# Making the Case For Computational Offloading in Mobile Device Clouds

**Afnan Fahim, Abderrahmen Mtibaa, and Khaled A. Harras**

June 2013
CMU-CS QTR-120
CMU-CS-13-119

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Carnegie Mellon University, Qatar campus.
The author can be reached at afahim@qatar.cmu.edu or amtibaa@cmu.edu or kharras@cs.cmu.edu.

**Abstract**

It is common practice for mobile devices to offload computationally heavy tasks off to a cloud, which has greater computational resources. We consider an environment in which computational offloading is made among mobile devices. We call such an environment a *mobile device cloud* (MDC). In this work, we first highlight the gain in computation time and energy consumption that can be achieved by offloading tasks to nearby devices inside a mobile device cloud. We do this by emulating network conditions that exist for different communication technologies provided by modern mobile devices. We then present a platform that allows creation and offloading of tasks by a mobile devices to nearby devices. Such a platform consists of an API, an accompanying Android application deployable across MDC devices, and a test bed to measure power being consumed by a mobile device. Finally, we create and utilize a testbed, which consists of four Android devices and energy measurement equipment, in order to validate our intuitions and qualify the gain in time and energy which we deduced from the emulation experiments. Using this test bed we show up to 50% gain in time and 26% gain in energy by employing task offload in MDC's versus executing tasks locally.

* Funded by the CMUQ Seed Funding

**Keywords:** Mobile Cloud Computing, Computational Offloading, Mobile Device Cloud

# Contents

# List of Figures

# 1 Introduction

It is common practice for mobile devices to offload computationally heavy tasks off to a cloud, which has greater computational resources. Solutions have been presented which partition any given task into separate parts and offload these to a cloud so as to minimize the time taken to carry out the task [CIM+11]. However, this type of offloading is expensive due to high energy costs and the high latency which exists between the cloud and the offloading mobile device. As an answer to this problem, 'Cloudlets' were proposed: smaller clouds placed closer to users which would make mobile task offloading less expensive in terms of energy waste and time consumption [SBCD09]. The idea of reducing communication costs by executing closer to the offloader device was then extended to introduce mobile device cloud computing - where the idea is to offload tasks to nearby devices, be they mobile or stationary - so as to reduce communication costs and latency [SLAZ12].

Mobile devices are now powerful. Recent studies forecast that, by 2014, mobile usage will take over desktop usage [tag13]. Mobile devices are increasingly becoming resource intensive, and with the advent of wearable computing devices like the Pebble and Google Glass, the need for task offloading is even more severe since these devices come with limited processing capabilities [goo13]. Thus a solution that would allow such devices to save time and energy by offloading to nearby devices rather than offloading to the cloud has lots of practical implications.

In this document we propose task offloading in a network of connected mobile devices, which we call a mobile device cloud (MDC). We define the device which is offloading computation as the *offloader*, and a device which is carrying out computation on behalf of another device as the *offloadee*. We define a task as a combination of the *data* that it takes in as input, and the *computation* that the task needs to perform on this data in order to yield a result. An application is comprised of a lot of such tasks, and the more data and computation intensive these tasks are, the more *energy* is required to perform them, and the more *time* it takes to complete these tasks. Figure 1 shows three options for computational offloading. Namely, a device can offload to a cloud which is far away, to a cloudlet which is a smaller set of servers located inside a building, nearby coffeshops, etc., or to an MDC. Each of these different choices offers different trade offs that need to be kept in mind when making the decision of which platform to offload to.

Attempts have already been made to minimize the time or energy loss by offloading these *heavy* tasks off to the cloud [CIM+11], or a nearby cloudlet [SBCD09]. Mobile device cloud computing attempts to not only overcome unavailability of such infrastructures, but also to save time and energy by offloading tasks to a nearby set of mobile devices that can carry out the tasks on behalf of the offloader.

This work highlights the potential gain in energy and time which can be achieved by offloading computation among devices in a MDC. We investigate if it is possible to gain energy and time by offloading computation by employing offloading in MDC. Answering this question involves surveying what different communication technologies are available to computational devices, and finding out what their trade offs are in terms of bandwidth, RTT, etc. We perform a set of experiments that emulate environments with the same network characteristics as those provided by these communication techniques so as to be able to determine whether it makes sense to (1) offload computation at all and (2) if so, for what combinations of data and computation should these tasks be offloaded so as to conserve time and energy. Using the emulation test bed, we show potential gain in both time and energy, up to 50% and 23% respectively, which can be achieved by offloading to other mobile devices in an MDC.

We propose an experimental platform that allows carrying out testing in the context of MDC's. With this platform, we create a test bed which allows us to measure the energy being consumed while a device is performing different tasks using different communication technologies, as well as the time taken to offload tasks to other devices and receive the result. We use this test bed to carry out experimentation which allows us to obtain insights into what kinds of tasks should be offloaded and in what scenarios is it better to offload tasks to a mobile device cloud versus executing it on the offloader device itself. Using the MDC platform, we have shown that it is possible to gain in time and energy, up to 50% and 26% respectively, by offloading within MDC's, as opposed to executing tasks on the offloader itself.

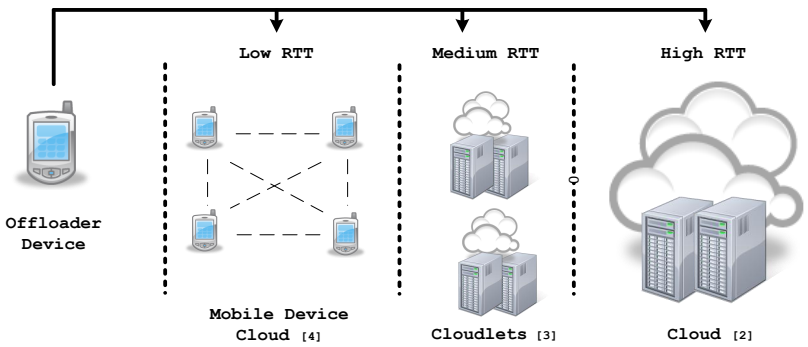



Figure 1: Scenarios for offloading computation

The rest of the document is organized as follows. Section II outlines related work regarding mobile cloud computing and offloading in mobile device clouds. We make the case for mobile cloud computing in Section III by use of the emulation test bed we have created. We outline the design of the MDC platform that we have built in Section IV, and present results and findings of experimentation carried out using this platform in Section V. We discuss our conclusions and on going work in Section VI.

