CASES: Context Adaptation for Energy Savings in Mobile Applications

Patrick Ferreira[1] **and Lucas Wanner**[1]

[1]University of Campinas, Brazil

ra175480@fee.unicamp.br, lucas@ic.unicamp.br

Abstract. *In this article, we present CASES (Context-Adaptive System for Energy Savings), a framework for the Android system capable of managing device resources in a context-sensitive way, in order to extend the battery life of smartphones. For this, CASES will perform the scheduling of tasks according to predefined criteria, such as the user's usage pattern, current location and other variables of the device environment. CASES will determine the optimal plan and pass it on to the other system services in the form of instructions determining how and at what rate they should be executed. Analyzing complete cycles of battery discharge (from 100% to 0%), it is verified that the framework, when in automatic optimization mode, presents an expressive energy saving, which on average is close to 33%. In a dynamic scenario in which the user can partially recharge the device throughout the day, the battery life prolongation reaches the equivalent of 91% savings.*

Resumo. *Neste artigo, apresentamos CASES (Context-Adaptive System for Energy Savings), um* framework *para o sistema Android capaz de gerenciar recursos do dispositivo de maneira sensível ao contexto, a fim de estender a duração da bateria dos* smartphones*. Para tanto, CASES irá desempenhar o escalonamento de tarefas de acordo com critérios predefinidos, tais como o padrão de uso do dispositivo, localização atual e outras variáveis do ambiente. CASES determinará o plano de ação ótimo e o encaminhará para os demais serviços do sistema na forma de instruções, determinando como e com que frequência os mesmos devem ser executados. Analisando ciclos completos de descarga da bateria, foi verificado que o framework, quando em modo de otimização automática, apresenta uma expressiva economia de energia, a qual, em média, é próxima de 33%. Em um cenário dinâmico onde o usuário pode recarregar parcialmente o dispositivo ao longo do dia, o prolongamento de duração da bateria alcança o equivalente a 91% de economia.*

1. Introduction

Mobile and wearable computing devices, such as smartphones, are rapidly evolving from being used for sporadic communication to being used for sensing, analyzing, disseminating, and continuously presenting various information.

New applications such as health monitoring, fitness, and augmented reality typically have high power consumption, and embedded mobile computing devices are already facing a growing energy gap, where device power is growing faster than the energy storage density. Despite the reduction of active energy base consumption with new chip manufacturing processes, in the new Dennard silicon sizing regime [Taylor 2013], the increase of the logical density and of memory to each generation contributes with increased

complexity and power. In this way, battery life has become one of the main complaints of smartphone users, as well as one of the main criteria in the choice of new devices [Ferreira et al. 2011, Kelly 2014]. Advanced mobile applications also give us, however, ample opportunity for optimization and adaptation of different quality of service metrics. This translates into a broad band of adaptation between quality of service and energy consumption that can be adjusted to, for example, maintain a minimum battery life according to user needs and the context of use.

A task scheduler aware of energy consumption and energy charging patterns could, for example, fine-tune the duty cycle of sensing tasks and postpone non-urgent tasks (such as system updates and backups) or migrate tasks between different cores in a multicore system with heterogeneous processors. When the battery charging potential is not reaching its maximum potential (for example, when using a low power charger), the scheduler could seek to reduce power consumption until the battery saturates its charge, in order to reduce the total charging time of the system. If more power is available than necessary (for example, when the battery is fully charged and the charger is still connected to the power), the scheduler could react, increasing system performance. The system scheduler would become not only a task scheduler but also a resource scheduler that adapts to the context of use.

