BLAST: Battery Lifetime-constrained Adaptation with Selected Target in Mobile Devices

Pietro Mercati
Computer Science Department
UC San Diego
San Diego, CA, USA
pimercat@eng.ucsd.edu

Vinay Hanumaiah
Advanced Systems Engineering Lab
Samsung Research America
Mountain View, CA, USA
vinay@samsung.com

Jitendra Kulkarni
Advanced Systems Engineering Lab
Samsung Research America
Mountain View, CA, USA
j.kulkarni@samsung.com

Simon Bloch
Advanced Systems Engineering Lab
Samsung Research America
Mountain View, CA, USA
s.bloch@samsung.com

Tajana Rosing
Computer Science Department
UC San Diego
San Diego, CA, USA
tajana@ucsd.edu

ABSTRACT
Mobile devices today contain many power hungry subsystems and execute different applications. Standard power management is not aware of the desired battery lifetime and has no visibility into which applications are executing. However, power consumption is strongly dependent on which applications are executed. In this work, we propose a novel power characterization strategy for mobile devices called *application-dependent power states (**AP-states**)*. Based on that, we formulate a management problem to improve performance under battery lifetime constraints, and we implement the management framework on a real Android device. We call our framework **BLAST**: **B**attery **L**ifetime-constrained **A**daptation with **S**elected **T**arget. The goal of such framework is to maximize performance while letting the device battery to last at least for a certain required lifetime, and only requires the user to select the desired target lifetime. The implementation does not require OS modifications and can be ported and installed to any Android device. We experimentally verify that our strategy can still meets user experience requirements with a selected target battery lifetime extension of at least 25%.

Keywords
Mobiles, Android, Power Management, Battery, User Experience.

1. INTRODUCTION
Mobile devices such as smartphones and tablets contain a variety of power hungry subsystems (CPU, GPU, camera, display, antennas, etc.) and execute applications with different requirements: from browsing, to multimedia, to gaming and many more. Power consumption heavily depends on the application running in foreground (e.g. the one showing on the display), as it mostly determines the usage of different parts of the system. Also, it is the application which attracts user's attention, thus influencing user experience the most. Such intense activity contributes at making the battery lifetime as short as few hours for most devices [1,2]. Therefore, power management to trade battery lifetime and user experience is a primary requirement for mobiles.

In the last decade, the target of mobile designers and developers has shifted from high performance to high user-experience. The concept of user-experience depends on a number of variables, from device specifications and performance to user personal profile and level of attention. However, we can define it as the scenario in which device behavior meets user expectations. Therefore, in the case of mobiles we can identify two main factors determining user-experience: (i) **application behavior** and (ii) **battery lifetime**. The first refers to the case in which the user is satisfied with application execution (for example, a Youtube video that reproduces smoothly or a 3D game with high frames per second). The second indicates the case in which

MOBIQUITOUS 2015, July 22-24, Coimbra, Portugal

DOI 10.4108/eai.22-7-2015.2260051

the achieved battery lifetime is as long as the one expected by the user. The two targets are contrasting, as there is a tradeoff between them: if power is optimized for providing a minimum required level of user experience, this could mean trading on battery lifetime, if the level of expectation is high (for example for a user playing 3D games). On the other hand, if the goal is to reach a predefined battery lifetime, this could penalize the behavior of some applications. This means that even if both are factors affecting user experience, either one of the two can be the constraint of power management, but not both at the same time. A comprehensive management solution requires the possibility of dynamically switching from one strategy to the other, depending on the user's main concern at the time: application behavior or battery lifetime.

Recent publications mainly address the first problem [1,3,4,20,21]. These approaches all require some description of user-experience to adapt, which is provided either by the user's configuration, or with user experience modeling. However, no unique model for user-experience depending on application behavior is widely accepted so far, they all suffer from inaccuracies due to the heterogeneity and high complexity of user profiles [16]. To the best of our knowledge, no work addresses the problem of maximizing performance while ensuring that a minimum battery lifetime for mobiles is met. Such scenario is better explained by a motivational example in the next subsection