## 2 Related Work

With the rise in demand of computational resources by mobile applications, various solutions for computation offloading to more powerful surrogate machines, known as cyber foraging, have been proposed by Satyanarayanan et. al. [Fli12]. Recent solutions include CloneCloud [CIM+11] and MAUI [CBkC+10]. CloneCloud decides, for any given task, whether to execute this task locally or to offload it to a remote cloud. It does not rely on developer effort, and by carrying out static and dynamic analysis, it partitions any given application into tasks that can be offloaded to other devices. It aims to minimizes the execution time of an application by offloading some of its constituent tasks off to a cloud, while executing the rest locally. MAUI relies on developer effort to convert mobile applications in a managed code environment to better support fine-grained real-time offload decision making; it also considers the possibility of offloading to different types of high-end infrastructures depending on their RTT in order to conserve energy. The impact of large RTT's on power consumption when offloading computation is further examined and utilized as an incentive for bringing resource-rich computational infrastructure, known as Cloudlets [SBCD09] closer to mobile devices.

Serendipity [SLAZ12] and Cirrus [SAZN12] devise solutions and architectures for making mobile device clouds possible. Cirrus looks into the spectrum of devices that can be used as part of a mobile device cloud, and proposes a holistic solution to cyber foraging which involves offloading not only to other mobile devices, but also to computers installed on moving vehicles or placed in different areas of a building. Serendipity is the first work that aims to develop and test a system that handles task allocation in mobile device clouds, and use emulation to explore the possible speedups gained and energy conserved using offloading in mobile device clouds. However, Serendipity does not consider all the technologies available to mobile devices in the present day, and it does not consider different cases of data and computation to see which ones are energy efficient and time saving as compared to others. Also, Serendipity focuses on mobile-mobile offloading only and does not consider the other two scenarios of offloading to a cloudlet or a cloud. As opposed to this approach, our work considers the full spectrum of communication technologies available to mobile devices today, and also considers all the different infrastructures a mobile device can offload tasks to (Figure 1). For these

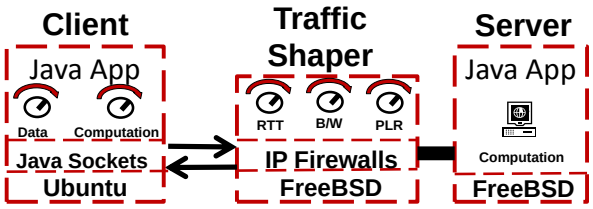



Figure 2: A high level architecture of our Emulation Testbed

combinations of communication technologies and infrastructures, our work identifies what combinations of data and combination are most efficient in terms of time and energy conservation, and confirms these insights by evidence from real world experimentation.

# 3 Making the case for Offloading in MDC

In this section, we investigate what combinations of data and computation would make a task suitable for offloading versus executing locally, considering the different communication technologies available. A variety of methods exist to carry out our investigation. We list four methods: (i) analytical modeling, (ii) network simulation, (iii) network emulation, and (iv) real world experiments. The potentials and limitations for each method have been widely discussed [Jai91]. Simulation, while offering a high degrees of freedom and reproducibility, is mainly criticized for inaccuracies in capturing realistic mobility and wireless medium characteristics. Real world testbeds, however, are very much limited in scope and induce high management overhead.

Because the emulation approach provides a balance between simulation and real world experiments, we implement an emulation testbed as a first step towards evaluating the potential gain of data and computation offloading in mobile environments. The test bed evaluates the gain achieved using any of the available communication technologies for different combinations of data and computation. We identify five types of communication technologies that could be used to offload tasks, namely, Bluetooth 3.0, Bluetooth 4.0, WiFi Direct, WiFi and 3G. We consider offloading to all three scenarios as shown in Figure 1, namely, to an MDC, to a cloudlet or to a cloud.

## 3.1 Emulation Testbed

We implement an emulation test bed in order to test task offloading in the context of MDC's. This test bed allows us to measure the energy consumed and time taken to complete tasks being offloaded from one mobile device to another. It consists of (1) a client, (2) a server and (3) a traffic shaper. These components are outlined below.

### 3.1.1 Client

The Client application represents an offloader device in an MDC. It is a Java Sockets based application which allows a user to define tasks as combinations of Data and Computation (in MB and MFLOP respectively). Once such tasks are created, they are sent to a server to be computed. Computation which is carried out on the server is abstracted as a number of additions of different predefined matrices composed of floating points. Data being sent to the server is represented as `String` objects of fixed lengths the size of which is equal to the specified data size in MB.

This application sends a user-specified percentage of the defined task to a Server application(described below) at a predefined IP address and port number, i.e., it sends the calculated MFLOP value representing

the computation to be offloaded, and a String object the size of which equals the data to be offloaded. The client application keeps track of the total time it takes to complete the offload operation, and receives the *server computation time* (how much time it took to execute the task by the Server application). The Client application carries out each offload operation three times and calculates the average total offload time and average server computation time. It then writes these times to a file.

The client application logs the time taken to (1 ) offload and (2) compute the task, as well as the calculated energy consumed by the offload operation. We only take the energy consumed due to communication into consideration. We define *completion time* as the total time to send data, carry out computations on each of the connected devices in parallel, and receive the result from each. We define *energy consumed* as the communication energy that is spent while offloading the task to the offloadee device, and it includes both the energy spent by the offloader to send the task as well as the energy spent by the offloadee to receive the task. We calculate energy consumed by multiplying the communication time with the corresponding power value in Table 1. These values have been taken from work carried out by Friedman et. al. [FKK11], and information present in the specification documents provided by the manufacturers of the chip set being used by the respective communication technology [bt413]. Friedman et. al have devised an experimental platform to measure the energy consumed by a mobile device while the device performs different tasks. We assume that the energy required to send and receive data using the communication technology is the same, and thus use the power measurements for sending operations only when calculating energy.