In this article, we present CASES (Context-Adaptive System for Energy Savings), a framework for the Android system capable of managing device resources in a context-sensitive way, in order to extend the battery life of smartphones. For this, CASES will perform the scheduling of tasks according to predefined criteria, such as the user's usage pattern, current location and other variables of the device environment. The energy saving plan varies according to the context changes in the device. CASES will determine this plan and will pass to the other services of the system instructions determining how and at what rate they should be executed. The analysis of complete cycles of battery discharge (from 100% to 0%) shows that the framework, when in automatic optimization mode (without considering the device's location), has an expressive energy saving, which on average is close to 33% (comparing the mode of maximum and minimum processing in the device). When considering device location, the framework reaches battery life prolongation equivalent to 91% savings.

2. Related Work

We list in this section works that seek to optimize energy consumption based on task scheduling, reduction of processing and data exchange, and adaptation to the context of the device. The principle of action of each of them served as a foundation for the formation of the functions which CASES framework is endowed.

In [Zhao et al. 2011], the authors attempt to forecast the remaining battery life from a context-sensitive system. The method relies on multiple linear regressions of battery usage to identify the current usage pattern and estimate how much battery time remains. The model predicts with reasonable accuracy the battery life in predefined contexts where only specific components are used.

The work in [Martins et al. 2015] aims to optimize battery life through management and interventions made in services running in the background for well-known and/or common cases on most devices. The system that will perform the actions of this project is named Tamer. It will perform its action by blocking or limiting the time and frequency

of execution for services that are consuming too much energy.

Another way to save energy in costly processes, for example in streaming video in [Trestian et al. 2012], is to get content in lower qualities than the original. In this research, the authors seek to clarify the ways of performing this type of service in which less energy is spent. Different scenarios of network signal strength, number of connected users and different streaming qualities for the same video are explored, in addition to repeating the tests in different data exchanges protocols.

Therefore, the improvement proposed by this article is assembling a framework capable of managing system tasks in the way to save energy using those previous methods. But, differently from the other works, the framework will attempt to find the best way to implement those strategies at the same time and for more general kinds of tasks, not only for those ones that use only the WiFi, the processor or any hardware else separately, allowing an easier implementation on any Android device and requiring fewer modifications for any existing application.

The last related work to be cited is not another research, but an energy manager native from Android Marshmallow and it's going to deal with CASES during its perform, it's called Doze. This is a resource used by Android to save energy and, even disabling all possible configuration about its interventions, it'll keep making some actions trying to save energy over other applications running, and it includes CASES. The authors disabled all configurations that prevent CASES from managing energy of the system, what mitigates a lot Doze's effects over our framework, but sometimes it will affect the sampling rate of CASES as it's going to be shown in graphics ahead. Nevertheless, it's going to be clear that Doze didn't make the system sleep much more than it was intended, not affecting significantly our sampling.

3. System Architecture

The main objective of CASES is saving energy from mobile phones through the tasks scheduling and the use of applications methods that perform equivalent actions under different energy costs, without any significant performance loss for the device. To do so, the system uses a methodology based on identifying the current context, determining which mode of saving is the most appropriate and informing the other services about the execution instructions.

The first key point for running the system is the Data Collection Service. It is responsible for periodically taking samples from certain smartphone sensors and storing them for later use. Among the information collected by this service are: Battery level, the current condition of the same (charging, discharging, USB connected, etc.), the last moment the cell phone was charging and battery level in the same, user's GPS location and moment of this sample. The service repeats at certain time intervals, varying at a sampling rate from every 30 minutes until it can be suspended indefinitely, all at the discretion of the current energy saving features and plan, which is defined by system manager.

The Manager is the second key point for system execution, a service being able to identify user's usage pattern of the device and determining which saving plan is the least restrictive possible and is able to extend the remaining battery for long enough. Manager's decision making regarding the current mode of action is based on three aspects:

- **Remaining Time Prediction**: A linear regression is performed on data from the Data Collection Service on battery level at each time interval. In this way, it is

possible to obtain a forecast of how long the battery will last, based on the standard use of the device;
- **Device's Location**: It is assumed that when the user is at home, for example, it is easier for him to connect the device to the charger. Therefore, you can allow the system to consume a little more resources, since the smartphone will probably be recharged soon;
- **User Command**: The Manager's decision will be made according to the user's preference. If the user has selected the constant savings mode, for example, even if there are surplus resources, the manager will continue to restrict the consumption of the system. However, if the preference was for the full performance mode, the device continues to consume resources normally until the end of battery's load.