The power management of today's mobile devices is implemented at the OS level and regards mainly CPU and GPU. These two are the most power consuming subsystems [5]. Display is also very power hungry, but it should be managed independently as it is critical to user-experience [20]. Therefore, we do not consider it in this work. Other subsystems either have no power management control (e.g. antennas) or have proprietary kernel code (e.g. modem, DSPs), which makes it difficult to modify and evaluate. For example, the Android operating system, which is based on the Linux kernel, has modules called *Frequency Governors* to implement the power management policy for the CPU and GPU [6]. The *performance* governor always sets maximum frequency, while the *powersave* governor always sets minimum frequency. Similarly, the *conservative* governor allows for low power consumption, at the cost of potential performance loss. Today's standard governor, the *ondemand*, scales frequency over time depending on CPU (or GPU) utilization. Such approach has two main limitations: (i) it is agnostic of which application is currently running on the device and (ii) it does not account for battery lifetime.

In this paper, we propose **BLAST**: **B**attery **L**ifetime-constrained **A**daptation with **S**elected **T**arget. BLAST is a novel power management framework for mobile devices, which dynamically adapts to different applications while ensuring a predefined (e.g. selected) battery lifetime. Our contributions are summarized below:

1. We formulate an application and battery lifetime-aware power management problem for mobiles.
2. We propose the concepts of *Application-dependent Power state* (***AP-state***), *battery discharging profile* and *energy tank* to determine power management decisions.
3. We develop BLAST: a lightweight, ready-to-use, high-level and portable implementation on a real Android smartphone, which does not require OS modifications and thus can be easily extended to any mobile device.
4. The proposed implementation is in the user space and it is compatible with any frequency governor in the kernel.

With a set of experiments conducted on real devices executing common Android applications we demonstrate the effectiveness of our strategy in guaranteeing the predefined battery lifetime and compare against device native power management. Also, we show that our strategy can still meets user experience requirements with a selected target battery lifetime extension of at least 25%. This claim is demonstrated by testing the framework with real users. The average rating of users is within 5% for a battery lifetime improvement of 25%. The remaining of this paper is organized as follows: Section 2 reports the related work, Section 3 describes the management problem formulation and framework implementation, and Section 4 shows our experimental results. Finally, Section 5 concludes this paper.

1.1 Motivating Example

Assume that two users X and Y are leaving work to get back home by train. They both usually use their smartphone on the way home, but while user X enjoys playing 3D videogames, user Y prefers to read emails or browse through news websites. The train takes 1 hour to bring them home, and during that period of time they absolutely want their smartphone to not run out of battery, no matter what the quality of application behavior is. Once home, they are both going to put the device into charge.

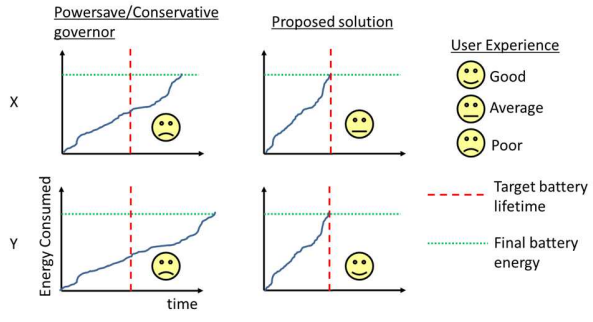


Figure 1. Illustration of motivating example

One obvious solution would be to lower the operating conditions, for example by setting the *powersave* or the *conservative* governor. However, such approach presents

three downsides. First, it requires the user to be aware of what a *governor* is and how it works, and to be able to install and use an interface application to change it. Second, the use of the *powersave* governor is likely to extend the lifetime of the battery way beyond 1 hour, depending on the initial state of charge. This is because it is not aware of battery energy. As a result, the *powersave* governor may hurt application behavior more than what is required to meet the constraint on battery lifetime. Third, the *powersave* governor is not aware of which application is executing. However, in the scenario described, user Y (mail and news) is likely to consume less energy than user X (3D games). Therefore, the performance level required to meet the same target lifetime is different for the two users.

Our solution, on the contrary, only requires the user to set the desired minimum battery lifetime (1 hour in this example), that is, the selected target. Then, the framework automatically detects the battery state of charge and the executing applications, and regulates energy consumption thanks to the *AP-states* by adapting the maximum CPU and GPU frequency. The result is that both user X and Y will have a working smartphone for the next hour after leaving work. The presented example is better shown in the qualitative plots in Figure 1, which show the energy consumed over time for user X and Y respectively when using a *powersave* or *conservative* governor and when using our proposed solution. We also highlight the user experience achieved in the four cases (either good, average or poor).

