



Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Energy-efficient and network-aware offloading algorithm for mobile cloud computing

Chathura M. Sarathchandra Magurawalage^a, Kun Yang^{a,*}, Liang Hu^b, Jianming Zhang^c

^a School of Computer Science and Electronic Engineering, University of Essex, Colchester, UK

^b School of Computer Science and Technology, Jilin University, Changchun, China

^c School of Computer and Communication Engineering, Changsha University of Science and Technology, Changsha 410114, China

ARTICLE INFO

Article history:

Received 10 January 2014

Received in revised form 25 April 2014

Accepted 19 June 2014

Available online xxxx

Keywords:

Mobile cloud computing

Mobile clone

Mobile cloudlet

Offloading algorithm

Energy efficiency

ABSTRACT

We propose a new system architecture for mobile cloud computing (MCC) that includes a middle layer sitting between mobile devices and their cloud infrastructure or clones. This middle layer is composed of cloudlets and is thus called a cloudlet layer. Cloudlets are deployed next to IEEE 802.11 access points and serve as a localized service point in close proximity to mobile devices to improve the performance of mobile cloud services. On top of this new architecture, an offloading algorithm is proposed with the main aim of deciding whether to offload to a clone or a cloudlet. The decision-making takes into consideration the energy consumption for task execution and the network status while satisfying certain task response time constraints. We also introduce a data caching mechanism at cloudlets to further improve the overall MCC performance. Simulation results demonstrate the effectiveness and efficiency of the proposed system architecture and offloading algorithm in terms of response time and energy consumption.

© 2014 Published by Elsevier B.V.

1. Introduction

The past few years have witnessed a rapid shift in computing from the desktop to the cloud. To keep pace with advances in both wireless network technologies and mobile smart phones, there is an increasing need for provision of cloud services to mobile users via mobile wireless networks. This new research field is called mobile cloud computing (MCC) [1–5]. Because mobile devices, even modern smart phones, are constrained in size and weight, their resources for computation and communication are limited compared to their desktop counterparts [6]. Therefore, it makes sense to offload heavy mobile applications to more powerful machines in the cloud. Computing and

service delivery are possible because of the advanced sensors built into most mobile phones currently on the market; these sensors include accelerometers, magnetometers, GPS chips, gyroscopes, and pressure sensors. The more sensors a device has, the more data need to be analyzed in various domains at the same time, which accentuates the need for more computational power. One critical issue in MCC is how battery power for mobile devices can be spared [7]. One effective approach is to offload some tasks from the mobile device to a remote cloud server for execution. This can also potentially reduce the task execution time because of the power of cloud servers. Kumar and Lu investigated the power consumption of mobile devices, including whether offloading can increase battery life [8]. Offloading should only occur when it is beneficial to the mobile application. Thus, tasks should only be offloaded to the cloud if the sum of the data transmission cost and the energy cost is smaller than when the tasks are executed locally on the mobile device.

* Corresponding author.

E-mail addresses: csarata@essex.ac.uk (C.M. Sarathchandra Magurawalage), kunyang@essex.ac.uk (K. Yang), hul@jlu.edu.cn (L. Hu), jmzhang@csust.edu.cn (J. Zhang).

<http://dx.doi.org/10.1016/j.comnet.2014.06.020>

1389–1286/© 2014 Published by Elsevier B.V.

Please cite this article in press as: C.M. Sarathchandra Magurawalage et al., Energy-efficient and network-aware offloading algorithm for mobile cloud computing, Comput. Netw. (2014), <http://dx.doi.org/10.1016/j.comnet.2014.06.020>

We contribute to the exciting MCC research field by investigating a key problem for mobile application offloading. There is much work being carried out in this area, which largely falls into two categories: (1) task partitioning, which involves dividing an application into offloadable partition(s) and a local partition [2]; and (2) virtual machine (VM) selection [9], which involves choosing an appropriate VM to which to offload a partition. VMs are the key component of a cloud and they provide virtual resources such as CPU, memory, storage, and network interfaces in the same way as physical resources do. One common element of this research is the assumption that there is a perfect network connection and sufficient bandwidth between a mobile application on a mobile device and the remote cloud VM. This may not be an unrealistic assumption for wired networks, for which network bandwidth is usually abundant or at least not scarce. However, this is not the case for wireless networks [10], for which network bandwidth is not as abundant and a network connection may sometimes not even be available; this is more of a problem for mobile cellular networks such as 3G.

To address this issue, we use a new network-awareness perspective: in addition to considering network status parameters such as bandwidth and delays, we also take network types into account. We consider two types of wireless network in this study: IEEE 802.11 (i.e., WiFi) and mobile cellular networks such as 3G and LTE. These are the two mainstream wireless technologies that people interact with on a daily basis. Considering one without the other is not realistic and leads to a situation in which a mobile cloud system is not as efficient as it should be. This consideration is reflected in our proposed MCC system architecture, in which a middle layer is introduced between mobile devices and their corresponding cloud clones. This middle layer, called the cloudlet layer, is deployed next to WiFi access points (APs) in the proximity of mobile devices. The aim is to run clone equivalents named cloudlets in this layer, as illustrated in Fig. 1. The benefits of this approach are twofold. First, it can take advantage of the higher bandwidth of WiFi and switch the network connection from 3G to WiFi. Second, data caching can be carried out on cloudlets to some extent for applications such as data downloading from the Internet. 3G networks are typically proprietary and are not open to MCC service providers, whereas WiFi networks can be easily deployed. Otherwise, MCC service providers can relatively easily install their offloading software on the home servers of end users or on gateways next to their home APs. Our proposed MCC infrastructure is intended for MCC service providers rather than network providers or operators. As a result, our proposed offloading algorithm focuses on the offloading location (clone or cloudlet) when offloading a mobile application. By contrast, most of the current literature on offloading focuses on *whether* to offload a mobile application or not. Cloudlets can also be placed next to cellular base stations, as illustrated by the blue line (connecting a mobile device, cloudlet, and clone) in Fig. 1. However, this issue is beyond the scope of the present study and will be investigated in future work.

In summary, we propose a new MCC system architecture that contains a cloudlet middle layer above the exist-

ing cloud server infrastructure. We also propose an offloading algorithm that decides on where to offload a given mobile application. The objectives of the offloading algorithm are twofold: (1) to minimize the service response time, which is a direct factor of the quality of experience of cloud end users and (2) to save the battery life of mobile devices.

In comparison to the existing literature on MCC offloading, we make the following two major contributions:

- We propose a new MCC system architecture with a cloudlet layer sitting between mobile devices and their traditional cloud infrastructure or clones. Cloudlets are deployed next to IEEE 802.11 APs and serve as a localized service point in close proximity to mobile devices, which improves the performance of MCC services.
- On top of this new architecture, an offloading algorithm decides whether to offload to a clone or a cloudlet. The decision-making takes into consideration the energy consumption for task execution and the network status, while satisfying certain task response time constraints. Note that the energy efficiency considered here is from the perspective of mobile devices rather than the cloud servers.

The remainder of the paper is organized as follows. Related work is described in Section 2. Section 3 presents the proposed MCC system architecture, detailing each component and their relationships. Section 4 describes the proposed energy-efficient and network-aware offloading algorithm and an energy model in the context of our proposed MCC system architecture, and the two types of wireless network considered. The simulation results and performance analysis are presented in Section 5. The paper concludes with Section 6.