4. Methods

The model put into practice administered the consumption of an application's service itself, whose sole purpose was to download a predetermined data package from the Internet. This service downloads this content recurrently and we would measure the savings ratio obtained when executed with or without CASES intervention.

4.1. Download Service

The Download Service resorted to a given time interval. The interval size was obtained by instructions from the Management Service, as described below. For convenience, we chose to download an audio content, given the ease of making it available in different recording qualities. We could then use this property to adjust how much energy would be spent on each download run, since the higher the file quality, the larger the media size and thus the energy expenditure [Trestian et al. 2012]. Whenever the Download Service was run, it would instantiate an AsyncTask class that would make a content request to the server, display a progress notification of the download and saves it in the device memory itself. The quality of downloaded content was also configured by system management according to available resources. This service had a short piece of code dedicated to identifying the instruction left by the system manager. This information was saved to a public archive in the device's memory, in which there were instructions on when the Download Service should be run again and what download quality should be required.

4.2. System Functions Distribution

The construction of CASES took place from the task assignment to different services and classes. Classes were constructed to obtain readings of all sensors and device alerts - such as GPS and battery -, identification of current connectivity status (WiFi, 3g, disconnected, etc.), linear regression and communication with servers - asynchronous tasks which run independently of their respective services. The actions of the system are taken based on the execution of the services and calls of broadcast requested, both, by the application itself. To do so, we use the above-mentioned classes in order to receive all the necessary parameters about the context and being able to respond with the appropriate measures.

4.2.1. Data Collection Service

This recurring service is responsible for periodically carrying out readings of the relevant sensors on the mobile phone and storing them for later use. The service collects battery

information - level, voltage, storage condition, current charging status-, GPS position and local time. With this information, it is possible to know when and where the user most demands the battery and then make predictions, through linear regressions, of how long a certain level of charge will last.

4.2.2. Management Service

The Management Service assumes the role of system manager. Through data collected by the Data Collection Service, it will trace the user profile and will make their intervention decisions according to device's current situation and the pattern of use by the user. The manager has its operation mode defined by the user in the application interface. If the user opts for performance mode or constant economy, the services will always execute at their maximum or minimum capacity, respectively, regardless of context. When the user chooses the automatic saving mode, the manager acts to adapt to the context continuously. The device will autonomously switch between performance and economy modes as energy resources change over time.

This manager will evaluate if, with the current battery level and the standard use of the device, the tendency is to arrive at the end of the day with or without remaining charge. In the first case, the manager allows system services to operate at their maximum capacity. For the second, it will impose restrictions on other services operation. The restrictions applied by the manager to other services are divided into two types: postponement of their next execution and reduction of the quality of the tasks - in the case of services that download media, for example.

When the Download Service operates in maximum performance mode, the content is downloaded every 5 minutes and with maximum quality. When it verifies that the command left by the manager is to save energy, the audio content is downloaded every 15 minutes and with low quality, aiming to save energy while performing an equivalent action. Even the Data Collection Service is subject to the interventions of CASES system manager. In performance mode, samples are taken every 30 minutes, while, in economy mode, every 60 minutes. Because activating certain sensors of the cell phone - GPS, for example - requires considerable energy consumption, so it is an action that can save energy when subject to the scheduling of tasks.

