CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Integrating mobile and cloud resources management using the cloud personal assistant
Authors	O'Sullivan, Michael J.;Grigoras, Dan
Publication date	2015
Original Citation	O'SULLIVAN, M. J. & GRIGORAS, D. 2015. Integrating mobile and cloud resources management using the cloud personal assistant. Simulation Modelling Practice and Theory, 50, 20-41. doi:10.1016/j.simpat.2014.06.017
Type of publication	Article (peer-reviewed)
Link to publisher's version	http://www.sciencedirect.com/science/article/pii/S1569190X14001130 - 10.1016/j.simpat.2014.06.017
Rights	© 2015 Elsevier Inc. All rights reserved. NOTICE: this is the author's version of a work that was accepted for publication in Simulation Modelling Practice and Theory. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version was subsequently published in Simulation Modelling Practice and Theory [Volume 50, January 2015, Pages 20–41] http://dx.doi.org/10.1016/j.simpat.2014.06.017 - https://creativecommons.org/licenses/by-nc-nd/4.0/
Download date	2026-10-27 20:09:59
Item downloaded from	https://hdl.handle.net/10468/1955

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Integrating Mobile and Cloud Resources Management using the Cloud Personal Assistant

Michael J. O'Sullivan*, Dan Grigoras
Department of Computer Science, Western Gateway Building, University College Cork, College Road, Cork, Ireland

***Corresponding Author, Telephone +353-21-420-5983**
Email Addresses: *m.osullivan@cs.ucc.ie, grigoras@cs.ucc.ie*

Abstract: The mobile cloud computing model promises to address the resource limitations of mobile devices, but effectively implementing this model is difficult. Previous work on mobile cloud computing has required the user to have a continuous, high-quality connection to the cloud infrastructure. This is undesirable and possibly infeasible, as the energy required on the mobile device to maintain a connection, and transfer sizeable amounts of data is large; the bandwidth tends to be quite variable, and low on cellular networks. The cloud deployment itself needs to efficiently allocate scalable resources to the user as well. In this paper, we formulate the best practices for efficiently managing the resources required for the mobile cloud model, namely energy, bandwidth and cloud computing resources. These practices can be realized with our mobile cloud middleware project, featuring the Cloud Personal Assistant (CPA). We compare this with the other approaches in the area, to highlight the importance of minimising the usage of these resources, and therefore ensure successful adoption of the model by end-users. Based on results from experiments performed with mobile devices, we develop a no-overhead decision model for task and data offloading to the CPA of a user, which provides efficient management of mobile cloud resources.

Keywords: mobile; cloud; resources; management; model

1. Introduction

With the increasing numbers of smart mobile devices in use, they are rapidly becoming a portable, capable, and personalised computing platform for the end user in different modes of mobility. These devices feature Wi-Fi and cellular network interfaces, which open them up to the resources of the world-wide-web. Many end-users turn to them rather than a desktop or laptop computer for a large range of tasks.

As mobile devices become more capable with dedicated apps and mobile optimised websites, the demand for applications and functionality on these devices, in terms of user expectations, grows. This is still problematic, as despite hardware advances, resources such as CPU capacity, power, memory, and storage are still small compared to their desktop counterparts. In this context, mobile cloud computing has emerged as a viable solution: demanding applications and tasks can be offloaded from the mobile to the cloud for execution. Completed work or results of offloaded tasks can be returned to the mobile device when complete.

In this complex equation of communication between the mobile device and the cloud, the management and use of the limited resources available on the device plays a central role. Many mobile cloud solutions published in the research literature require continuous, high-quality connectivity to the cloud, and involve large amounts of data transfer. There are several resource considerations that come into play. For example, the continuous transfer of data between the mobile device and the cloud will come at a high energy cost from the device battery. This cost only grows if the device

is using the cellular network connection, due to variation in signal level, along with the variable network bandwidth, and resulting data-rate. It may not be feasible for a mobile device to exhaust such energy continuously for the duration of the cloud interaction. As another example, as the bandwidth of the network connection may vary over time, levels of performance from the resulting data-rate cannot be guaranteed. This is very significant in the areas of real-time applications where minimal latency is crucial.

As offloaded tasks and applications execute on cloud infrastructure, cloud resources such as storage are used to compliment the local resources of the mobile device. The management of resources allocated to the mobile cloud at the cloud infrastructure also has to be taken into consideration. As the mobile device is a personal computing device, how this translates to the resources of a public cloud is important; existing approaches have proposed offloading portions of applications to the cloud, or even entire mobile operating systems. If all mobile device users offload large and complex data to support such operations, then the management of energy and physical resources at the cloud must be a factor.

The objective of this paper is to devise and examine the best practices in resource management, in the area of mobile cloud computing, and to derive a model for data and task offloading to the cloud, while considering the limited available resources. We will look at some of the previous approaches and models to energy and bandwidth resource management in the research literature, in the context of mobile cloud. We will augment these models with mobile cloud considerations, and apply them to existing approaches that have been taken for mobile cloud implementations, to understand how these approaches utilise the resources being studied. From this, we will devise and highlight the best practice approaches that solutions in the mobile cloud domain should adopt for managing these resources. We will then contrast these approaches and the resulting implications from the models with our cloud middleware solution, Context Aware Mobile Cloud Services (CAMCS) [1], currently under development, to highlight how a disconnected approach from the cloud can be of great benefit to the conservation of resources on the mobile device, by minimising the usage of the scarcely available energy and bandwidth. This is enabled by the Cloud Personal Assistant (CPA) [2], a component of CAMCS, which works to complete user tasks in the cloud in a disconnected fashion. We will also discuss how our middleware solution can avoid a large allocation of resources to the mobile cloud at the cloud server side, by the use of cloud services in SOA fashion, rather than allocation of entire virtual machines. We will present the results of several experiments performed with mobile devices, to derive a decision model for data and task offloading to the CPA of a user of the CAMCS middleware. This model considers the available resources and the nature of the data to be offloaded, as part of the offload decision process, without requiring additional overhead from the mobile device. As a result of the adoption of these best practices and the derived model, our middleware can achieve the goal of efficient resource management in the mobile cloud.

To summarise, the contributions of this paper are:

- Examination of existing energy models for mobile devices applied to mobile cloud approaches.
- Examination of bandwidth utilisation for mobile cloud approaches.
- Examination of cloud infrastructure resource requirements for mobile cloud approaches.

- Devise best practices for managing and utilising the aforementioned resources.
- A no-overhead decision model for task and data offloading based on results from experiments performed on the mobile devices.
- Highlight how our cloud middleware meets the best practice requirements outlined.

The remainder of this paper is organised as follows; Section 2 looks at the related work. Section 3 introduces a model of our cloud middleware, CAMCS, along with the CPA component. Section 4 will analyse the energy resource management applied to existing solutions and our middleware. Section 5 will analyse bandwidth management applied to existing solutions and our middleware. Section 6 examines the cloud infrastructure resource considerations for the mobile cloud. Section 7 describes experiments performed to derive our offload decision model for the mobile device, and presents the results of these experiments. Section 8 outlines the derived offload decision model based on the results of the experiments. Section 9 presents a discussion on best practices along with analysis of our model, followed by our conclusions in Section 10.

2. Related Work

We are unaware of any other related work that has examined and analysed the modelling of resource management in the domain of mobile cloud computing related to energy, bandwidth, and cloud server requirements. We believe this to be the first work to study existing mobile cloud approaches in such a way. However, many existing mobile cloud solutions have focused on the conservation of various resources such as energy and bandwidth at the mobile device, one component of the mobile cloud.

Code offload solutions [3, 4] have often cited the energy required in running an application locally on the mobile device as a consideration for offload; they profile the estimated energy cost in running an application locally, and then calculate an energy cost for running an application by offloading it to cloud infrastructure. Only if the estimated energy cost for offloaded execution is less than the energy cost for local execution, will offload actually take place. The consideration itself on the energy cost of offloading is actually dependent on the estimated bandwidth currently available on the network connection. This may vary over time. Disconnections may also occur while the application code is offloaded. In this case, the entire offloaded execution and resulting resource usage was wasted, and the task must be re-executed locally. Very significantly, in the MAUI approach [3], in experiments that were carried out using the cellular 3G connections, the offload decision maker chose never to offload. It always opted for local execution rather than offload onto the cellular network. With CloneCloud [4], another code offload approach, the authors do not try to evaluate their mechanism on a cellular network; they only evaluate it on Wi-Fi. exCloud [5] is another code offload approach which considers finer grained code migration by focusing offload on the stacks stored in memory for an object-oriented program, associated with method calls; this is called stack-on-demand execution. For example, when a ClassNotFound exception is thrown, the code is offloaded to the cloud, which presumably features the missing library. Offload can also occur when an OutOfMemory exception is thrown.

A Quality of Service (QoS) and power-management aware framework by Ye et al [6] focuses on mobile cloud service migration based on power and QoS constraints. If it is too costly to use a remote service from the device, or the latency is too high such that QoS will not be met, the cloud service can be migrated to the device, and can be used locally at lower cost. The work proposes algorithms for migration, along with the decision process for local versus remote execution. The solution also supports chained services, as the user may use many services in combination, and supports services that may not be migrated, due to constraints such as being tied to a database. Wang and Deters [7] developed a web-service mash-up solution to deliver SOA based services to mobile devices from the cloud. As part of the mashup approach, they support the use of multiple services; for example, the output of one service can be the input to another. The COSMOS platform by Sankaranarayanan et al [8], supported by the proposed SMILE middleware as part of the platform, also focuses on the idea of using existing services together to deliver relevant information and functionality to mobile devices. The difficulty, but potential of the platform, is in allowing services to share data, rather than work in isolation, as most do now. They analyse the potential and outline the practical difficulties involved in their endeavour.

Alfredo [9], an application partitioning solution, operates by dividing a mobile application up into various components; these components are then distributed amongst local computing infrastructure, where each component is executed. The actual partitioning corresponds to an application graph, which is dynamically created based on what is required to be optimised, such as “optimise in such a way that will result in the lowest energy/bandwidth usage”, or “optimise for minimal execution time”. Of course, there is no guarantee here that optimising the energy usage will optimise the bandwidth or time; therefore it becomes something of a trade-off in optimising one resource over another.

Cloudlets [10] and their variations are another approach to mobile cloud that, like Alfredo, use offloading to local cloud infrastructure that may be located nearby, such as at the Wi-Fi access point of a coffee shop or an airport. The mobile device stores a virtual machine overlay, which is then sent to the computing infrastructure, combined with a full virtual machine, and then the personal applications of the user run on the infrastructure. Some execution is then offloaded to remote cloud infrastructure. The idea of this solution is to overcome the variable bandwidth and resulting latency that comes from offloading to a distant cloud data centre. This could prove very useful for real-time applications with minimal latency requirements. How a Cloudlet approach will scale for many users in terms of its own resources, how the Wi-Fi network performance will cope with multiple users, and how the Cloudlet will handle mobility, is yet to be seen. Other works have extended upon the Cloudlet concept, such as the work by Verbelen et al [11], which defines a Cloudlet as a collection of computing devices within a small area, such as laptops, and even other mobile devices. An OSGI-based implementation breaks up an application, similar to the Alfredo approach, and distributes the components to other devices within the Cloudlet for execution.

Remote display [12] is a similar approach to Cloudlets. Based on virtual machines that run in the cloud, the visual output (the GUI) of the virtual machine is sent back to the mobile device, and the user interacts with it using their device. This will suffer more than the Cloudlet approach from latency, as the virtual machine is not local. The full virtual machine runs in the cloud; there is no personal overlay like with Cloudlets. The virtual machine runs a desktop operating system, such as Windows.

The analysis of energy usage by the antenna in the mobile device is also covered to some extent in the literature. As outlined by Schulman et al [13], much of the energy cost associated with using the network interface of the mobile device actually comes from what is known as the “tail” energy of the network interface. Once the communication has actually completed, the network interface stays active for a few more seconds before sleeping, to receive any remaining packets from the server. This can be quite costly for a short communication. The cost of this grows substantially as different network operations take place. The work in the paper is in the implementation of a system called Bartendr, which groups network requests together, so as to reduce the number of tail energy occurrences. Balasubramanian et al [14] describes the high cost in “ramp” energy, to actually transition the network interface into a high power state from sleep or low power states, as also being very significant. Their TailEnder approach also tries to group together network requests so as to minimise the ramp and tail energy. In looking at the actual energy usage figures of the antenna, the work by Heinzelman et al [15] describes the energy used by the antenna of a micro-sensor in a wireless sensor network, as a product of the electrical power required for the antenna components, along with amplification power, the size of the data to transmit, and the distance from the base-station.