2. Related work

Our literature review focuses on the two main contribution areas: MCC system architecture and offloading algorithms. Since the core is offloading algorithms the discussion on MCC system architecture is from the perspective of offloading support.

2.1. MCC infrastructure for offloading

CloneCloud [3] is an elastic cloud execution framework for mobile devices. It incorporates automatic static and dynamic analysis of code at runtime before migration/offloading and dynamic profiling to partition applications, without having to make any changes to the code. Code partitioning is at the thread level with fine granularity. This approach is less scalable because bootstrapping is required for every new application and only limited execution environments and input conditions are considered during off-line code preprocessing. The Thinkair [4] and Cuckoo [11] frameworks provide scalable, online, and method-level code offloading mechanisms with on-demand resource

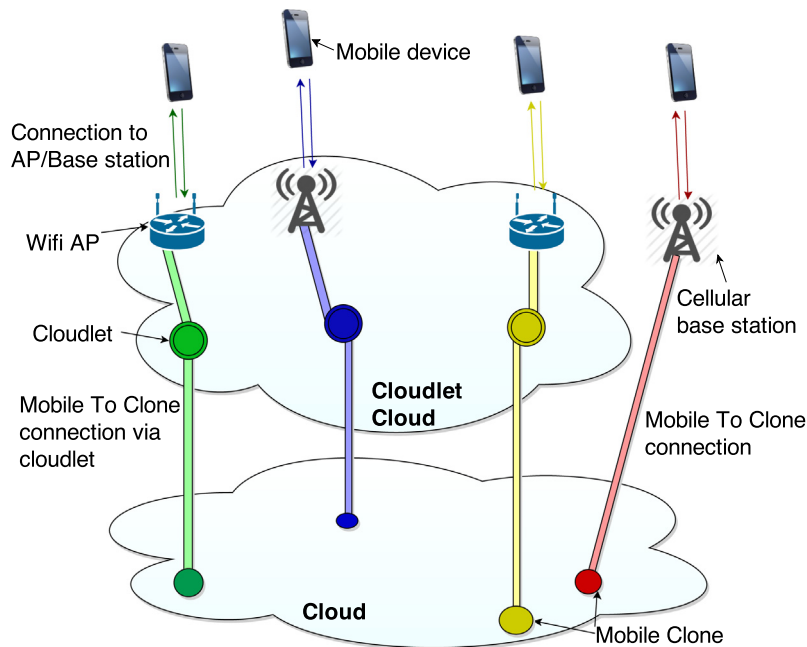


Fig. 1. Illustration of task offloading with clones and cloudlets.

allocation [12]. Cuckoo computational offloading is possible to any resource running a Java VM. Thinkair provides scalability by connecting to a VM with a pure smart phone operating system in the cloud for offloading. Thus, it eliminates the environmental, input, and output restrictions placed on offloaded code by other offloading frameworks such as CloneCloud [3]. The Thinkair execution controller decides whether a method should be offloaded or not. This decision is based on information collected by hardware, software, and network profilers. Thinkair exploits parallelism on the cloud side by allowing the creation, resumption, and destruction of multiple VMs in the cloud for parallelization. In addition, Thinkair does not choose where the cloud clone is hosted. The clone host location may affect the network latency and round trip time, which may have a negative effect on energy efficiency.

Clone2Clone [13] allows users to create their own clone in the cloud or to request CloneDS, the Clone2Clone directory service, to create a clone on their behalf. This introduces a new paradigm in which there is dedicated VM in the cloud for each mobile device. Clones can form secure peer-to-peer networks for content sharing, searching, and distributed code execution. This eliminates the dilemma of having unpredictable and energy inefficient wireless networks, considering that clones in the cloud have solid always-on, high-bandwidth networks. Other mechanisms can be used to offload computationally heavy tasks to the cloud [3,4,14,15], for which the architecture is also limited to networks with low bandwidth and high latency.

CDroid [16] uses another MCC approach in which a secured tunnel is established between a mobile device and its cloud clone for all Internet traffic. The cloud clone appears as a local resource for the mobile device. This improves web navigation, compression and caching of

web pages, and blocking unwanted advertising and virus scanning applications prior to installation. A major drawback is that CDroid requires an always-on connection to the cloud clone and all traffic must go through the clone, which is not energy-efficient.

VM-based cloudlets [17–19] are decentralized and widely distributed internet infrastructure components whose computation and storage resources can be leveraged by nearby mobile devices. Satyanarayanan et al. emphasized the necessity of physical proximity of the cloudlet to the wireless AP so that mobile devices benefit from only having to traverse one hop using high-bandwidth, low-latency wireless networks [17]. Kimberley is the current reference for cloudlet implementation. It uses two approaches to introduce the VM state to the infrastructure. (1) The VM is suspended and its processor, memory, and disk states are transferred to the cloudlet, then the VM resumes at the exact point at which it was suspended. (2) In dynamic VM synthesis, the mobile device delivers a small VM overlay to the cloudlet infrastructure, which already possesses the base VM from which this overlay was derived. Parallelized VMs can be spawned on demand on the cloudlet side to achieve faster execution. Kimberley does not consider energy utilization on the mobile side, but resource-heavy operations can drain the battery of a mobile device. Device mobility is one of the major cloudlet issues for mobile users on the move while connected to cloudlets. Kimberley does not consider this factor. Our proposed architecture addresses the aforementioned limitations of Kimberley.

Placement of a number of cloudlets belonging to various service providers next to each other can occupy a lot of space. Collaboration among cloud service providers to form a coalition for transparent customer service could

lead to broader cloudlet coverage. Niyato et al. used game theory to model coalitions among cloud service providers [20].

The MCC architectures described above only consider the implications of using highly resource-rich computers on the cloud side when delegating tasks. Recent studies have investigated the use of other nearby mobile devices with common interests when offloading computationally heavy tasks. Nguyen et al. showed that mobility increases MCC processing capacity [21]. Mobility also affects the performance and resilience of mobile clouds. This means that addition of even a small number of highly mobile nodes to a highly localized network can significantly improve the processing capacity and resilience. The main drawback of this approach is that the battery life of current mobile devices is not sufficiently energy-efficient. Cloud clones inhabit one place at one time, with little regard for mobility, in contrast to the highly mobile devices they serve. Hence, a comprehensive investigation of clone placement and clone migration methodologies is required.

2.2. MCC offloading algorithm

Lin et al. proposed a context-aware algorithm that uses historical log records to determine whether to offload tasks or not [22]. The offloading algorithm considers the user location at a certain time of day when tasks are offloaded for remote execution. A task is offloaded if the energy consumption of the mobile device when the task was previously offloaded is lower than when it was executed locally for the same time of day and the same geographic location. This approach relies on historical records that might not be valid for present conditions and could lead to inaccurate offloading decisions.

Kovachev et al. addressed adaptive computation offloading as an optimization problem using integer liner programming [23]. This approach considers available memory and CPU and energy usage as the criteria for offloading. The algorithm dynamically chooses what services to offload by solving a new optimization problem each time parameters such as the available bandwidth and memory change in the model. The offloading decision model of Wu et al. takes network unavailability into consideration [24]. The model uses an application partitioning algorithm, and an offloading decision module intelligently decides on whether to offload by considering the network availability for remote execution. The CRoSS algorithm also selects the best host for offloading according to the link cost [25]. The link cost includes both the link failure rate and the bidirectional transmission rate.