Table 1: Power Consumption of Sending Data using Wireless Technologies [FKK11]

| Technology | Power (mW) |
|---|---|
| Bluetooth 4.0 | 50 |
| Bluetooth 3.0 | 520 |
| WiFi Ad Hoc | 1548 |
| WiFi | 1568 |
| 3G | 2500 |

#### 3.1.2 Server

The Server application represents an offloadee device in an MDC. The Server application is also a Java Sockets based application. It listens to any tasks that the client application might offload to it. Once it receives a task sent by the client application, it executes the task and measures how much time it took to carry out the computation. It then sends this time value back to the Client application and continues listening for other tasks. In our work, we do not emulate is the computation capability of the offloadee device, and this is something we are considering as future work.

#### 3.1.3 Traffic Shaper

The main goal of the traffic shaper is to emulate the network conditions that are provided by the different communication technologies that we consider in our experimentation. All the packets going from the Client application to the Server application first go through a traffic shaper. Once the packets go through the traffic shaper, a certain level of RTT and bandwidth constraint are introduced to the connection which exists between the Client and the Server to emulate the network conditions provided by the type of technology being emulated. We use bandwidth values from the work carried out by Friedman et. al [FKK11] as well as specification documents provided by technology developers [bt413]. In case of RTT, we actually measure the RTT that exists for each of the communication technologies, as well as the RTT that exists when communicating with a Cloud. These measurements are carried out by pinging a server listening on each of the technologies and infrastructures that we are considering five times, and taking a average of the time it

takes to ping the server and receive the result. Since Cloudlets are not widely available, we pick two cases of RTT's, of 15ms and 30ms. We assume that no packet loss exists in the connections that we are emulating.

We consider two choices for building a setup that allowed us to emulate the environment - namely - NIST Net [CS03] and Dummynet [Riz97].

**NIST Net** is implemented as a kernel module extension to the Linux operating system, and is a general purpose tool for emulating network performance. It allows any computer to be converted into a router and this box can then be used to vary network conditions between different connected devices. We installed NIST Net on a VMWare virtual machine running on an Ubuntu machine. It was very challenging to work with NIST Net since it is a kernel module and thus was implemented on top of specific Ubuntu kernels, which aren't supported anymore. Since 2005, NIST decided to stop maintaining this software [nis13], and thus researchers have been relying on a combination of patches for different versions of Ubuntu kernel - and even these patches are outdated. In light of all these problems, we then decided to look into other emulation methods.

**Dummynet** is also implemented as a kernel module, on top of Free BSD, and is still supported by the FreeBSD Foundation. It is built on top of the IP Firewall framework, and using it, any connection coming in or going out of the system can be altered to introduce packet loss, latency and bandwidth in the connection. Since these functionalities fit our requirements, we decided to move forward with this piece of software as our emulation tool.

#### 3.1.4 Testbed Implementation

The client application runs on an Intel Core 2 Duo machine running on Ubuntu 12.04 LTS. We call this machine the *host* machine. The server application runs on a *virtual* machine which is hosted on top of the host machine which runs the client application. This virtual machine runs FreeBSD. We install dummynet (the trafficshaper) as a kernel module in the virual machine that runs the server application. The high level architecture of our testbed, with the client, server and traffic shaper, can be seen in Figure 2.

### 3.2 Experimental Methodology

In our experiments, we vary the computational size (denoted by MFLOP) and data being sent (denoted by MB). We measure the *completion time* and the *energy consumed* for these different combinations of computational size and data being offloaded to another device. In each of the experiments we offload half of the computation and carry out the rest of the computation on the offloader device itself.

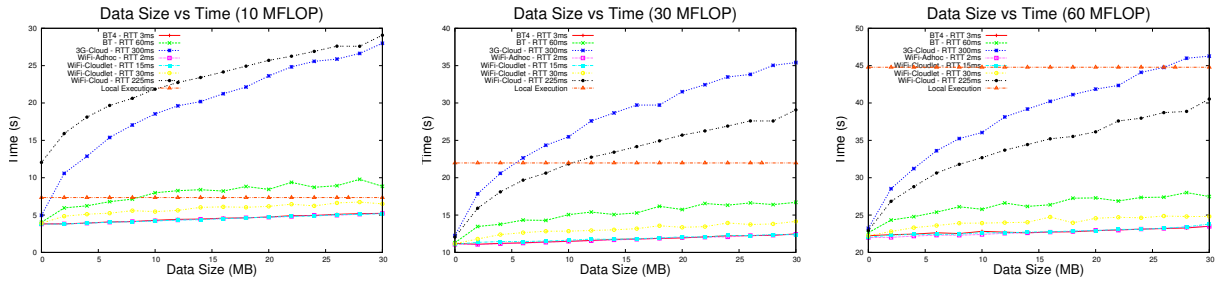
We note that 10, 30 and 60 MFLOP correspond to computational complexities of low, medium and highly complex applications. A detailed method to estimate the MFLOP of a given application is given in Appendix A. For each of these three cases we vary the data being offloaded along with the task (0 - 30 MB at intervals of 2 MB). For each of these cases we measure the time taken to carry out the task offload, and use this time to calculate how much energy was consumed by the task offload operation.

### 3.3 Results

The results of the emulated experiments are outlined in Figures 3 and 4. We present results for three cases of computation - low, medium and high. Each of these cases corresponds to a certain value of computation in MFLOP, as described in the previous section on Experimental Methodology. These results provide us with insights into the potential gains in time and energy that can be achieved by offloading in MDC's.

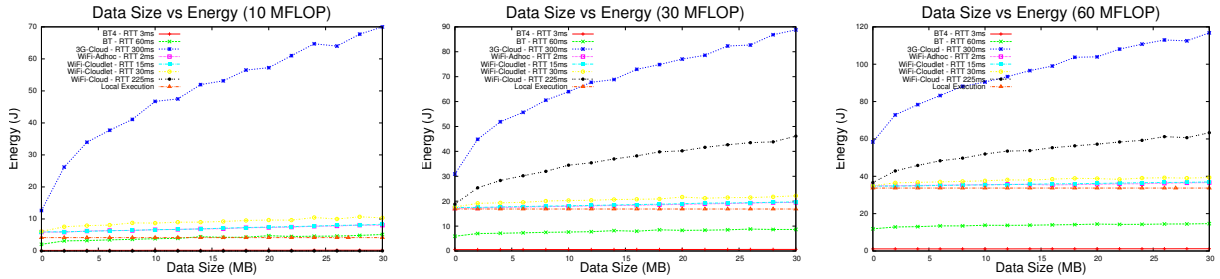
#### 3.3.1 Case of Low Computation (10 MFLOP)

The first set of experiments deal with offloading tasks of low computational intensity - the same as that of a 10 move chess game. The results for this experiment are given in Figures 3(a) and 4(a). Figure 3(a) shows




(a) Low Computation (b) Moderate Computation (c) High Computation

Figure 3: Completion time for (a) Low, (b) Medium and (c) High computationally intensive tasks.




