP module. Also applying the Genetic Algorithm (GA) for the offloading system, Deng et al. [184], however, made some appropriate modifications to the generic GA for better matching the needs of the decision problem for multiple mobile services in workflows. The experiments were conducted with different parameters of the GA, such as population size, number of iteration, mutation probability, and crossover probability.

*6.5.5. Markov models*

In [106], the approach using Markov Decision Process (MDP) is applied to solve the offloading problem in cloudlet systems with the existence of intermittent connection phenomena due to the mobility and cloudlet capacity. This technique was proved to outperform conventional baseline schemes with fixed offloading decisions. The intermittent connection and admission policies adopted by cloudlets may be the causes of offloading failures. Therefore, the authors took into account the random features of mobility pattern, availability and admission control of cloudlets, and workload on user devices when constructing MDP model. At each phase of the execution of given applications, MDP is solved with the current system state to an obtain optimal offloading policy, which minimizes computation and communication cost of user devices. In the context of real-time systems presented in [185], Markov chain is used to model the transition of a task sequence in an offloaded program and optimization problem solving is to seek for online-fetching policies that minimize the energy consumption for transferring fetched data to the server under a deadline constraint.

*6.5.6. Game theory*

In computation offloading systems, there are many scenarios that multiple users need to make offloading decisions at the same time and the benefit depends on the choices of the others. For example, in the three-tier architecture (local, cloudlets, and remote) presented in [186], multiple mobile devices can selfishly select one of three tiers to offload their computation. Another example was described in [187], where autonomous devices can decide any of multiple access points for offloading. Therefore, the authors in [186, 187] came up with the idea of using game theory, namely the Nash equilibrium, to formulate the stated problems.

*6.5.7. Deep Learning*

One of the hottest trends today is to solve the optimization problem using Deep Learning. Not outside this direction, several studies also investigated the application of this approach in computation offloading. In [188], the authors formulate and build a Deep Supervised Learn- ing model for the offloading decision problem to achieve maximum benefit on the system cost. The local overhead and the limitation in network communication and computation are taken into account in the optimization problem. Recently, a Deep Reinforncement Learning (DRL)-based framework [189] has been suggested to deal with highly complex offloading problems, for which finding solutions that could fully adapt to the variety of scenarios in MEC is extremely difficult.

Table 6: A summary of computation offloading algorithms.

| Work | Algorithm (Year) | Infrastructure | Core idea | Parameters | Evaluation metrics | Applications |
|---|---|---|---|---|---|---|
| Gu et al. [176] | Fuzzy (2004) | Cloud | Minimize the offloading overhead while relieving the memory constraint on the mobile device | Available memory, available wireless bandwidth | Offloading delay, bandwidth requirement, avg. interaction stretch | Image editor, graphical molecular editor, text editor |
| Flores and Srirama [180] | Fuzzy (2013) | Cloud | More than 'yes' or 'no' answer but assign a degree of truth to an offloading decision | Bandwidth, data transferred, CPU instance, video execution | The responsiveness of the cloud (latency) | Mobile calendar prediction, social group formation |
| Wolski et al. [181] | Bayes (2008) | Simulation computer | Predict remote cost (time for tranferring data & remote execution) and local cost | bandwidth between local/remote sites | Offloading performance (regret in $ms$) for different task sizes | Simulation task |
| Zhang et al. [161] | Bayes (2010) | Amazon EC2 & S3 | Use Naïve Bayes to find the optimal weblet configuration (# of weblets on device and cloud) | Device status (CPU, memory, network consumptions), user preference (expected # of images processed), history data | Average throughput, average CPU usage | Image processing, augmented reality |
| Folino and Pisani [183] | Genetic (2014) | VMs based on the ORACLE virtualbox | GP-based tool for building the decision tree-based model deciding whether to offload | Parameters in 4 categories: application, user, network, device | Error ratio, false negative & false positive rates | Dataset A (call, message), B (game, image processing), C (multimedia), real scenario (OCR) |
| Deng et al. [184] | Genetic (2015) | Intel Core i7 CPU with 2.3 GHz, 8 GB of RAM | Consider the dependency relations among component services and optimize execution time and energy consumption | Population sizes, # of iterations, mutation/crossover probabilities | The optimal fitness, fault-tolerance, scalability | Service workflows with random component services and control structures |
| Zhang et al. [106] | Markov (2015) | Mobile Cloudlet | Solve MDP to obtain optimal threshold policy for mobile user (minimize computation and offloading cost) | Users' mobility pattern, cloulets' admission control (cloudlet availability) | Expected cost, offloading rate | Linear structured face recognition program |

Table 6 Continued: A summary of computation offloading algorithms.

| Work | Algorithm (Year) | Infrastructure | Core idea | Parameters | Evaluation metrics | Applications |
|---|---|---|---|---|---|---|
| Ko et al. [185] | Markov (2017) | Cloud | Model the sequential task transition as a Markov chain, apply stochastic optimization to design the online-fetching policies | Task data size, latency requirement, # of candidate tasks, prefetching duration | Expected energy consumption (in $dB$) | Simulation |
| Cardellini et al. [186] | Game theory (2016) | Heterogeneous (mobile nodes, cloudlets, cloud) | Formulate the problem of selfish-selecting infrastructure (any of the 3 tiers) as a generalized Nash equilibrium problem | # of users, # of cloudlet, task execution time, maximum power consumption | Probability of infrastructure selection, # of iterations | Simulation in MATLAB |
| Jovsilo and Dan [187] | Game theory (2017) | Elastic cloud, non-elastic cloud | Formulate the problem of selecting wireless AP for computation offloading as Nash equilibrium | # of mobile users, average input data size | Cost ratio, offloading difference ratio, # of iterations | Simulation |
| Yu et al. [188] | Deep Learning (2017) | MEC | Formulate the offloading decision as a multi-label classification problem and develop a Deep Supervised Learning (DSL) method to minimize the computation and offloading cost | Data number for training, distance between UE and SCceNB | Offloading accuracy | Simulation |
| Wang et al. [189] | Deep Reinforncement Learning (2019) | MEC | Address the challenges of task dependency in offloading decision by using DRL-based offloading framework | # of tasks, # of dependencies between tasks, communication-to-computation ratio, transmission rate | Avg. latency | Random application graphs (synthetic DAG) |

*6.5.8. Other algorithms*

Many recent works focused on scheduling problem in computation offloading. Under the stringent constraints of application completion time and unstable operating conditions, the problem of how to efficiently offload computation-intensive tasks from local to more-resourced clouds is more challenging. Several strategies were proposed to address such challenges. Guo et al. [190] developed a distributed resource scheduling algorithm that reduces energy consumption and satisfies the task dependencies and completion time constraints. The constructed model takes both CPU clock frequency control at local devices and transmission power allocation in clouds into consideration. In [191], taking into account the impact of wireless network characteristics and the delay constraints of applications, the authors suggested an algorithm that schedules the wireless transmissions to minimize the communication cost but still guarantee the responsiveness of applications. Another scheduling algorithm for the offloading problem in LTE-based networks was presented in [192]. Due to the increase in complexity and the relationship between the problems involved, today, the offloading task is normally formulated by jointly considering different sub-problems like task distribution, channel assignment, power allocation, or data offloading [119, 98]. Table 6 provides additional information and shows a summary of basic characteristics of the selected offloading algorithms.

## 7. Offloading Frameworks

At early stages, computation offloading was carried out by simply migrating full processes [194] or even a full Virtual Machine (VM) [195, 104] to the remote infrastructure. In order to provide more fine-grained offload of mobile code, programmers have to specify, which methods or applications could or should be moved to the servers for execution as well as the offloading strategies to adapt with the changing of network conditions as well [196, 175]. With the objectives of reducing these heavy burdens on application developers and providing a more flexible way for outsourcing expensive tasks on smartphones, different frameworks have been suggested to facilitate the offloading implementation and deployment.

In this section, we survey selected existing computation offloading frameworks that allow smartphone applications to dynamically offload the intensive parts of their execution to the cloud in order to improve the performance and save the energy consumption. Most of the frameworks aim to reduce the energy consumption on devices while still guarantee or even improve the performance of applications [4, 193, 110]. Some others try to offer the scalability [7, 180] or the capability of transparent code migration [5, 8]. We summarize the basic features of the selected computation offloading frameworks in Table 7.

Typically, a computation offloading framework integrates all the common discussed components of the computational offloading process, namely profilers, partitioner, energy models, and an offloading decision engine. Most of these components are responsible for a particular phase in the offloading process except for the profiler that participates in all most phases.

In the initial step, developers use the profilers to gather the necessary training data to construct power models, which are then applied to dynamically estimate the runtime energy consumption of the smartphones at different granularities. When an application is started, the profiling data related to environmental conditions like network availability and quality is among the parameters used for deciding where and when to perform module offloading. During the execution, profilers continuously keep track of variant important hardware and software information of the smartphone and feed them into the energy models. The data obtained after the end of applications, such as the energy consumption, execution time, amount of resources used, and network parameters, is then used to improve the accuracy and efficiency of applications partitioning and offloading decision for future invocations. In other words, the data from profilers is indispensable for optimization and adaptation problems in computation offloading.

Besides that, other important issues need to be taken into consideration when building a framework is the offloading mechanism. This includes different technical problems concerning the implementation and deployment of computation offloading in reality. First, depending on the platform on which smartphone is operating, an appropriate application runtime environment is used for the offloading framework. It can be either Microsoft .NET Common Language Runtime (CLR) [4] used by the Microsoft Windows or Java VM [176], Dalvik VM [5, 7, 8, 9, 193], or Android Runtime (ART) [197] used by the Android OS.

Second, there are several strategies to execute offloadable modules in the cloud. The two most common ones are Remote Procedure Calls (RPCs) [4], which performs offloading for some specific computational tasks with pre-specified remoteable annotations and using VMs [5], which allows to carry out computation offloading for general cases. In RPC-based frameworks, a remote version for each method marked as *remoteable* is created by the compiler and loaded onto the server. Whereas for VM-based frameworks, a device clone or a replication of the smartphone VM image is deployed in the cloud. For transferring local states, data, and control from user devices to the server, the serialization technique can be used to convert necessary data structure into an array of bytes, which is suitable for transmission [4]. At the server side, the opposite process, i.e., deserialization, is performed for parsing data. The transmission of the computation results and returned states from the server to local is conducted in the same manner.