In two previous studies on offloading algorithms, bandwidth was the only network parameters considered [8,26]. Furthermore, the authors assumed that the same power consumption is required for sending and receiving data, but Feeney and Nilsson showed that wireless network interfaces exhibit a complex range of consumption behaviors [27]. Hence, factors such as packet size, the number of broadcasts, and point-to-point traffic need to be considered when designing energy-aware offloading protocols. Wen et al. considered the features of wireless channels and showed that execution policies depend on the input

data size and completion deadline for the application, as well as the wireless transmission model [28].

Clock frequency is an important computing parameter. The mobile device energy can be optimized by scheduling the clock frequency via dynamic voltage scaling [29] because the CPU clock frequency is approximately linearly proportional to the voltage supply. Another important issue that is often neglected is multisite offloading. A user may have connectivity to more than one wireless network and the mobile device may be within reach of a VM on each network. In such cases, the mobile device can decide whether to use two VMs or to offload to only one VM. For example, there may be cases in which the connected VM does not have enough resources to complete offloaded tasks within the set deadline. Thus, the mobile device may decide to offload code simultaneously to two VMs on two different wireless networks, such as a WiFi network and a cellular network. A previous study on multisite offloading only considered a graph partitioning approach in finding a solution to the partitioning problem [30].

3. Proposed offloading architecture

In this section, we describe user issues for conventional offloading architectures and propose an architecture with solutions that overcome these issues. We also validate the solutions.

3.1. Scenarios

A mobile device user may offload code or data to a dedicated cloud clone directly. The mobile device communicates with the clone using a cellular network via the Internet. Because it uses networks with low bandwidth and high latency, this conventional offloading approach can be improved.

Because of mobility, device users may lose connectivity to a cloudlet, but might reconnect to the same cloudlet later or might connect to a different cloudlet. A user may also lose connectivity to a cloudlet, but might not reconnect to it for a long time. In all these cases, the offloaded code and data should not be lost during provision of a seamless service.

3.2. Proposed MCC system architecture

The novel feature of our proposed architecture is that it comprises two supported layers: a mobile clone cloud and a cloudlet cloud. Current popular offloading frameworks do not support both layers simultaneously. Our architecture is built on the Thinkair framework [4] and is shown in Fig. 2. Our aim in this paper is only to propose a conceptual model of the architecture. Its implementation is still ongoing and will be finalized in future work.

A mobile clone is a VM hosted by a public cloud service provider with an application offloading server (AOS). Every mobile device is assigned a clone in the cloud for offloading and caching purposes. Clones can communicate with other clones and with cloudlets. A cloudlet is a VM hosted by a resource-rich machine placed next to an AP or a cellular

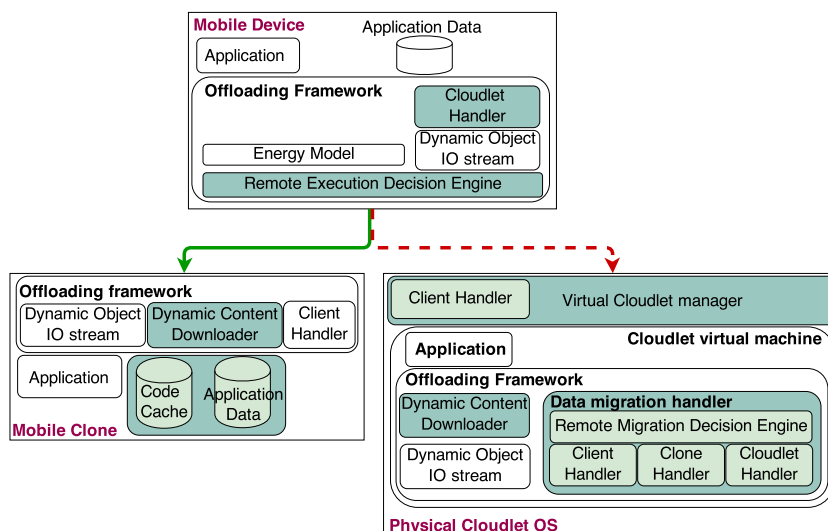


Fig. 2. Proposed architecture for MCC offloading.

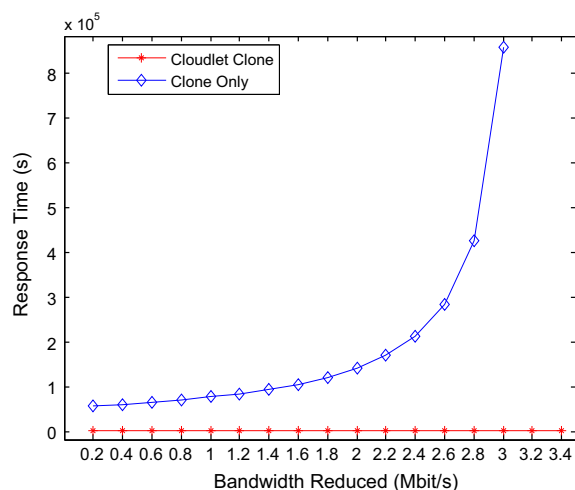


Fig. 3. Response time for a conventional architecture and Cloudlet + Clone when the bandwidth is reduced by a fixed amount.

base station. Cloudlets also hold a type of AOS. They are interconnected using a separate network for transferring user data.

3.2.1. Components

This section describes component functions and their interconnections, which are denoted by arrows in Fig. 2. Device–clone links are shown by red arrows, device–cloudlet links by green arrows, and cloudlet–clone links by blue arrows. A discussion of the modules inherited from the Thinkair architecture is beyond the scope of this paper. Thus, only the new components are explained.

3.2.1.1. Cloudlet handler. When a user is in within range of a cloudlet, the cloudlet handler connects the users mobile device to the nearest cloudlet. Cloudlets can be placed next

to WiFi APs and to cellular base stations, so the mobile device can save energy by only turning on the WiFi interface when it is within reach of a cloudlet under the control of the cloudlet handler. The handler queries the nearest mobile base station for the nearest cloudlets. If a cloudlet is available for connection, the cloudlet handler turns on the WiFi interface and connects to the cloudlet using a well-known public IP.

3.2.1.2. Remote execution decision engine. The remote execution decision engine is the intelligent component of the mobile offloading framework. It decides on *Whether* and *Where* to offload data. In other words, it decides if an offloadable block of code (method) annotated by the programmer should be offloaded (*Whether*) to a clone or cloudlet (*Where*). The embedded algorithm used to make this decision is presented in Section 4.

3.2.1.3. Dynamic content downloader. Dynamic offloader downloads are placed in the clone and in the cloudlet. The downloader downloads input data for tasks when a mobile device offloads computing tasks through the internet, so that the mobile device does not have to transmit input data with the code, which saves energy. The input data are downloaded using the fixed network to which the clone or cloudlet is connected, as shown by black arrows in Fig. 2. This module is equipped with a utility that can be used to download content by providing a URL, such as GNU Wget [31].

3.2.1.4. Data migration handler. The data migration handler manages data transmitted between a mobile device and its clone. Similar to Thinkair [4], the offloading framework on the mobile device connects to the client handler on the cloudlet, and the dynamic migration handler connects to the dynamic object input/output stream on the clone and the mobile device. The clone handler in the cloudlet can initiate a session to connect to the client handler in the

clone. The client handler in the cloudlet waits and accepts new connections from mobile devices. Only mobile device can be connected to one cloudlet at any instant. If WiFi coverage fluctuates, the connection to a cloudlet may drop when the user is moving. The remote data migration decision engine decides on whether to migrate unfinished jobs with their data to the clone or not.