Finally, note that the target lifetime for the proposed solution should be selected in a defined range. This is better clarified by Figure 2. The lower bound is represented by the battery lifetime obtained with all cores active executing at maximum frequency (e.g. with *performance* governor), while the upper bound is given by a single core active executing at minimum frequency (e.g. with *powersave* governor).

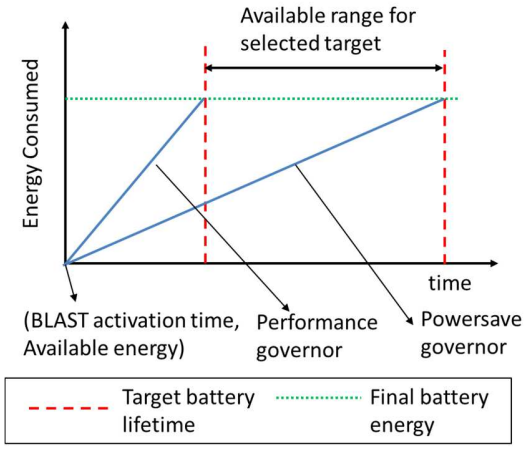


Figure 2. Range for selected target

2. RELATED WORK

Power management is an extensively investigated area of research, from server systems, to desktops and laptops, to mobiles [7]. The characterizing aspect of mobile devices with respect to other systems is the reduced form factor, which limits battery size [5]. For this reason, researchers spent many efforts in the last decade in power analysis, modeling and management for mobiles.

Publications [1] and [5] analyze phone power consumption and investigate the impact of different user activities and different applications. Work in [1] also demonstrates that CPU, GPU and screen are the most power consuming subsystems in modern smartphones. Yoon et al. propose Appscope, a tool for Android energy metering, and characterize power consumption for different applications in reference [8]. Paper [9] proposed a framework to estimate power consumption of different applications from battery power traces. These publications highlight that energy consumption for mobiles is highly influenced by different applications and that CPU and GPU are crucial in determining battery lifetime.

For this reason, power models for mobile devices have been proposed recently. Reference [10] develops a power model based on user activity for an Android-based smartphone, using regression techniques. Work in [11,12] estimates power consumption through adaptive modeling based on monitored performance activity, and integrate it in the *MPower* app, which provides the user suggestions to improve power efficiency. *MPower* collects measures on the target device and transmit them to a server for power estimation. Performing the estimation online would result in performance overhead. In general power models may not be practical for runtime power management, due to computation overhead.

Recent work on power management for mobiles focuses on user-experience determined by application behavior. For doing this, some techniques allow the user to configure personal preferences and application priority levels [13,14]. Other techniques, instead, are based on user-experience models. The strategy in [4] increases CPU frequency in response to user interaction, to minimize perceived delay. Publication [1] presents a model for user typical activity session duration, and use it to compare various power management strategies. Work in [15] proposes a scheduling algorithm for energy-based fair queuing, aiming at optimizing activity and idle periods for user comfort. Such techniques achieve better energy efficiency only if user's requirements are not too strong, as they target application behavior. However, they do not give guarantees on battery lifetime. Moreover, models for user experience may be inaccurate, due to diversity of user profiles [16]. Our technique does not require user experience models, as it targets battery lifetime. Moreover, it only requires the user to configure the desired battery lifetime. Also, the implementation of such techniques requires modifying the operating system, which may affect portability across

devices. Li et al. propose an intelligent and self-adaptive scheme for mobile power management, called SmartCap [21]. The objective of SmartCap is to automatically configure the CPU frequency subject to user experience requirements. The proposed approach is shown to significantly outperform the standard *ondemand* governor. Our work is fundamentally different as we try to maximize performance subject to a battery lifetime constraint. SmartCap, on the other hand, aims at minimizing power consumption while meeting a user experience (e.g. performance) constraint. As discussed in the introduction, these two problems are complementary to each other. Also, Smartcap focuses on CPU solely, while we include also GPU frequency control in our implementation.