Finally, the cloud infrastructure and the software installation for the execution of offloadable modules in the cloud are important factors that need to be considered when developing an offloading frameworks. With the strong development and the ubiquity of MCC, it is promising to improve the mobile users' experience in terms of battery

| Framework | Objectives | Platform | Offload Level | Partitioning Approach | Offloading Mechanism | Testing Applications |
|---|---|---|---|---|---|---|
| Cloudlets [104] (2009) | Cloudlet-based offloading | Virtual-box | Entire apps | Migrating entire images or partitioning VM images for parallelism | Dynamic VM synthesis | Abiword, GIMP, Gnumeric, Kpresenter, PathFind, and SnapFind |
| MAUI [4] (2010) | Energy saving, guaranteeing the execution time | Microsoft .NET | Method | Manual annotations | LP model, based on the inputs of profilers and solver | Face recognition, video& chess game, language translator |
| CuckCoo [6] (2010) | Simplifying the development & deployment of offloading apps | Android | Method | Using AIDL to define interfaces for intensive and interactive parts | Decision depending on the availability of remote resources | Object recognition & face detection |
| Clone-Cloud [5] (2011) | context-aware, seamless code migration | Dalvik VM | Thread | Offline generation of partitions based on static program analysis & profiling data | Selecting the most appropriate partition at runtime depending on the environmental conditions | Virus scan, image search, and behavior profiling |
| ThinkAir [7] (2012) | Scalability, multiple-user | Android, Virtual-Box | Method | Manual annotations | Decision depending on the data from profilers, historical execution, and cloud resources | N-queens, face detection, virus scan, and image merging |
| COMET [8] (2012) | Transparent code offload for multi-thread apps | Dalvik VM | Thread | Not mentioned | Applying DSM, decision depending on the workload | Image editors, turn-based games, trip planner, and math tools |
| EMCO [180] (2013) | Energy saving, scalability | Android & GCM | Task | Not mentioned | Using Fuzzy logic & asynchronous notification, based on mobile and cloud variables | Cloud-based resource intensive mobile applications |
| COSMOS [9] (2014) | Performance improvement & efficient cloud resources usage | Android VM | Task | Not mentioned | Selecting VM-instance using greedy strategy based on the network and resource conditions | Face detection, voice recognition, and chess game |
| Jade [193] (2015) | Energy-aware offloading for Android apps | Java VM | Class | An offloadable class must implement one of two remoteable interfaces | Decision depending on the profiling information (system status and execution costs) | Face detection and path finding (Dijkstra) |
| NFC [110] (2017) | Energy saving & performance improvement | Android, NFC API | Method | Not mentioned | Enabling-disabling strategy (lowest data transmission delay & highest bandwidth of all NFC protocols) | N-Queens, RSA key generation/encryption, and gaming/puzzles |

Table 7: A comparison of computation offloading frameworks.

life, data storage capability, processing power, and service reliability [198].

In the last few years, many other frameworks for computation offloading have been proposed. Contrary to previous studies, which mainly focused on the problem of where and how to offload, these recent works put more effort on the optimization problem for suggested offloading frameworks. In [199], the authors propose a demand-based offloading framework for Android systems, in which user devices apply a dynamic partition algorithm to potential offloading applications to maximize energy saving depending on the demand. Also, the goal of maintaining Quality of Service (QoS) is considered along with the allocation of shared resources. An offloading framework, called Mirror [200], has been designed specifically for sophisticated games, which helps to improve the performance and reduce the energy consumption of the games. The framework presented in [201] aimed to jointly optimized timing predictability and energy dissipation for embedded systems, e.g., Android smartphone.

A more recent lightweight framework, which has also been tested on Android systems, called ULOOF [202], exploits the benefits of mobile edge computing to offer better timing performance and energy efficiency. In [203], the authors present a framework that can produce faster responses to offloading requests from multiple users while still maximizing energy savings. This objective is examined under the constraints of completion time and bandwidth. Also investigating the offloading problem in MEC, the CCO framework [204], on the other hand, concentrates on enabling cross-edge collaboration in computation offloading for partitionable applications.

In the context of a large-scale IoT system, the FogFlow framework [205] was proposed to address different issues in most of other existing fog computing frameworks for IoT smart city platforms. This framework offers a standard programming model for service developers, enables the share and reuse of contextual data across services, and helps to improve performance of existing solutions. For the development of Fog Computing technology that aims to tackle the challenges coming from large-scale IoT systems, service orchestration is an indispensable part [206]. Service orchestration refers to the process involved in the design, management, and delivery of end-to-end services. Besides the mass adoption of IoT technology, the dynamicity and heterogeneity of devices and networks make the challenges for service orchestration be even greater.

## 8. Open Research Questions

We conclude our survey of computation offloading for smartphones with a discussion of selected open research questions, which, we hope, will help researchers and practitioners to contribute to the field and to understand current limitations.

### *8.1. Offloading Framework Standardization*

Even though a number of computation offloading frameworks have been developed, software developers and users still have difficulties in making their applications offloadable. Most constraints and annotation of offload candidates are handled manually [4, 7], which makes the system inflexible and application-specific. Automatic systems have been suggested to overcome these problems [5, 8], however, they normally require modifications of the mobile core system. In order to support developers to benefit most from computation offloading, it is necessary to have a set of general rules and guidelines on developing and optimizing offloadable applications. In [22], the authors briefly present such instructions for designers to build their systems more efficiently. However, these guidelines are still quite generic and lack many important aspects of offloading systems like optimization problems and security.

Additionally, the variety of mobile platforms and cloud infrastructures also leads to many challenges in standardizing offloading frameworks. For example, different smartphone models provide different ranges of available information regarding hardware status and system statistics, so that their energy models are quite varying. This limits offloading frameworks to one or a few smartphones models.

### *8.2. Optimization Problems*

In the early time when the idea of computation offloading was just introduced, the attention of most researchers mainly focused on the question of how to migrate computation to the remote servers rather than on offloading strategies. Recently, many approaches have been proposed to solve the offloading problem in different contexts and scenarios (cf. Section 6.5). Overall, the application of these optimization techniques in practice is still limited and their efficiency is not always clear. This is particularly due to a non-constant overhead.

Moreover, the optimization also depends on the objectives and requirements of smartphone applications. There are many applications do most of their processing locally, others have high workload on data communication over the network. This has a high influence on the expected benefits of offloading techniques. Besides, some additional optimizations can be applied to specific applications to further enhance the performance and save more energy on the systems. For example, in [3], the authors showed the possibilities to adjust the fidelity level of receiving data from the server in order to lower the energy consumption on handheld devices. Alternatively, in [207], a technique called *offloading shaping* was introduced to reduce the demand for resources on mobile devices by sometimes performing additional cheap computation locally before offloading. Other schemes and algorithms need to be used for computation offloading for real-time applications [208, 209] in order to satisfy the strict constraint of application completion times. Thus, there is always a trade-off between the generality and efficiency, which makes optimization an ongoing challenge.

### 8.3. Context Awareness

Context awareness is one of the most timely features being investigated as static offloading is not always beneficial [17]. We are talking about the ability of systems to sense and react to the changing of user location, system states, and environmental conditions. Context awareness has been considered in different phases of computation offloading. First, application partitioning algorithms need to be aware of the current resource status of the local system, characteristics and resource demand of offloading applications, from which remoteable partitions can be identified optimally [165]. Repartitioning applications might be required every time the parameters change. Second, offloading decision making heavily depends on context information, such as network quality and availability of cloud servers. A context-aware framework is able to monitor and adapt its offloading decisions to the current context [5, 210]. For example, in the case of poor network quality the communication might be too expensive (in term of energy and/or time) and local execution would be preferable compared to migrating to cloud systems.

Even though there have been several works on developing context-aware offloading algorithms and systems [17, 18, 13], experiments are often simplified to only cover selected environmental parameters that influence offloading operations. More general models and techniques are needed to ensure the adaption of systems to a wide range of variables. Moreover, more real-world field testing is needed to complement lab experiments.

### 8.4. Offloading in Mobile Cloud Computing

The current main problem of MCC is how to exploit its full potential in augmenting mobile devices [198, 211]. Related to the cloud infrastructure, another problem arising in computation offloading is to select the most appropriate system in a very heterogeneous environment to most improve the overall performance. Generally, mobile cloud architecture can be divided into different tiers: a local tier of mobile end-devices like smartphones, tablets, wearables, and cameras; a middle tier of nearby clouds like cloudlets or mobile edge computing; and a remote tier of Internet cloud systems [186, 138]. Here, offloading in mobile cloud computing is a special case of context awareness. Depending on the current state, environmental conditions, and available cloud services, user devices can offload their computation to any part of the infrastructure [210] or even to a hybrid cloud [126, 149] in order to maximize offloading benefits. Experiments have shown many positive results, however, many practical aspects need to be investigated in more detail.

### 8.5. Fault Tolerance

During computation offloading process, the network conditions and the availability of cloud services can vary significantly due to the user's mobility. This may cause offloading operations failing, which could be either because of lost connections or server-related problems. It is, therefore, crucial for offloading systems to handle these failures and guarantee the continuous and successful execution of offloaded applications. When connection problems happen, systems could try to reconnect until a timeout expires. Otherwise, the devices need to make a decision on whether to execute applications locally or to make use of vertical handoff between different wireless media [212, 213] to continue the computation offloading process. Similarly, if server failures occur, user devices could switch to another available server or ignore the current offloading operation and perform the computation at local. In case of server switching, handheld equipments can execute either horizontal handoff between cloud servers in the same tier or vertical handoff between resources in different tiers, e.g., cloud to cloudlet or vice versa [214].

All of these operations consume extra processing time and energy on local devices. So, an effective offloading strategy needs to cover such scenarios as well by also taking into account the extra costs for handling the error and recovering the operation in the decision-making problem. Up to now, most of the frameworks mainly focus on offloading only; failure handling is normally considered separately.

### 8.6. Security and Privacy

Computation offloading requires data exchange between mobile devices and cloud servers, therefore, ensuring the confidentiality of private user data during the offloading process is an important task. Security needs to be maintained from all three perspectives of mobile cloud computing including user device, network transmission, and cloud server [215].

From the end system perspective, security and privacy are handled in both upload and download procedures. The constraints related to the privacy of data should be taken into consideration when making offloading decisions. The data/modules that contain or access much private information may have less possibility for remote execution or require to apply additional privacy-preserving techniques before offloading [216]. Also, there should be mechanisms to detect and prevent malware or malicious code in the download streams when receiving the results from clouds [217]. Looking at the data transmission over (public) networks, utilizing security protocols and data encryption methods can limit information leakage [218]. On the cloud side, trustworthiness is a must [219, 220]. The servers can also suffer unwanted intrusions, thus, protecting user data on the servers is required. This issue can be addressed by applying authentication and access control as well as other security technologies [221]. More details on security-related questions can be found in other excellent surveys [222, 21, 215, 19].

The development of security solutions for computation offloading, however, comes with an increased complexity and costs, are big concerns as they may reduce the offloading benefits [223]. Currently, the number of works on security and privacy problems in computation offloading is

still limited and the integration of suggested solutions into offloading software frameworks remains challenging.

### *8.7. Supporting Tools & Frameworks*

For most mobile applications, the offloading potentials are highly dependent on the concrete implementation. This is because not all the methods or modules in an application can be offloaded to the cloud due to constraints such as required hardware support (e.g., GPS and sensors) or the user interface [4]. In practice, programmers are required to identify these constraints and determine offloading candidates manually, which is normally non-trivial. It is, therefore, challenging to evaluate the effectiveness of an application in terms of maximizing computation offloading benefits. Frameworks are needed to support developers addressing this issue.