(a) Low Computation (b) Moderate Computation (c) High Computation

Figure 4: Energy consumed by (a) Low, (b) Medium and (c) High computationally intensive tasks.

that in terms of time, gains can be made by offloading versus executing locally. We can see that the "Local Execution" line crosses with the offload time at a certain point. We can thus infer that, for tasks made up of data greater than this point, it is a better idea to execute such tasks locally since these tasks would consume lesser time being executed locally versus being executed in parallel using task offload. Specifically, we find it efficient in terms of time to offload tasks of less than 8 MB using Bluetooth, WiFi, or Cloudlets, and in this way it is possible to achieve up to 50% gain in time by offloading to one device.

In terms of energy, Figure 4(a) shows less of a gain to be made in terms of energy. As we can see, the horizontal line only intersects with one of the communication technologies (Bluetooth), and the potential gain that can be made in terms of energy by offloading a task of such low computation is very low, and applicable only to a small data size. Specifically, for tasks less than 15 MB, energy can be conserved by offloading using Bluetooth, by up to 25%; above that value, it is more energy efficient to execute the task locally. Also, we see that up to 80% gain in energy can be achieved by using Bluetooth 4.

### 3.3.2 Case of Moderate Computation (30 MFLOP)

The second set of experiments deals with moderate computational intensity, the same as popular video games, as shown in Appendix A, Table 2. In the case of moderate sized computation, as shown in Figure 3(b), we see that because we have now increased computation, we can now save more time by offloading. In terms of time, we now see that for tasks with data sizes up to 5 MB, we see gain of up to 40% gain in time by offloading to another device for any of the different technologies and infrastructures that we have considered. In addition, for all the data sizes we have considered (up to 30 MB), except for offloading to the Cloud, time gain is seen by offloading using any of the communication technologies and infrastructures that we have considered.

In terms of energy, as seen in Figure 4(b), up to 44% gain in energy was registered by offloading using

Bluetooth. We also see that the line denoting local execution doesn't cross the line denoting Bluetooth execution, thus showing that now for all the data sizes that we have considered, in terms or energy, gain was registered by offloading the task rather than executing it locally.

### 3.3.3 Case of High Computation (60 MFLOP)

The third case deals with high computation - the same complexity as that of an application recognizing objects from a live video feed (See Appendix A, Table 2). In the case of a high sized computation (as shown in Figure 3(c)), we observe that with the larger amount of computation, there is a greater potential for saving time by offloading than executing locally. For data sizes of of less than 5 MB, it makes sense to offload, as up to 50% time can be conserved as is the case in WiFi Ad-hoc, Bluetooth 4 and offloading to a nearby cloudlet. For data sizes greater than 5 MB, it time can still be gained by offloading, but not to the cloud, since offloading to the cloud has very high latency costs associated with it. It also needs to be noted that for all the technologies, the time it takes to complete the offload operation steadily increases as data size increases so for potentially very data-intensive tasks greater than the upper limit we considered (30 MB),time can be conserved if we execute them locally versus offloading them to the cloud.

In terms of energy (Figure 4(c)), for data sizes less than 2 MB, it makes sense to offload the computation to any communication technology or infrastructure available, and a maximum gain of 23% in energy can be observed. Above 5 MB, it still makes sense to offload but not to the cloud, because the latency associated with offloading to the cloud shrouds the potential gain that can be achieved by offloading to a MDC.

### 3.3.4 Summary of Results

Among all of the graphs in Figure 3, we can see that there is a lot of time gain in offloading to a MDC rather than offloading to a cloud. We have registered up to 80% savings in time by offloading to an MDC as opposed to offloading to the cloud. We have also registered up to 20% savings in time by offloading to a MDC as opposed to a cloudlet located nearby. Gains for offloading to a cloud are significantly lesser and apply to fewer cases of data. Similar insights can be seen about energy gains. As it can be seen across Figure 4, for computationally less intensive tasks, energy can be gained by executing them locally versus offloading, and the gains are up to 80%. However we have only registered a maximum of 50% gain in energy that can be achieved by offloading to a cloudlet, and no gain at all by offloading to a cloud. With these results we can decide, given the computational complexity of the task, and the availability of different communication technologies and infrastructures, whether to offload the task or execute locally.

## 4 MDC Experimental Platform

Researchers in mobile cloud computing resort to implementing or migrating representative resource heavy applications on mobile devices over which they evaluate new architectures, task scheduling algorithms, or different offloading techniques. Since appropriate, flexible, and open source mobile applications are not easily accessible, this approach is time consuming and takes the focus away from the main research contributions. Even with the effort exerted in integrating research contributions with representative applications, results are coarse grained, potentially application dependent, and take away the ability of evaluating future applications that might not exist yet.

Based on this observation, we believe there is a need for a generic flexible platform that can be utilized by researchers to freely test mobile cloud computing resource sharing and offloading solutions. This tool should decouple two main components that characterize any mobile application: the amount of data as well as the computational load that any task or job will require. These two components of the application should also be easily broken down into distributable sub-tasks that researchers can control in real-time. Similar to simulations, this flexibility in the generic platform allows researchers to test their solutions over a fine-grained range of parameters that can represent a wider spectrum of current and future applications.


Figure 5: Snapshot of Energy Test Bed

We introduce our mobile device cloud (MDC) experimental platform for mobile cloud computing research. Our platform consists of (1) an energy test bed, (2) an API for mobile-to-mobile task offloading, and (3) an Android application built using the API to carry out experimentation on real world MDC's.

The platform makes way for further research in the context of MDC's. It can be used, for instance, to test various offloading strategies, and identifying the best solutions for saving on time and energy in the context of real world MDC's. One particular work uses the platform to maximize the lifetime of an MDC by sharing of tasks among the network connected devices [MFHA13], where lifetime is defined as the time it takes for at least one of the devices of an MDC to deplete its battery.