Other techniques for mobile power management are developed for specific applications. Reference [17] presents a joint Dynamic Voltage and Frequency Scaling (DVFS) for CPU and GPU targeting 3D games. Work in [18] makes Youtube more energy efficient by intelligently scheduling download activities. Being specific to certain applications, such techniques cannot be extended to full phone power management. Instead, our technique is developed to be compatible with any application.

A particular case is made for display power management. As shown in reference [1], display brightness plays a fundamental role in user experience; therefore it should be managed independently from CPU, GPU and other subsystems. For example, the authors of paper [19] develop a technique to adapt voltage scaling of OLED displays to video streaming while accounting for user satisfaction. In this work we do not include display management, but we show how it can be integrated.

To the best of our knowledge, our work is the first that formulates a management problem for mobiles considering battery lifetime as a constraint rather than as an objective function, and implements a portable and lightweight framework for managing power consumption on Android devices executing real applications.

3. MANAGER FORMULATION AND IMPLEMENTATION

In this section, we first show the assumptions of our work and key observations. Based on that, we describe the concepts of *AP-states*, *battery discharging profile* and *energy tank*, and the management problem formulation. Finally, we describe the solution strategy and the framework implementation.

The target platform of our work is a battery-powered mobile handheld device equipped with DVFS-enabled CPU and GPU, controllable from the userspace. This is common in modern devices, for which the operating system exposes control capabilities at the *sysfs* interface, like setting the maximum frequency. Both CPU and GPU have predefined voltage/frequency operating points. The battery power consumption and charge level are sampled from the *sysfs* interface as well, without the need of external equipment. In this work we use the *ondemand* governor, except when clearly stated. However, note that the proposed solution is implemented in the userspace thus it is compatible with any frequency governor. This is better shown in the results section

The key for formulating and solving a management problem constrained by battery lifetime is having a model for battery power consumption. Battery power at a generic time t can be expressed as in Equation (1).

$$P_{batt}(t) = \sum_i P_i(t) \quad (1)$$

Here P_i is the power consumption of the i^{th} subsystem. Unfortunately, commodity mobile devices usually do not have per-subsystem power information available at runtime. Moreover, as mentioned in the introduction, not all subsystems can be controlled at runtime to adjust power consumption. Therefore we rewrite battery power as in Equation (2).

$$P_{batt}(t) = P_c(t) + P_{nc}(t) \quad (2)$$

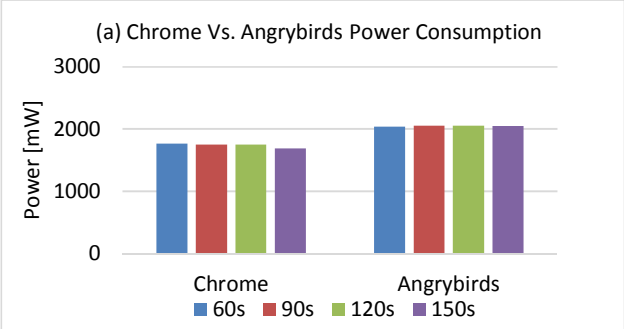
Where P_c represents the power contribution that can be controlled at runtime and P_{nc} is the contribution which cannot be controlled. In this work, we assume P_c to be a function of CPU and GPU frequency, therefore $P_c = P_c\,(t, f_{CPU}, f_{GPU})$. As for the non-controllable contribution P_{nc}, this is heavily determined by the behavior of the application running in foreground, which determines CPU and GPU load, number of active CPUs, antennas usage, task scheduling, and other factors in which we either have no control or that are already managed by other entities in the device. For example, in Android the number of active CPUs is controlled by the userspace daemon called *mpdecision*.