A good example is SmartDiet [224]. It can be used as an assisting toolkit for many existing method-level offloading frameworks. It performs the specification of constraints and partitioning applications automatically based on the traces of method executions. Besides, the toolkit also calculates the communication cost and energy-saving potential of offloading tasks. The output helps developers modifying and improving their applications in the terms of energy efficiency as well as computation offloading potentials. However, there is room for many improvements, e.g., in terms of further program characterization and mobile device support.

Additionally, developers also have difficulties in setting up offloading scenarios and profiling data during the working process due to the diversity of system states and network variables. Thus, initial experiments are normally quite simplistic, exploring only a few parameters. For this reason, simulation frameworks need to support simulating different offloading operations and scenarios, i.e., helping with detailed parameter studies. One possibility would be to integrate offloading frameworks with network simulation or even dedicated MCC simulators. However, the integration of cost models (energy and execution time), communication models, and offloading policies in such integrated simulator is still quite challenging.

## 9. Conclusion

In this paper, we investigated research solutions in the scope of computation offloading. Our work is mainly associated with the objective of saving energy. We provide a review of existing methods for measuring or estimating the energy consumption of smartphones as a fundamental basis for computation offloading. We examine the functionalities of different software components as well as algorithms and strategies proposed in the literature. We also perform a comprehensive comparison among several existing offloading frameworks. Finally, we discuss relevant research challenges that explain need to be solved in order to make computation offloading a success. Overall, most previous works mainly concentrated on the feasibility of offloading, i.e. answering the questions what and how to offload. Recent research tries to solve the offloading problem in different contexts and scenarios, in which the questions where and when to offload become the main focus. Along with that, optimization issues are also concerned.

## References


[1] A. Rudenko, P. Reiher, G. J. Popek, G. H. Kuenning, Saving Portable Computer Battery Power Through Remote Process Execution, ACM SIGMOBILE Mobile Computing and Communications Review 2 (1) (1998) 19–26. doi:10.1145/584007.584008.

[2] U. Kremer, J. Hicks, J. Rehg, A Compilation Framework for Power and Energy Management on Mobile Computers, in: 14th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2001), Springer, Cumberland Falls, KY, 2001, pp. 115–131. doi:10.1007/3-540-35767-X_8.

[3] Z. Li, C. Wang, R. Xu, Computation Offloading to Save Energy on Handheld Devices: A Partition Scheme, in: International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2001), ACM, Atlanta, GA, 2001, pp. 238–246. doi:10.1145/502217.502257.

[4] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, MAUI: Making Smartphones Last Longer with Code Offload, in: 8th ACM International Conference on Mobile Systems, Applications and Services (MobiSys 2010), ACM, San Francisco, CA, 2010, pp. 49–62. doi:10.1145/1814433.1814441.

[5] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, CloneCloud: Elastic Execution Between Mobile Device and Cloud, in: 6th Conference on Computer Systems (EuroSys 2011), ACM, Salzburg, Austria, 2011, pp. 301–314. doi:10.1145/1966445.1966473.

[6] R. Kemp, N. Palmer, T. Kielmann, H. Bal, Cuckoo: A Computation Offloading Framework for Smartphones, in: M. Gris, G. Yang (Eds.), 2nd ICST International Conference onMobile Computing, Applications, and Services (MobiCASE 2010), Springer, Santa Clara, CA, 2010, pp. 59–79. doi:10.1007/978-3-642-29336-8_4.

[7] S. Kosta, A. Aucinas, P. Hui, R. Mortier, X. Zhang, ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, in: 31st IEEE Conference on Computer Communications (INFOCOM 2012), IEEE, Orlando, FL, 2012, pp. 945–953. doi:10.1109/INFCOM.2012.6195845.

[8] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, X. Chen, COMET: Code Offload by Migrating Execution Transparently, in: 10th USENIX Conference on Operating Systems Design and Implementation (OSDI 2012), USENIX, Hollywood, CA, 2012, pp. 93–106.

[9] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, E. Zegura, COSMOS: Computation Offloading As a Service for Mobile Devices, in: 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2014), ACM, Philadelphia, PA, 2014, pp. 287–296. doi:10.1145/2632951.2632958.

[10] A. Ferrari, S. Giordano, D. Puccinelli, Reducing your local footprint with anyrun computing, Elsevier Computer Communications 81 (2016) 1–11. doi:10.1016/j.comcom.2016.01.006.

[11] M. V. Barbera, S. Kosta, A. Mei, V. C. Perta, J. Stefa, Mobile Offloading in the Wild: Findings and Lessons Learned Through a Real-life Experimentwith a New Cloud-aware System, in: 33rd IEEE Conference on Computer Communications (INFOCOM 2014), IEEE, Toronto, Canada, 2014, pp. 2355–2363. doi:10.1109/INFOCOM.2014.6848180.

[12] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, R. Buyya, Mobile Code Offloading: From Concept to Practice and Beyond,

IEEE Communications Magazine (COMMAG) 53 (3) (2015) 80–88. doi:10.1109/MCOM.2015.7060486.

[13] A. Bhattacharya, P. De, A survey of adaptation techniques in computation offloading, Journal of Network and Computer Applications 78 (C) (2017) 97–115. doi:10.1016/j.jnca.2016.10.023.

[14] K. Kumar, J. Liu, Y.-H. Lu, B. Bhargava, A Survey of Computation Offloading for Mobile Systems, ACM/Springer Mobile Networks and Applications (MONET) 18 (1) (2013) 129–140. doi:10.1007/s11036-012-0368-0.

[15] Z. Sanaei, S. Abolfazli, A. Gani, R. Buyya, Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges, IEEE Communications Surveys & Tutorials 16 (1) (2014) 369–392. doi:10.1109/SURV.2013.050113.00090.

[16] M. Shiraz, M. Sookhak, A. Gani, S. A. A. Shah, A Study on the Critical Analysis of Computational Offloading Frameworks for Mobile Cloud Computing, Journal of Network and Computer Applications 47 (2015) 47–60. doi:10.1016/j.jnca.2014.08.011.

[17] A. u. R. Khan, M. Othman, F. Xia, A. N. Khan, Context-Aware Mobile Cloud Computing and Its Challenges, IEEE Cloud Computing 2 (3) (2015) 42–49. doi:10.1109/MCC.2015.62.

[18] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, A. Qureshi, Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions, Journal of Network and Computer Applications 48 (2015) 99–117. doi:10.1016/j.jnca.2014.09.009.

[19] M. Ali, S. U. Khan, A. V. Vasilakos, Security in cloud computing: Opportunities and challenges, Information Sciences 305 (2015) 357–383. doi:10.1016/j.ins.2015.01.025.

[20] E. Ahmed, A. Gani, M. K. Khan, R. Buyya, S. U. Khan, Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges, Journal of Network and Computer Applications 52 (2015) 154–172. doi:10.1016/j.jnca.2015.03.001.

[21] M. Alizadeh, S. Abolfazli, M. Zamani, S. Baharun, K. Sakurai, Authentication in mobile cloud computing: A survey, Journal of Network and Computer Applications 61 (2016) 59–80. doi:10.1016/j.jnca.2015.10.005.

[22] M. Golkarifard, J. Yang, A. Movaghar, P. Hui, A Hitchhiker's Guide to Computation Offloading: Opinions from Practitioners, IEEE Communications Magazine (COMMAG) 55 (7) (2017) 193–199. doi:10.1109/MCOM.2017.1600481.

[23] P. Mach, Z. Becvar, Mobile Edge Computing: A Survey on Architecture and Computation Offloading, IEEE Communications Surveys & Tutorials 19 (3) (2017) 1628–1656. doi:10.1109/COMST.2017.2682318.

[24] P. Hu, S. Dhelim, H. Ning, T. Qiu, Survey on fog computing: architecture, key technologies, applications and open issues, Journal of Network and Computer Applications 98 (2017) 27–42. doi:10.1016/j.jnca.2017.09.002.

[25] F. Jameel, Z. Hamid, F. Jabeen, S. Zeadally, M. A. Javed, A Survey of Device-to-Device Communications: Research Issues and Challenges, IEEE Communications Surveys & Tutorials 20 (3) (2018) 2133–2168. doi:10.1109/COMST.2018.2828120.

[26] R. Mahmud, R. Kotagiri, R. Buyya, Fog Computing: A Taxonomy, Survey and Future Directions, in: D. M. B., L. KC., Y. L., E. A. (Eds.), Internet of Everything, Springer, 2018, pp. 103–130. doi:10.1007/978-981-10-5861-5_5.

[27] D. Xu, Y. Li, X. Chen, J. Li, P. Hui, S. Chen, J. Crowcroft, A survey of opportunistic offloading, IEEE Communications Surveys & Tutorials 20 (3) (2018) 2198–2236. doi:10.1109/COMST.2018.2808242.

[28] K. Peng, V. Leung, X. Xu, L. Zheng, J. Wang, Q. Huang, A Survey on Mobile Edge Computing: Focusing on Service Adoption and Provision, Wireless Communications and Mobile Computing 2018 (Apr. 2018). doi:10.1155/2018/8267838.

[29] H. Wu, Multi-Objective Decision-Making for Mobile Cloud Offloading: A Survey, IEEE Access 6 (2018) 3962–3976. doi:10.1109/ACCESS.2018.2791504.

[30] C. Jiang, X. Cheng, H. Gao, X. Zhou, J. Wan, Toward Computation Offloading in Edge Computing: A Survey, IEEE Access 7 (2019) 131543–131558. doi:10.1109/ACCESS.2019.2938660.

[31] B. Cao, L. Zhang, Y. Li, D. Feng, W. Cao, Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework, IEEE Communications Magazine (COMMAG) 57 (3) (2019) 56–62. doi:10.1109/MCOM.2019.1800608.

[32] M. Sharifi, S. Kafaie, O. Kashefi, A Survey and Taxonomy of Cyber Foraging of Mobile Devices, IEEE Communications Surveys & Tutorials 14 (4) (2012) 1232–1243. doi:10.1109/SURV.2011.111411.00016.

[33] A. u. R. Khan, M. Othman, S. A. Madani, S. U. Khan, A Survey of Mobile Cloud Computing Application Models, IEEE Communications Surveys & Tutorials 16 (1) (2014) 393–413. doi:10.1109/SURV.2013.062613.00160.

[34] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao, S. Tarkoma, Modeling, Profiling, and Debugging the Energy Consumption of Mobile Devices, ACM Computing Surveys (CSUR) 48 (3) (2016) 39:1–39:40. doi:10.1145/2840723.

[35] M. A. Khan, A survey of computation offloading strategies for performance improvement of applications running on mobile devices, Journal of Network and Computer Applications 56 (2015) 28–40. doi:10.1016/j.jnca.2015.05.018.

[36] A. Shye, B. Scholbrock, G. Memik, Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures, in: 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42), ACM, New York City, NY, 2009, pp. 168–178. doi:10.1145/1669112.1669135.

[37] Y. Xiao, R. Bhaumik, Z. Yang, M. Siekkinen, P. Savolainen, A. Yla-Jaaski, A System-Level Model for Runtime Power Estimation on Mobile Devices, in: IEEE/ACM International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing (GreenCom-CPSCom 2010), IEEE, Hangzhou, China, 2010, pp. 27–34. doi:10.1109/GreenCom-CPSCom.2010.114.