When a mobile device connects to a cloudlet, the URL, clone authentication details, and details for the previous cloudlet to which it was connected (if applicable) are transferred to the cloudlet. Then the remote migration decision engine connects to the clone and automatically downloads the cached data and code to the cloudlet to speed up the offloading process. When the mobile device connects to the next nearest cloudlet, the handler of the new cloudlet connects to the handler of the previous cloudlet to retrieve the code/data. If the user is not too late, the new cloudlet may receive the requested data. If the old cloudlet does not receive any requests for user data after a short time period, it transfers the data back to the user's dedicated clone to prevent data loss. This is ideal if many cloudlets are available when the user is moving, as it provides a seamless cloudlet service, which we assume occurs in crowded areas such as busy shopping malls and town centers.

3.2.1.5. Code cache and application data. Our approach utilizes any available clone storage by allowing the offloading framework to cache frequently offloaded codes with application input data in the clone. A record of the cached code and data is also kept so that mobile devices can discover whether the code and/or input data already exist on clone so that they do not have to be offloaded. In addition, when a user loses connection to a cloudlet, the cloudlet storage system can transmit unfinished/finished jobs back to the user's dedicated cloudlet so that the clone can execute the code and/or the user can access the data later.

3.2.1.6. Virtual cloudlet manager. The offloading framework in the cloudlet is placed in a VM (cloudlet VM) hosted by a visualization platform such as KVM [32].

3.3. Features of the proposed MCC system architecture

- 1. Dynamic adaptation to a changing environment:** In the proposed architecture, the offloading framework [4] implements code migration such that the code is offloaded to be executed remotely at the most suitable time. Two offloading destinations are available to the offloading framework, clones and cloudlets, so the framework can decide *Where* to offload, depending on the user's environment. Users offloaded data are always backed up to their dedicated clones at all times to prevent data loss.
- 2. Performance improvement via cloudlets and clones:** Offloading of code to more resource-rich servers in cloud clones can improve the performance of mobile devices. With the increasing bandwidth available, users expect a faster mobile experience. The bottleneck for this is the mobile network status, specifically the response time and available bandwidth

when performing online operations. In the proposed architecture, VM-based cloudlets [17] are deployed closer to the mobile device (at a one-hop distance) as temporary points for code execution because the distance and number of hops directly affect the response time and energy consumption [33]. Users can access cloudlets via high-bandwidth WiFi networks. This can be perceived as bringing the cloud closer to mobile devices. However, we consider clones as permanent execution points because we assume that every mobile device has a dedicated clone for offloading.

- 3. Faster code execution via caching and data localization:** Long-term caches of remote code can be stored in the user-specific clone, so that the mobile device will not have to send code to the clone when offloading and cloudlets can download cached code from the clone to reduce mobile data traffic.

4. Dynamic offloading algorithm

The offloading algorithm embedded in the *remote execution decision engine* decides on *Whether* and *Where* to offload data. The novelty of this algorithm is that it considers more than one offloading location as a parameter when deciding *Where* to offload. By contrast, conventional offloading algorithms support just one offloading location [34]. In the remainder of the paper, we assume that cloudlets are only located next to WiFi APs and mobile devices access clones using their cellular network via the Internet, as shown by green¹ and red lines in Fig. 1.

Before application execution, the decision-making process seeks to reduce the total energy consumed by the mobile device prior to execution [28]. We apply a novel energy- and time-aware offloading algorithm that takes two offloading locations into account, as shown in Fig. 1. The energy model considers ramp, tail, and maintenance measures when estimating energy, unlike energy models used by existing offloading frameworks.

We use a simple mobility model similar to that of Huang et al. [35]. We assume that the availability of wireless networks and their data transmission rates vary according to the current location of the mobile device. In contrast to Huang et al. [35], we assume that the availability of networks for the mobile user may change during the remote execution time for one application.

We first show how to estimate the task execution time and the device energy consumption. Then we describe the algorithm that uses these estimates to make offloading decisions. The notations used are listed in Table 1.

4.1. Estimating the response time and energy for local execution

The execution time for task j is estimated as

$$t_{mob}^j = \frac{t^j}{\mu_{cpu}}. \quad (1)$$

¹ For interpretation of color in Fig. 1, the reader is referred to the web version of this article.

Table 1

Notation conventions.

Symbol	Unit	Description
j	MIPS	Number of instructions for completion of task j
T	Seconds	Task hard deadline
t_{mob}^j	Seconds	Execution time for task j on a mobile device
t_{cloud}^j	Seconds	Execution time for task j in a cloud (cloudlet or clone)
E_{local}	J	Energy consumption by a mobile device for task execution
E_{cloud}	J	Energy consumption by a mobile device when a task is offloaded to a cloud
E_{nic}	J	Energy consumed by a network interface (E_{nic}^{wifi} , E_{nic}^{3g} , E_{nic}^{lte})
μ_{cpu}	MIPS	Mobile CPU speed
μ_{cloud}	MIPS	Cloud speed
B	Kbps	Transmission bandwidth
d_j	KB	Data that need to be transmitted for task j
P_{trans}	Watt	Power consumption of the network interface
P_{basic}	Watt	Baseline power consumption when the mobile device is idle
P_{comp}	Watt	Power consumption of mobile CPU
κ_v	%	Proportional reduction in execution time compares to mobile execution
γ_v	%	Proportional reduction in energy compared to mobile execution

The energy consumption for local execution is given by

$$E_{local} = (P_{comp} \times t_{mob}^j). \quad (2)$$

4.2. Estimating the response time and energy for offloading

The response time for offloading of tasks to a remote cloud server is calculated as

$$t_{cloud}^j = \frac{d_j}{B} + \frac{t^j}{\mu_{cloud}}. \quad (3)$$

We assume that a mobile device accesses its cloud clone or cloudlet using a WiFi and/or cellular network, so calculation of the energy consumption for operations performed on these interfaces is crucial. Zhang et al. carried out a detailed study of the power consumption of mobile device components using PowerTutor, an energy estimation tool that considers CPU, LCD, GPS, WiFi, audio, and cellular interfaces [36]. We used a similar model for WiFi and cellular networks, although we only consider power consumed when a task is executed on a cloud and when data are transferred. The energy consumption is estimated according to

$$E_{cloud} = P_{basic} \times t_{cloud}^j + E_{nic}, \quad (4)$$

where P_{basic} is the baseline power consumption of the mobile device when it is idle, t_{cloud}^j is the time for task execution in the cloud, and E_{nic} is the energy consumed by the network interface when offloading to the cloud. If $E_{nic} = E_{nic}^{wifi}$, then the equation yields the energy consumption for offloading to a cloudlet.

As empirically observed by Zhang et al. [36], the packet rate, not the bit rate, determines the power state of the WiFi interface. In other words, the packet size does not influence power consumption given a fixed channel state and packet rate. However, the authors did not take the energy for connection establishment into account [37]. We use a similar model but also take the connection estab-

lishment energy E_e into account. We express the system energy cost for establishing and transferring n bytes as

$$E_{nic}^{wifi} = E_e + n \cdot E_{trans}. \quad (5)$$

We assume that there is no significant difference in power consumption between receiving and transmitting packets, so we use the same values to represent transmitted and received energy.