## 4.1 Energy Test Bed

In order to calculate the energy consumed by different operations carried out by a device in a mobile device cloud, we need to know the power taken for each of the different operations that the device performs. We create an energy test bed to be able to make these measurements. We create this by taking out the battery of the device being tested and soldering wires coming from a power supply into the battery contacts of the device. The power supply that we use comes with a built in ammeter and voltmeter. We then provide a constant voltage according to the manufacturer specifications and power the device on. Using the current and voltage readings from the ammeter and voltmeter respectively, we are able to determine the power being consumed by the phone at any instance. Figure 5 shows a Samsung Galaxy SII device connected to the power supply. In the particular scenario shown in this figure, the device is consuming current of 3.6 A and Voltage of 0.14 V, and this means that the instantaneous power being consumed by the device is 0.504 W.

We carry out different tasks on the device and measure the power being consumed for each of these tasks. All of these tasks are carried out for a minute each to account for system load fluctuations, and for each of the tasks, a base reading is taken before performing the task itself. Thus the power being consumed by the specific task can be calculated by subtracting the base value from the total power being consumed while the task is being performed. We carry out such experiments for all of the communication technologies we have identified above, apart from Bluetooth 4.0 since support for it is not currently available on Android phones. The testing is carried out on Samsung Galaxy SII and SIII phones.

Figure 6 compares different energy measurements while performing wireless transfers using Bluetooth

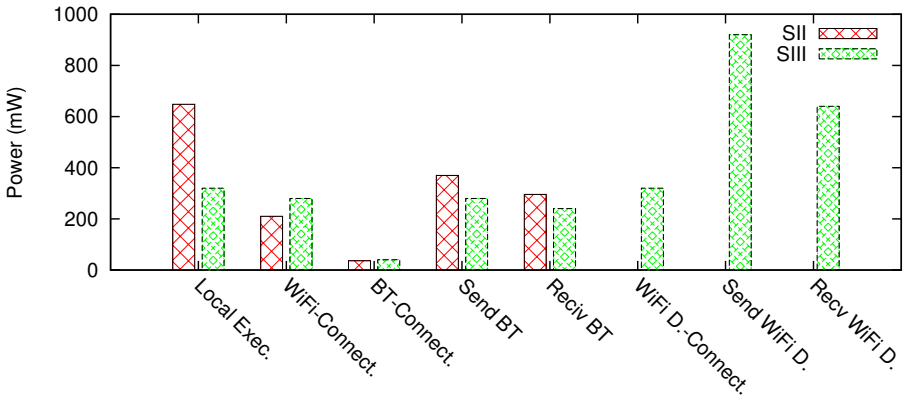



Figure 6: Results from Energy Testbed Experimentation

(BT) and WiFi Direct (WiFi D.) between two Samsung SII devices and two Samsung SIII devices. We send the same data size using both Bluetooth and WiFi Direct, and show that Bluetooth is 80% to 120% more energy efficient than WiFi Direct. Moreover, we notice that sending data costs 10% to 25% more energy than receiving data independently of the wireless communication used. This plot confirms the fact that WiFi Direct is an energy expensive technology; in fact, SIII with WiFi Direct radio on and connected to another SIII, consumes almost the same energy than SIII sending via Bluetooth to another SIII device. We note that the Samsung SII does not implement WiFi Direct, so we plot only the SIII measurements in Figure 6.

## 4.2 API

For researchers to carry out experimentation on MDC's, a set of functionalities needs to be provided so that they can test different offloading strategies. We develop an API that allows task definition and sending and receiving of tasks using different communication technologies. We implement the API using Java and the Android framework. Below we have defined the different features of this API.

- `PreOffloader` - Allows a user to specify a task as a composition of data (in MB) and computation (in MFLOP). It supports user data entry using the phone's interface, or creation of multiple tasks dynamically.
- `LocalExecutor` - Runs as a service in a background process. It takes in a task and executes it on the offloader device itself, before returning the task to the `TaskOffloader` class.
- `BluetoothOffloader` - Takes as input a task and a percentage of how much of that task needs to be offloaded. It divides the percentage of the task to be offloaded amongst devices paired with the offloader devic. Following this, it sends the offloadee portions of the task via bluetooth, and then executes the tasks on the offloadee devices and itself in parallel.
- `WiFiDirectOffloader` - Works in a similar fashion as the `BluetoothOffloader` , except that it uses WiFi Direct to offload the tasks. Execution is also carried out in parallel.
- `RemoteOffloader` - The RemoteOffloader sends a task to a given IP address. We used this functionality to offload tasks to a Cloudlet or a Cloud based device.

**Implementation Details** - The API executes the fraction of the task that needs to be executed locally in a separate thread in the background using Android's `IntentService` facility, and once the local execution

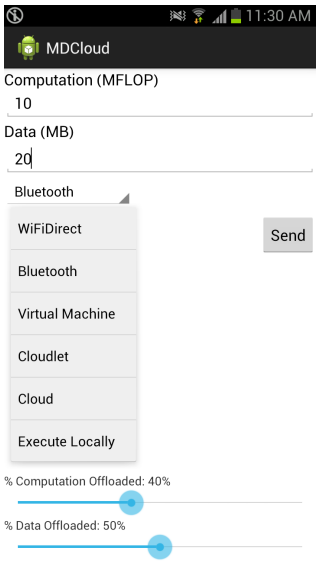



Figure 7: Screenshot of MDCloud Application

of the computation is completed, the time taken to run this computation is written to a file. Parallel to this execution, another thread divides the computation and data into as many parts as offloadee devices in the mobile cloud, creates as many threads as these devices, and uses each of these threads to offload to each offloadee device its share of the task. The sending and receiving of tasks for each communication technology is carried out by using `ServerSocket` and `Socket` on top the relevant API for each of the technologies provided by Android.

Since the devices being used in our mobile device cloud run on multi core processors, we want to leverage that in order to make sure phones with more parallel processing capabilities executed the code in more efficient ways so that offloadee computation time can be reduced. This would help us test the scenario where a device with less computational capability offloads to a set of more powerful devices. In order to achieve maximum performance gain when executing computation in parallel, we uses `AsychTasks`, a functionality provided by Android through which we set high priority to the threads carrying out the computations. This optimization, however, does not show much reduction in computational time, but this is the maximum gain we obtain given the options that Android's 4.1 version provides developers with.