[38] L. Zhang, B. Tiwana, R. Dick, Z. Qian, Z. Mao, Z. Wang, L. Yang, Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones, in: IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2010), IEEE, Scottsdale, AZ, 2010, pp. 105–114.

[39] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, Y.-M. Wang, Fine-Grained Power Modeling for Smartphones Using System Call Tracing, in: 6th Conference on Computer Systems (EuroSys 2011), ACM, Salzburg, Austria, 2011, pp. 153–168. doi:10.1145/1966445.1966460.

[40] R. Mittal, A. Kansal, R. Chandra, Empowering Developers to Estimate App Energy Consumption, in: 18th ACM International Conference on Mobile Computing and Networking (MobiCom 2012), ACM, Istanbul, Turkey, 2012, pp. 317–328. doi:10.1145/2348543.2348583.

[41] F. Xu, Y. Liu, Q. Li, Y. Zhang, V-edge: Fast Self-constructive Power Modeling of Smartphones Based on Battery Voltage Dynamics, in: 10th USENIX Conference on Networked Systems Design and Implementation (NSDI 2013), USENIX, Lombard, IL, 2013, pp. 43–55.

[42] A. Pathak, Y. C. Hu, M. Zhang, Where is the energy spent inside my app? Fine Grained Energy Accounting on Smartphones with Eprof, in: 7th ACM European Conference on Computer Systems (EuroSys 2012), ACM, Bern, Switzerland, 2012, pp. 29–42. doi:10.1145/2168836.2168841.

[43] D. Li, S. Hao, J. Gui, W. G. J. Halfond, An Empirical Study of the Energy Consumption of Android Applications, in: IEEE International Conference on Software Maintenance and Evolution (ICSME 2014), IEEE, Victoria, Canada, 2014, pp. 121–130. doi:10.1109/ICSME.2014.34.

[44] N. Balasubramanian, A. Balasubramanian, A. Venkataramani, Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications, in: 9th ACM SIGCOMM Conference on Internet Measurement (IMC 2009), ACM, Chicago, IL, 2009, pp. 280–293. doi:

10.1145/1644893.1644927.

[45] W. Jung, C. Kang, C. Yoon, D. Kim, H. Cha, DevScope: A Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components, in: 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2012), ACM, Tampere, Finland, 2012, pp. 353–362. doi:10.1145/2380445.2380502.

[46] A. Rice, S. Hay, Decomposing Power Measurements for Mobile Devices, in: 8th IEEE International Conference on Pervasive Computing and Communications (PerCom 2010), IEEE, Mannheim, Germany, 2010, pp. 70–78. doi:10.1109/PERCOM.2010.5466991.

[47] A. Carroll, G. Heiser, An Analysis of Power Consumption in a Smartphone, in: USENIX Annual Technical Conference (ATC 2010), USENIX, Boston, MA, 2010, pp. 21–21.

[48] M. Segata, B. Bloessl, C. Sommer, F. Dressler, Towards Energy Efficient Smart Phone Applications: Energy Models for Offloading Tasks into the Cloud, in: IEEE International Conference on Communications (ICC 2014), IEEE, Sydney, Australia, 2014, pp. 2394–2399. doi:10.1109/ICC.2014.6883681.

[49] Q.-H. Nguyen, J. Blobel, F. Dressler, Energy Consumption Measurements as a Basis for Computational Offloading for Android Smartphones, in: 14th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC 2016), IEEE, Paris, France, 2016. doi:10.1109/CSE-EUC-DCABES.2016.157.

[50] A. Schulman, T. Schmid, P. Dutta, N. Spring, Phone Power Monitoring with BattOr, in: 17th ACM International Conference on Mobile Computing and Networking (MobiCom 2011), Demo Session, IEEE, Las Vegas, NV, 2011.

[51] R. Trestian, A.-N. Moldovan, O. Ormond, G.-M. Muntean, Energy consumption analysis of video streaming to Android mobile devices, in: 18th IEEE/IFIP Network Operations & Management Symposium (NOMS 2012), IEEE, Maui, HI, 2012, pp. 444–452. doi:10.1109/NOMS.2012.6211929.

[52] M. Dong, L. Zhong, Self-constructive High-rate System Energy Modeling for Battery-powered Mobile Systems, in: 9th International Conference on Mobile Systems, Applications, and Services (MobiSys 2011), ACM, Bethesda, MD, 2011, pp. 335–348. doi:10.1145/1999995.2000027.

[53] S. Tarkoma, M. Siekkinen, E. Lagerspetz, Y. Xiao, Smartphone Energy Consumption: Modeling and Optimization, Cambridge University Press, 2014.

[54] C. Isci, M. Martonosi, Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data, in: 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36), IEEE, San Diego, CA, 2003, pp. 93–104. doi:10.1109/MICRO.2003.1253186.

[55] C. Yoon, D. Kim, W. Jung, C. Kang, H. Cha, AppScope: Application Energy Metering Framework for Android Smartphone Using Kernel Activity Monitoring, in: USENIX Annual Technical Conference (ATC 2012), USENIX, Boston, MA, 2012, pp. 387–400.

[56] H. Inamura, T. Kamiyama, T. Konishi, K. Ohta, Extending Battery Lifetime in Smartphones with Power Efficient Task Management and Energy Aware Design Tool, International Journal of Informatics Society 6 (1) (2014) 3–10.

[57] Y. Zhang, Y. Liu, X. Liu, Q. Li, Enabling Accurate and Efficient Modeling-based CPU Power Estimation for Smartphones, in: IEEE/ACM 25th International Symposium on Quality of Service (IWQoS 2017), IEEE, Vilanova i la Geltrú, Spain, 2017, pp. 1–10. doi:10.1109/IWQoS.2017.7969112.

[58] S. Kim, H. Kim, J. Kim, J. Lee, E. Seo, Empirical Analysis of Power Management Schemes for Multi-core Smartphones, in: 7th International Conference on Ubiquitous Information Management and Communication (ICUIMC 2013), ACM, Kota Kinabalu, Malaysia, 2013, pp. 109:1–109:7. doi:10.1145/2448556.2448665.

[59] Z. Jiang, S. Mao, Energy Delay Tradeoff in Cloud Offloading for Multi-Core Mobile Devices, IEEE Access 3 (2015) 2306–2316. doi:10.1109/ACCESS.2015.2499300.

[60] C. Gao, A. Gutierrez, M. Rajan, R. G. Dreslinski, T. Mudge, C.-J. Wu, A Study of Mobile Device Utilization, in: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2015), IEEE, Philadelphia, PA, 2015, pp. 225–234. doi:10.1109/ISPASS.2015.7095808.

[61] M. Zhu, K. Shen, Energy Discounted Computing on Multicore Smartphones, in: USENIX Annual Technical Conference (ATC 2016), USENIX, Denver, CO, 2016, pp. 129–141.

[62] Y. Zhang, X. Wang, X. Liu, Y. Liu, L. Zhuang, F. Zhao, Towards Better CPU Power Management on Multicore Smartphones, in: Workshop on Power-Aware Computing and Systems (HotPower 2013), ACM, Farmington, PA, 2013, pp. 11:1–11:5. doi:10.1145/2525526.2525849.

[63] A. Rice, S. Hay, Measuring mobile phone energy consumption for 802.11 wireless networking, Pervasive and Mobile Computing 6 (6) (2010) 593–606. doi:10.1016/j.pmcj.2010.07.005.

[64] G. P. Perrucci, F. H. P. Fitzek, G. Sasso, W. Kellerer, J. Widmer, On the Impact of 2G and 3G Network Usage for Mobile Phones' Battery Life, in: European Wireless Conference (EW 2009), IEEE, Aalborg, Denmark, 2009, pp. 255–259. doi:10.1109/EW.2009.5357972.

[65] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, Characterizing Radio Resource Allocation for 3G Networks, in: 10th ACM Internet Measurement Conference (IMC 2010), ACM, Melbourne, Australia, 2010, pp. 137–150. doi:10.1145/1879141.1879159.

[66] O. Kassinen, E. Harjula, J. Korhonen, M. Ylianttila, Battery Life of Mobile Peers with UMTS and WLAN in a Kademlia-based P2P Overlay, in: 20th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2009), IEEE, Tokyo, Japan, 2009, pp. 662–665. doi:10.1109/PIMRC.2009.5450083.

[67] L. Wang, J. Manner, Energy Consumption Analysis of WLAN, 2G and 3G interfaces, in: IEEE/ACM International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing (GreenCom-CPSCom 2010), IEEE, Hangzhou, China, 2010, pp. 300–307. doi:10.1109/GreenCom-CPSCom.2010.81.

[68] E. Harjula, O. Kassinen, M. Ylianttila, Energy Consumption Model for Mobile Devices in 3G and WLAN Networks, in: IEEE Consumer Communications and Networking Conference (CCNC 2012), IEEE, Las Vegas, NV, 2012, pp. 532–537. doi:10.1109/CCNC.2012.6181134.

[69] N. Ding, D. Wagner, X. Chen, A. Pathak, Y. C. Hu, A. Rice, Characterizing and Modeling the Impact of Wireless Signal Strength on Smartphone Battery Drain, ACM SIGMETRICS Performance Evaluation Review 41 (1) (2013) 29–40. doi:10.1145/2494232.2466586.

[70] Y. Xiao, Y. Cui, P. Savolainen, M. Siekkinen, A. Wang, L. Yang, A. Ylä-Jääski, S. Tarkoma, Modeling Energy Consumption of Data Transmission Over Wi-Fi, IEEE Transactions on Mobile Computing (TMC) 13 (8) (2014) 1760–1773. doi:10.1109/TMC.2013.51.

[71] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, A Close Examination of Performance and Power Characteristics of 4G LTE Networks, in: 11th ACM International Conference on Mobile Systems, Applications, and Services (MobiSys 2013), ACM, Low Wood Bay, United Kingdom, 2012, pp. 225–238. doi:10.1145/2307636.2307658.

[72] J. Manweiler, R. Roy Choudhury, Avoiding the Rush Hours: WiFi Energy Management via Traffic Isolation, in: 9th International Conference on Mobile Systems, Applications, and Services (MobiSys 2011), ACM, Bethesda, MD, 2011, pp. 253–266. doi:10.1145/1999995.2000020.

[73] R. Friedman, A. Kogan, Y. Krivolapov, On Power and Throughput Tradeoffs of WiFi and Bluetooth in Smartphones, IEEE Transactions on Mobile Computing (TMC) 12 (7) (2013) 1363–1376. doi:10.1109/TMC.2012.117.

[74] L. Sun, R. K. Sheshadri, W. Zheng, D. Koutsonikolas, Modeling WiFi Active Power/Energy Consumption in Smartphones, in: 34th IEEE International Conference on Distributed Computing Systems (ICDCS 2014), IEEE, Madrid, Spain, 2014, pp. 41–51.

doi:10.1109/ICDCS.2014.13.

[75] M. O. Khan, V. Dave, Y.-C. Chen, O. Jensen, L. Qiu, A. Bhartia, S. Rallapalli, Model-Driven Energy-Aware Rate Adaptation, in: 14th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2013), ACM, Bengaluru, India, 2013, pp. 217–226. doi:10.1145/2491288.2491300.