Models of energy consumption for cellular networks consist of three components [38]: (1) *ramp* energy e_{ramp} , (2) *transmission* energy e_{trans} , and (3) *tail* energy e_{tail} . Unlike Zhang et al. [36], we consider radio resource control states. The power amplifier of a mobile cellular interface switches to a higher power mode to counter the drop in signal strength [39] when transferring and receiving data. Cellular base stations use feedback received from mobile devices some 800–1600 times per second, and choose an appropriate modulation scheme and data rate. Hence, a strong signal allows for high data rates and shorter data transfer times. It can be concluded that when the signal is weak, data transfer take longer to complete and the radio interface draws higher power. Considering these factors, the energy model for cellular data transmission is given by

$$E_{nic}^{3g} = e_{ramp} + e_{trans} + e_{tail} \times t_{tail}. \quad (6)$$

4.3. Decision-making

At runtime, the offloading framework dynamically requests the cloud server speed of the cloudlet μ_{cloud}^{ct} from the connected cloudlet; we assume that the speed of the dedicated clone, μ_{cloud}^{cl} , is already known by the mobile device. The framework then calculates the proportional reduction in execution time for offloading the task to a clone or cloudlet as

$$\kappa_v = \frac{t_{mob} - t_{cloud}^v}{t_{mob}}, \quad (7)$$

where the subscript v can be either ct or cl , denoting cloudlet and clone, respectively. If Eq. (7) yields a positive value,

offloading will not be time-efficient. A hard deadline T_j is defined for task j . This deadline must be met and it is preferable to choose the offloading location that executes the task in the shortest time, even though energy consumption has a higher priority than execution time in the offloading decision engine.

The proportional reduction in energy consumption for offloading a task to a clone or a cloudlet is given by

$$\gamma_v = \frac{E_{local} - E_{cloud}^v}{E_{local}}. \quad (8)$$

If Eq. (8) yields a positive value, offloading will not be energy-efficient. It is obvious that if $\gamma_{mob} = 0$ and $\kappa_{mob} = 0$, then tasks are executed locally. If $\gamma_{cl} > 0$ and $\kappa_{cl} > 0$, then it is energy- and time-efficient to offload the task to a dedicated clone. If $\gamma_{ct} > 0$ and $\kappa_{ct} > 0$, then it is energy- and time-efficient to offload the task to a connected cloudlet or clone.

The proposed offloading approach is presented in Algorithm 1, which favors the execution time over energy consumption as the criterion for task offloading. This is because the application will fail to reach quality of service (QoS)/quality of experience (QoE) requirements if the task deadline is not met. By contrast, the execution time has a direct affect on energy consumption as defined in Section 4.2. If offloading of a selected task to either a clone or a connected cloudlet cannot improve the execution time, the task will not be offloaded (line 1). If the cloudlet can execute the task faster than the clone can, and the energy cost is approximately equal for offloading to the cloudlet or the clone while meeting the deadline, the task will be offloaded to the cloudlet, and vice versa (lines 5–9). Next, if the mobile device can save more energy by offloading to the cloudlet than to clone while meeting the deadline, the task will be offloaded to the cloudlet, otherwise the clone will be selected as the offloading destination (lines 10–15). If none of the above conditions are met, the task is executed locally (line 16).

5. Performance evaluation and analysis

The name Cloudlet + Clone is used hereafter to refer to our proposed architecture. The terms CloudletClone and CloneOnly in figures refer to Cloudlet + Clone and a conventional architecture, respectively.

5.1. Simulation set-up

We conducted several simulations to study the performance of the proposed MCC system architecture and the offloading algorithm. We implemented the algorithm using CloudSim 2.0, an extensible tool kit for modeling and simulation of cloud computing environments that supports modeling of VMs on a simulated node of a data center [40,41]. The effectiveness and efficiency of our approach were evaluated mainly in terms of the average service response time and the average energy consumption by a mobile device. Comparisons were made among the following three scenarios: mobile only, clone only, and Cloudlet + Clone.

Algorithm 1. Energy-efficient and network-aware decision algorithm

```

1:  $\forall v$  calculate  $\gamma_v$  &  $\kappa_v$ 
2: if  $\kappa_{cl} \leq 0$  &  $\kappa_{ct} \leq 0$  then
3:   return Do not offload
4: else
5:   if  $\kappa_{ct} > \kappa_{cl}$  &  $\gamma_{ct} \approx \gamma_{cl}$  &  $t_{cl}^j \geq T$  then
6:     return Offload to Cloudlet
7:   else if  $\kappa_{cl} > \kappa_{ct}$  &  $\gamma_{cl} \approx \gamma_{ct}$  &  $t_{cl}^j \geq T$  then
8:     return Offload to Clone
9:   end if
10:  if  $\gamma_{ct} > \gamma_{cl}$  &  $t_{cl}^j \geq T$  then
11:    return Offload to Cloudlet
12:  else if  $\gamma_{cl} > \gamma_{ct}$  &  $t_{cl}^j \geq T$  then
13:    return Offload to Clone
14:  end if
15: end if
16: Do not offload

```

The simulation environment was configured as follows. We assumed that the network state was constant for every offloading instance, for example, that the bandwidth and link latency for the uplink and downlink were constant throughout the execution process for one task. The CPU intensity of a task is represented in million instructions per second (MIPS). The proposed architecture has two types of network available for offloading. When offloading, the time taken to execute a task in a particular environment may be taken into consideration because of task headline values to achieve the promised QoE level. Hence, we measured differences in response time between Cloudlet + Clone and other popular architectures for task execution.

For the evaluation we assumed that the CPU power (MIPS) of the remote server is double that of the mobile device and that the size is fixed throughout. This is a fair assumption because the resources are much greater for VMs in the cloud than for mobile devices and can be resized on demand. Hence, the computational power of a cloud clone or cloudlet can exceed double the power of a mobile device. We chose a mobile device speed of 500 MIPS for a clone and 1000 MIPS for a cloudlet.

We assumed that the maximum bandwidth of the wireless network for Cloudlet + Clone is 300 Mbit/s since the cloudlets are located next to WiFi APs, although the bandwidth can be even higher for wireless 802.11n devices. For offloading frameworks that only support clones, the bandwidth is 15 Mbit/s and the latency is 0.020 s when accessing a clone hosted by a cloud service provider. The average latency was measured experimentally using a real mobile device with a 3G or WiFi network enabled in various geographic locations.

5.2. Response time

The aim of these simulations was to demonstrate the Cloudlet + Clone efficiency in executing offloaded code.

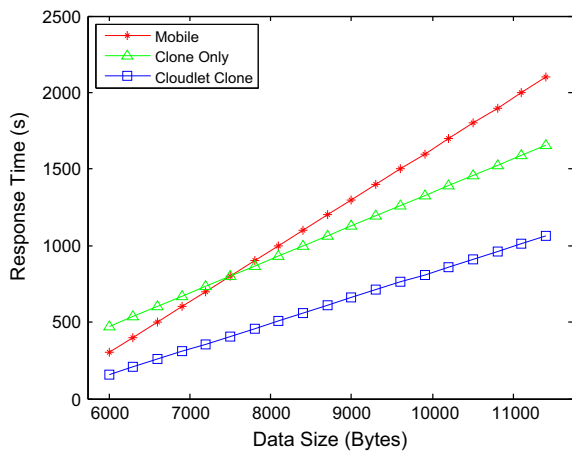


Fig. 4. Task response time as a function of input data size.