The modelling of data bandwidth in the mobile context does not seem to have been examined extensively. While there are published figures, which specify what the upper bandwidth of a given type of connection should be (such as EDGE, HSDPA, or Wireless 802.11b/g/n/ac), how this can vary on the mobile device is largely not investigated. Some approaches in the past have looked at the allocation of bandwidth in regards to the GSM networks handoff bandwidth [16] [17]. Xu et al [18] proposed a fair scheme for dynamic bandwidth allocation in WCDMA networks. Tocado et al [19] developed an Android based monitoring application that can collect information about cellular networks on the move, and they used this data to analyse the performance of cellular networks; for example, they take a train journey from the city to the countryside to evaluate network changes en-route. Work by Gomes et al [20] examines a splitting and merging approach to cellular base stations that are backed by emerging Radio-over-Fibre technology. By organising cells into multiple tiers, based on splitting and merging of the cells, they aim to meet a number of different optimisations, such as an increase in network capacity during busy periods, and to reduce it when demand is low. The optimisations of the multi-tier organisation can support other objectives such as energy saving and cost saving/revenue maximisation. Kivekäs et al [21] have looked at how bandwidth is influenced by the design of the mobile device handset itself, along with the positions of the hand of the user holding the device, and the position of the device relative to a model of a human head.

Several works exist in terms of allocating cloud resources to meet the demands and QoS requirements of resource requests from clients. Liang et al [22] proposes an economical cost model to deal with offloaded tasks. Their approach aims at identifying an optimal allocation of cloud resources to offloaded tasks, and rewarding those instances that provide resources with an income. Rahimi et al [23] examine using a tiered cloud based system to execute mobile applications as a Location-Time-Workflow (LTW). Their heuristic algorithm, MuSIC, can reach a 73% optimal allocation of resources for mobile applications to meet specified QoS requirements. Liang et al [24] used a Semi-Markov decision process to model mobile cloud domains, where requests for resources could be transferred to neighbouring cloud domains, with the aim of maximising rewards for the providers of the domains involved for completing requests, while also considering the cost of such request

transfers. Using resources from different clouds to satisfy requests has been explored in other works. Kaewpuang et al [25] developed a framework that allows different cloud providers to share their server resources to meet the resource requirements of their users, again, with a focus on maximising the revenue for each of the cloud providers that participates in sharing. They also focus on solving optimisation problems that indicate when to share, and the optimal number of resources (such as application servers) required for sharing to meet request demands. The SAMI model, by Sanaei et al [26], is a multi-tier infrastructure framework, which aims to meet latency, resource requirements, and security concerns, by utilising existing infrastructure resources from trusted mobile network operators (MNO) instead of the cloud. An MNO can also delegate to trusted third party infrastructure providers, and utilise cloud infrastructures, meaning it can also act as an arbitrator for cloud resource requests. In addition, similar to the approach we take with our CAMCS middleware in this paper, this framework utilises an SOA approach to using services in the cloud to meet service requests from users. The PhD thesis of A. Beloglazov [27] studies distributed dynamic virtual machine consolidation techniques and algorithms to meet energy efficiency requirements, while being able to meet user demand.

3. System Model

This section presents the model of our cloud-based middleware system under development, which is aware of the resource management considerations outlined in this paper.

3.1 Cloud Middleware

The purpose of our cloud middleware system is to complete tasks passed to it by the user, from a thin-client running on the mobile device. Each user of the middleware, known as Context Aware Mobile Cloud Services (CAMCS) [1], is allocated a Cloud Personal Assistant (CPA) [2], which carries out the tasks on behalf of the user. A given task is completed using existing cloud and web-based services. We do not instantiate cloud-based resources such as virtual machines for each user. However, the user's CPA can gain access to resources in the cloud owned by their user, such as storage. Services used to complete the tasks can range from information services (locality point of interest data, mapping data), to e-commerce services (reservations systems, online shopping, etc.). These services already exist on the web and in cloud deployments; they are used with SOA techniques, such as Simple Object Access Protocol (SOAP) or Representational State Transfer (REST) – see Fig. 1.

Each task has a name, type, and a randomly generated ID associated with it, so that the task can be uniquely identified. Also, the user can specify parameters that can be used to complete the task, such as in the case of a restaurant reservation, the location/name of the restaurant, time, date, and number of guests in the dining party. The middleware uses a discovery service to locate a restaurant booking service, and contacts that service with the parameters to make the reservation. A confirmation is returned and stored with the task data when complete. When the mobile device is available, the result is sent back to the thin client. The task itself is stored in a task history list, located with the CPA, so that the user can run the task again in the future. The middleware containing each users CPA can be deployed on virtual machines running in different clouds. Therefore, the CPA can move from the different deployments to one that is located nearest to the users location. In this way, the

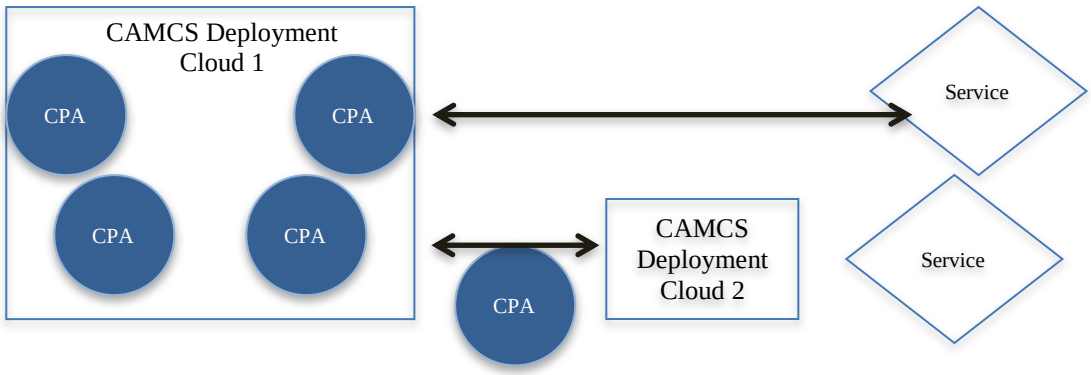

Fig. 1. The CAMCS Middleware. It is deployed on VM's in different clouds. Each deployment contains CPA's belonging to various users. They complete tasks for their users by using existing cloud and web-based services. The CPA's can move between different deployments.

middleware can support scalability for multiple users, and minimise the time required for the thin client to communicate with the user's CPA.

3.2 Thin Client

The thin client application runs on the mobile device of the user. It is responsible for communicating with the user's CPA in the cloud. The user can input their details of a task to be completed into the client, which will then forward them to their CPA. The client will also receive the result of a completed task, and the user can then use the client to view this result.

Sending a task to the CPA only requires the user to select the type of task they want to complete, and the other details associated with it (name, parameters). The client sends these to the CPA in a client-server fashion, and simply receives an acknowledgement from the CPA that the task details have been received. Therefore, the amount of data sent over the connection will be very small compared to the size of the data being transferred in other mobile cloud computing works.

The important detail here is that once the task has been passed to the CPA from the thin client on the mobile device, the CPA takes full responsibility for completing the task. The mobile device can disconnect from the CPA at this time. The result is stored with the CPA until the mobile device re-connects to a network and the CPA can once again communicate with it. Therefore, the client does not need a continuous connection with the CPA or cloud deployments. As task data is sent to the CPA in a client server fashion, with an acknowledgement, and the only other communication that will take place is the sending of the task result from the CPA back to the mobile device, the frequency of use of the network interface is twice per task – see Fig. 2.

4. Mobile Cloud Energy Management

We now turn our attention to examining the management of energy in the mobile cloud computing domain. The limited energy provided by the mobile device battery is used by the hardware on the mobile device, as well as the software. Some software and services running on the mobile device can drain more energy than others. This, combined with large displays and the network communication make it a key resource. We have described in our introduction how many mobile cloud solutions require a continuous connection to the cloud-based infrastructure. This continuous connectivity

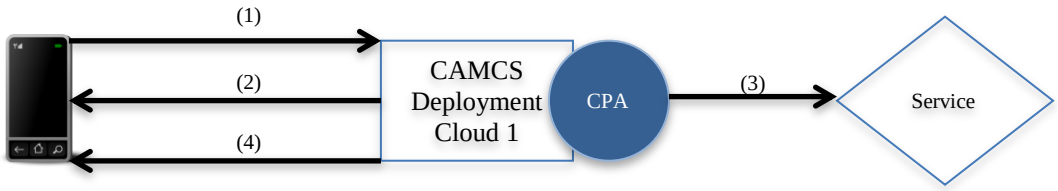

Fig. 2. CAMCS Task Execution Model. The thin client running on the mobile sends a task to the users CPA in a nearby CAMCS deployment (1), and receives an acknowledgement (2); the mobile device can then disconnect. CAMCS delegates the task to the user's CPA, which uses a cloud service to complete the task (3). Later, the result is returned to the mobile device (4). No continuous connection to the middleware is required.

is very undesirable. We first look at describing the energy usage of the antenna of a mobile device to send and receive data, before augmenting this usage with other aspects. We will then apply this model to the existing solutions, before examining it with our own.

4.1 Antenna Energy Modelling

Heinzelman et al [15] described the energy usage of the antenna of a micro-sensor in a wireless sensor network to send data, as follows:

$$E_{Tx}(k, d) = E_{Tx\text{-}elec}(k) + E_{Tx\text{-}amp}(k, d)$$
$$E_{Tx}(k, d) = E_{elec} * k + \varepsilon_{amp} * k * d^2 \qquad (1)$$

where $E_{Tx}(k, d)$ is the energy required to transmit a k-bit message a distance d to the base-station. E_{elec} = 50nJ/bit is the assumed energy dissipated by the antenna for the electrical circuitry, and ε_{amp} = 100 pJ/bit/m^2 is the assumed energy dissipated for the amplification of the antenna.

The energy usage to receive a message is given as follows:

$$E_{Rx}(k) = E_{Rx\text{–}elec}(k)$$
$$E_{Rx}(k) = E_{elec} * k \qquad (2)$$

where $E_{Rx}(k)$ is the energy required to receive a k-bit message. The assumed energy dissipation E_{elec} is the same as in (1).

In the Bartendr project, Schulman et al [13] estimates the energy cost for transmitting a stream of data. The power required by the transmitter to send this data will vary depending on the strength of the signal of the mobile device; the weaker the signal, the more power that would be required. This also corresponds to (1), even though it does not consider signal strength as a factor. This is understandable, as a sensor position may be fixed in the wireless sensor network, unlike a mobile device, which will constantly move. However, even though signal strength is not a factor, the amplification required will be greater if the signal is weaker.

Continuing to look at Bartendr, the system tries to schedule a data stream of size S to be sent over a given time period T. They divide up the data-stream into N frames of fixed size chunks, and the time period T is divided into slots. One slot is defined as a time period when one frame can be sent. The data rate and width of the slots will vary; the predicted signal strength and the observed median throughput over the time interval T are used to estimate the power consumption and width for each slot. With a given predicted $signal_\zeta$ in slot ζ, they estimate the communication energy as:

$$\frac{Signal_To_Power(signal\zeta) * \frac{S}{N}}{Signal_To_Throughput(\text{signal}\zeta)} \quad (3)$$

where *Signal_To_Power()* maps the given signal strength to a power value, and *Signal_To_Throughput()* maps the given signal strength to a throughput value. The mapping values are based on empirical measurements carried out by the authors, by examining and recording the power required for different transmitter states. They found that the power values were smaller for a stronger signal, and that the signal itself varied by location, as one would expect.

In the TailEnder project work by Balasubramanian et al [14], their experiments recorded the actual energy values, along with the times, for the ramp-up and tail energy, using both cellular 3G and GSM networks, as well as Wi-Fi. These were used to construct an energy model for each network, where the energy to transfer an *x*-byte file was given as:

$$R(x) + M + E \quad (4)$$

where $R(x)$ is the sum of the ramp energy, and the energy to transmit the *x*-byte file, M is the maintenance energy required to keep the transmitter on, and E is the tail energy. For the Wi-Fi connection, there is no ramp or tail energy as the interface is kept active, but there is an energy cost for scanning and association with an access point. Interestingly, the tail time for the 3G connection with their tests lasts 12.5 seconds. This value is actually set by the network provider. With such a high value, they observed that up to 60% of the total energy that went into a file transfer on the 3G connection, was from the tail energy alone, where no actual data-transfer was taking place. To contrast, the tail time from the GSM connection was only 6 seconds.