**Usage** - So far we have implemented this API on the Android framework. This application can be installed on any number of devices in a mobile device cloud. It can be used to set up a mobile device cloud where one device is an offloader and the rest of the devices are offloadees. The application's interface can then be used to specify how much data and computation needs to processed, and what percentage of these should be offloaded and what percentage should be executed locally. The interface also allows choosing the communication technology that should be used in order to offload the task and receive the result.

## 4.3 MDCloud Application

We implement an application called MDCloud that uses the API described above, and runs on Android 4.1. The interface of the application allows a user to create a task by specifying the amount of computation and data (in MFLOP and MB, respectively). It also allows the user to select the communication technology to be used to offload the task from among a list of choices given in a drop down menu. This list also contains 'Cloud' and 'Cloudlet' options using which the user can offload to one of those infrastructures hosted at

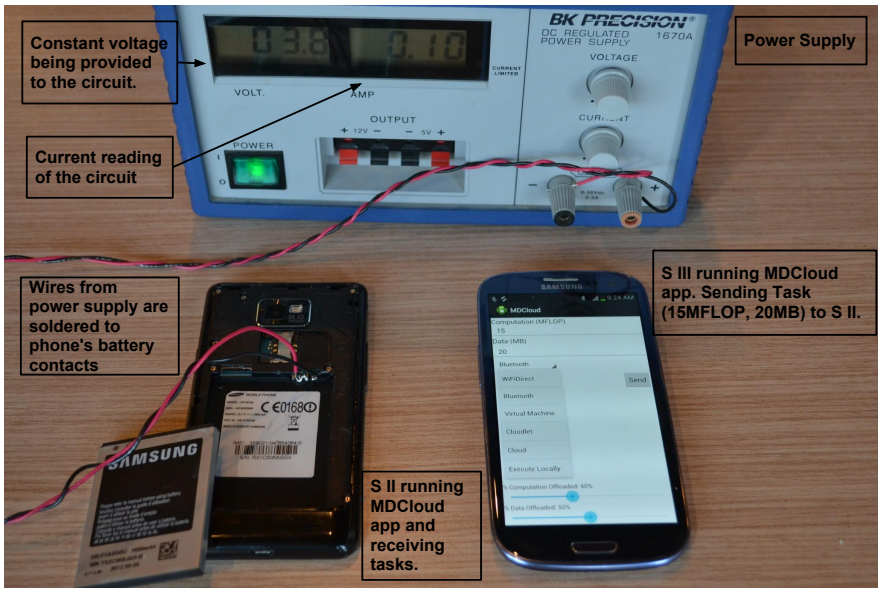



Figure 8: MDC Testbed - A Scenario

specified IP addresses. In addition, the interface also provides two seek-bars that allow the user to specify what percentage of computation and data should be offloaded, and the remaining percentage is executed locally. Once the task has been offloaded, the application keeps track of the time it takes to get the result from all the offloadees, as well as the time it takes for each offloadee to complete it's assigned computation.

This platform allows users to generate tasks with different computational loads (measured in total floating point operations and denoted in MFLOP) and relevant data input (measured and denoted in MB). It also provides APIs that enable building more specialized applications that can offload sub-tasks, set by the user, using various wireless technologies, such as WiFi, Bluetooth, or WiFi Direct. The MDC platform enables the application to log the total response time for each task (*i.e.,* the time when the task was initiated to the time when the results are sent back to the initiating user). It also logs the computational time for every device as well as the data transfer time separately.

The application allows the user to select the number of connected devices from a pool of devices within its proximity, as well as the amount of data and computational load to be offloaded to each connected device. When the user executes a task generation and offloading scenario by pressing the send button, the original task is, therefore, fragmented and the selected percentages are offloaded to remote devices as specified by the user. The remaining sub-tasks are executed locally. We install the application on multiple phones and use the application of offload tasks from one phone to multiple offloadees. More details on the scenarios under which we test the MDCloud application can be seen in the next section.

A screen shot of one of the uses of the application can be seen in Figure 7. In the particular scenario pictured in this figure, the application is set up to offload 10 MFLOP of computation and 20 MB of data using Bluetooth, where 40% of the computation and 50% of the data would be offloaded to the remote paired devices while the rest would be executed locally.

# 5 The MDC Testbed

We create an experimentation testbed which uses the MDC platform to gain insights into the energy and time tradeoffs when offloading in real world MDC's. Our experimental testbed consists of two Samsung

Galaxy SII and two Samsung Galaxy SIII phones, all running the MDCloud application we have described in the previous section. A snapshot describing the testbed can be seen in Figure 8. In the particular scenario pictured in the figure, a MDC consisting of two devices has been set up, where one of the devices (the SIII) is offloading a task to another device (the SII), and the power being consumed by SII is being measured.

Using the MDCloud Application, we attempt to get practical insights useful for making offloading decisions in mobile device clouds. We thus define four scenarios, and using their results, make conclusions about what strategy would be good for developing an offloading algorithm. In all of these scenarios we chose Bluetooth as the standard communication technology, because of its widespread availability in almost all modern smart devices.

## 5.1 Corroborating Emulation

In this scenario, we consider the case of two devices in a mobile device cloud, where one device is the offloader and the other is an offloadee. We consider moderate and high computation (30 and 60 MFLOP, respectively), and we vary data between 0 and 20 MB. In all cases, we divide the task into two equal parts: one for offloading and the other for executing locally. These experiments are of similar nature to the ones carried out using the emulation testbed as outlined in Section III. The idea is to corroborate the findings of the emulated testbed to see if similar results are achieved, to be able to justify that our emulated testbed mimicked real world MDC's testbed correctly.

### 5.1.1 Case of Moderate Computation (30 MFLOP)

The results for this experiment can be seen in Figures 9(c) and 9(a), where we have plotted data on the horizontal axis and time and energy on the vertical axes. As can be seen, for tasks lesser than 3 MB, up to 40% gain in time and up to 20% gain in energy can be achieved by offloading half of the task to another device in the MDC. However, since the computation is negligible, we see that in Figure 9(a), the lines denoting time consumed by offloading crosses the line denoting time taken by local computation earlier in the graph, showing that only for a small subset of the experiment does it make sense in terms of time conservation to offload versus executing locally. Similarly in the case of energy (Figure 9(a)), the data component of a task that can actually show a conservation in energy has a very small upper limit.