The number of instructions W required to execute a task for a given data size of L is $W = XL$, where X can vary depending on the type of application [42,43]. Fig. 4 depicts a scenario in which the input size increases by 300 bytes as the instructions increase by 50,000 MIPS. This is a typical scenario in which the instructions to be executed increase exponentially in all cases when the input data size is increased for one task. To evaluate the proposed architecture, we assumed that the number of instructions increases by 166 MIPS per byte. Cloudlet + Clone performs best in this case because it is equipped with a network with a larger bandwidth compared to clone-only conventional architectures. The response time increases exponentially with the input data size in all cases. Depending on the task deadline, this response time might not be good enough because it directly affects the QoE of mobile applications.

We also considered a case in which the number of instructions for execution can vary depending on the task, regardless of the size of the input data. Fig. 5 shows how

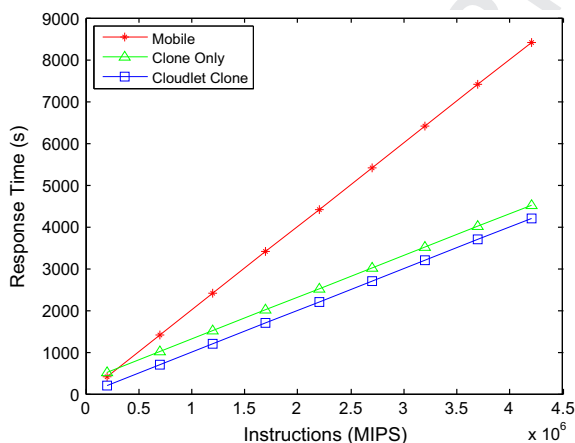


Fig. 5. Task response time as a function of the number of instructions.

the response time increases with the number of instructions. It is apparent that it is much quicker to offload the code to a remote server (clone or Cloudlet + Clone) because of the lack of computational resources in mobile devices. In particular, Cloudlet + Clone is even quicker when offloading because data transmission is much quicker than via traditional offloading frameworks owing to the smaller network latency and larger bandwidth. It is clear that the bottleneck for offloading computation is the transmission time. Two features of the proposed architecture overcome: (1) the high bandwidth of the offloading framework and (2) the use of data caching.

The proposed architecture supports local data caching on a clone and a cloudlet. This means that when offloading, a mobile device does not have to transfer input data with the code if the data can be downloaded via the Internet. The input data do not have to be sent with the code if they are already stored in the clone. In such cases, if the code is offloaded to a cloudlet, the cloudlet will retrieve these data from the user's dedicated clone using the fixed network. Current popular offloading frameworks do not support this functionality. As shown in Fig. 6, applications benefit from having an offloading framework with data caching enabled in comparison to traditional offloading frameworks (clone without data caching) and clones with data caching enabled. Fig. 7 compares the response time when offloading for traditional clones and for Cloudlet + Clone with data caching enabled.

Every offloading framework considers the state of its available network to decide whether it is worth offloading. The bandwidth of the network is a function of the network state. Hence, it is not appropriate to assume that the network state remains constant until a user finishes using an offloadable mobile application. Bandwidth reduction can negatively affect the performance of the offloading framework. By reason of the previous statement, it is important to analyze the effect in a situation in which the available network bandwidth decreases. The results are shown in Fig. 3 for a fixed reduction in bandwidth when offloading the same task with the same number of

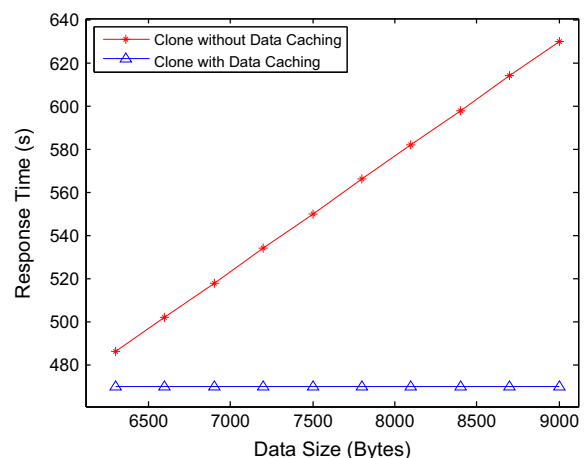


Fig. 6. Response time as a function of input data size with data caching (proposed architecture without Cloudlet) and without data caching in a clone (conventional architecture).

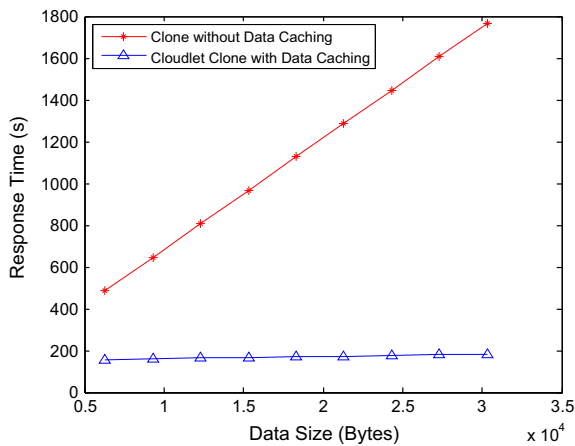


Fig. 7. Response time as a function of input data size without data caching on a clone (conventional architecture) and with data caching in Cloudlet + Clone (proposed architecture).

input data. It is evident that the reduction in response time when offloading is far greater owing to the small bandwidth of traditional offloading frameworks, whereas the proposed architecture supplies more bandwidth to the offloading framework by adding WiFi-based cloudlet support.

5.3. Energy consumption

We simulated the energy consumption of the CPU and the network interface of a mobile device. The aim was to

Table 2

Wireless energy model for downloading x bytes of data over 3G and WiFi networks.

	3G	WiFi
Transfer energy	$0.025(x) + 3.5$	$0.007(x) + 5.9$
Maintenance	0.02 J/s	0.05 J/s
Tail energy	0.62 J/s	NA
Tail-time	12.5 s	NA

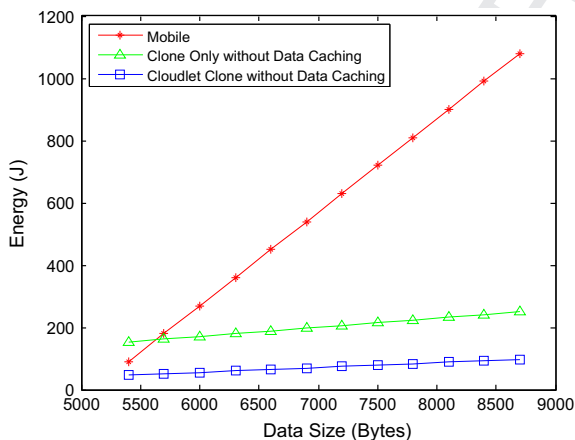


Fig. 8. Energy consumption of a mobile device with a clone only and Cloudlet + Clone when increasing the input data size without data caching.

give an indication of the energy consumption of the mobile device rather than to report precise measurements. When calculating the energy consumption of a mobile device, we consider the CPU energy consumption for local execution of a task, and the energy consumption of the network interface when offloading. For a CPU with approximately 500 MIPS, the energy consumption is 0.9 W [8]. The parameters listed in Table 2 were used to estimate the energy per byte [38].