4.2 Mobile Cloud Considerations

Looking at these other works in the previous Section 4.1, we can see there are many factors in modeling the energy required for network communication on the mobile device, such as the size of the data, the distance from the base-station, the throughput, along with the ramp and tail energy characteristics of the network interfaces. In applying these parameters to the mobile cloud domain, we need to consider how the mobile cloud makes use of the network interfaces. Does it differ from traditional-client server approaches where one simply makes a request and receives a response?

The idea of the mobile cloud is to offload complex calculation and applications to the cloud. In the related work, we have mentioned several approaches to this in the literature. One such approach is code offload [3, 4]. Such approaches package up the code of some application, normally methods in object-oriented languages, along with the state of objects, which are then transferred to the server. Once the server has completed execution, the results are returned, in the form of the changed state of the objects; these need to be merged with the local copies of the objects on the mobile device.

The work by Kumar and Lu [28] has already proposed an energy saving model for the decision to offload some computation to the cloud from the mobile device. The energy saved on the mobile device by offloading the computation to the cloud is given by:

$$\frac{C}{M} \times \left(P_c - \frac{P_i}{F}\right) - P_{tr} \times \frac{D}{B} \quad (5)$$

where C is the number of instructions, M is the number of instructions the mobile device can execute per second, P_c is the power required to execute the computation at the mobile device, P_i is the power the mobile device uses while idle, F is the number of times faster the server is compared to the mobile device, P_{tr} is the power required to transmit the computation to the cloud, D is the number of bytes that make up the data, and B is the network bandwidth. Energy is saved when the formula produces a positive result.

This formula does not consider the technicalities of actually performing a code offload. We can say that if we have a method code-base size S_m, outgoing object size S_{oo}, and incoming returned object size S_{oi} then going by (1) and (2), we need to consider the total energy cost of offload E_{total} as:

$$E_{total} = n\ ((E_{elec} * (S_m + S_{oo}) + \varepsilon_{amp} * (S_m + S_{oo}) * d^2 + E_{prof}) + (E_{elec} + (S_{oo})) + E_{merg}) \quad (6)$$

where n is the number of times a method offload occurs, E_{prof} is the profiling energy used to determine if the offload should take place, and E_{merg} is the amount of energy used at the mobile device to merge the changed objects back into the virtual machine (such as adding new objects, updating existing objects, and removing old objects, along with the corresponding memory allocations/de-allocations). Equation (6) takes into account (1) and (2) for the offload and the return of the result to the mobile device. If we consider that n may occur 10 times in the course of one application execution, then the energy usage clearly increases ten-fold with such an approach.

Let us consider the Cloudlet approach. Here, there will be energy expended to transfer the VM overlay to the local cloud infrastructure. Eventually, any changes made to the application settings and data will ultimately need to be returned to the mobile device before the user leaves the venue where the Cloudlet is located. Looking at (1), we can estimate this energy as:

$$E_{total} = O + I + n\ ((E_{elec} * (\mathrm{k})) + m\ (E_{elec} * (\mathrm{k}) + \varepsilon_{amp} * (k) * d^2) \quad (7)$$

where O will be the initial energy to transfer the virtual machine overlay to the Cloudlet, and I will be the energy required to transfer the modified overlay back to the mobile device. Energies O and I could correspond to (1) and (2). n will factor up the energy consumption for the number of times the Cloudlet sends data to the mobile device. It is hard to predict this value. It could be extremely high, if the Cloudlet actually displays the visual output of the virtual machine on the mobile device; consider how often the display GUI would be refreshed, and this would need to be updated on the device display, possibly even if no change has taken place. m will factor up the number of times the mobile device sends data to the Cloudlet, which could be much smaller than n. The primary energy saving aspect of the Cloudlet approach is that d will be very small by contrast compared to sending data to a cellular base-station for offloading to remote cloud infrastructure.

Let us consider our disconnected middleware approach. If a user wishes to offload some task to their CPA, to send the task, there will only be one outward communication to the cloud, describing the task. This will be a simple message of k bytes, with no application code. There will be no immediate response data except for

a request acknowledgement. When the task is complete, the result will be pushed to the thin client on the mobile. This will be a message of *m*-bytes, which simply carries a result message, with no changed object states.

Therefore, the task offload energy is given by (1), and the response, given by (2), will consist only of a HTTP Status with no data. When the task has completed, the result will also be given by (2), $E_{Rx}(m)$. As the focus here is on disconnected operation, there is no continuous communication that needs to be factored into the formulas as in (6) and (7), along with no additional overhead (such as initial virtual machine offload).

5. Mobile Cloud Bandwidth Management

During operation, the mobile device will be connected to a multitude of different networks, be it a Wi-Fi network, or different base-stations in the cellular network. Bandwidth allocation, along the lines of guaranteeing QoS, is a topic that has been researched extensively. However, it is difficult to predict what the bandwidth will be on a given network.

At the cellular network level, the bandwidth will vary depending on the type of cellular connection used (GPRS, EDGE, 3G, HSPA). While each of these has a defined upper limit on the bandwidth specified, the actual bandwidth that the user will experience in practice will vary, and will depend on aspects such as signal strength, interference, and location.

At the wireless LAN level, depending on the different version of the 802.11 that a Wi-Fi network adheres to (b/g/n/ac), the user will experience a far more stable bandwidth with little variation from the nature of the network; what may cause trouble is the number of users accessing the Wi-Fi network from the same access point.

The question for mobile cloud is that given the variable nature of the bandwidth of these networks, how do we manage the available bandwidth in such a way that this will not impact the user experience with the mobile cloud? At first glance, since no given bandwidth can be guaranteed with the mobility factor of a mobile device, we have to design to use as little bandwidth as possible. This can be a troublesome aspect given some of the existing approaches to mobile cloud we have discussed, due to the required continuous connection, and large data transfer.

Additionally, from the perspective of a software developer actually developing mobile cloud applications, they will not be able to have any control over bandwidth allocation to their applications, as this work is typically carried out at the network layer [16 - 18]. The allocation is decided at the base station or access point.

5.1 Bandwidth Requirements

While we cannot say with certainty what bandwidth is required by many of the mobile cloud approaches we have reviewed, we can make an estimate on how much will be used based on how they use the network connection.

Approaches that require a continuous connection will undoubtedly require more bandwidth than what is used in an approach with disconnected fashion. If we look at the Cloudlet approach, where the network connection is repeatedly used to transfer the output of the virtual machine to the mobile client, then the expected bandwidth usage is going to be large. On a Wi-Fi network, which is the premise of use of Cloudlets, we can expect the bandwidth to initially be large, and relatively constant,

compared with the data connection; however this may change as the number of users of that Cloudlet grows. As the limited bandwidth of the Wi-Fi connection is used up, the user experience will suffer from the time delay induced as a result of the drop in the data rate. The same principle can be applied to cellular networks where remote display technologies may be used, except in this case that the bandwidth will be very small to start with, and shared by far more users of the base-station. As a result, the main concern with these approaches is that they have a requirement that a certain amount of bandwidth must be available for use when required, to ensure that they work as expected for the user, without any delays from sub-optimal bandwidth allocation.

Approaches such as code-offload and application partitioning may not be as dependent on the amount of bandwidth as Cloudlets or remote displays. A continuous connection is required to either the cloud infrastructure running the virtual machine, or to the other computing nodes in the network where the application partitions have been sent to, but data does not need to be repeatedly sent over the network connection. Once code or partitions have been offloaded from the mobile device, it need only receive the results back when the remote execution is complete. The actual data to be sent/received between the mobile and the infrastructure is the serialised application code and objects. These will likely be much smaller in size when compared with the graphical output of virtual machines. Also, in the way that the updated graphical output of a virtual machine needs to be sent to the mobile continuously (even if the display has not changed), the same does not apply for these approaches, because the mobile device only receives the results of a completed execution. As a result, unless a disconnection occurs, there is no negative user experience impact, as it is not as time sensitive as the virtual machines.

If we consider the disconnected operation aspect of our CAMCS middleware, then the bandwidth is something of little concern. Users offload a task description to their CPA within the CAMCS middleware. The description is just made up of a limited number of ASCII characters that describe the task, the type of task, and any parameters. We speculate that the data sent will be even smaller than the data sent on code-offload or application partitioning approaches, so our middleware brings the same advantages as far as bandwidth is concerned as these approaches, low data usage, no time sensitive delay, and no continuous use of the data connection.

There are other features of our middleware that will further reduce the requirement for data transfer, and hence bandwidth utilisation. A history of completed tasks and related task data for each user is stored with their CPA. If the user wants to run a variation of a past task, or re-run a task entirely, such as order office supplies from an e-commerce website, then the user does not need to send this task information to their CPA, all over again; they can simply signal an old task to run again with a small HTTP request signal using the task ID, and send any updated parameters if required, therefore reducing the request size and the bandwidth utilisation even further. Also, the CPA is able to undertake work under its own initiative for the user. For example, if the user has, in the past, requested traffic information for a work route on weekdays, then, without the user having to send this task to their CPA over the network connection, the CPA will fetch this task data independently, and send it back to the mobile. This is similar to the Google Now service, except the initiative to fetch this information is not taken from the mobile device (and hence, requiring a request to be made), but by the CPA; therefore we only have a one way-communication, “response”, to the mobile.

There are scenarios where we expect the size of the data sent to the CPA can be very large. For example, if the user sends large files such as images or datasets to the cloud for some processing task, or possibly for storage, then we can expect the size of the task data to be sent to the CPA to be very large. Next, we discuss how to effectively manage the available bandwidth in such a situation.

5.2 Bandwidth Management

As part of a positive user experience as far as bandwidth is concerned (i.e. minimise use, not require large amounts), not only must we design to not use large amounts of it, but what is available must be used intelligently.

Where a high amount of bandwidth is required, it can possibly be reserved in advance of needing it, if it can be reasonably anticipated. Cloudlets or remote display approaches could reserve and allocate an amount of bandwidth as needed to cope with their expected demand. In this way, it could ensure bandwidth is available so that they function optimally with no time delay. Of course, this may not be fair to other users of the Wi-Fi access point, and can reduce the amount of bandwidth available to them. A financial cost may also be incurred for this reserving of bandwidth in advance. However, reserving bandwidth will still fail to deal with unexpected bursts in required capacity.

Code-offload and application partitioning approaches may not need to reserve bandwidth in advance. They can simply make the best use of bandwidth already available, which can speed up the offload of the code, and return of any changed object state. It may be argued that if these approaches did reserve bandwidth, it may still reduce the wait time of the user, but it may not reduce the time enough to justify the high cost of reservation.

We have already discussed how the mobile uses little bandwidth as a result of the small request size made to the CPA in the cloud, and because of its disconnected operation, it is not time sensitive. However, despite the fact that little bandwidth is used, bandwidth that is available must be used intelligently. Furthermore, there are cases when the size of the data sent to the CPA from the mobile are very large, such as when sending large files to be stored as described earlier. In this case, if bandwidth is low, then there is certainly going to be a delay for the user. If using the cellular network rather than the Wi-Fi, it will take even longer. The additional cost will be incurred to the user if for example, the low bandwidth results in not only a slow upload, but if data needs to be re-transmitted across the network if there are errors.

The thin client application running on the mobile device, must adapt to the current situation. If, for example, the available bandwidth is low, transfer of a file must be deferred to a later point in time, when the available bandwidth increases (the same principle applies to other resources such as energy, if the battery is low, upload should be deferred). As far as we are aware, this has been implemented in many apps today, such as Dropbox, where files will only be uploaded on a Wi-Fi connection if the user chooses. However, if there is a need to upload on a cellular network, this should be possible, but only when the bandwidth is suitable. The same principle of deferring upload can be used on a busy Wi-Fi network, or when the Wi-Fi signal is poor. This will benefit other users of the network, as bandwidth is not used up with a large task upload that may fail if the signal is very poor and ultimately drops.

We can also defer the offload of tasks that may not be urgent. The user can specify how urgent a task is, and even give a time when it should be completed by. Urgent tasks, or tasks that have a specified completion time coming up, can be offloaded

immediately, even on a network with poor bandwidth, if this is the case. Tasks that are not urgent, or tasks with a specified completion time at some point later in the day, can be deferred until the bandwidth is available.

In scenarios where the battery energy is low, it may consume energy to actively test the bandwidth available on the connection. Even though data transfer is small, the network interface must be activated. In such cases, it is best to apply a policy based on the connection (Wi-Fi or cellular), and the current measured signal strength, which can normally be determined from an API on the mobile operating system. This is explored and developed further in Sections 7 and 8.

6. Cloud Resource Management

This section will look at how resources on the cloud can be managed while provisioning services to mobile clients for the various mobile cloud approaches. At the cloud, one will find resources at the different "Service Layers" (IaaS and PaaS) that can be provisioned as resources to clients, such as virtual machines providing compute capacity, storage in the form of file systems and databases, memory, and networking capacity. Developers can also find resources for developing and deploying tools, such as application runtime environments, message queues, replication and backup facilities, and load balancers for scalability.