### 5.1.2 Case of High Computation (60 MFLOP)

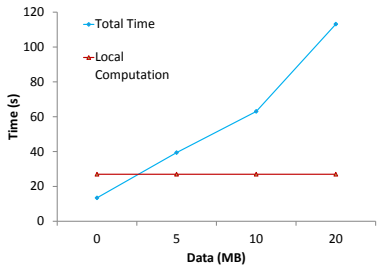
The results for this experiment can be seen in Figures 9(d) and 9(b) where we have plotted data on the horizontal axis and time and energy on the vertical axes. For a higher MFLOP value, we see a larger gain in both energy and time conservation that we can achieve by offloading the task to another device. We see that we can gain up to 50% gain in time and 26% gain in energy by offloading half of the task to another device.

As can be observed, the crossing that is seen in the graphs 9(d) and 9(b) has shifted *horizontally*. This means that since we are now considering higher computation, a larger set of tasks can show a gain in both time and energy in case we offload them to another device in an MDC, and these tasks can now have a higher upper limit for the amount of data that they are composed of.

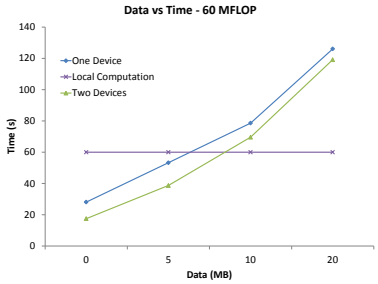
What we have seen in these two experiments corroborates what we saw in the emulation experiments we ran in Section III. We can thus confirm that in the case of real world MDC's it is definitely possible to gain time and energy by offloading to other devices.

## 5.2 Offloading to Multiple Devices

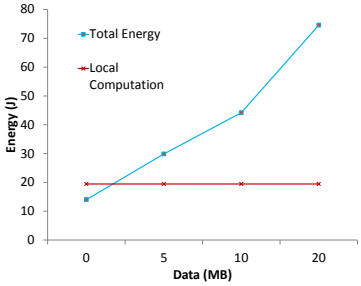
In this scenario, we consider the case of three devices in a mobile device cloud, where one device is the offloader and the remaining two are offloadees. We consider a task of fixed computation size (60 MFLOP),



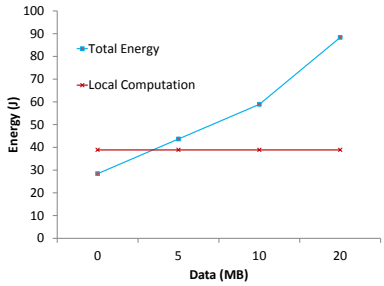

(a) Data vs Time - 30 MFLOP



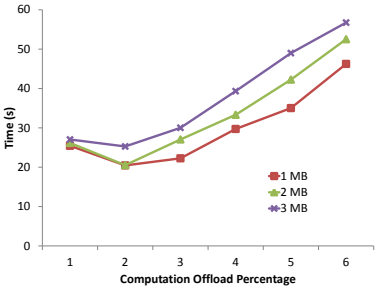

(b) Data vs Time - 60 MFLOP



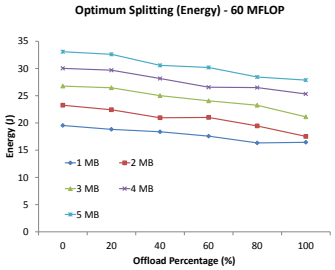

(c) Data vs Energy - 30 MFLOP



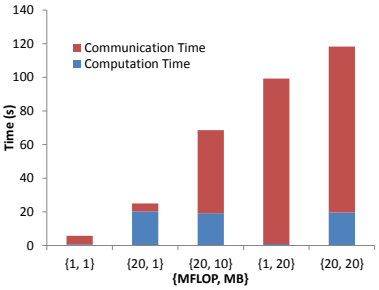

(d) Data vs Energy - 60 MFLOP




(e) Optimal Offload w.r.t Time




(f) Optimal Offload w.r.t. Energy




(g) Computation vs Communication

Figure 9: Results from MDC Testbed Experiments

and we very the data from 5 MB, 10 MB and 20 MB. We carry out two experiments. In the first experiment, we offload half the task to another device and execute the rest of the task locally. This first experiment has already been carried out in section 5.1.2. In the second experiment we divide the offloaded task equally among the offloader and two offloadee devices such that each device carries out 33% of the task. We measure the time consumed in each of these different experiments. The goal of this experiment is to determine if gain in time can be maid by offloading a task to multiple devices instead of just one.

The results can be seen in Figure 9(b). From our experiment, we can see up to 40% gain in time by offloading to two devices as compared to one. We can also see that for higher data values, the distance between the time taken to execute on one device and the time taken to execute on two devices is narrowing. This is because for higher data values, the overhead that comes from having a lot of data being sent mitigates the potential gain that can be made by distributing the computation among multiple devices.

## 5.3 Offload Distribution

We now consider the case of a mobile device cloud consisting two devices, where one is the offloader and the other is the offloadee. We offload different tasks composed of a fixed amount of computation (60 MFLOP), and we vary the data from 0 to 5 MB.We carry out five experiments, one for each data size. In each experiment, we vary the percentage of computation that is being offloaded to the offloadee device, between 0% and 100%. The goal of this experiment is to determine the optimal percentage to offload so as to maximize gain in time and energy. Determining such optimal distribution in runtime will be investigated in future work. We also calculate the energy being consumed in each of these experiments to see what is the optimum offload percentage which ensures maximum gain in time and energy.

The results for time measurements can be seen in Figure 9(e). As we can see, for lower data values (1-4 MB), there line that represents the time taken is in the shape of a curve. In all these cases, the lowest time it took to complete the offload operation is when 20% of the task was offloaded to the offloadee device and the rest was executed locally. We can also see that by varying the percentage of the task that is being offloaded versus being executed locally, we can see up to 51% gain in the time it takes to complete the offload operation.

The results for energy calculations for the above set of experiments can be seen in Figure 9(f). We can see that for all the data values that we have considered, the most gain in energy is made by offloading 100% of the task to another device. We can also see that up to 16% conservation in energy can be achieved by offloading the whole task to the offloadee device versus executing the whole task locally.