4.3. Context Adaptation

CASES will learn how the user spends energy throughout the battery samples collected by the Data Collection Service. Therefore, a linear regression will be traced once we have more than 1 sample collected, allowing us to estimate how much energy by time the device will spend if it keeps on following the same pattern of use. Thus, if the battery level is below the line traced on this regression, it means CASES has to save energy and it will operate in Economy Mode. If the level is above the line, it will operate in Performance Mode, cause we have more energy than usual. Note that the user can change his way of using the smartphone, spending more or less energy, and the framework will automatically adapt to this, according to the samples being collected, which will change the slope of the regression line. Very old samples (3 days old or more) are discarded, then we know how the user uses the device currently, because the regression gives us the standard use of the device. Naturally, CASES will take into consideration other circumstances, such as

device's current place, but it is going to be explained ahead in the algorithm of Automatic Mode (the mode where CASES is context aware).

5. Results

We implemented CASES system in a Nexus 5 smartphone with Android 6.0 as operating system. So, to measure the results of the system execution, several battery discharge cycles were performed from 100% to 0%, analyzing how it behaved under different scenarios. We disabled the system management (Doze) over CASES framework in the "developer mode", intending to be free from system interventions (in the end of the paper it's going to be shown that the Android system remains making interventions, although less often than before).

An interface created for the application was used, from which it was possible to change the way that system was executed across buttons (Figure 1). Those buttons affect the framework execution form, changing it through different operating modes, setting specific locations (like *home* position) and handling services manually.

- **Start Service**: Forces the execution of all the recurring framework services (Download, Data Collection and Management);
- **Stop Service**: Forces the stop of all the recurring framework services (Download, Data Collection and Management);
- **Set Home**: The current location is defined as the user's home position. This feature is useful for identifying when the user is close to possible recharge sources (within their home) and we can consume a little more of energy;
- **Economy Mode**: This mode causes the System Manager to order the other services to be executed at long time intervals, as well as performing content downloads under a lower quality rate, minimizing the hardware active time;
- **Performance Mode**: In this mode, the Manager allows applications to run at as high rate as they wish and perform downloads under maximum quality;
- **Automatic Mode**: This is the context-adaptive mode of energy saving. The Management Service distributes commands to the others according to the amount of current resources: Under resources excess, performance prevails, whereas, in case of scarcity, the economy mode is applied. Figure 2 shows it's simplified diagram;
- **Off Mode**: In this mode, only the Data Collection Service runs, in order to obtain the telephone energy expenditure data before it begins to perform download tasks and the like. This mode intends to collect info about how long the battery lasts to only keep the cell phone on.

5.1. Collected data

The first measure was how much the battery was able to keep the device performing no activities from CASES (just letting the battery discharge by keeping the cell phone on), whose purpose was to identify the device's default discharge. We selected the Off Mode, in which only Data Collection was executed, in order to draw device's default discharge, whose sampling was approximately one point every 30 minutes. After collecting the data, repeating this cycle a few times, and grouping the samples, the graph shown in Figure 3 was assembled. Note that, for this scenario, the battery lasts about 26 hours.

The second step was then to acquire samples of the device discharge curve when the energy device's consumption was maximum. To do so, we select *Performance Mode*

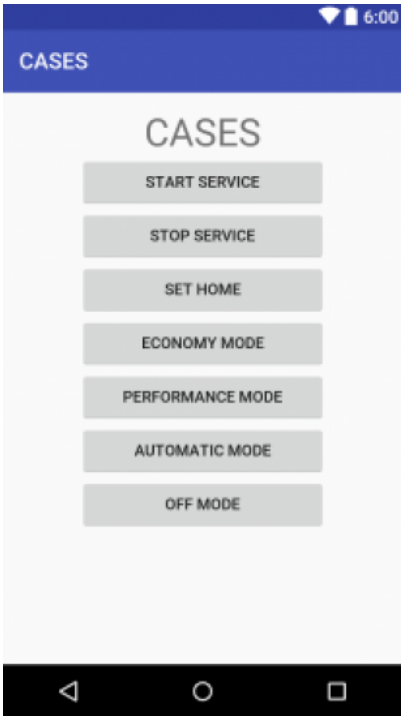

Figure 1. Interface of CASES framework.