6.1 Cloud Resource Cost Model

The services offered by cloud deployments are normally offered through the interface of a virtual machine (VM); that is, for each resource, such as a hardware resource (e.g. CPU), or a developer resource, (e.g. a database), the user or application interacts with them by using a VM. CPU time is allocated to each VM for computation tasks, and database systems run on these VM's. To run each VM, a hypervisor is required to sit on top of the hardware. Therefore from an energy and financial standpoint, the number of VM's required for the mobile cloud requirements, and how much of the physical resources are required, will be of concern.

It is difficult to precisely approximate VM hardware costs, as described in [27]. Therefore, for each VM running on a server in the cloud, we can estimate a simple cost model for our purposes, as the summation of the costs required to run the physical resources required on the server. There will be a CPU cost, C_{cpu}, network input and output costs, C_{input} and C_{output}, a storage cost, C_{stor}, and a memory cost, C_{mem}. All these costs can be viewed as both energy costs and financial costs, as ultimately the financial cost will be the energy required to power these physical resources. If n is the total number of VM's, possibly different, running on a server, the total server cost, C_{server_total} for a cloud server is:

$$C_{server_total} = \sum_{1}^{n} \; (C_{cpu} + C_{input} + C_{output} + C_{stor} + C_{mem}) \qquad (8)$$

For m, the number of servers required on the cloud for scalability, we can factor up equation (8) by m. As a result, to keep the energy and financial cost as low as possible, we need to minimise as much of the hardware costs C as possible, and of course, n and m.

In the next section, we will examine how each of the discussed approaches to the mobile cloud makes use of the server resources, and estimate a cost based on (8).

6.2 Mobile Cloud Usage

Several of the previous approaches take different paths in terms of requirements at the cloud side, depending on what role the cloud will play.

With VM-based approaches such as the Cloudlets and Remote Display technologies, virtual machines capable of running end-user operating systems with GUI's such as Microsoft Windows or Linux distributions such as Ubuntu or Red-Hat will be required. They will also require resources such as imaging and storage, to store a user's VM image and for storing the image itself and any files. Cloudlets, as described in Section 2, only have a base VM stored on the infrastructure. The overlay image, containing the user's applications, data, and settings, is stored on the mobile device of the user. It is combined with the base VM to form the full VM at runtime. As a result, it will not need as much storage at the cloud deployment. However, the overhead of having to transfer the overlay images between the mobile and the server may be undesirable. Furthermore, breaking up and combining the base and overlay VMs may need special modification of the images and operating system software. There is also the issue of having to determine what should be offloaded to the remote cloud infrastructure, and what should be kept local on the Cloudlet. The energy requirements for this decision making process, along with merging the VM's, are unclear.

From this, we can determine that for Cloudlets, we will actually have two "server" costs, the costs incurred by the local Cloudlet, and the costs incurred at the remote cloud server. Let us categorise the costs from equation (8) by indicating if the resources are local and remote. The cost will be:

$$C_{total} = \sum_{1}^{n} (C_{r_cpu} + C_{r_stor} + C_{r_input} + C_{r_output} + C_{r_mem}) + (C_{l_cpu} + C_{l_input} + C_{l_output} + C_{l_stor} + C_{l_mem}) \quad (9)$$

In equation (9), an r in a cost C indicates the resource is remote on the cloud infrastructure, and l indicates the resource is local on the Cloudlet. In equation (9), the remote storage cost will only be required for the VM image that can run whatever software is required to carry out the work offloaded by the Cloudlet; what form this takes is unclear. The local storage on the Cloudlet C_{l_stor} has to store the base virtual machines. As described earlier, applications and settings are stored on the overlay VM on the mobile device of the user. The local input and output costs C_{l_input} and C_{l_output} need to accommodate the energy required not only for transferring the visual VM GUI state to the mobile device and received input from the user, but also the costs to transfer the VM overlay to and from the mobile device. C_{l_stor} will account for any temporary storage of user data for the VM operation; no user data required for VM operation (except for temporarily offloaded jobs) will be stored at the remote server, so there is no contribution to C_{r_stor} for this. C_{l_cpu} will also have the overhead of the offload decision process.

If we consider remote viewing approaches, the cost model will be modelled as equation (8). For remote viewing, we only have a remote server cost. Each of n users will require their own virtual machine. The VM for each user and all user data will need to be stored on the server, so C_{stor} will be very high. Again, we can expect C_{input} and C_{output} to be large as to accommodate the energy required to transfer the visual VM GUI state to the mobile device and receive input from the user.

Code offload and application partitioning approaches may be more desirable. These need only have specific software running on the cloud resources to support

their operations. However, it is generally not described what resources or specific software is required at the cloud. For example, for CloneCloud [4], we know that a copy of the mobile device operating system must be available and running on the server. MAUI [3] has an MAUI runtime required at the server, which is presumably not as resource-hungry as an entire mobile operating system. How these are deployed (standalone executable, application containers, operating system services) is unknown. MAUI was implemented in C#, so it stands to reason that the .NET CLR was used. For Alfredo [9], as an application partitioning solution, each local computing node must have OSGI available, as this is the framework that is used to split up and distribute the application components.

For application partitioning approaches such as Alfredo, we can model the resource usage as equation (8), except that n will be the number of remote computing nodes used for offloading the application partitions. Each node may or may not run virtual machines with the supporting OSGI platform (a desktop machine node could, but a mobile device node is unlikely to). C_{cpu} at a node will be smaller than in VM approaches because each node only runs a smaller part of a whole application on the platform, and if no VMs are deployed on the nodes, then each node only runs one operating system. C_{input} and C_{output} will correspond to the transfer cost of serialised application code and data. C_{stor} will not have any additional contribution from the mobile cloud approach. C_{mem} will be contributed to by the OSGI runtime.

Code offload approaches are similar to the model for application partitioning. n will be the number of cloud servers required as scaling takes place for the VM running the software or complete mobile operating system. C_{cpu} will be small, as the server only has to execute offloaded code, along with the required software for the offload process; there is little contribution to this cost from the mobile cloud. C_{input} and C_{output} will again correspond to the transfer cost of serialised application code and data. C_{stor} is tricky to estimate; if like MAUI, the server runs a software application to execute offloaded code, it would be small, but if the CloneCloud approach is used, this will be greater, because the entire mobile operating system is cloned into the cloud server; how many OSs are required for each user and their software is unclear. C_{mem} will depend on the amount of code and data offloaded and will vary with each offload.

With each of the discussed approaches above, disconnection is a concern. If the mobile device becomes disconnected, then the exhausted energy at the cloud in completing work that has been offloaded will be wasted.

Similar to the code offload and application partitioning approaches, the CAMCS middleware only requires a runtime application container running in a virtual machine, along with NoSQL database storage for storing user and task details. The VM can be scaled as required to cope with demand, and indeed we expect it to be replicated at cloud deployments globally, for user proximity benefits. We do not need to provision virtual machines and the corresponding required hardware for each user of the middleware.

As a result of the architecture of our middleware, going by equation (8) again, we expect that n can be as low as one but will get larger as scaling occurs. One area where C_{cpu} may grow is in the case that the middleware makes a decision process to carry out work on behalf of the user, without the user requesting it (which will also result in C_{input} being zero). C_{input} and C_{output} will be very small; as we have discussed, the nature of the character data will likely be even smaller than serialised application code. C_{stor} will be mainly attributed to the database storage. There is no significant storage required by the middleware itself. C_{mem} will not incur any significant

overhead from the middleware. Importantly, one cloud deployment can serve several users and their CPA's.

Aside from making independent decisions for the user, C_{cpu} will not result in any significant overhead from the middleware, because the work of completing a task will be distributed to existing SOA services located elsewhere in the cloud and the web. This is an important consideration; various cloud and web services exist already, and our approach relies on these to complete work for the user. For example, Amazon provides the S3 storage service. Therefore, our approach does not require VM resources to allocate storage for a user. As another example, if a user wishes for their CPA to find restaurants based on a meeting scheduled in a calendar and make a booking for them, we can utilise services such as Google Calendar, TripAdvisor, and online booking systems such as booktable.com (of course, they must expose a public API). These providers offer their services with their own server infrastructure, so we do not have to consider VM resources allocated specifically for this computation. The CPA relies on the nature of distributed SOA services, and does not operate under the same constraints of a VM that has to provide dedicated resources to carry out a piece of work. The CPA simply contacts services running on other servers, and stores result data, along with implementing some coordination logic. A CPA does not perform demanding computation itself that operates subject to available VM resources and constraints. In this sense, the CAMCS middleware simply acts as a trusted third party mediator in the cloud, to bring dispersed cloud services together, to offer the user our concept of a mobile device as an intelligent, portable, lightweight-computing terminal.

7. Thin Client Modelling Experiments for the Mobile Cloud

In order to realise a positive user experience of the mobile cloud for the user, a model must be developed which considers the resources outlined in this paper. A mobile user will primarily be concerned with the energy consumed on the mobile device, and time taken for mobile cloud communication. In light of this, we performed several experiments to evaluate task offloading over a cellular 3G network, to our CAMCS middleware in the cloud.

Our experimental devices were a Samsung Galaxy S3, and a Google Nexus 5, running on the Vodafone Ireland network. Our thin client software is installed on each device. The Galaxy is our primary development device; the Nexus was introduced at a later stage, and the reasons for this will be explained shortly. Power saving mode was disabled on the Galaxy, the Nexus has no such mode. The research questions we aimed to answer with these experiments were the following:

1. What effect does the varying quality of the cellular signal (signal strength) have on task offloading?
2. What effect does the distance between the mobile device and the mobile cloud deployment have on task offloading?
3. What effect does the varying quality of the cellular signal (signal strength) have on the power draw while task offloading is in progress?

The results from the experiments performed to answer these questions, drove the creation of our offloading decision model for the thin client on the mobile device.

7.1 Signal Strength Experiments

Evaluating the effect of the signal strength on the performance of the task offload has been performed in various ways in related work. We explained in Section 2 that code offload approaches normally evaluate the state of the network before the offload decision is made. The result of this evaluation is given to an optimisation solver. The solver then makes the decision to offload the code to the cloud, if an objective function can be met, such as saving energy. This network evaluation/profiling introduces overhead at the mobile device. Our aim is to eliminate this overhead to benefit the user experience. Nevertheless, we need to know something about the network state for the offload.

Our Galaxy S3 mobile device runs the Android OS operating system, as does the Google Nexus 5. Android OS provides some information through its PhoneStateListener API, regarding the state of the cellular network. According to the documentation, this includes signal strength (as an RSSI value or measured in Decibel-milliwatts [dBm]). By registering with the Listener, an application can receive updates whenever the strength of the signal changes. Our thin client was adapted to listen for these updates. With this approach, we do not have to introduce additional overhead to profile the network state, as this information is already collected by the Android OS. Our first experiment was therefore to answer research question 1, by evaluating offload performance with varying signal strengths.

In our experiments, the GSM signal strength was divided into ranges of dBm values: [-110, -100], [-99, -90], [-89, -80], and [-79, -70] dBm. The range between -110 and -100 is the poorest signal range, while in our experiments, the range between -79 and -70 dBm is the best signal range. The signal range can go higher than this, though on our devices, it was difficult to get a steady signal in the [-69, -60] range and above. The reason for using the GSM signal strength, rather than the CDMA signal strength, considering the use of the 3G network, is discussed in Section 8.1 In one of our previous works [29] we looked at other application models that can be facilitated by the CPA and CAMCS. We modified one of these, a file synchronisation application, to evaluate the timing of the offload of image files of varying size to the CPA; the CPA then forwards these files to cloud storage providers such as Dropbox and Facebook. However, for these experiments, we disabled this forwarding; as soon as the file was offloaded to the CPA as part of the file synchronisation task, the server immediately responded without doing anything. Files are converted into a String format with Base-64 encoding for offload. This encoding adds overhead, therefore the images were sized beforehand such that when the overhead was taken into account and added to the offload data size, the size of the five image files used was 2MB, 1MB, 500KB, 250KB, and 125KB. For each of these files, 25 offloads were performed over the cellular 3G network, to a cloud server, for each of the 4 signal ranges, totalling 100 offloads for each of the five files. This experiment was repeated twice, once for a cloud server located at the Virginia, USA site of Amazon EC2, and one for a cloud server located within our University. The aim of this was to answer research question 2. The Amazon EC2 server is a t1.micro instance, featuring 613MB RAM, and 1 vCPU with 2 EC2 Compute Units. Our University server features a 1.7Ghz CPU and 2GB RAM. Offload experiments were performed between the hours of 10am and 5pm on weekdays.