Deriving a model for $P_c(t)$ at each time t leads to inaccurate estimations for two reasons: (i) only battery power consumption can be monitored from userspace in commodity devices, therefore it is hard to isolate the two components P_c and P_{nc} from observations, and (ii) the sampling rate at the userspace should be at least 1 second to avoid excessive overhead. What matters is not instantaneous battery power consumption, but rather its average over time, or consumed energy, shown in Equation (3), and approximated in Equation (4). In this Equation, P_{avg} is the average battery power consumed over the time period T.

$$E_{batt} = \int_{t=t_0}^{t=t_0+T} P_{batt}(t)dt \quad (3)$$

$$E_{batt} = P_{avg} * T \quad (4)$$

The fundamental observation we make is that once the operating conditions are fixed (in our case f_{CPU} and f_{GPU}), then the average power consumption tends to stable values over time for different applications. To motivate this fact, we show a simple measurement of battery energy in Figure 3. In this experiment, we measure the battery power consumption on a Nexus 5 smartphone while executing Chrome (browsing, scrolling & zooming) and Angrybirds for 150 seconds with CPU frequency fixed at 1.5Ghz and GPU at 390Mhz. Then we calculate the average power consumption on time windows of different durations. In Figure 3(a) we report the average power consumption over a time window of different duration. We notice that the average is almost the same for a given application, but changes for different apps. Referring to Equation (2), in this case the difference between the two applications is determined by P_{nc}, as frequency is constant in both cases. In Figure 3(b) we measure the energy consumed over time. For doing this we sample battery power consumption and battery capacity and relate that to the total battery energy (derived from datasheets). We observed that the energy consumed is almost linear over time, with different slopes for different applications. This is an example of the simple fact that playing 3D games for one hour drains battery more than browsing.



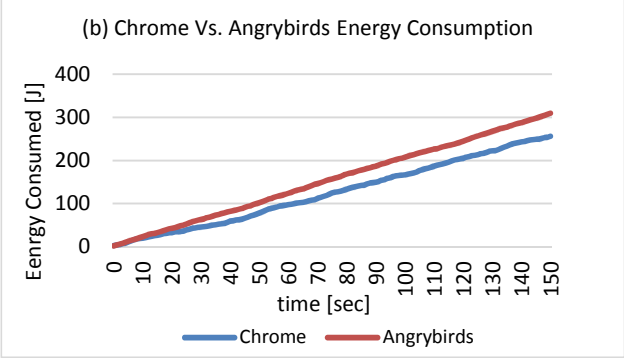


Figure 3. Power & energy consumption of Chrome & Angrybirds running on Nexus 5

Based on these two observations, we associate a value of average power consumption to each pair $f = (f_{CPU}, f_{GPU})$, independently for each application. Then, we create a table for each app such as the one shown in Table I. In this table, $P_{APP_i}(f_j)$ is called *AP-state*, and it is the average battery power consumed by application i in state j and T_{ij} is the total time spent by the device executing application i in state j. The values of P_{APP} and T are updated at runtime based on the monitored operating conditions, battery power and application executing in foreground. AP-states are stored in memory and there is at most one list of AP-states for each application.

Given this, the energy battery consumption can be rewritten as in Equation (5).

$$E_{batt} = \sum_i \sum_j P_{APP_i}(f_j) * T_{ij} \quad (5)$$

In other words, an application running on the device Contributes an average value to the total energy consumption. This depends on the execution frequency and is weighted with the time spent executing it. At this point, Equation (5) allows us to formulate the management problem.

The management problem requires the specification of a **target battery lifetime** t_{charge} and of an **energy budget** E_{budget}. Note that the energy budget could be simply set equal to the remaining battery charge. The final goal of management is to not consume more than energy E_{budget} before time t_{charge}. The manager activates at a constant time rate and we indicate with t_k the current time step. The time of manager activation is t_0 and the initial battery energy is E_{start}. The problem can be formulated then as in Equation (6).

$$Max(f_{k+1}) \; s.t. \; E_{batt_{k+1}} < E_{target_{k+1}} \quad (6)$$

Where f_{k+1} is the frequency selected for the next time interval and $E_{target}$is the constraint on battery energy consumption. The value of E_{target} is calculated at each time instant based on the concept of *battery discharging profile*. This is a function of energy over time, starting from point (t_0, E_{start}) and ending at point (t_{charge}, E_{budget}). For practical reasons, in this work we assume the discharging profile to be linear in time, therefore $E_{prof_k} = mt_k + E_{start}$. This is motivated by the result shown in Figure 3(b), for which battery discharging can be well approximated by a linear function. Note that the concept is general and different discharging profile (e.g. piece-wise linear, quadratic, etc.) may lead to different management behavior.