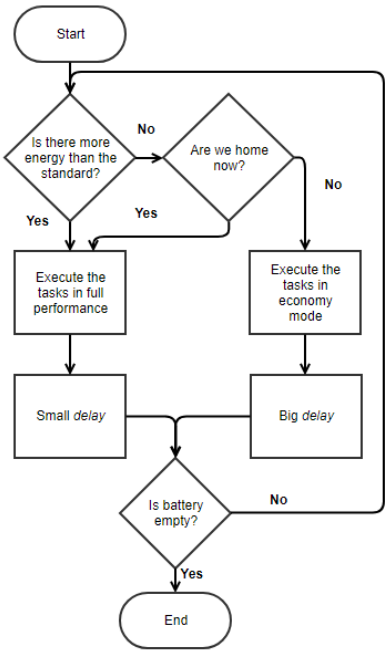

Figure 2. Operating diagram in Automatic Mode.

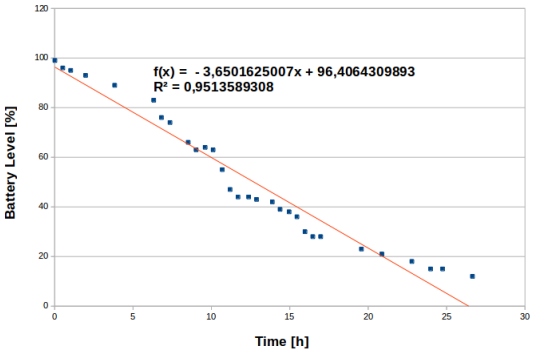

Figure 3. Device's standard discharge samples.

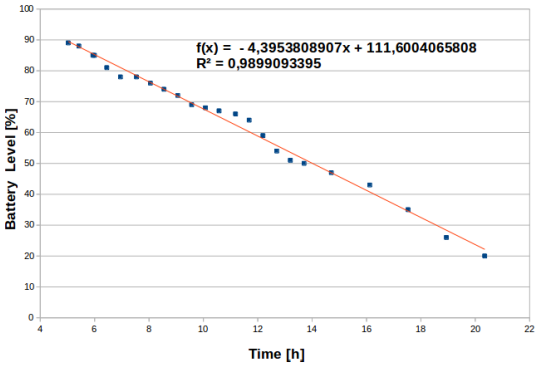

Figure 4. Samples of device discharge in Performance Mode.

and let the smartphone successively downloading the content on its highest quality in 5 minutes intervals. Time which was, certainly, sufficient to complete the download of the previously designated file on the network used in this process. The discharge cycle was repeated a few times and the samples were also pooled, resulting in the graph shown in Figure 4. For this scenario, the battery lasted about 20 hours before the device shut down.

Now it's necessary to pay attention to the way the savings are measured and compared, because it'll be based on the economy during tasks execution. The energy spent to keep the cell phone powered is approximately constant, it means the phone won't last for more than 26 hours even if we execute nothing on it. We can think the energy consumption as a negative charge. This way, we can consider the battery discharge duration as a sum of charge enough to support the time the cell phone stands without performing any task plus the charge spent on the tasks (which is a "negative charge"). Then, it's clear that we can apply the superposition principle and, supposing a linear level discharge (percent-

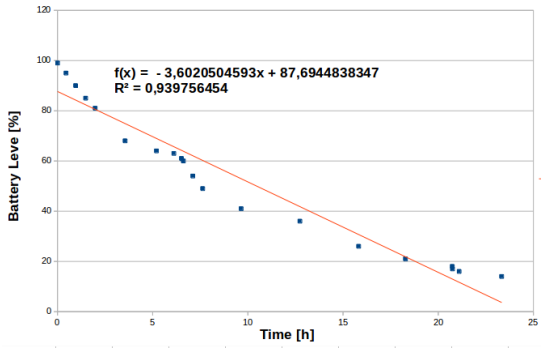

Figure 5. Samples of the device optimized discharge.