[76] J. Li, J. Xiao, H. Azzouz, J. W.-K. Hong, R. Boutaba, PowerGuide: Accurate Wi-Fi Power Estimator for Smartphones, in: 16th Asia-Pacific Network Operations and Management Symposium (APNOMS 2014), IEEE, Hsinchu, Taiwan, 2014, pp. 1–6. doi:10.1109/APNOMS.2014.6996567.

[77] L. M. Feeney, M. Nilsson, Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment, in: 20th IEEE Conference on Computer Communications (INFOCOM 2001), IEEE, Anchorage, AK, 2001, pp. 1548–1557.

[78] D. Qiao, S. Choi, A. Jain, K. G. Shin, MiSer: An Optimal Low-Energy Transmission Strategy for IEEE 802.11a/h, in: 9th ACM International Conference on Mobile Computing and Networking (MobiCom 2003), ACM, San Diego, CA, 2003, pp. 161–175. doi:10.1145/938985.939003.

[79] E. Rantala, A. Karppanen, S. Granlund, P. Sarolahti, Modeling energy efficiency in wireless internet communication, in: 1st ACM Workshop on Networking, Systems, and Applications for Mobile Handhelds (MobiHeld 2009), ACM, Barcelona, Spain, 2009, pp. 67–68. doi:10.1145/1592606.1592624.

[80] C.-Y. Li, C. Peng, S. Lu, X. Wang, Energy-based Rate Adaptation for 802.11n, in: 18th ACM International Conference on Mobile Computing and Networking (MobiCom 2012), ACM, Istanbul, Turkey, 2012, pp. 341–352. doi:10.1145/2348543.2348585.

[81] A. Garcia-Saavedra, P. Serrano, A. Banchs, G. Bianchi, Energy Consumption Anatomy of 802.11 Devices and its Implication on Modeling and Design, in: 8th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT 2012), ACM, Nice, France, 2012, pp. 169–180. doi:10.1145/2413176.2413197.

[82] M. Altamimi, A. Abdrabou, K. Naik, A. Nayak, Energy Cost Models of Smartphones for Task Offloading to the Cloud, IEEE Transactions on Emerging Topics in Computing 3 (3) (2015) 384–398. doi:10.1109/TETC.2014.2387752.

[83] N. Warty, R. K. Sheshadri, W. Zheng, D. Koutsonikolas, A First Look at 802.11n Power Consumption in Smartphones, in: 1st ACM International Workshop on Practical Issues and Applications in Next Generation Wireless Networks, ACM, Istanbul, Turkey, 2012, pp. 27–32. doi:10.1145/2348714.2348720.

[84] P. H. J. Perälä, A. Barbuzzi, G. Boggia, K. Pentikousis, Theory and Practice of RRC State Transitions in UMTS Networks, in: IEEE Global Telecommunications Conference (GLOBECOM 2009), 5th IEEE Broadband Wireless Access Workshop, Honolulu, HI, 2009. doi:10.1109/GLOCOMW.2009.5360763.

[85] K. Dufková, M. Bjelica, B. Moon, L. Kencl, J.-Y. Le Boudec, Energy Savings for Cellular Network with Evaluation of Impact on Data Traffic Performance, in: European Wireless Conference (EW 2010), IEEE, Lucca, Italy, 2010, pp. 916–923. doi:10.1109/EW.2010.5483431.

[86] M. Gupta, A. T. Koc, R. Vannithamby, Analyzing Mobile Applications and PowerConsumption on SmartPhone over LTE network, in: International Conference on Energy Aware Computing (ICEAC 2011), IEEE, Istanbul, Turkey, 2011, pp. 1–4. doi:10.1109/ICEAC.2011.6403623.

[87] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, O. Spatscheck, An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance, ACM SIGCOMM Computer Communication Review (CCR) 43 (4) (2013) 363–374. doi:10.1145/2534169.2486006.

[88] S. Zhang, Z. Zhao, H. Guan, D. Miao, H. Yang, Statistics of RRC State Transition Caused by the Background Traffic in LTE networks, in: IEEE Wireless Communications and Networking Conference (WCNC 2013), IEEE, Shanghai, China, 2013, pp. 912–916. doi:10.1109/WCNC.2013.6554685.

[89] B. Dusza, C. Ide, L. Cheng, C. Wietfeld, An Accurate Measurement-Based Power Consumption Model for LTE Uplink Transmissions, in: 32nd IEEE Conference on Computer Communications (INFOCOM 2013), Poster Session, IEEE, Turin, Italy, 2013, pp. 49–50. doi:10.1109/INFCOMW.2013.6970731.

[90] B. Cao, L. Zhang, Y. Li, D. Feng, W. Cao, Output Power Levels of 4G User Equipment and Implications on Realistic RF EMF Exposure Assessments, IEEE Access 5 (2017) 4545–4550. doi:10.1109/ACCESS.2017.2682422.

[91] M. Yan, C. Chan, A. Gygax, J. Yan, L. Campbell, A. Nirmalathas, C. Leckie, Modeling the Total Energy Consumption of Mobile Network Services and Applications, Energies 12 (1) (2019) 184. doi:10.3390/en12010184.

[92] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, J. C. Zhang, What Will 5G Be?, IEEE Journal on Selected Areas in Communications (JSAC) 32 (6) (2014) 1065–1082. doi:10.1109/JSAC.2014.2328098.

[93] R. Q. Hu, Y. Qian, An Energy Efficient and Spectrum Efficient Wireless Heterogeneous Network Framework for 5G Systems, IEEE Communications Magazine (COMMAG) 52 (5) (2014) 94–101. doi:10.1109/MCOM.2014.6815898.

[94] S. Rizvi, A. Aziz, M. T. Jilani, N. Armi, G. Muhammad, S. H. Butt, An investigation of energy efficiency in 5G wireless networks, in: International Conference on Circuits, System and Simulation (ICCSS 2017), IEEE, London, United Kingdom, 2017, pp. 142–145. doi:10.1109/CIRSYSSIM.2017.8023199.

[95] M. E. Khoda, M. A. Razzaque, A. Almogren, M. M. Hassan, A. Alamri, A. Alelaiwi, Efficient Computation Offloading Decision in Mobile Cloud Computing over 5G Network, ACM/Springer Mobile Networks and Applications (MONET) 21 (5) (2016) 777–792. doi:10.1007/s11036-016-0688-6.

[96] L. Yang, H. Zhang, M. Li, J. Guo, H. Ji, Mobile Edge Computing Empowered Energy Efficient Task Offloading in 5G, IEEE Transactions on Vehicular Technology (TVT) 67 (7) (2018) 6398–6409. doi:10.1109/TVT.2018.2799620.

[97] H. Guo, J. Liu, J. Zhang, Efficient Computation Offloading for Multi-Access Edge Computing in 5G HetNets, in: IEEE International Conference on Communications (ICC 2018), IEEE, Kansas City, MO, 2018, pp. 1–6. doi:10.1109/ICC.2018.8422238.

[98] Z. Ning, X. Wang, J. J. Rodrigues, F. Xia, Joint Computation Offloading, Power Allocation, and Channel Assignment for 5G-Enabled Traffic Management Systems, IEEE Transactions on Industrial Informatics 15 (5) (2019) 3058–3067. doi:10.1109/TII.2019.2892767.

[99] S. K. Routray, M. K. Jha, L. Sharma, S. Sarkar, A. Javali, R. Tengshe, Energy Consumption Aspects of 5G Waveforms, in: International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET 2018), IEEE, Chennai, India, 2018, pp. 1–5. doi:10.1109/WiSPNET.2018.8538674.

[100] M. Matalatala, M. Deruyck, E. Tanghe, L. Martens, W. Joseph, Simulations of beamforming performance and energy efficiency for 5G mm-wave cellular networks, in: IEEE Wireless Communications and Networking Conference (WCNC 2018), IEEE, Barcelona, Spain, 2018, pp. 1–6. doi:10.1109/WCNC.2018.8377212.

[101] M. Höyhtyä, O. Apilo, M. Lasanen, Review of Latest Advances in 3GPP Standardization: D2D Communication in 5G Systems and Its Energy Consumption Models, Future Internet 10 (1) (2018) 3. doi:10.3390/fi10010003.

[102] L. Corral, A. B. Georgiev, A. Sillitti, G. Succi, A Method for Characterizing Energy Consumption in Android Smartphones, in: 2nd International Workshop on Green and Sustainable Software (GREENS 2013), IEEE, San Francisco, CA, 2013, pp. 38–45. doi:10.1109/GREENS.2013.6606420.

[103] I. Giurgiu, O. Riva, G. Alonso, Dynamic Software Deployment from Clouds to Mobile Devices, in: 13th ACM/IFIP/USENIX International Middleware Conference (Middleware 2012), Springer, Montréal, Canada, 2012, pp. 394–414. doi:10.1007/978-3-642-35170-9_20.

[104] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The Case

for VM-Based Cloudlets in Mobile Computing, IEEE Pervasive Computing 8 (4) (2009) 14–23. doi:10.1109/MPRV.2009.82.

[105] T. Verbelen, P. Simoens, F. De Turck, B. Dhoedt, Cloudlets: Bringing the cloud to the mobile user, in: 3rd ACM workshop on Mobile cloud computing and services (MCS '12, ACM, Low Wood Bay, United Kingdom, 2012, pp. 29–36. doi:10.1145/2307849.2307858.

[106] Y. Zhang, D. Niyato, P. Wang, Offloading in Mobile Cloudlet Systems with Intermittent Connectivity, IEEE Transactions on Mobile Computing (TMC) 14 (12) (2015) 2516–2529. doi:10.1109/TMC.2015.2405539.

[107] M. H. ur Rehman, C. Sun, T. Y. Wah, A. Iqbal, P. P. Jayaraman, Opportunistic Computation Offloading in Mobile Edge Cloud Computing Environments, in: 17th IEEE International Conference on Mobile Data Management (MDM 2016), IEEE, Porto, Portugal, 2016, pp. 208–213. doi:10.1109/MDM.2016.40.

[108] X. Chen, L. Jiao, W. Li, X. Fu, Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing, IEEE/ACM Transactions on Networking (TON) 24 (5) (2016) 2795–2808. doi:10.1109/TNET.2015.2487344.

[109] Z. Cheng, P. Li, J. Wang, S. Guo, Just-in-Time Code Offloading for Wearable Computing, IEEE Transactions on Emerging Topics in Computing 3 (1) (2015) 74–83. doi:10.1109/TETC.2014.2387688.

[110] K. Sucipto, D. Chatzopoulos, S. Kosta, P. Hui, Keep Your Nice Friends Close, but Your Rich Friends Closer - Computation Offloading Using NFC, in: 36th IEEE Conference on Computer Communications (INFOCOM 2017), IEEE, Atlanta, GA, 2017, pp. 1–9. doi:10.1109/INFOCOM.2017.8057147.