Table 1. AP-states

FREQ. LIST	POWER	COUNTER
f_0	$\boldsymbol{P_{APP_i}(f_0)}$	$\boldsymbol{T_{i0}}$
f_1	$\boldsymbol{P_{APP_i}(f_1)}$	$\boldsymbol{T_{i1}}$
...	...	...
f_N	$\boldsymbol{P_{APP_i}(f_N)}$	$\boldsymbol{T_{iN}}$

If the device consumes less power than what is allocated for using at a certain time instant t_k, the energy left over is added to the *energy tank* E_{tank}. On the other hand, if it consumes more power, the excess energy is subtracted from E_{tank}. Therefore, in the end the target energy is equal to:

$$E_{target_{k+1}} = E_{prof_{k+1}} + E_{tank_{k+1}} \quad (7)$$

To solve this management problem we leverage a heuristic. Depending on which application i is currently running, the heuristic selects the highest frequency f_N and checks whether the condition in Equation (6) is met for $P_{APP_i}(f_N)$. If not, it selects f_{N-1} and checks again. If no valid frequency is found for the current time instant, then the manager selects f_0. The heuristic is feasible given the limited number of CPU and GPU predefined operating frequencies.

3.1 Framework Implementation

We implemented the management framework on a Nexus 5 smartphone. This device has a Qualcomm Snapdragon 800 chipset, with a quad-core Krait 400 CPU. Its maximum battery capacity is 31,257J. The four cores have a frequency range from 300Mhz to 2.26Ghz in 14 fixed operating points, with $f_{CPU_{13}}$ being the maximum frequency. It also has an Adreno 330 GPU with frequency range from 200Mhz to 450Mhz, in 4 fixed operating points, being f_{GPU_3} the maximum. For simplicity, he CPU and GPU frequency pairs adopted for our AP-states are associated with the following rule: $(f_{CPU_{13-10}}, f_{GPU_3})$, $(f_{CPU_{9-6}}, f_{GPU_2})$, $(f_{CPU_{5-3}}, f_{GPU_1})$, $(f_{CPU_{3-0}}, f_{GPU_0})$.

Figure 4 presents the block diagram of our implementation.

The **App Monitor** is an independent program, which periodically checks the executing processes and writes on the file *Foreground App* the name of the application currently executing in foreground.

The **AP-states Monitor** periodically samples the operating conditions of the multicore platform (CPU and GPU frequency) and the battery power with a rate of 1 second. Based on this and on the current foreground application, it updates the values of AP-states and writes it back to the file *AP-states*. If APP_j is executing, the current sampled power is P_{curr} and the current operating conditions are equal to f_k, then the k^{th} AP-state of APP_j is updated as in equation (8).

$$P_{APP_j}(f_k) = P_{APP_j}(f_k) * \left(\frac{T_{jk}}{T_{jk}+1}\right) + \frac{P_{curr}}{T_{jk}+1} \quad (8)$$

Also, the value of T_{jk} is increased by 1.

The **Power Controller** loads the *Target Battery Lifetime* and the *Initial Energy Budget* configured by the user when starting execution. Then, it periodically applies operating conditions (CPU and GPU frequency), by solving the problem shown in Equation (6) based on the values of energy tank, available energy (left from E_{start}) and discharging profile. The activation rate of the power controller is also 1 second.