## 5.4 Computation vs. Communication

We consider the case of a mobile device cloud consisting two devices, where one is the offloader and the other is the offloadee. We offload different tasks composed of different amounts of data and computation. We define *low*, *medium* and *high* data as 1 MB, 10 MB and 20 MB respectively, and we define *low* and *high* computation as 1 MFLOP and 20 MFLOP. The goal of this experiment is to determine which factor (data or computation) consumes more energy during the offload operation.

The results of this experiment can be seen in Figure 9(g). For these common tasks that we have considered, what we observe is that communication can take up to 100 times more time than computation does. We also see that an increase in 20 MB of data inside a task can cost 4 times more time than increasing 20 MFLOP of computation in the task being considered.

This analysis shows that the bottleneck when it comes to offloading is the data that the task is composed of, and not the computation. Having more computation provides avenues for more gain to be achieved in terms of time of time and energy conservation, while having more data means a reduction in the energy and time that can be achieved by offloading tasks into an MDC.

# 6 Summary & Ongoing Work

In this work we have shown that mobile devices can be used to save both time and energy when it comes to executing computationally heavy tasks. We have shown a potential gain in both time and energy, up to 50% and 23% respectively, which can be achieved by offloading to other mobile devices in an MDC, and we have corroborated these results by carrying out experimentation on our MDC test bed. We also present different insights into the factors that affect the offloading decision by carrying out further testing on our MDC test bed.

We also provide an API that allows algorithms for offloading decisions to be tested on an actual mobile device cloud consisting on multiple devices. We have also used the MDC platform to carry out experiments that have given us insights into how to decide whether to offload a particular task or not, and what potential strategies could be used to make this decision. We have shown up to 50% gain in time and 26% gain in energy by offloading, and these results corroborate what we learned from our emulation testbed results.

In the future we would like to publish the MDC API that we have built so that other researchers in the field of mobile cloud computing can create testbeds and experiment different offloading strategies. We would also like to enahnce the MDC Application interface so that it allows us to specify different percentages of task to be sent to different devices, according to the characteristics of the devices. We would also like to carry out real world mobile cloud computing experiments by giving devices running the MDC Application to different students on campus to keep with them throughout the day and evaluating gains in time and energy by employing task offloading.

While we have looked into making offloading decisions based on the contents of the task at hand, we would also like to explore which device a task should be offloaded to given information about the devices existing inside the MDC. We plan on leveraging social context and contact history of the device holder and the device itself respectively to determine which devices are most likely to respond the fastest and thus result in the most energy efficient offloading choice. For each of the different strategies used to determine which device a task should be offloaded to, we would then run simulations against actual contact history data sets to discover which of the strategies are most energy/time efficient in real world situations.

# References


[bt413] Bluetooth low energy, 2013.

[CBkC+10] Eduardo Cuervo, Aruna Balasubramanian, Dae ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *MobiSys'10*, pages 49–62, 2010.

[CIM+11] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[CS03] Mark Carson and Darrin Santay. Nist net: a linux-based network emulation tool. *SIGCOMM Comput. Commun. Rev.*, 33(3):111–126, July 2003.

[DLP03] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003.

[FKK11] R. Friedman, A. Kogan, and Y. Krivolapov. On power and throughput tradeoffs of wifi and bluetooth in smartphones. In *INFOCOM, 2011 Proceedings IEEE*, pages 900–908, 2011.

[Fli12] Jason Flinn. Cyber foraging: Bridging mobile and cloud computing. *Synthesis Lectures on Mobile and Pervasive Computing*, 7(2):1–103, 2012.

[goo13] Techcrunch google glass api article, 2013.

[Jai91] Raj Jain. The art of computer system performance analysis: techniques for experimental design, measurement, simulation and modeling. *New York: John Willey*, 1991.

[MFHA13] Abderrahmen Mtibaa, Afnan Fahim, Khaled Harras, and Mostafa Ammar. Towards resource sharing in mobile device clouds: Power balancing across mobile devices. In *Proceedings of the second edition of the MCC workshop on Mobile cloud computing*, MCC '13, New York, NY, USA, 2013. ACM.

[nis13] Nist net home page, 2013.

[Riz97] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM Comput. Commun. Rev.*, 27(1):31–41, January 1997.

[SAZN12] Cong Shi, Mostafa H. Ammar, Ellen W. Zegura, and Mayur Naik. Computing in cirrus clouds: the challenge of intermittent connectivity. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, MCC '12, pages 23–28, New York, NY, USA, 2012. ACM.

[SBCD09] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

[SLAZ12] Cong Shi, Vasileios Lakafosis, Mostafa H. Ammar, and Ellen W. Zegura. Serendipity: enabling remote computing among intermittently connected mobile devices. In *MobiHoc*, pages 145–154, 2012.

[tag13] Microsoft tag survey, 2013.

# Appendices

## A Mapping Everyday Applications to MFLOPs

In order to figure out what MFLOP values can be considered as "low", "medium" and "high", we need a method to be able to measure the MFLOP values that are used by common applications that we use in our every day lives. We have outlined this method below.

### A.1 Method to calculate MFLOPs of an application

Our mobile testbed is implemented on the Android platform. We use a benchmarking application called Linpack [DLP03] which stress tests the CPU and provides the number of MFLOP that the CPU is able to handle. While this benchmarking operation is running, we note down the CPU load - a functionality provided by Android Developer Tools. We calculate the "MFLOP per unit load" by dividing the total MFLOP consumed in the benchmarking process by the load. Using this value, we are then able to run any arbitrary application and measure how many MFLOPs are being consumed by an application by multiplying the CPU load with the MFLOP per load unit.

### A.2 Mappings of Applications to MFLOPs

We used the above method to measure MFLOP values of most common every applications on the Android platform. For each of the three categories we are considering, we identify one application that corresponds to that category. We have identified a chess game called Chess for Android low compute intensive application. For this game, we measure MFLOP when the application is making a move in high difficulty mode. For a moderate sized computation application, we identify a common video game called Angry Birds Space. For the category of high computation we identify an object recognition application called Google Goggles (running in continuous mode, where the application identifies objects in real time from a live video input). We install these applications on a Samsung Galaxy SIII device. We use the above method to measure the MFLOP values being consumed be each of these applications. We carry out each experiment three times and take an average of the computed MFLOP value. The results for these experiments can be seen in Table 2.

Table 2: Mapping Tasks to MFLOP

| Application | MFLOP |
|---|---|
| Chess Game | 10 |
| Video Game | 30 |
| Object Recognition in Video Feed | 60 |