[111] Z. Chang, Z. Zhou, T. Ristaniemi, Z. Niu, Energy Efficient Optimization for Computation Offloading in Fog Computing System, in: IEEE Global Communications Conference (GLOBECOM 2017), IEEE, Singapore, Singapore, 2017, pp. 1–6. doi:10.1109/GLOCOM.2017.8254207.

[112] L. Liu, Z. Chang, X. Guo, S. Mao, T. Ristaniemi, Multiobjective Optimization for Computation Offloading in Fog Computing, IEEE Internet of Things Journal 5 (1) (2017) 283–294. doi:10.1109/JIOT.2017.2780236.

[113] G. Orsini, D. Bade, W. Lamersdorf, Computing at the Mobile Edge: Designing Elastic Android Applications for Computation Offloading, in: 8th IFIP Wireless and Mobile Networking Conference (WMNC 2015), IEEE, Munich, Germany, 2015, pp. 112–119. doi:10.1109/WMNC.2015.10.

[114] J. Dolezal, Z. Becvar, T. Zeman, Performance Evaluation of Computation Offloading from Mobile Device to the Edge of Mobile Network, in: IEEE Conference on Standards for Communications and Networking (CSCN 2016), IEEE, Berlin, Germany, 2016, pp. 1–7. doi:10.1109/CSCN.2016.7785153.

[115] J. Ren, G. Yu, Y. Cai, Y. He, F. Qu, Partial Offloading for Latency Minimization in Mobile-Edge Computing, in: IEEE Global Communications Conference (GLOBECOM 2017), IEEE, Singapore, Singapore, 2017, pp. 1–6. doi:10.1109/GLOCOM.2017.8254550.

[116] M. S. Elbamby, C. Perfecto, M. Bennis, K. Doppler, Toward Low-Latency and Ultra-Reliable Virtual Reality, IEEE Network 32 (2) (2018) 78–84. doi:10.1109/MNET.2018.1700268.

[117] Y. Mao, J. Zhang, K. B. Letaief, Dynamic Computation Offloading for Mobile-Edge Computing with Energy Harvesting Devices, IEEE Journal on Selected Areas in Communications (JSAC) 34 (12) (2016) 3590–3605. doi:10.1109/JSAC.2016.2611964.

[118] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, Y. Zhang, Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks, IEEE Access 4 (2016) 5896–5907. doi:10.1109/ACCESS.2016.2597169.

[119] H. Sun, F. Zhou, R. Q. Hu, Joint Offloading and Computation Energy Efficiency Maximization in a Mobile Edge Computing System, IEEE Transactions on Vehicular Technology (TVT) 68 (3) (2019) 3052–3056. doi:10.1109/TVT.2019.2893094.

[120] C.-F. Liu, M. Bennis, H. V. Poor, Latency and Reliability-Aware Task Offloading and Resource Allocation for Mobile Edge Computing, in: IEEE Global Communications Conference (GLOBECOM 2017), 6th International Workshop on Emerging Technologies for 5G and Beyond Wireless and Mobile Networks (ET5GB), IEEE, Singapore, Singapore, 2017, pp. 1–7. doi:10.1109/GLOCOMW.2017.8269175.

[121] M. Bagheri, M. Siekkinen, J. K. Nurminen, Cloud-Based Pedestrian Road-Safety with Situation-Adaptive Energy-Efficient Communication, IEEE Intelligent Transportation Systems Magazine 8 (3) (2016) 45–62. doi:10.1109/MITS.2016.2573338.

[122] Q.-H. Nguyen, M. Morold, K. David, F. Dressler, Adaptive Safety Context Information for Vulnerable Road Users with MEC Support, in: 15th IEEE/IFIP Conference on Wireless On demand Network Systems and Services (WONS 2019), IEEE, Wengen, Switzerland, 2019, pp. 28–35. doi:10.23919/WONS.2019.8795475.

[123] F. Hagenauer, C. Sommer, T. Higuchi, O. Altintas, F. Dressler, Vehicular Micro Clouds as Virtual Edge Servers for Efficient Data Collection, in: 23rd ACM International Conference on Mobile Computing and Networking (MobiCom 2017), 2nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services (CarSys 2017), ACM, Snowbird, UT, 2017, pp. 31–35. doi:10.1145/3131944.3133937.

[124] F. Hagenauer, T. Higuchi, O. Altintas, F. Dressler, Efficient Data Handling in Vehicular Micro Clouds, Elsevier Ad Hoc Networks 91 (2019) 101871. doi:10.1016/j.adhoc.2019.101871.

[125] M. Chen, Y. Hao, Y. Li, C.-F. Lai, D. Wu, On the Computation Offloading at Ad Hoc Cloudlet: Architecture and Service Modes, IEEE Communications Magazine (COMMAG) 53 (6) (2015) 18–24. doi:10.1109/MCOM.2015.7120041.

[126] Y. Jararweh, A. Doulat, O. AlQudah, E. Ahmed, M. Al-Ayyoub, E. Benkhelifa, The Future of Mobile Cloud Computing: Integrating Cloudlets and Mobile Edge Computing, in: 23rd International Conference on Telecommunications (ICT 2016), IEEE, Thessaloníki, Greece, 2016, pp. 1–5. doi:10.1109/ICT.2016.7500486.

[127] I. Yaqoob, E. Ahmed, A. Gani, S. Mokhtar, M. Imran, S. Guizani, Mobile ad hoc cloud: A survey, Wiley Wireless Communications and Mobile Computing (WCMC) 16 (16) (2016) 2572–2589. doi:10.1002/wcm.2709.

[128] W. Shen, B. Yin, X. Cao, L. X. Cai, Y. Cheng, Secure Device-to-Device Communications over WiFi Direct, IEEE Network 30 (5) (2016) 4–9. doi:10.1109/MNET.2016.7579020.

[129] X. Wu, R. Miucic, S. Yang, S. Al-Stouhi, J. Misener, S. Bai, W.-h. Chan, Cars Talk to Phones: A DSRC Based Vehicle-Pedestrian Safety System, in: 80th IEEE Vehicular Technology Conference (VTC 2014-Fall), IEEE, Vancouver, Canada, 2014. doi:10.1109/VTCFall.2014.6965898.

[130] D. Chatzopoulos, C. Bermejo, E. ul Haq, Y. Li, P. Hui, D2D Task Offloading: A Dataset-Based Q&A, IEEE Communications Magazine (COMMAG) 57 (2) (2018) 102–107. doi:10.1109/MCOM.2018.1700873.

[131] J. Feng, L. Zhao, J. Du, X. Chu, F. R. Yu, Computation Offloading and Resource Allocation in D2D-Enabled Mobile Edge Computing, in: IEEE International Conference on Communications (ICC 2018), IEEE, Kansas City, MO, 2018, pp. 1–6. doi:10.1109/ICC.2018.8422776.

[132] Y. He, J. Ren, G. Yu, Y. Cai, D2D Communications Meet Mobile Edge Computing for Enhanced Computation Capacity in Cellular Networks, IEEE Transactions on Wireless Communications (TWC) 18 (3) (2019) 1750–1763. doi:10.1109/TWC.2019.2896999.

[133] G. Qiao, S. Leng, Y. Zhang, Online Learning and Optimization for Computation Offloading in D2D Edge Computing and Networks, Mobile Networks and Applications (2019) 1–12doi:10.1007/s11036-018-1176-y.

[134] X. Chen, L. Pu, L. Gao, W. Wu, D. Wu, Exploiting Massive D2D Collaboration for Energy-Efficient Mobile Edge Computing, IEEE Wireless Communications 24 (4) (2017) 64–71. doi:10.1109/MWC.2017.1600321.

[135] J. Wen, C. Ren, A. K. Sangaiah, Energy-Efficient Device-to-Device Edge Computing Network: An Approach Offloading

Both Traffic and Computation, IEEE Communications Magazine (COMMAG) 56 (9) (2018) 96–102. doi:10.1109/MCOM.2018.1701054.

[136] J. Xu, L. Chen, K. Liu, C. Shen, Designing Security-Aware Incentives for Computation Offloading via Device-to-Device Communication, IEEE Transactions on Wireless Communications (TWC) 17 (9) (2018) 6053–6066. doi:10.1109/TWC.2018.2854579.

[137] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog Computing and Its Role in the Internet of Things, in: 1st Workshop on Mobile Cloud Computing (MCC 2012), ACM, Helsinki, Finland, 2012, pp. 13–16. doi:10.1145/2342509.2342513.

[138] K. Dolui, S. K. Datta, Comparison of Edge Computing Implementations: Fog Computing, Cloudlet and Mobile Edge Computing, in: Global Internet of Things Summit (GIoTS 2017), IEEE, Genève, Switzerland, 2017, pp. 1–6. doi:10.1109/GIOTS.2017.8016213.

[139] J. Zhu, D. S. Chan, M. S. Prabhu, P. Natarajan, H. Hu, F. Bonomi, Improving Web Sites Performance Using Edge Servers in Fog Computing Architecture, in: 7th IEEE International Symposium on Service-Oriented System Engineering, IEEE, Redwood City, CA, 2013, pp. 320–323. doi:10.1109/SOSE.2013.73.

[140] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, M. Nemirovsky, Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing, in: 19th IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD 2014), IEEE, Athens, Greece, 2014, pp. 325–329. doi:10.1109/CAMAD.2014.7033259.

[141] I. Stojmenovic, S. Wen, The Fog Computing Paradigm: Scenarios and Security Issues, in: Federated Conference on Computer Science and Information Systems, IEEE, Warsaw, Poland, 2014, pp. 1–8. doi:10.15439/2014F503.

[142] M. Aazam, E.-N. Huh, E-HAMC: Leveraging Fog Computing for Emergency Alert Service, in: 13th IEEE International Conference on Pervasive Computing and Communication (PerCom 2015), Workshop on Pervasive Networks for Emergency Management (PerNEM 2015), IEEE, St. Louis, MO, 2015, pp. 518–523. doi:10.1109/PERCOMW.2015.7134091.

[143] Y. Cao, S. Chen, P. Hou, D. Brown, FAST: A Fog Computing Assisted Distributed Analytics System to Monitor Fall for Stroke Mitigation, in: IEEE International Conference on Networking, Architecture and Storage (NAS 2015), IEEE, Boston, MA, 2015, pp. 2–11. doi:10.1109/NAS.2015.7255196.

[144] M. Arif, G. Wang, T. Wang, T. Peng, SDN-Based Secure VANETs Communication with Fog Computing, in: G. Wang, J. Chen, L. T. Yang (Eds.), Security, Privacy, and Anonymity in Computation, Communication, and Storage, Lecture Notes in Computer Science, Springer, Melbourne, Australia, 2018, pp. 46–59. doi:10.1007/978-3-030-05345-1_4.

[145] X. Meng, W. Wang, Z. Zhang, Delay-Constrained Hybrid Computation Offloading With Cloud and Fog Computing, IEEE Access 5 (2017) 21355–21367. doi:10.1109/ACCESS.2017.2748140.

[146] S. Chen, Y. Zheng, K. Wang, W. Lu, Delay Guaranteed Energy-Efficient Computation Offloading for Industrial IoT in Fog Computing, in: IEEE International Conference on Communications (ICC 2019), IEEE, Shanghai, China, 2019, pp. 1–6. doi:10.1109/ICC.2019.8761199.