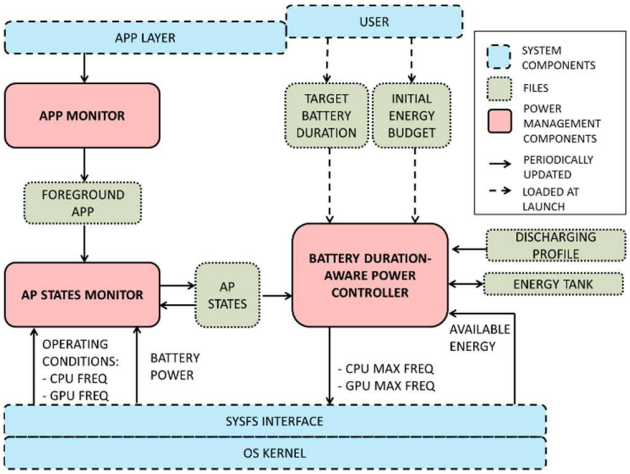


Figure 4. Block Diagram of implemented framework

Since the framework is implemented in the userspace, the minimum activation rate to avoid overhead is 1 second. However, the activation rate of Frequency Governors in the kernel space is much higher (20ms is the standard for the *ondemand* governor [6]). Then, we must guarantee that our Power Controller is compatible with any governor. Therefore the output of the Power Controller is not a fixed frequency value, but it is a maximum operating frequency for all the four cores. This can be set by writing values to the appropriate *sysfs* file. Consequently, we decided to configure the AP-states Monitor so that it updates AP-states based on the current maximum value of frequency among the 4 cores. This choice guarantees that every time a certain maximum frequency configuration f_j is selected, the average contribution to the total power consumption will not be higher than $P_{APP_i}(f_j)$, which is exactly what we require to meet a certain t_{charge}.

The two programs that compose this implementation, e.g. the App monitor and the AP-states Monitor/Power Controller, are written in C and cross compiled with Android NDK tools to run on ARM-based platforms. To run the framework, it is sufficient to load the binaries of the two programs in the device and execute them. No modification to the operating system is required.

4. RESULTS

In this section, we describe the experimental results obtained by measurements on the target platforms. To perform comparisons on real applications, we wrote a program that allows to record and replay display touch event traces. In this way, comparisons are performed on the same trace of events.

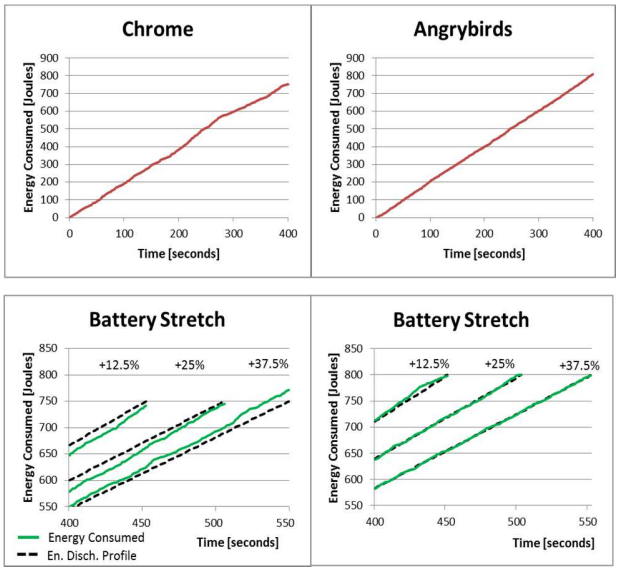


Figure 5. Standard energy consumption and battery stretch

For the first experiment, we recorded two traces on the Nexus 5 from two popular Android applications: Chrome and Angrybirds. We chose to show results for these two applications because they are representative corner cases of typical mobile usage: browsing and gaming [16]. In the first, we are browsing through popular webpages. In the second we are playing the game. The first two plots of Figure 5 report the discharging curves for the two traces when no control is active. This allows us to fix an initial reference t_{charge} equal to 400 seconds and energy budgets E_{budget} of respectively 750 Joules and 800 Joules for Chrome and Angrybirds. Considering this we show how to configure the control for *Battery Stretch*, that is, for a longer t_{charge}. The second two plots of Figure 5 show three cases of battery stretch for the applications, respectively +12.5% (450s), +25% (500s) and +37.5% (550s). Reported values are the real energy consumed over time (bold green line) and the energy discharging profile E_{prof} (black dotted line). For practical reasons, the x-axis only reports the final section of the curves. We can notice that in all three cases, the total final consumed energy is less or equal than the predefined target, therefore the power management goal is met. The only exception is the case of +37.5% battery stretch for Chrome. In this case the final target exceeded. This means that t_{charge} for this case was set too long and the controller cannot meet it even with constant minimum frequency set. Moreover, a subjective evaluation suggested that while in the case of +25% stretch the application behavior is still satisfactory, in the +37.5% stretch both applications become blurry.