The timing of the offload was captured using logging placed into the Android thin client code. This data was imported into IBM SPSS Statistics version 21, and we

generated boxplots to study how the offload time varied for each of the file sizes, under the varying ranges of signals.

For the following results until stated otherwise, the Samsung Galaxy S3 device was used for the experiments. Fig. 3 shows the boxplots for the offload time (in seconds) of the 2MB image to the CAMCS middleware deployed on the Amazon EC2 server (left boxplot) and the University College Cork server (right boxplot), against the four GSM signal ranges. We note from this that the median time for the [-99, -90], [-89, -80], and [-79, -70] dBm ranges are very similar, just under or at 10 seconds. The only notable difference is for the [-110, -100] range, where the median time for the Amazon server is just over 30 seconds, and just under 30 seconds for the UCC server.

This kind of result was also repeatedly observed for the offload times of each of the other file sizes (Fig. 4-7). When the GSM signal strength is in the dBm ranges [-99, -90], [-89, -80], and [-79, 70], the median offload times are almost the same, but the median offload time for the [-110, -100] range is always higher. Of note, is that for the 500KB, 250KB, and 125KB files, there is barely any difference in the median offload times at all for the three strongest signal ranges. At this point, the offload time would appear to be dominated by network overhead, rather than payload transfer.

Fig. 8 shows the same boxplot for the 500KB file offloaded to the UCC server, where the experiment was performed on the Google Nexus 5 device, instead of the Samsung Galaxy S3; this was to see if the same timing trend held true for our other device. We can see that the trend does hold. The median offload time was very close for the [-99, -90], [-89, -89], and [-79, -79] dBm signal ranges, 2-4 seconds, where the signal strength for the [-110, -100] dBm range was about 9 seconds.

From this, to answer to research question 1, we would conclude that if the GSM signal strength is greater than -100dBm, there is no significant difference in the offload times as the signal strength varies.

7.2 Mobile Cloud Distance Experiments

To answer research question 2, we further analysed the data collected to answer research question 1. Fig. 9 shows a boxplot, comparing the offload time of a 2MB file to the Amazon EC2 middleware deployment, and the University College Cork middleware deployment in the [-99, -90] dBm signal range. The mobile device used

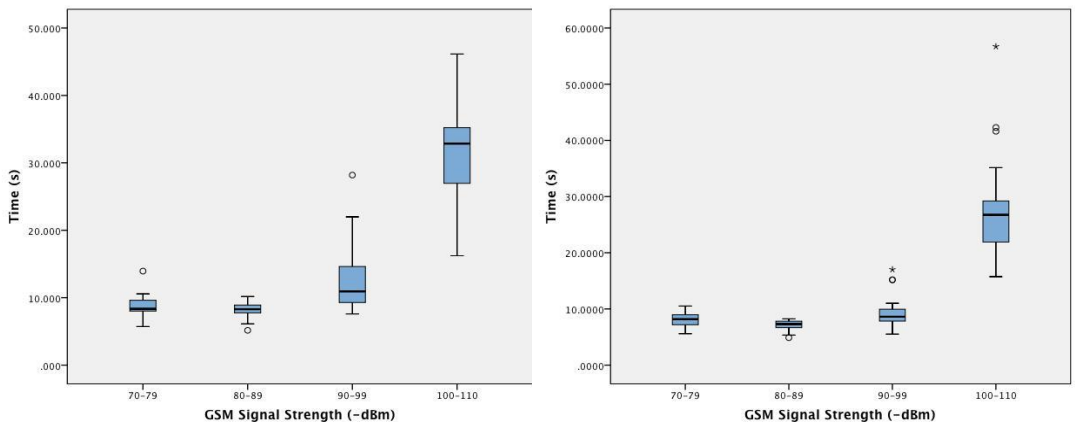

Fig. 3. 2MB Image Offload Boxplots. Plot of offload time (in seconds) against GSM signal strength (in -dBm) for offloads to an Amazon EC2 VM (left) and a UCC cloud VM (right) over the cellular 3G network. For Fig. 3-8, 25 offloads were performed for each range of signal strength, totalling 100 offloads of the file. For Fig. 3-7, the Samsung Galaxy S3 device was used.

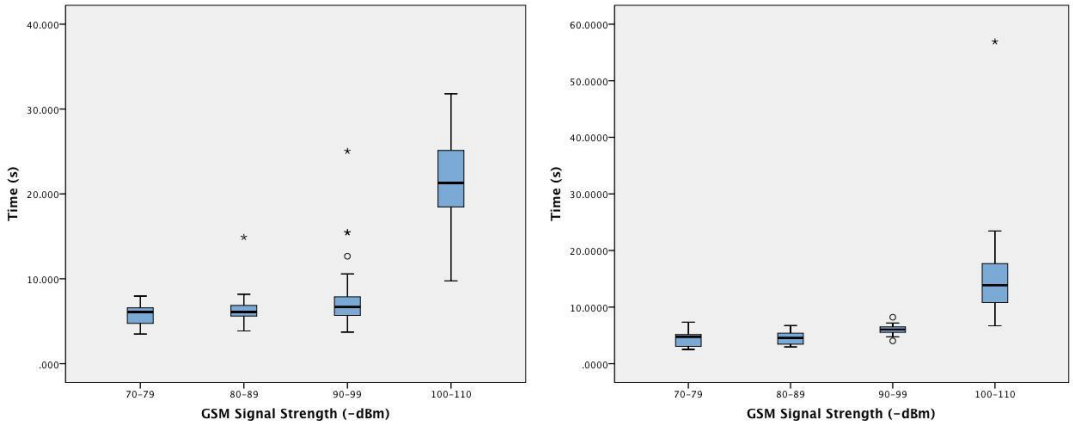

Fig. 4. 1MB Image Offload Boxplots. Plot of offload time (in seconds) against GSM signal strength (in -dBm) for offloads to an Amazon EC2 VM (left) and a UCC cloud VM (right) over the cellular 3G network.

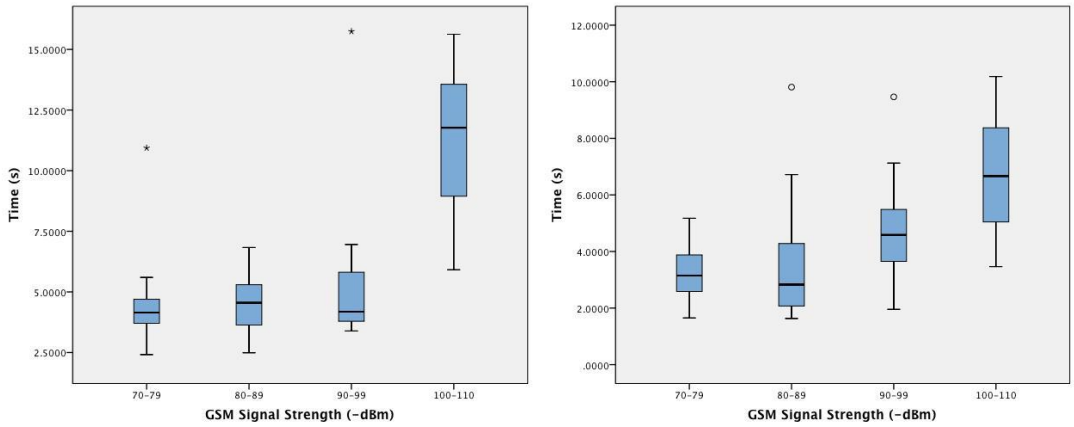

Fig. 5. 500KB Image Offload Boxplots. Plot of offload time (in seconds) against GSM signal strength (in -dBm) for offloads to an Amazon EC2 VM (left) and a UCC cloud VM (right) over the cellular 3G network.

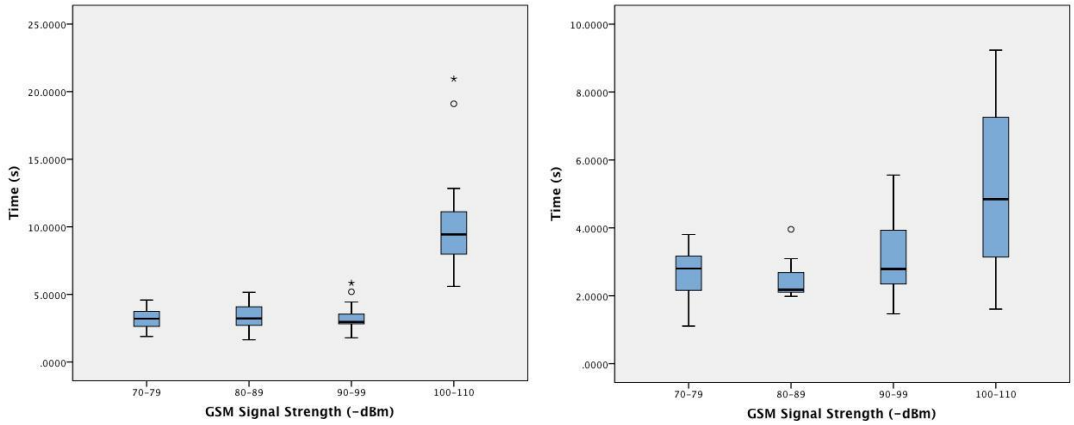

Fig. 6. 250KB Image Offload Boxplots. Plot of offload time (in seconds) against GSM signal strength (in -dBm) for offloads to an Amazon EC2 VM (left) and a UCC cloud VM (right) over the cellular 3G network.

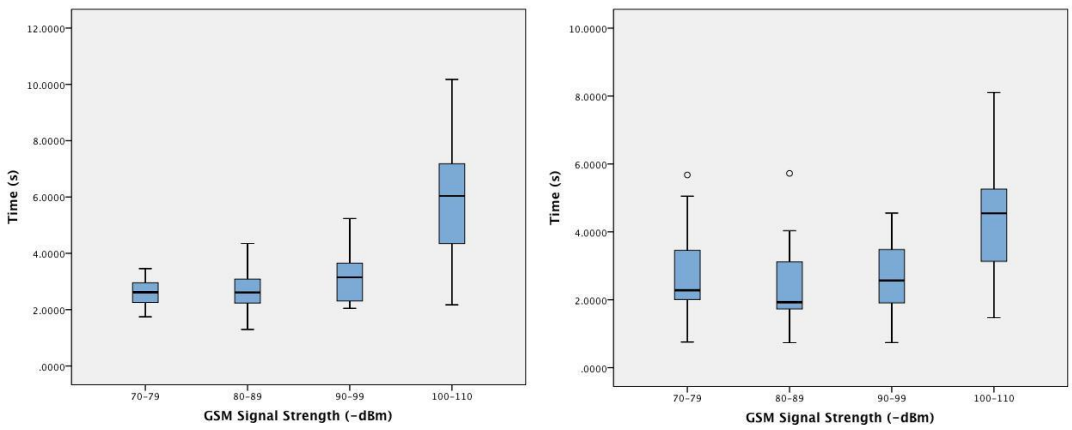

Fig. 7. 125KB Image Offload Boxplots. Plot of offload time (in seconds) against GSM signal strength (in -dBm) for offloads to an Amazon EC2 VM (left) and a UCC cloud VM (right) over the cellular 3G network.

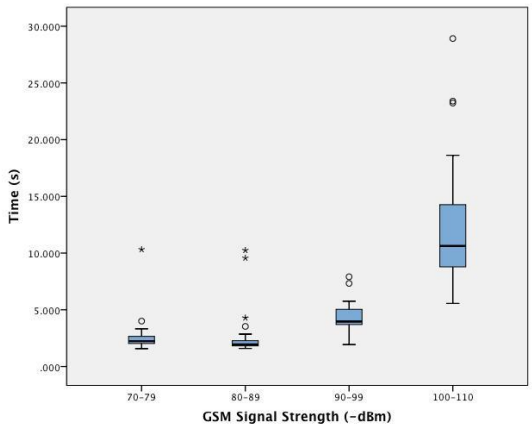

Fig. 8. 500KB Image Nexus 5 Offload Boxplots. Plot of offload time (in seconds) against GSM signal strength (in -dBm) for offloads to a UCC cloud VM over the cellular 3G network, with the Google Nexus 5 device.

was the Samsung Galaxy S3, and it was located in Cork city when performing the offloads. The Amazon EC2 virtual machine is running in the Virginia, USA Amazon datacentre.