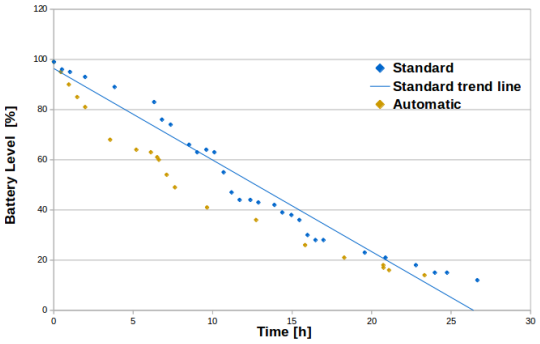

Figure 6. Comparison between *Off Mode* (device's standard) and *Automatic Mode*.

age level, not directly the voltage level), we can say that the time the battery will stand has the form "y = ax + b", where "a" is a negative constant depending on which tasks the system is executing.

It's already known that the maximum duration of the system is about 26 hours, on no task performing, and its minimum is about 20 hours when executing on full performance. It's evident that the tasks decreased -6 hours in charge duration time. The percentage results will be based on how much this time changes, because we only affect the way that those system tasks (Data Collection, Download and Management) consume energy, and the energy spent to keep the cell phone turned on is fixed, we cannot change it in the proposed work. For example: If any test below performs the same tasks but saving energy somehow the battery lasts for 23 hours, we would say it's a saving of 50%, because 3h is a half of the 6h spent originally to perform those tasks

We then proceed to the effectiveness verification scenario of the developed system. The data exposed reflects the performance of the system in *Automatic Mode*. In this mode, the System Manager checks every 3 minutes whether the battery life prospects are to last for more or less time than the default usage profile of the device. When there is more power than the standard, for that given moment of the day, the manager authorizes the Download Service to perform its tasks in maximum quality and at intervals of 5 minutes. When there is less power than usual, Manager restricts downloads to occur in reduced quality and only every 15 minutes. Samples collected under this regime are shown in the graph of Figure 5. In this last scenario, the battery kept its charge for about 23 hours - 2 hours longer than in the Performance Mode -, indicating savings of approximately 33%.

5.1.1. Comparison between standard discharge and under optimization

With the automatic mode data, we can compare them to device's standard discharge curve (obtained under *Off Mode*) in figure 6. Both run from the same load level, so the system manager assumes that there is enough load to perform high quality downloads at the highest rate (every 5 minutes), authorizing the services for such a performance. However, when the current battery level becomes lower than that obtained by the linear regression of the cell phone's standard use (this regression line can be checked in figure 5, because it's the device standard use), the manager identifies the situation and begins to save resources. The interval between downloads grows up to 15 minutes and the quality of the downloads is set to the lowest value. Only at about 15% of battery's capacity, we are able to be

with more load than the standard level, and soon thereafter, the battery exhausts and, as a result, the device turns off.

5.1.2. Influence of Location on Battery Life

Finally, we tested the efficiency of the system when we take into account the current location of the device and the probability of the user submitting it to loading - even if incomplete - throughout the day. It is assumed that when the smartphone is inside the home or the workplace of the user, the probability of loading is high, so that more resources can be spent, while this probability in other places is given as low.

Thus, after setting the home position of the user, the system was started, and the System Manager was instructed to delay the new execution of other services within 45 minutes and opt for options with lower processing costs (low quality downloads) until the user entered the home position. Upon entering this position, the Data Collection Services readings indicate to the Manager the occurrence of the event, which allows the services to run at intervals of less than 5 minutes and under maximum performance (maximum quality downloads).

With the charger plugged in, the cell phone will continue to gain load even at the highest throughput rate because of system's compensation for the time it spent under a restricted spending regime. In Figure 7, we see the graph of the resulting discharge, where, at about 20%, the recharge of the battery occurred, which raised it to approximately 30%.