[147] Y. Wang, K. Wang, H. Huang, T. Miyazaki, S. Guo, Traffic and Computation Co-Offloading With Reinforcement Learning in Fog Computing for Industrial Applications, IEEE Transactions on Industrial Informatics 15 (2) (2018) 976–986. doi:10.1109/TII.2018.2883991.

[148] M. Zeng, Y. Li, K. Zhang, M. Waqas, D. Jin, Incentive Mechanism Design for Computation Offloading in Heterogeneous Fog Computing: A Contract-Based Approach, in: IEEE International Conference on Communications (ICC 2018), IEEE, Kansas City, MO, 2018, pp. 1–6. doi:10.1109/ICC.2018.8422684.

[149] J. Du, L. Zhao, J. Feng, X. Chu, Computation Offloading and Resource Allocation in Mixed Fog/Cloud Computing Systems With Min-Max Fairness Guarantee, IEEE Transactions on Communications 66 (4) (2018) 1594–1608. doi:10.1109/TCOMM.2017.2787700.

[150] B. G. Ryder, Constructing the Call Graph of a Program, IEEE Transactions on Software Engineering SE-5 (3) (1979) 216–226. doi:10.1109/TSE.1979.234183.

[151] M. Smit, M. Shtern, B. Simmons, M. Litoiu, Partitioning Applications for Hybrid and Federated Clouds, in: Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2012), Toronto, Canada, 2012, pp. 27–41.

[152] C. Wang, Z. Li, Parametric Analysis for Adaptive Computation Offloading, ACM SIGPLAN Notices 39 (6) (2004) 119–130. doi:10.1145/996893.996857.

[153] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, G. Alonso, Calling the Cloud: Enabling Mobile Phones as Interfaces to Cloud Applications, in: 10th ACM/IFIP/USENIX International Middleware Conference (Middleware 2009), Springer, Urbana-Champaign, IL, 2009, pp. 83–102. doi:10.1007/978-3-642-10445-9_5.

[154] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, D. Milojicic, Adaptive Offloading Inference for Delivering Applications in Pervasive Computing Environments, in: 1st IEEE International Conference on Pervasive Computing and Communications (PerCom 2003), IEEE, Dallas-Fort Worth, TX, 2003, pp. 107–114. doi:10.1109/PERCOM.2003.1192732.

[155] S. Ou, K. Yang, A. Liotta, An Adaptive Multi-Constraint Partitioning Algorithm for Offloading in Pervasive Systems, in: 4th IEEE International Conference on Pervasive Computing and Communications (PerCom 2006), IEEE, Pisa, Italy, 2006, pp. 116–125. doi:10.1109/PERCOM.2006.7.

[156] E. Abebe, C. Ryan, Adaptive application offloading using distributed abstract class graphs in mobile environments, Journal of Systems and Software 85 (12) (2012) 2755–2769. doi:10.1016/j.jss.2012.05.091.

[157] V. Jamwal, S. Iyer, Automated Refactoring of Objects for Application Partitioning, in: 12th Asia-Pacific Software Engineering Conference (APSEC 2005), IEEE, Taipei, Taiwan, 2005. doi:10.1109/APSEC.2005.45.

[158] J. Niu, W. Song, M. Atiquzzaman, Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications, Journal of Network and Computer Applications 37 (2014) 334–347. doi:10.1016/j.jnca.2013.03.007.

[159] T. Verbelen, T. Stevens, F. De Turck, B. Dhoedt, Graph partitioning algorithms for optimizing software deployment in mobile cloud computing, Future Generation Computer Systems 29 (2) (2013) 451–459. doi:10.1016/j.future.2012.07.003.

[160] M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, B. Priyantha, F. Zhao, Energy-Optimal Software Partitioning in Heterogeneous Multiprocessor Embedded Systems, in: 45th Annual Design Automation Conference (DAC 2008), ACM, Anaheim, CA, 2008, pp. 191–196. doi:10.1145/1391469.1391518.

[161] X. Zhang, S. Jeong, A. Kunjithapatham, S. Gibbs, Towards an Elastic Application Model for Augmenting Computing Capabilities of Mobile Platforms, in: 11th ACM/IFIP/USENIX International Middleware Conference (Middleware 2010), Springer, Chicago, IL, 2010, pp. 161–174. doi:10.1007/978-3-642-17758-3_12.

[162] S. Park, Q. Chen, H. Han, H. Y. Yeom, Design and evaluation of mobile offloading system for web-centric devices, Journal of Network and Computer Applications 40 (2014) 105–115. doi:10.1016/j.jnca.2013.08.006.

[163] D. Kovachev, R. Klamma, Framework for Computation Offloading in Mobile Cloud Computing, International Journal of Interactive Multimedia and Artificial Intelligence 1 (7) (2012) 6–15. doi:10.9781/ijimai.2012.171.

[164] B. Gao, L. He, L. Liu, K. Li, S. A. Jarvis, From Mobiles to Clouds: Developing Energy-Aware Offloading Strategies for Workflows, in: 13th ACM/IEEE International Conference on Grid Computing (GRID 2012), IEEE, Beijing, China, 2012, pp. 139–146. doi:10.1109/Grid.2012.20.

[165] H. Wu, W. Knottenbelt, K. Wolter, Y. Sun, An Optimal Offloading Partitioning Algorithm in Mobile Cloud Computing, in: 13th International Conference on Quantitative Evaluation of Systems (QEST 2016), Springer, Québec, Canada, 2016, pp. 311–328. doi:10.1007/978-3-319-43425-4_21.

[166] M.-R. Ra, B. Priyantha, A. Kansal, J. Liu, Improving Energy Efficiency of Personal Sensing Applications with Heterogeneous Multi-Processors, in: ACM Conference on Ubiquitous Computing (UbiComp 2012), ACM, Pittsburgh, PA, 2012, pp. 1–10. doi:10.1145/2370216.2370218.

[167] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, S. Madden, Wishbone: Profile-based Partitioning for Sensornet Applications., in: 6th USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2009), USENIX, Boston, MA, 2009, pp. 395–408.

[168] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, A. Chan, A Framework for Partitioning and Execution of Data Stream Applications in Mobile Cloud Computing, ACM SIGMETRICS Performance Evaluation Review 40 (4) (2013) 23–32. doi:10.1145/2479942.2479946.

[169] K. Sinha, M. Kulkarni, Techniques for Fine-Grained, Multi-site Computation Offloading, in: 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2011), IEEE, Newport Beach, CA, 2011, pp. 184–194. doi:10.1109/CCGrid.2011.69.

[170] S. H. Mortazavi, M. Salehe, C. S. Gomes, C. Phillips, E. de Lara, CloudPath: A Multi-Tier Cloud Computing Framework, in: 2nd ACM/IEEE Symposium on Edge Computing (SEC '17), ACM, San Jose, CA, 2017, pp. 1–13. doi:10.1145/3132211.3134464.

[171] C. Cicconetti, M. Conti, A. Passarella, Low-latency Distributed Computation Offloading for Pervasive Environments, in: IEEE International Conference on Pervasive Computing and Communications (PerCom 2019, IEEE, Kyoto, Japan, 2019, pp. 262–271. doi:10.1109/PERCOM.2019.8767419.

[172] A. Gallidabino, C. Pautasso, The Liquid WebWorker API for Horizontal Offloading of Stateless Computations, Journal of Web Engineering 17 (6) (2018) 405–448. doi:10.13052/jwe1540-9589.17672.

[173] J. Flinn, S. Park, M. Satyanarayanan, Balancing Performance, Energy, and Quality in Pervasive Computing, in: 22nd IEEE International Conference on Distributed Computing Systems (ICDCS 2002), IEEE, Vienna, Austria, 2002, pp. 217–226. doi:10.1109/ICDCS.2002.1022259.

[174] K. Kumar, Y.-H. Lu, Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?, IEEE Computer 43 (4) (2010) 51–56. doi:10.1109/MC.2010.98.

[175] R. K. Balan, M. Satyanarayanan, S. Y. Park, T. Okoshi, Tactics-Based Remote Execution for Mobile Computing, in: 1st ACM International Conference on Mobile Systems, Applications, and Services (MobiSys 2003), ACM, San Francisco, CA, 2003, pp. 273–286. doi:10.1145/1066116.1066125.

[176] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, D. Milojicic, Adaptive Offloading for Pervasive Computing, IEEE Pervasive Computing 3 (3) (2004) 66–73. doi:10.1109/MPRV.2004.1321031.

[177] S. Ou, K. Yang, Q. Zhang, An Efficient Runtime Offloading Approach for Pervasive Services, in: IEEE Wireless Communications and Networking Conference (WCNC 2006), IEEE, Las Vegas, NV, 2006, pp. 2229–2234. doi:10.1109/WCNC.2006.1696642.

[178] D. Huang, P. Wang, D. Niyato, A Dynamic Offloading Algorithm for Mobile Computing, IEEE Transactions on Wireless Communications (TWC) 11 (6) (2012) 1991–1995. doi:10.1109/TWC.2012.041912.110912.

[179] G. Chen, B.-T. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, R. Chandramouli, Studying Energy Trade Offs in Offloading Computation/Compilation in Java-Enabled Mobile Devices, IEEE Transactions on Parallel and Distributed Systems 15 (9) (2004) 795–809. doi:10.1109/TPDS.2004.47.

[180] H. Flores, S. Srirama, Adaptive Code Offloading for Mobile Cloud Applications: Exploiting Fuzzy Sets and Evidence-based Learning, in: 4th ACM Workshop on Mobile Cloud Computing and Services (MCS 2013), ACM, Taipei, Taiwan, 2013, pp. 9–16. doi:10.1145/2482981.2482984.

[181] R. Wolski, S. Gurun, C. Krintz, D. Nurmi, Using Bandwidth Data To Make Computation Offloading Decisions, in: IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), IEEE, Miami, FL, 2008, pp. 1–8. doi:10.1109/IPDPS.2008.4536215.

[182] P. Nawrocki, B. Śnieżyński, J. Czyżewski, Learning Agent for a Service-Oriented Context-Aware Recommender System in a Heterogeneous Environment, Computing & Informatics 35 (5) (2016) 1005–1026.

[183] G. Folino, F. S. Pisani, Automatic offloading of mobile applications into the cloud by means of genetic programming, Applied Soft Computing 25 (2014) 253–265. doi:10.1016/j.asoc.2014.09.016.

[184] S. Deng, L. Huang, J. Taheri, A. Y. Zomaya, Computation Offloading for Service Workflow in Mobile Cloud Computing, IEEE Transactions on Parallel and Distributed Systems 26 (12) (2015) 3317–3329. doi:10.1109/TPDS.2014.2381640.

[185] S.-W. Ko, K. Huang, S.-L. Kim, H. Chae, Energy Efficient Mobile Computation Offloading via Online Prefetching, in: IEEE International Conference on Communications (ICC 2017), IEEE, Paris, France, 2017, pp. 1–6. doi:10.1109/ICC.2017.7997341.