Looking at Fig. 9, we see that there is little difference between the median offload times for each server. The standard deviation is higher for the Amazon server; this may be expected because it is further away than the UCC server and naturally because of varying bandwidth and usage over the traversed networks. This experiment was repeated for all the file sizes, and there was little difference for all, so the boxplots for the other files are omitted. We do however include one other boxplot, Fig. 10, which shows the same experiment for the 250KB file. In this case, the results for the closer UCC server show higher standard deviation and larger interquartile range than the Amazon server results, although the Amazon server data features outliers not found in the UCC server data. From this data, we would conclude that when modelling the offload performance, there is no need to consider the server distance, from a time perspective. Of course, this applies to our CAMCS middleware and thin client application. Consider a real-time application, such as in experiments performed in the work by Clinch et al [30], where latency caused by server distance did have an impact on results on the user experience while playing games run on Cloudlets.

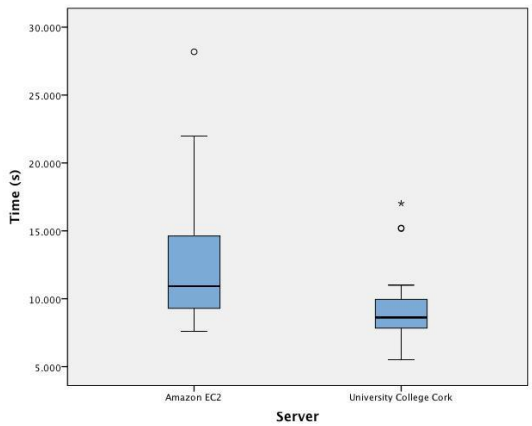

Fig. 9. 2MB Image Offload Server Comparison Boxplots. Plot of offload time (in seconds) for offloads to both an Amazon EC2 VM and a UCC Cloud VM, in the GSM signal range of [-99, -90] dBm, over the cellular 3G network. Data taken from 25 offloads of the file for both servers in the [-99, -90] dBM range of Fig. 3.

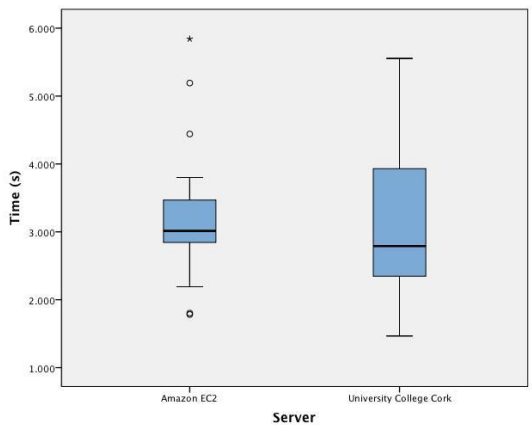

Fig. 10. 250KB Image Offload Server Comparison Boxplots. Plot of offload time (in seconds) for offloads to both an Amazon EC2 VM and a UCC Cloud VM, in the GSM signal range of [-99, -90] dBm, over the cellular 3G network. Data taken from 25 offloads of the file for both servers in the [-99, -90] dBM range of Fig. 6.

7.3 Power Experiments

Answering research question 3 was a more difficult endeavour. For the Android OS, there is no publically available power or energy API. One can use a graph in the device Settings to determine how much battery energy has been used by the different OS components and applications in percentages, but this is not fine grained, the figures are only estimates. In the MAUI code offload paper [3], the authors used external measuring equipment placed between the battery and the device to construct a model. The implications of this for a developer are that they cannot dynamically evaluate power consumption of an application they develop.

In our research, we discovered the Trepn [31] profiler tool, developed by Qualcomm. Qualcomm manufacture the processors used in various mobile devices and the tool can be used to profile various aspects of the mobile device performance. A profiler application is simply installed on the mobile device, and is started. The user can then carry out whatever tasks they want to profile the performance of. The profiler is then stopped, plugged into the Eclipse IDE, and a graph printout shows the collected data. Unfortunately, for our primary Samsung Galaxy S3 development device, it does not profile battery power usage. This is why at this late stage the

Google Nexus 5 device was introduced, which is fully compatible with the profiler. Therefore, the following experiments were carried out with the Google Nexus 5 device. For this experiment, the profiler was started. We conducted offloads of the 2MB file over the [-110, -100], [-99, -90], and [-89, -80] dBm GSM signal ranges. The results for the [-89, -80] and [-110, -100] dBm ranges (in the aforementioned order) are featured in Fig. 11-12. For each experiment, the profiler measured the battery power before, during, and after the offloads, in Watts. Just for reference, Fig. 13 shows a power draw trace gathered when no offload was taking place. Aside from

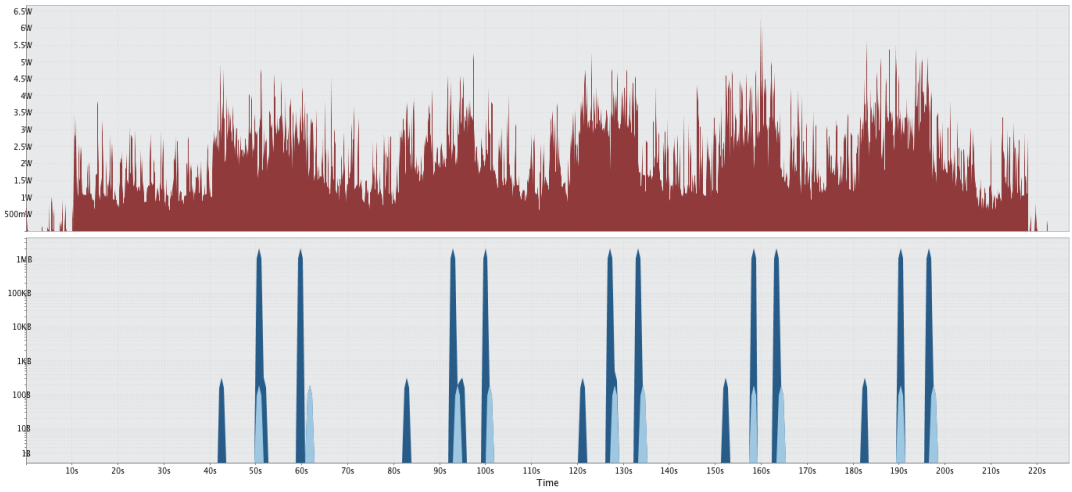

Fig. 11. 2MB Image Offload Trepn Profiler Graphs from [-89, -80] dBm GSM Signal Range. Top graph shows battery power usage (Watts), bottom graph shows mobile data usage (Bytes), both over time (Seconds), during offload to a UCC cloud VM over the cellular data connection with the Google Nexus 5. Five offloads occurred here. Offloads occur in the periods between the tall blue spikes in the bottom graph. Therefore, spikes appear in five pairs; one pair for each offload. The first spike in a pair is the time at the start of the data transfer, and the second spike is the time at the end of the data transfer. The smaller spikes in the bottom graph result from DNS queries and handshaking. Dark blue represents data sent, light blue represents data received.

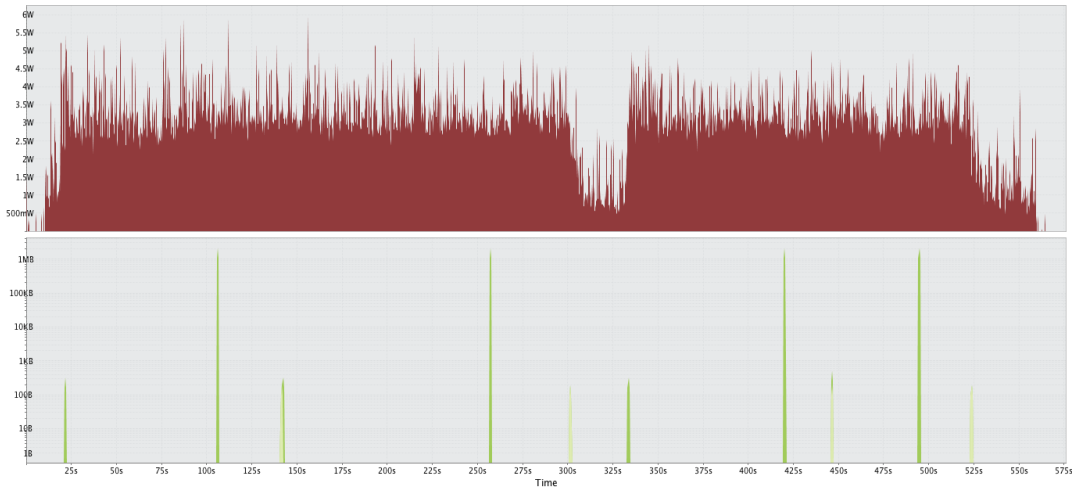

Fig. 12. 2MB Image Offload Trepn Profiler Graphs from [-110, -100] dBm GSM Signal Range. Top graph shows battery power usage (Watts), bottom graph shows mobile data usage (Bytes), both over time (Seconds), during offload to a UCC cloud VM over the cellular data connection with the Google Nexus 5. Two offloads occurred here. Offloads occur in the periods between the tall green spikes in the bottom graph. Therefore, spikes appear in two pairs; one pair for each offload. The first spike in a pair is the time at the start of the data transfer, and the second spike is the time at the end of the data transfer. The smaller spikes in the bottom graph results from DNS queries and handshaking. Dark green represents data sent, light green represents data received.

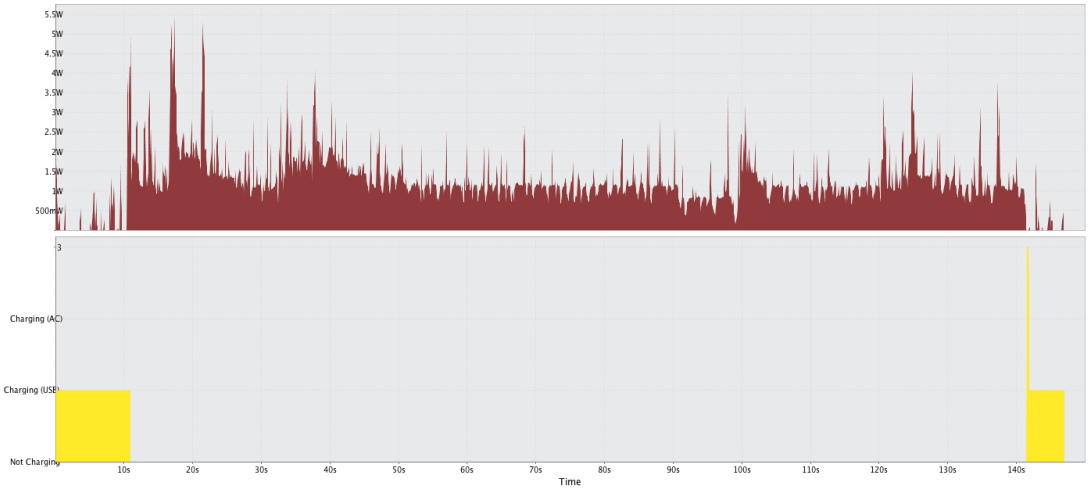

Fig. 13. No Offload Activity Trepn Profiler Graphs. Shows battery power usage (top) and charge state (bottom) while no offload was occurring with the Google Nexus 5. Aside from the occasional spikes, power usage rests between 1W and 1.5W. We also see that during the time the device is plugged into a USB charger (time periods when the yellow rectangular areas on the bottom graph are visible on the left and right), power usage drops further.

occasional spikes, the power consumption remains levelled out between 1W and 1.5W for the trace with no offload taking place. This base power draw could of course vary depending on what other applications and services are running on the device at the time.

The interesting results show that there was little difference for each of the signal ranges tested. When the offloads start, there is an initial ramp up of energy used, before the offload starts. Once the offload starts, the power for each of the offloads performed in all signal ranges shows many peaks of between 5W and 7W, but all graphs level out between 3W and 3.3W. After the offload completes, the power does not drop for a few seconds; this is the tail energy as demonstrated in the related works. The same trend was found in the [-99, -90] dBm GSM signal range, and therefore the graph is not shown.

We expected that the power used would be greater as the signal strength decreased, especially for the [-110, -100] dBm range, as with the results from Section 7.1. This however does not seem to be the case. Of course, it should go without saying that because of the longer offload duration of the poorer signal strength, especially the [-110, -100] dBm range, that the transfers take more time, and as such, the power draw from the battery will occur for a longer duration of time.