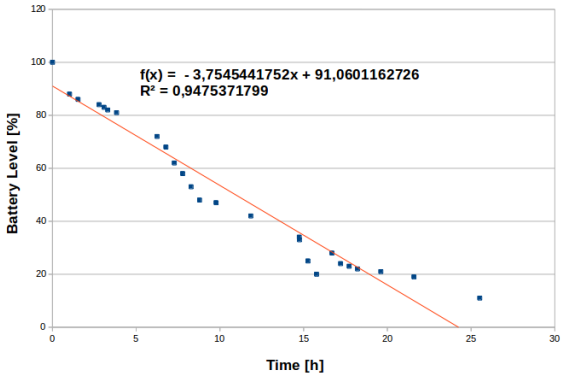

Figure 7. Discharge samples when the device recharges for 15 minutes.

Simulating that the user recharges the device for 15 minutes in the unloading process, the system was able to maintain the device for 25.5 hours, an equivalent saving of 91% (5.5h/6h = 91.7%) over the maximum performance mode. Note that the objective of allowing short recharges during the execution is simulation a real situation, where the user only has time to apply fast charges to the device. This way, CASES' objective would be making the battery last long enough until the next recharge without shutting down.

6. Conclusions

It was possible to notice that the tactic of reducing the amount of energy spent on the same task - opting for inferior quality in the downloads - and scheduling those tasks to more favorable moments - when there is more load - was able to prolong the battery life, without significant performance loss. By measuring the device's download rate by performing downloads at the highest rate and in higher quality, the battery's life time decreases by

about 6 hours, compared to the scenario where no task is performed besides keeping the device turned on for the user. However, with the use of CASES system, the same task (Download Services and Data Collection) could be performed for further 2 hours (without recharging the device), characterizing 33% extension of the battery autonomy. It was still possible to reach a 91% increase in battery life when we consider the possibility of the device receiving brief recharges during its cycle - which usually happens in practice. This shows how efficient a context-adaptive system can become.

For comparison, in [Martins et al. 2015], the authors achieved 27% savings by scheduling most energy expensive services without considering users habits. Here we got at least 33% savings by scheduling services to moments that the framework estimates the user will use most this services and by considering the current place of the device, always checking if the services are consuming too much energy, of course. This shows how the efficiency can increase with context adaptation.

It is important to emphasize that CASES not only saves energy, but makes it having a significantly small loss of device's performance, using resources when the device is not spending its maximum power and when this energy would not be so costly. That's the improvement of a context-adaptive system and why it may be beneficial to implement this in a wider range of scenarios.

One difficulty with taking measurements on the Android system is that, even though you change developer settings to the most appropriate debugging modes, Android often interferes with running the framework services because of Doze. This can become difficult to handle since the operating system may decide to severely defer service execution or even cancel it, which renders a whole measurement cycle unusable.

Acknowledgements

This work was supported by the National Council for Scientific and Technological Development (CNPq).

References

Ferreira, D., Dey, A. K., and Kostakos, V. (2011). Understanding Human-smartphone Concerns: A Study of Battery Life. In *Proceedings of the 9th International Conference on Pervasive Computing*, pages 19–33.

Kelly, G. (2014). iOS 8 hit by WiFi and battery life complaints. Forbes Margazine.

Martins, M., Cappos, J., and Fonseca, R. (2015). Selectively taming background android apps to improve battery lifetime. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '15, pages 563–575, Berkeley, CA, USA. USENIX Association.

Taylor, M. B. (2013). A Landscape of the New Dark Silicon Design Regime. *IEEE Micro*, 33(5):8–19.

Trestian, R., Moldovan, A. N., Ormond, O., and Muntean, G. M. (2012). Energy consumption analysis of video streaming to android mobile devices. In *2012 IEEE Network Operations and Management Symposium*, pages 444–452.

Zhao, X., Guo, Y., Feng, Q., and Chen, X. (2011). A system context-aware approach for battery lifetime prediction in smart phones. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 641–646. ACM.