Fig. 8 compares the energy consumption for a task executed on a mobile device only, a clone only, and Cloudlet + Clone as the input data size for the same task increases, which also increases the number of instructions to complete. The results show that the Cloudlet + Clone architecture saves a significant amount of energy owing to its high bandwidth and low-latency networks. Figs. 9 and 10 compare energy consumption with data caching enabled in a clone only and Cloudlet + Clone, respectively, with energy consumed when the task is executed on a mobile device and on a clone without data caching. In all cases,

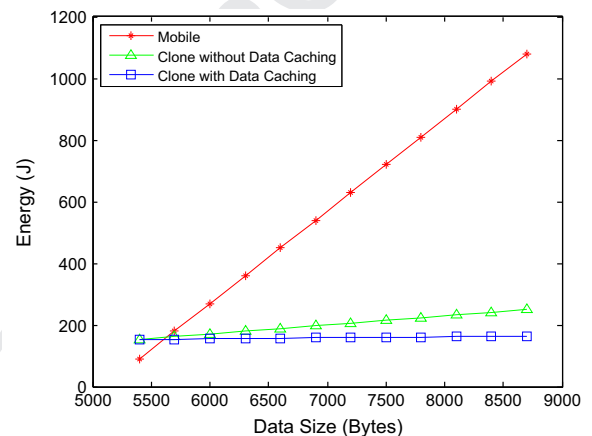


Fig. 9. Energy consumption of a mobile device when offloading without data caching in a clone (conventional architecture) and with data caching enabled in a clone (proposed architecture without Cloudlet).

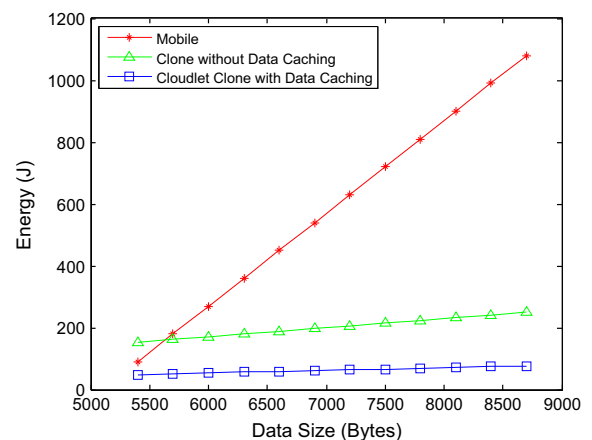


Fig. 10. Energy consumption of a mobile device when offloading without data caching in a clone (conventional architecture) and with data caching enabled in Cloudlet + Clone.

Cloudlet + Clone performs better than the comparative architecture because of the reduction in transmitted data.

6. Conclusions and future work

We proposed a new MCC system architecture called Cloudlet + Clone, containing a new middle layer called a cloudlet layer. This cloudlet layer sits between mobile devices and their traditional cloud infrastructure or clones. Cloudlets are deployed next to WiFi APs and serve as a localized service point in close proximity to mobile devices to improve the performance of mobile cloud services in terms of response time. An offloading algorithm for deciding *Whether* and *Where* to offload is applied on top of the new architecture. The decision-making takes into account the availability of two types of wireless network, WiFi and 3G, with the aim of saving battery life for mobile devices while satisfying the response time constraints of applications. We demonstrated the efficiency of the proposed architecture by comparison with conventional offloading architectures in simulations. Our immediate aim for future work is to carry out experiments on our OpenStack-based cloud test-bed.² Integration of our coarse-grained offloading algorithm with a VM placement algorithm such as the one proposed by Chang et al. [5] is planned as one of our next steps. Our energy model will be refined and the overhead for collecting energy consumption information will also be evaluated.

Acknowledgement

The work reported here was partially funded by the EU FP7 Projects MONICA (GA-2011-295222), CLIMBER (GA-2012-318939), and EVANS (GA-2010-269323).

References

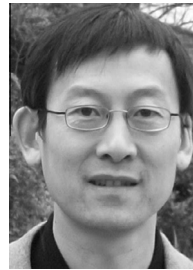
- [1] S. Qureshi, T. Ahmad, K. Rafique, S. ul Islam, Mobile cloud computing as future for mobile applications – implementation methods and challenging issues, in: 2011 IEEE International Conference Cloud Computing and Intelligence Systems (CCIS), 2011, pp. 467–471.
- [2] L. Yang, J. Cao, S. Tang, T. Li, A.T.S. Chan, A framework for partitioning and execution of data stream applications in mobile cloud computing, in: R. Chang (Ed.), IEEE CLOUD, IEEE, 2012, pp. 794–802.
- [3] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, Clonecloud: elastic execution between mobile device and cloud, in: Proceedings of the 6th Conference on Computer Systems, EuroSys '11, ACM, 2011, pp. 301–314.
- [4] 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: Proceedings of IEEE INFOCOM 2012, 2012, pp. 945–953.
- [5] D. Chang, G. Xu, L. Hu, K. Yang, A network-aware virtual machine placement algorithm in mobile cloud computing environment, in: 2013 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), 2013, pp. 117–122.
- [6] IEEE, Standard for local and metropolitan area networks part 16: air interface for fixed and mobile broadband wireless access systems amendment 2: physical and medium access control layers for combined fixed and mobile operation in licensed bands and corrigendum 1, IEEE Std 802.16e-2005 and IEEE Std 802.16-2004/Cor 1-2005 (Amendment and Corrigendum to IEEE Std 802.16-2004), IEEE, 2006.
- [7] J.D. Power and Associates, US Wireless Smartphone and Traditional Mobile Phone Satisfaction Studies, 2012.
- [8] K. Kumar, Y.-H. Lu, Cloud computing for mobile users: can offloading computation save energy?, Computer 43 (2010) 51–56.
- [9] H. Wu, Q. Wang, K. Wolter, Methods of cloud-path selection for offloading in mobile cloud computing systems, in: IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom), 2012, pp. 443–448.
- [10] Y. Wu, G. Min, A. Al-Dubai, A new analytical model for multi-hop cognitive radio networks, IEEE Trans. Wireless Commun. 11 (2012) 1643–1648.
- [11] R. Kemp, N. Palmer, T. Kielmann, H. Bal, Cuckoo: a computation offloading framework for smartphones, in: Mobile Computing, Applications, and Services, Springer, 2012, pp. 59–79.
- [12] G.F. Nan, Z.F. Mao, M.Q. Li, Y. Zhang, S. Gjessing, H.G. Wang, M. Guizani, Distributed resource allocation in cloud-based wireless multimedia social networks, IEEE Netw. Magaz. 28 (4) (2014) 74–80.
- [13] S. Kosta, C. Perta, J. Stefa, P. Hui, A. Mei, Clone2clone (c2c): Enable Peer-to-Peer Networking of Smartphones on the Cloud, Technical Report TR-SK032012AM, T-Labs, Deutsche Telekom, 2012.
- [14] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, Maui: making smartphones last longer with code offload, in: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, ACM, 2010, pp. 49–62.
- [15] K.K. Rachuri, C. Mascolo, M. Musolesi, P.J. Rentfrow, Sociableness: exploring the trade-offs of adaptive sampling and computation offloading for social sensing, in: Proceedings of the 17th Annual International Conference on Mobile Computing and Networking, ACM, 2011, pp. 73–84.
- [16] M.V. Barbera, S. Kosta, A. Mei, V.C. Perta, J. Stefa, CDroid: Towards a cloud-integrated mobile operating system, in: Proceedings of 2013 IEEE INFOCOM, 2013.
- [17] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for VM-based cloudlets in mobile computing, IEEE Pervasive Comput. 8 (4) (2009) 14–23.
- [18] R. Yu, Y. Zhang, S. Gjessing, W. Xia, K. Yang, Toward cloud-based vehicular networks with efficient resource management, IEEE Netw. 27 (2013) 48–55.
- [19] D.T. Hoang, D. Niyato, P. Wang, Optimal admission control policy for mobile cloud computing hotspot with cloudlet, in: 2012 IEEE Wireless Communications and Networking Conference (WCNC), 2012, pp. 3145–3149.
- [20] D. Niyato, P. Wang, E. Hossain, W. Saad, Z. Han, Game theoretic modeling of cooperation among service providers in mobile cloud computing environments, in: 2012 IEEE Wireless Communications and Networking Conference (WCNC), 2012, pp. 3128–3133.
- [21] A.-D. Nguyen, P. Senac, V. Ramiro, How mobility increases mobile cloud computing processing capacity, in: 1st International Symposium on Network Cloud Computing and Applications (NCCA), 2011, pp. 50–55.
- [22] T.-Y. Lin, T.-A. Lin, C.-H. Hsu, C.-T. King, Context-aware decision engine for mobile cloud offloading, in: 2013 IEEE Wireless Communications and Networking Conference Workshop (WCNCW), 2013, pp. 111–116.
- [23] D. Kovachev, T. Yu, R. Klamma, Adaptive computation offloading from mobile devices into the cloud, in: IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA), 2012, pp. 784–791.
- [24] H. Wu, D. Huang, S. Bouzeffrane, Making offloading decisions resistant to network unavailability for mobile cloud collaboration, in: 9th International Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013, pp. 168–177.
- [25] S. Ou, K. Yang, L. Hu, Cross: a combined routing and surrogate selection algorithm for pervasive service offloading in mobile ad hoc environments, in: Proceedings of the 2007 IEEE Global Telecommunications Conference, 2007, pp. 720–725.
- [26] R. Wolski, S. Gurun, C. Krintz, D. Nurmi, Using bandwidth data to make computation offloading decisions, in: IEEE 2008 International Symposium on Parallel and Distributed Processing, IPDPS 2008, 2008, pp. 1–8.
- [27] L. Feeney, M. Nilsson, Investigating the energy consumption of a wireless network interface in an ad hoc networking environment, in: Proceedings of INFOCOM 2001, the 20th Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, 2001, pp. 1548–1557.
- [28] Y. Wen, W. Zhang, H. Luo, Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud

² An Androidx86 VM for OpenStack was developed during the study; <https://sourceforge.net/projects/androidx86-openstack/>.

- clones, in: Proceedings of the 2012 IEEE INFOCOM Conference, 2012, pp. 2716–2720.
- [29] J.M. Rabaey, A. Chandrakasan, B. Nikolic, Digital Integrated Circuits, Prentice Hall, Englewood Cliffs, NJ, USA, 1996.
- [30] K. Sinha, M. Kulkarni, Techniques for fine-grained, multi-site computation offloading, in: 2011 IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2011, pp. 184–194.
- [31] H. Nicksic, GNU Wget, Available From the Master GNU Archive Site <<http://prep.ai.mit.edu>> and its Mirrors, 1998.
- [32] A. Kivity, Y. Kamay, D. Laor, U. Lublin, A. Liguori, KVM: the Linux virtual machine monitor, in: Proceedings of the Linux Symposium, vol. 1, 2007, pp. 225–230.
- [33] S. Abolfazli, Z. Sanaei, M. Alizadeh, A. Gani, F. Xia, An experimental analysis on cloud-based mobile augmentation in mobile cloud computing, IEEE Trans. Consumer Electron. 60 (2014) 146–154.
- [34] K. Kumar, J. Liu, Y.-H. Lu, B. Bhargava, A survey of computation offloading for mobile systems, Mob. Netw. Appl. 18 (1) (2013) 129–140.
- [35] D. Huang, P. Wang, D. Niyato, A dynamic offloading algorithm for mobile computing, IEEE Trans. Wireless Commun. 11 (6) (2012) 1991–1995.
- [36] 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: 2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010, pp. 105–114.
- [37] A. Rahmati, L. Zhong, Context-based network estimation for energy-efficient ubiquitous wireless connectivity, IEEE Trans. Mobile Comput. 10 (1) (2011) 54–66.
- [38] N. Balasubramanian, A. Balasubramanian, A. Venkataramani, Energy consumption in mobile phones: a measurement study and implications for network applications, in: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference, ACM, 2009, pp. 280–293.
- [39] S.C. Cripps, RF Power Amplifiers for Wireless Communications, second ed., Artech House, Inc., Norwood, MA, USA, 2006.
- [40] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. Pract. Exper. 41 (1) (2011) 23–50.
- [41] R. Buyya, R. Ranjan, R. Calheiros, Modeling and simulation of scalable cloud computing environments and the Cloudsim toolkit: challenges and opportunities, in: 2009 International Conference on High Performance Computing Simulation, HPCS '09, 2009, pp. 1–11.
- [42] W. Yuan, K. Nahrstedt, Energy-efficient soft real-time CPU scheduling for mobile multimedia systems, SIGOPS Oper. Syst. Rev. 37 (2003) 149–163.
- [43] W. Yuan, K. Nahrstedt, Energy-efficient CPU scheduling for multimedia applications, ACM Trans. Comput. Syst. 24 (2005) 292–331.



Chathura Magurawalage received his B.Sc. Hons. degree in Computer Science from the University Of Essex, UK. He joined the Network Convergence Laboratory at the University of Essex in 2012, after being awarded the University Of Essex scholarship to carry out research as a PhD student. His current research interests include Mobile Cloud Computing, Cloud computing and Computer Networks.



Kun Yang received his Ph.D. from the Department of Electronic & Electrical Engineering of University College London (UCL), UK. He is currently a full Professor and the Head of Network Convergence Laboratory (NCL) in the School of Computer Science and Electronic Engineering, University of Essex, UK. Before joining in University of Essex at 2003, he worked at UCL on several European Union research projects in the area of IP network management and service engineering. His current major research interests include wireless networks, heterogeneous wireless networks, fixed mobile convergence, future Internet technologies such as information centric networking, and cloud computing. He has published more than 150 journal and conference papers in the above areas. He serves on the editorial boards of both IEEE and non-IEEE journals. He is a Senior Member of IEEE, a Fellow of IET.



Liang Hu. Received his M.S. and Ph.D. degrees in Computer Science from Jilin University, in 1993 and 1999 respectively. Currently, he is a professor and doctoral supervisor at the College of Computer Science and Technology, Jilin University, China. His research areas are Network Security and Distributed Computing, including related theories, models and algorithms of PKI/IBE, IDS/IPS, and Parallel Computing. He has led a lot of research projects, and has published 100+ journal papers. He is a member of the China Computer Federation.



Jianming Zhang received his Ph.D. degree in 2010 from Hunan University, China. He received his M.S. and B.E. in 2001 and 1996, respectively, from the National University of Defense Technology and Zhejiang University, China. Currently, he is an associate professor in the School of Computer and Communication Engineering at Changsha University of Science and Technology, China. His main research interests lie in the areas of wireless multimedia sensor networks, pattern recognition and computer vision. He has served as an invited member of the technical program committee of several international conferences such as MobiQuitous 2013, mCloud 2013 and CloudID 2013. He is a member of ACM and CCF.