In terms of energy usage, while we cannot explicitly measure the energy without extra equipment, we can estimate it based on the power usage observed, and the time spent offloading. We also cannot specify an exact value for energy usage; as we see from Fig. 11-12, with varying power spikes during offloads, a constant value would not be appropriate. Instead, we specify a lower-bound estimate on the offload energy:

$$\Omega(e) = 3\mathrm{W}(t) \tag{10}$$

This formula is simply derived from the formula for Energy, $E = PT$, where E = energy, P = power (or work done), and T = time. In equation (10), we have e = energy in Joules, and t = time. 3 is derived from the minimum of 3W of power observed from Fig. 11-12 during the offloads.

Deriving an upper-bound estimate on the energy is less practical. In our experiments, the highest power usage spike observed during an offload was around 7W. Defining an upper bound from 7W would by and large grossly over-estimate the energy used. Instead, we define a weaker upper-bound estimate on energy:

$$\theta(e) = 3.3W(t) \quad (11)$$

In Fig. 11-12, ignoring the high spikes, the maximum offload energy used commonly levels off around 3.3W, and so we define (11) as such. The derivation and terms used are the same as (10).

From these results, it would appear that in terms of power draw, there is no need to include any specific power draw aspect into the model, as it does not vary considerably with stronger and weaker signal strengths. To lessen the energy utilised for offload, the speed of the offload must be maximised to reduce the time taken. From the results in Section 7.1, to achieve high speed, it is crucial to offload when the signal strength is greater than -100dBm. Of course, this result is an interesting difference to the findings for the Bartendr work by Schulman et al [13], which found a higher power draw for lower signal strengths as described in Section 4.1.

8. Modelling for Thin Client Offloading

Based on the experiments performed, and the obtained result data, we derive a simple model for the mobile cloud offload decision, which takes into account the resources under discussion. Section 8.1 provides some practical analysis of implementing such a model. Section 8.2 will present the decision model. Section 8.3 briefly describes the implementation of the mechanism for our Android based thin client.

8.1 Analysis

Considering the related works presented which provide models for energy usage at the radio level, it is difficult to incorporate these at the application level, as anybody developing a mobile cloud application may not have control over data size, radio state, or distance between the mobile device and the cell tower. When bandwidth is taken into account, regardless of bandwidth utilisation on the network, the user will want their tasks and data to be transferred regardless of the bandwidth utilisation of the network at some point. In the related works we have presented, we have already described the network state profiling and optimisation decision on local versus remote execution. Our aim is to remove this overhead, which can have a detrimental impact on the user experience, and waste resources.

The major obstacle in implementing such presented models at the application level is the limited options available to the developer. The Android API does not feature any functionality that allows the developer to inspect the power or energy state, or determine the state of the network quality without sending packets out over the network connection to evaluate RTT and bit-error rates.

Another area of difficulty with the Android API is in the details provided by the PhoneStateListener. A knowledgeable reader may have questioned why we based our signal range experiments for a 3G network transfer on the GSM signal strength; 3G networks use UMTS with CDMA instead of GSM. Indeed, one can read both the GSM and CDMA signal strength from the PhoneStateListener in Android. However,

Table 1. Offload Decision Model Variables.

Variable	Description
$p \in (0, 1)$	Offload Priority
s	Offload Data Size in Kilobytes
$r \in (-51,\ldots,-120)$	Signal Strength in dBm
$b \in (0,\ldots,1)$	Battery Percentage
$o \in (0, 1)$	Offload Decision Variable

implementing this functionality is dependent on the NIC manufacturer. On our Samsung Galaxy S3, if a developer queries the CDMA signal strength, while connected to a 3G network, the method call will return -1. This indicates it cannot read the CDMA signal strength. -1 is also returned if the developer queries the GSM bit-error rate, another method call in the Android API. The only data our Samsung Galaxy S3 device provided was the GSM signal strength. As such this was our only option to try and gauge signal quality, without adding additional network overhead. As such, for a developer, the minimal signal information that can be relied on is the GSM signal strength measured in RSSI or dBm, at least in Europe presently.

8.2 Modelling the Offload Decision

Based on our experimental results in Section 7, and our desire to not add any additional overhead to the mobile device, we introduce a simple model for offloading over the cellular network connection. For the model, we define the variables outlined in Table 1.

We then define the offload decision model as:

$$o = \begin{cases} 1 & p = 1 \\ 1 & decide(r, s, b) = true \\ 0 & decide(r, s, b) = false \end{cases} \tag{12}$$

The three case stack in (12) has 3 possible outcomes. We introduce a priority variable *p*. If this is set to 1, then regardless of the signal strength *r* or battery percentage *b*, offload will take place immediately; hence *o* is set to 1. The user can specify if a task is high priority using the thin client. The second and third cases rely on a decide function, which encapsulates a set of rules to determine if offload should take place. If the device is connected to Wi-Fi, offload will always take place immediately, but the decision process can easily be applied to this connection as well, in terms of the battery power, or even extended further to take into account performance based on Wi-Fi signal strength.

The decide function takes three parameters, *r*, *s*, and *b*. The algorithm pseudo code is presented in listing 1.

The comments in Listing 1 describe how the algorithm decides if the offload should occur. Any developer, or even the user, may choose their own battery policies for offload decisions; this is just our particular implementation; one may even leave the battery percentage values in the cases to the user to decide as a preference. The important decision processes in this algorithm are the cases; we do not offload at all if the signal strength is weaker than -100dBm. Otherwise, we consider the size of the data offload and the current battery situation. For cases where the data offload size is

Listing 1: Algorithm decide
Inputs: GSM signal strength *r*, data size *s* (in Kilobytes), battery percentage *b*
Output: true if offload should occur, false otherwise

```
if -100 < r                     //If signal is weaker than -100dBm
    return true                 //Do not offload
else
    case s > 500                //Case: data size greater than 500KB
        if b > 0.25             //If battery percentage is greater than 25%
            return true         //Offload
        else
            return false        //Do not offload
    case s <= 500               //Case: data size less than or equal to 500KB
        if b >= 0.10            //If battery percentage is greater than 10%
            return true         //Offload
        else
            return false        //Do not offload
    end case
end
```

greater than 500KB, in our results, the offload time would vary depending on the size. For cases where the data offload size is less than or equal to 500KB, there was little difference in the offload times. As described in Section 7.3, for sizes below 500KB the time is mostly dominated by network overhead, rather than payload size. Where the battery drops to 10% or lower, the user more than likely wants to conserve the battery life of the device for important calls or messages, rather than having additional network activity occur.

It is worthwhile briefly mentioning a situation where the signal strength changes during the offload. If the signal strength weakens to a value less than -100dBm, perhaps the offload should be suspended. We do not implement this in our model. If an offload has started, then regardless of how the signal strength changes, it should continue until completion. In the related works, disconnection during offload is not discussed, except for in the MAUI [3] framework. With MAUI, if the mobile device suffers a disconnection from the cloud while code has been offloaded, after a certain timeout period, the local application will execute the offloaded code itself. This is a waste of resources in the case where offload has successfully taken place, but no result has been received. It is our opinion that starting an offload in our model, and using up time and energy as a result, should not be made to count for nothing if the signal strength deteriorates.

8.3 Model Implementation

Our existing thin client Android application was modified to implement the model in Section 8.2. Whenever the user enters the details of a task to be offloaded to their CPA, for example, a file to be offloaded to a cloud storage provider as described in our previous work [29], the task is forwarded to a TaskOffloadHandler class. This class features our own custom implementation of a Queue data structure, the TaskOffloadQueue. Whenever a task is passed to the TaskOffloadHandler, unless it has been assigned priority, the task is placed into this queue, along with some additional offload data, such as the data-size. If the user has assigned priority to a

task, it is offloaded immediately. Our thin client uses RESTful web service architecture to send task data to the CAMCS middleware. The task is converted into JSON format for the transfer. Files, such as the image files in our example, are Base-64 encoded to a String for offload. Based on the number of ASCII characters in the JSON String and any Base-64 encoded files, the data size is calculated.

At this point, if the queue was previously empty, the Handler will register with the Android PhoneStateListener, to receive updates when the signal strength changes. When a signal strength change is detected (which is received as an RSSI value, which can be mapped to a dBm value), the queue is iterated, and for each task in the queue, the decide algorithm is executed. Based on the outcome of this, the task is offloaded, or the task remains in the queue. When the queue has finally been emptied of tasks, the Handler will unsubscribe from the PhoneStateListener. The user is notified via the Android notification tray when a task is being offloaded, and when the offload is complete.

9. Discussion

To enable a mobile cloud computing model that works seamlessly for the user, the resources we have described in this paper must be managed cost-effectively. For the end-user of the mobile device, the effectiveness of how energy and bandwidth are managed, will determine how successful the implementation of the model will be. Users want batteries that last longer, and have limited patience for delay from the network connection, so only if these resources are used wisely will we see users uptake this model. We already know that these resources are in limited supply to begin with. We now highlight the required elements for the three resources analysed in this paper to define the elements of a best practice model for management.

For energy, we have highlighted in Section 4 many of the previous works that have attempted to model the energy usage of the mobile device, and we can see that the models have brought the energy usage down to three factors: power for the radio, the amount of data transferred, and the distance from the base-station. To enable a successful implementation of the mobile cloud model, we need to minimise these factors. While we cannot control how power is supplied to the radio, we can control and minimise how much data is transferred over the connection, and we can specify the location of cloud deployments worldwide so that they are close to the user, minimising network delay where possible (we cannot force a user to move close to a base-station, nor expect them to stay there). The other factor in this regard is how often the network connection is used, as this also has a detrimental impact on the energy usage; it must also be kept to a minimum. Only when all these factors are minimised, we can realize a mobile cloud model that effectively manages energy for the user.

For bandwidth, we cannot control or guarantee any size allocation being made available to the mobile device, on the fly, or in advance. Only approaches that are going to have minimal bandwidth requirements in this regard will be successful and tolerated by the end user. We must develop systems under the assumption that we have minimal bandwidth available, and the mobile device must adapt its use of the mobile cloud appropriately to situations where a large amount is available, or where a small amount is available. If a large amount of bandwidth is available, such as on a Wi-Fi network, it should be used for the greatest advantage to the user, but only by assuming that little bandwidth is available, will all situations be tolerated by the user.

It is the same issue that applies to energy consumption; the amount of data that is transferred over the network must be minimal, and data transfer should occur sparsely. While approaches may take advantage of high bandwidth situations such as on Wi-Fi networks, and transfer large amounts of data at these times, the issue of fairness for other users connected to access points will be of concern as well.

While the end-user will likely not be concerned with what is happening on the cloud deployments at data centres, how the resources will be provisioned there, and what those resources are will be of technical importance to the implementation of the mobile cloud. Our belief is that not only the footprint on hardware requirements should be small on the cloud, but also the software requirements must not be extensive either. As more consumers of mobile cloud come online, the requirements at the cloud, resulting from factors such as personalisation of applications and data, must not grow. Providing a VM to each user for example may grow at an un-manageable rate. Replicating custom and personal software for users in the cloud (in the sense that users currently have personalised apps on their phones) may be difficult to scale, considering the size and complexity of the software. To achieve effective resource utilisation, we must minimize the factors we outlined in equation (8). We have seen that code offload approaches and our middleware do this most effectively. In addition, for our middleware, we believe that the use of existing software and services, already located in a distributed SOA fashion in various cloud deployments, that can be shared amongst many end-users will be the appropriate deployment model in the cloud moving forward. This is especially useful, as the VM is not relied upon to carry out intense computation and subject to the resource limitations and performance demand penalties that may come with a VM.

Our model for offload decision making over the cellular network in equation (12), in contrast to other works, does not impose additional profiling overhead on the device, further adding to the user experience, such as no energy or time penalties. We have observed from our experiments that offloading time increases significantly when the GSM signal strength falls below -100dBm; above this, offload times are very similar. We also observed that when the size of the data offloaded falls below 500KB, the median offload times are almost the same. In terms of the cloud deployment location, the results show that the actual deployment location relative to the user location had little difference in median times, but for the further away Amazon EC2 server, there was more variance. Battery power usage, aside from power spikes, was found to be the same regardless of the signal strength during offload.

In terms of the best practices outlined in this paper, the implications of these results from our experiments, now incorporated into the offload decision model, tell us that in order to manage energy usage effectively at the mobile device, the time for offload is the crucial factor. Minimising offload time is the goal to meet, and the best way to achieve this is to offload when the GSM signal strength is greater than -100dBm. The results also show the value of the low bandwidth approach. The smaller the data size, it goes without saying, the better performance; but for approaches that rely on transferring large volumes of data, practical application may be difficult, considering the substantial time overhead in comparison with small amounts of data. For example, consider the previously discussed remote display approaches. Unless compressed, transferring an image of a high resolution desktop output continuously over the network, judging by our experimental results, will result in a very poor user experience, and will use available resources very inefficiently. With small amounts of data, the network overhead is the primary time contribution, and so real time applications must focus on data size and frequency of use

optimisations, by making sensible use of data transferred when required. Of course, the fact that this transfer may have to take place continuously will also have a detrimental impact on the energy resources and performance. Our middleware does not require continuous data transfer, just the offload, and a result. The actual size of our task offloads in our previous work for task descriptions [2] were between 2KB and 5KB. Offloading such a small amount of data, while the device has a strong signal, will reduce offload time, and save energy.