[186] V. Cardellini, V. D. N. Personé, V. Di Valerio, F. Facchinei, V. Grassi, F. L. Presti, V. Piccialli, A game-theoretic approach to computation offloading in mobile cloud computing, Mathematical programming 157 (2) (2016) 421–449. doi:10.1007/s10107-015-0881-6.

[187] S. Jovsilo, G. Dan, A Game Theoretic Analysis of Selfish Mobile Computation Offloading, in: 36th IEEE Conference on Computer Communications (INFOCOM 2017), IEEE, Atlanta, GA, 2017, pp. 1–9. doi:10.1109/INFOCOM.2017.8057148.

[188] S. Yu, X. Wang, R. Langar, Computation Offloading for Mobile Edge Computing: A Deep Learning Approach, in: IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC 2017), IEEE, Montreal, Canada, 2017, pp. 1–6. doi:10.1109/PIMRC.2017.8292514.

[189] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, N. Georgalas, Computation Offloading in Multi-Access Edge Computing Using a Deep Sequential Model Based on Reinforcement Learning, IEEE Communications Magazine (COMMAG) 57 (5) (2019) 64–69. doi:10.1109/MCOM.2019.1800971.

[190] S. Guo, B. Xiao, Y. Yang, Y. Yang, Energy-Efficient Dynamic Offloading and Resource Scheduling in Mobile Cloud Computing, in: 35th IEEE Conference on Computer Communications (INFOCOM 2016), IEEE, San Francisco, CA, 2016.

[191] L. Tong, W. Gao, Application-Aware Traffic Scheduling for Workload Offloading in Mobile Clouds, in: 35th IEEE Conference on Computer Communications (INFOCOM 2016), IEEE, San Francisco, CA, 2016, pp. 1–9. doi:10.1109/INFOCOM.2016.7524520.

[192] W. Labidi, M. Sarkiss, M. Kamoun, Energy-Optimal Resource Scheduling and Computation Offloading in Small Cell Networks, in: 22nd International Conference on Telecommunications (ICT 2015), IEEE, Sydney, Australia, 2015, pp. 313–318. doi:10.1109/ICT.2015.7124703.

[193] H. Qian, D. Andresen, Jade: Reducing Energy Consumption of Android App, International Journal of Networked and Distributed Computing (IJNDC) 3 (3) (2015) 150–158. doi:10.2991/ijndc.2015.3.3.2.

[194] S. Osman, D. Subhraveti, G. Su, J. Nieh, The Design and Implementation of Zap: A System for Migrating Computing Environments, ACM SIGOPS Operating Systems Review 36 (SI) (2002) 361–376. doi:10.1145/844128.844162.

[195] B.-G. Chun, P. Maniatis, Augmented Smartphone Applications Through Clone Cloud Execution, in: 12th Workshop on Hot Topics in Operating Systems (HotOS 2009), USENIX, Monte Verità, Switzerland, 2009.

[196] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, H.-I. Yang, The Case for Cyber Foraging, in: 10th ACM SIGOPS European Workshop, ACM, Saint-Emilion, France, 2002, pp. 87–92. doi:10.1145/1133373.1133390.

[197] A. Yousafzai, A. Gani, R. M. Noor, A. Naveed, R. W. Ahmad, V. Chang, Computational offloading mechanism for native and android runtime based mobile applications, Journal of Systems and Software 121 (2016) 28–39. doi:10.1016/j.jss.2016.07.043.

[198] H. T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches, Wiley Wireless Communications and Mobile Computing (WCMC) 13 (18) (2013) 1587–1611. doi:10.1002/wcm.1203.

[199] J. Kumar, A. Malik, S. K. Dhurandher, P. Nicopolitidis, Demand-Based Computation Offloading Framework for Mobile Devices, IEEE Systems Journal 12 (4) (2017) 3693–3702. doi:10.1109/JSYST.2017.2706178.

[200] M. Jiang, O. W. Visser, I. Prasetya, A. Iosup, Mirror: A Computation-offloading Framework for Sophisticated Mobile Games, in: 18th IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM 2017), IEEE, Macau SAR, China, 2017, pp. 1–3. doi:10.1109/WoWMoM.2017.7974351.

[201] Z. Dong, Y. Liu, H. Zhou, X. Xiao, Y. Gu, L. Zhang, C. Liu, An Energy-efficient Offloading Framework with Predictable Temporal Correctness, in: 2nd ACM/IEEE Symposium on Edge Computing (SEC '17), ACM, San Jose, CA, 2017, p. 19. doi:10.1145/3132211.3134448.

[202] J. L. D. Neto, S.-Y. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, S. Secci, ULOOF: A User Level Online Offloading Framework for Mobile Edge Computing, IEEE Transactions on Mobile Computing (TMC) 17 (11) (2018) 2660–2674. doi:10.1109/TMC.2018.2815015.

[203] Z. Kuang, S. Guo, J. Liu, Y. Yang, A quick-response framework for multi-user computation offloading in mobile cloud computing, Future Generation Computer Systems 81 (2018) 166–176. doi:10.1016/j.future.2017.10.034.

[204] H. Zhao, S. Deng, C. Zhang, W. Du, Q. He, J. Yin, A Mobility-Aware Cross-Edge Computation Offloading Framework for Partitionable Applications, in: IEEE International Conference on Web Services (ICWS 2019), IEEE, Milan, Italy, 2019, pp. 193–200. doi:10.1109/ICWS.2019.00041.

[205] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, A. Kitazawa, Fogflow: Easy programming of iot services over cloud and edges for smart cities, IEEE Internet of Things Journal 5 (2) (2018) 696–707. doi:10.1109/JIOT.2017.2747214.

[206] M. S. de Brito, S. Hoque, T. Magedanz, R. Steinke, A. Willner, D. Nehls, O. Keils, F. Schreiner, A Service Orchestration Architecture for Fog-enabled Infrastructures, in: Second International Conference on Fog and Mobile Edge Computing (FMEC 2017), IEEE, Valencia, Spain, 2017, pp. 127–132. doi:10.1109/FMEC.2017.7946419.

[207] W. Hu, B. Amos, Z. Chen, K. Ha, W. Richter, P. Pillai, B. Gilbert, J. Harkes, M. Satyanarayanan, The Case for Offload Shaping, in: 16th ACM Workshop on Mobile Computing Systems and Applications (HotMobile 2015), ACM, Santa Fe, Mexico, 2015, pp. 51–56. doi:10.1145/2699343.2699351.

[208] A. Toma, J.-J. Chen, Computation Offloading for Real-time Systems, in: 28th Annual ACM Symposium on Applied Computing (SAC 2013), SAC '13, ACM, Coimbra, Portugal, 2013, pp. 1650–1651. doi:10.1145/2480362.2480670.

[209] A. Toma, S. Pagani, J.-J. Chen, W. Karl, J. Henkel, An Energy-Efficient Middleware for Computation Offloading in Real-Time Embedded Systems, in: IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2016), IEEE, Daegu, South Korea, 2016, pp. 228–237. doi:10.1109/RTCSA.2016.50.

[210] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, R. Buyya, mCloud: A Context-aware Offloading Framework for Heterogeneous Mobile Cloud, IEEE Transactions on Services Computing 10 (5) (2017) 797–810. doi:10.1109/TSC.2015.2511002.

[211] N. Fernando, S. W. Loke, W. Rahayu, Mobile cloud computing: A survey, Future Generation Computer Systems 29 (1) (2013) 84–106. doi:10.1016/j.future.2012.05.023.

[212] H. Petander, Energy-Aware Network Selection Using Traffic Estimation, in: 1st ACM Workshop on Mobile Internet Through Cellular Networks (MICNET 2009), ACM, Beijing, China, 2009, pp. 55–60. doi:10.1145/1614255.1614268.

[213] T.-s. Kim, R. Oh, S.-J. Lee, S.-H. Yoon, C.-H. Cho, S.-W. Ryu, Vertical Handover between LTE and Wireless LAN Systems based on Common Resource Management (CRRM) and Generic Link Layer (GLL), in: 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human (ICIS 2009), ACM, Seoul, South Korea, 2009, pp. 1160–1166. doi:10.1145/1655925.1656136.

[214] A. Ravi, S. K. Peddoju, Handoff Strategy for Improving Energy Efficiency and Cloud Service Availability for Mobile Devices, Wireless Personal Communications 81 (1) (2015) 101–132. doi:10.1007/s11277-014-2119-y.

[215] H. Suo, Z. Liu, J. Wan, K. Zhou, Security and Privacy in Mobile Cloud Computing, in: 9th International Wireless Communications and Mobile Computing Conference (IWCMC 2013), IEEE, Sardinia, Italy, 2013, pp. 655–659. doi:10.1109/IWCMC.2013.6583635.

[216] C. Wang, K. Ren, J. Wang, Secure and Practical Outsourcing of Linear Programming in Cloud Computing, in: 30th IEEE Conference on Computer Communications (INFOCOM 2011), IEEE, Shanghai, China, 2011, pp. 820–828. doi:10.1109/INFCOM.2011.5935305.

[217] A. Shabtai, U. Kanonov, Y. Elovici, Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method, Journal of Systems and Software 83 (8) (2010) 1524–1537. doi:10.1016/j.jss.2010.03.046.

[218] U. Khalid, A. Ghafoor, M. Irum, M. A. Shibli, Cloud Based Secure and Privacy Enhanced Authentication & Authorization Protocol, Procedia Computer Science 22 (2013) 680–688. doi:10.1016/j.procs.2013.09.149.

[219] D. Huang, X. Zhang, M. Kang, J. Luo, MobiCloud: Building Secure Cloud Framework for Mobile Computing and Communication, in: 5th IEEE International Symposium on Service Oriented System Engineering (SOSE 2010), IEEE, Nanjing, China, 2010, pp. 27–34. doi:10.1109/SOSE.2010.20.

[220] S. Bugiel, S. Nürnberger, A.-R. Sadeghi, T. Schneider, Twin Clouds: Secure Cloud Computing with Low Latency, in: 12th IFIP TC 6/TC 11 International Conference on Communications and Multimedia Security (CMS 2011), Springer, Gent, Belgium, 2011, pp. 32–44. doi:10.1007/978-3-642-24712-5_3.

[221] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, S. Jeong, Securing Elastic Applications on Mobile Devices for Cloud Computing, in: ACM Workshop on Cloud Computing Security (CCS 2009), ACM, Chicago, IL, 2009, pp. 127–134. doi:10.1145/1655008.1655026.

[222] M. La Polla, F. Martinelli, D. Sgandurra, A Survey on Security for Mobile Devices, IEEE Communication Surveys & Tutorials 15 (1) (2013) 446–471. doi:10.1109/SURV.2012.013012.00028.

[223] J. Liu, K. Kumar, Y.-H. Lu, Tradeoff between Energy Savings and Privacy Protection in Computation Offloading, in: ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED 2010), ACM, Austin, TX, 2010, pp. 213–218. doi:10.1145/1840845.1840887.

[224] A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, P. Hui, SmartDiet: offloading popular apps to save energy, ACM SIGCOMM Computer Communication Review (CCR) 42 (4) (2012) 297–298. doi:10.1145/2377677.2377739.