Our cloud middleware and the thin client can effectively manage all these resources to provide an effective model to satisfy the consumer of mobile cloud applications. If we contrast with the other approaches and their implementation details that we have outlined in this paper, our data transfer is minimal. We do not require continuous connections to cloud infrastructure once tasks have been offloaded to the users' CPAs. In addition, the CPA is capable of carrying out work and delivering results to users without the user having to request it. Results are stored with the CPA until the mobile device is available, and so it will support disconnected operation. Our model for offloading over the cellular network on the thin client considers the resources available, with no additional overhead. Our development goal is that the middleware will de deployed on multiple clouds, so that it is always close to the position of the user, and the CPA will move between these deployments; even if this showed little impact on our results, the shorter network traversals should beep the time variance low. The server side requirements only require an appropriate application container and database, and it does not require any physical resources to be allocated to each end user. Table 2 compares the discussed approaches with our middleware project, along the lines of how they manage the available resources discussed in this paper.

10. Conclusions

In this paper, we have presented an analysis of how mobile device resources such as energy and bandwidth, along with cloud infrastructure resources, can be managed effectively in the mobile cloud domain, and we have modelled best practice approaches for implementations. We applied these to existing works in the area, along with our Context Aware Mobile Cloud Services (CAMCS) cloud middleware and the Cloud Personal Assistant (CPA), the representative of the user within the middleware.

Our CPA works in the cloud to complete user assigned tasks, using existing cloud software and SOA services. Most importantly, it works in a disconnected fashion, which has several advantages in terms of resource management, compared with other existing approaches in the area. The network connection is not in continuous use, or required to be active as in other works. It is infrequently used, simply to send the task to the CPA from the mobile thin client, and receive the result later. The amount of data sent over the connection, the task description, and the result, is character-based and small in size. As a result of the small data size, and infrequent use of the network connection, the approach minimises energy and bandwidth use, when compared with some of the existing approaches described in this paper. The server side footprint at the cloud is also smaller compared to other approaches, as we do not require multiple cloud based hardware and software resources for each user of the middleware. The middleware runs within an application container, which can be deployed in multiple clouds, so that a user's CPA is nearby, minimising communication time variance.

Table 2. Mobile Cloud Computing Approach Resources. Comparison of the resource requirements of each of the discussed approaches to mobile cloud computing models, along with the CAMCS middleware project with the CPA by the authors

	Cloudlets	**Remote Display**	**Code Offload/Application Partitioning**	**Disconnected Operation (CPA)**
Continuous Connection	Yes	Yes	Yes	No
Energy	High due to virtual machine nature	High due to virtual machine nature	Varies depending on amount of data offloaded and frequency	Low as data transfer and frequency is small
Bandwidth	Moderate, 1-Wi-Fi hop	High, many hops	Varies depending on amount of data offloaded and frequency	Low as data transfer and frequency is small
Cloud	Hypervisor for user virtual machines	Hypervisor for user virtual machines	Varies (entire operating system, custom application)	Middleware application using existing software/ services
Decision Model	None, user initiated	None, user initiated	Varies, user initiated or network profiler with optimisation solver	No overhead model using existing information

We have also presented experiments conducted with mobile devices to judge the performance of offloading to the cloud from our thin client application, along with their results. We looked at the effects of signal strength, cloud deployment location, and power usage during offload, while offloading with various different data sizes, and ranges of GSM signal strength. From these results, conclusions were drawn which were implemented into an offload decision model for the cellular network, which imposes no additional overhead on the mobile device. The conclusions drawn from these results were discussed in the context of our approach, and the implications for other approaches.

Applying such management to the resources is important, because each user of the mobile cloud will have many tasks to be completed, and these resources will be required separately for each task; these resources must be used with the considerations outlined in this paper in mind. Effectively minimising the usage of these resources, and applying the best practices we have outlined, will be crucial in successful adoption of the mobile cloud computing model by the end user.

Our future work will involve continued development of the CPA, and our CAMCS cloud middleware project within which the CPA resides, while adopting the practices outlined in this paper, the focus being on minimal resource usage. We also intend to

extend our offload decision model to Wi-Fi networks. In addition, once our CAMCS middleware has been further developed with features such as service discovery, we will evaluate its usage of VM resources and performance under load with ordinary operation by end users.

Acknowledgement

The PhD research of Michael J. O'Sullivan is funded by the Embark Initiative of the Irish Research Council. The authors also wish to extend their gratitude to the reviewers of this paper for their helpful comments and suggestions for improvement.

References

[1] M. J. O'Sullivan, D. Grigoras. "User Experience of Mobile Cloud Applications – Current State and Future Directions", Proceedings of the 12th International Symposium on Parallel and Distributed Computing, Bucharest, Romania, 27-30 June, 2013, pp. 85-92.
[2] M. J. O'Sullivan, D. Grigoras. "The Cloud Personal Assistant for Providing Services to Mobile Clients", IEEE MobileCloud, Redwood City, San Francisco Bay, USA, 2013, pp. 477-484.
[3] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, et al. "MAUI: making smartphones last longer with code offload", Proceedings of the 8th international conference on Mobile systems, applications, and services, San Francisco, California, USA. 1814441, ACM, 2010, pp. 49-62.
[4] B-.G. Chun, S. Ihm, P. Maniatis, M. Naik and A. Patti, "CloneCloud: elastic execution between mobile device and cloud", Proceedings of the sixth conference on Computer systems, Salzburg, Austria, 1966473, ACM, 2011, pp. 301-314.
[5] R.K.K. Ma, L. King Tin, W. Cho-Li, "eXCloud: Transparent runtime support for scaling mobile applications in cloud', International Conference on Cloud and Service Computing (CSC), Hong Kong, PR China, 12-14 December, 2011, pp. 103-110.
[6] Y. Ye, N. Jain, X. Longsheng, S. Joshi, I-L. Yen, F. Bastani et al, "A Framework for QoS and Power Management in a Service Cloud Environment with Mobile Devices", Fifth IEEE International Symposium on Service Oriented System Engineering (SOSE), Nanjing, PR China, 4-5 June, 2010, pp. 236-243.
[7] Q. Wang, R. Deters, "SOA's Last Mile-Connecting Smartphones to the Service Cloud", Proceedings of the IEEE International Conference on Cloud Computing (CLOUD), Bangalore, India, 21-25 September, 2009, pp. 80-87.
[8] J. Sankaranarayanan, H. Hacigumus, and J. Tatemura, "COSMOS: A Platform for Seamless Mobile Services in the Cloud", 12th IEEE International Conference on Mobile Data Management (MDM), 2011, pp. 303-312.
[9] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: Enabling mobile phones as interfaces to cloud applications", Middleware 2009, pp. 83-102.
[10] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing", IEEE Pervasive Computing, Vol. 8, No. 4, pp. 14-23, October 2009.
[11] T. Verbelen, P. Simoen, F. De Turck and B. Dhoedt, "Cloudlets: bringing the cloud to the mobile user", Proceedings of the third ACM workshop on Mobile cloud computing and services, Low Wood Bay, Lake District, UK. 2307858, ACM, 2012, pp. 29-36.
[12] P. Simoens, F. De Turck, B. Dhoedt, and P. Demeester, "Remote Display Solutions for Mobile Cloud Computing", Computer, Vol. 44, No. 8, pp. 46-53, August 2011.
[13] A. Schulman, V. Navday, R. Ramjeey, N. Spring, P. Deshpandez, C. Grunewald et al. "Bartendr: A Practical Approach to Energy-aware Cellular Data Scheduling", Proceedings of the sixteenth annual international conference on Mobile computing and networking (MobiCom'10), September 20–24, 2010, Chicago, Illinois, USA, pp. 85-96.
[14] N. Balasubramanian, A. Balasubramanian, A. Venkataramani, "Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications", Proceedings of the 9th ACM SIGCOMM conference on Internet measurement (IMC '09), November 4–6, 2009, Chicago, Illinois, USA, pp. 280-293.
[15] W. R. Heinzelman, A. Chandrakasan, H. Balakrishnan. "Energy-Efficient Communication Protocol for Wireless Microsensor Networks", Proceedings of the Hawaii International Conference on System Sciences, January 4-7, 2000, Maui, Hawaii, USA, pp. 1-10.
[16] P. Ramanathan, K. M. Sivalingam, P. Agrawal, S. Kishore, "Dynamic resource allocation schemes during handoff for mobile multimedia wireless networks", IEEE Journal on Selected Areas in Communications, Vol. 17, No. 7, pp.1270-1283, July 1999
[17] B. Li, L. Yin, K. Y. M. Yong, S. Wu, "An Efficient and Adaptive Bandwidth Allocation Scheme for Mobile Wireless Networks Using an On-Line Local Estimation Technique", Wireless Networks, Vol. 7, No. 2, pp.107-116, March 2001
[18] L. Xu, X. Shen, J. W. Mark, "Dynamic bandwidth allocation with fair scheduling for WCDMA systems", IEEE Wireless Communications, Vol. 9, No. 2, pp. 26-32, April 2002.
[19] F. J. R. Tocado, A D. Zayas, P. M. Merino Gomez, "Characterizing Traffic Performance in Cellular Networks", IEEE Internet Computing, Vol. 18, No, 1, pp. 12-19, Jan.-Feb. 2014.
[20] P. H. Gomes, N. L. Saldanha da Fonseca, O. C. Branquinho, "Radio Resource Allocation and Green Operation for Mobile Access Networks Based on Radio-over-Fiber", IEEE Transactions on Mobile Computing, Vol. 13, No. 4, pp. 894-906, April 2014.
[21] O. Kivekäs, J. Ollikainen, T. Lehtiniemi, P. Vainikainen. "Bandwidth, SAR, and Efficiency of Internal Mobile Phone Antennas", IEEE Transactions on Electromagnetic Compatibility, Vol. 46, No. 1, pp. 71-86, February 2004.
[22] H. Liang, D. Huang, D. Peng. "On Economic Mobile Cloud Computing Model", Second International ICST Conference, MobiCASE 2010, Santa Clara, CA, USA, October 25-28, 2010
[23] M. R. Rahimi, N. Venkatasubramanian, S. Mehrotra1, A. V. Vasilakos. "On Optimal and Fair Service Allocation in Mobile

Cloud Computing", 20th August 2013, http://arxiv.org/abs/1308.4391, Last Accessed 30th December 2013.
[24] H. Liang, L. X. Cai, D. Huang, X. Shen, D. Peng, "An SMDP-Based Service Model for Interdomain Resource Allocation in Mobile Cloud Networks", IEEE Transactions on Vehicular Technology, Vol. 61, No. 5, pp. 2222-2232, June 2012.
[25] R. Kaewpuang, D. Niyato, P. Wang, E. Hossain, "A Framework for Cooperative Resource Management in Mobile Cloud Computing", IEEE Journal on Selected Areas in Communications, Vol. 31, No, 12, pp. 2685-2700, December 2013.
[26] Z. Sanaei, S. Abolfazli, A. Gani, M. Shiraz, "SAMI: Service-based arbitrated multi-tier infrastructure for Mobile Cloud Computing", 1st IEEE International Conference on Communications in China Workshops (ICCC), Beijing, PR China, 15-17 August 2012, pp. 14-19.
[27] Anton Beloglazov, "Energy-Efficient Management of Virtual Machines in Data Centers for Cloud Computing", PhD thesis, Department of Computing and Information Systems, The University of Melbourne, Australia, 2013.
[28] K. Kumar, Y-H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?", *Computer*, Vol. 43, No. 4, pp.51-56, April 2010.
[29] M. J. O'Sullivan, D. Grigoras, "Application Models Facilitated by the CPA", Proceedings of the 6th International Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications (MOBILWARE), Bologna, Italy, 11-12 November, 2013.
[30] S. Clinch, J. Harkes, A. Friday, N. Davies, M. Satyanarayanan, "How close is close enough? Understanding the role of cloudlets in supporting display appropriation by mobile users", IEEE International Conference on Pervasive Computing and Communications (PerCom), Lugano, Switzerland, 19-23 March, 2012, pp. 122-127.
[31] Trepn Profiler, https://developer.qualcomm.com/mobile-development/increase-app-performance/trepn-profiler, Last Accessed 26th May 2014.