We also conducted an experiment in which we asked 15 students from the university campus to check application behavior under battery stretch. To illustrate how portable our framework is, this study has been conducted instead on a Qualcomm APQ8064 development tablet. We asked people to play the popular game Temple Run [23] and to browse on popular websites (CNN and BBC) for one minute under 0%, 25% and 50% battery stretch conditions respectively. In order to avoid influencing their judgment, users were not informed about the nature of the control and the goal of the experiment. In addition, the battery stretch conditions are applied in a random order. At the end, users were asked to rate their experience with a number from 1 (bad) to 10 (good). Figure 6 reports the average value of scores for the

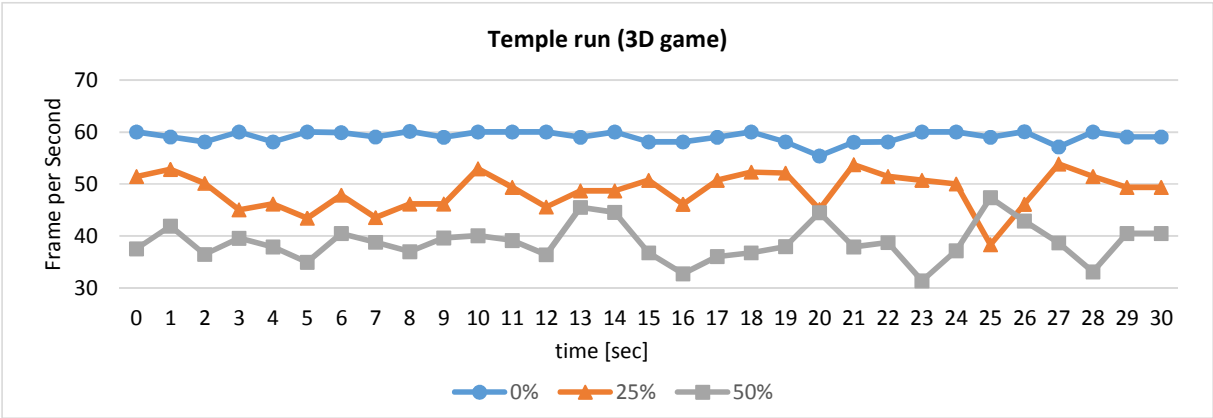


Figure 7. Frame per second (FPS) traces with different target lifetimes

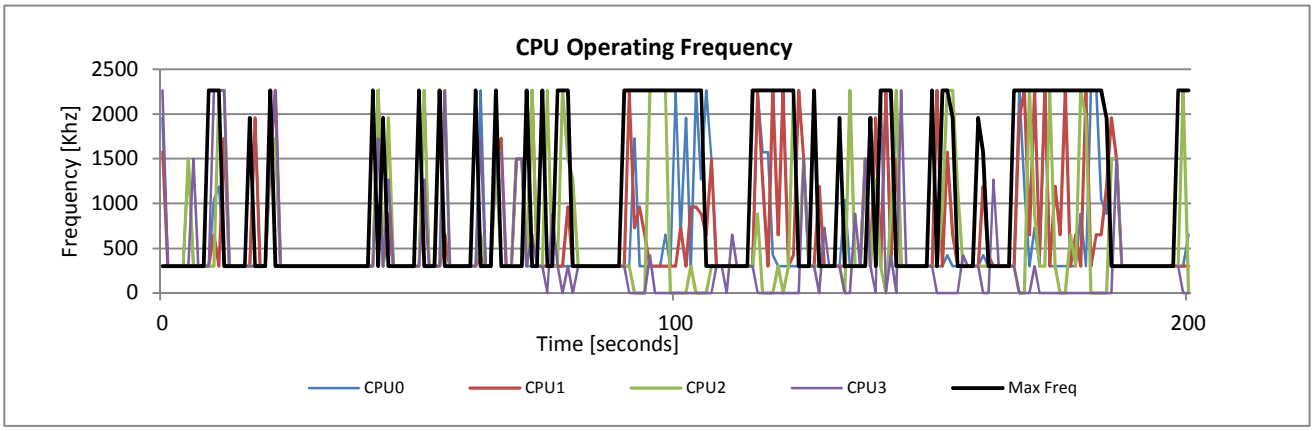


Figure 8. CPU operating frequency for Angrybirds +25% battery stretch

two applications. What we observe is that there is very low or no negative difference when the 25% stretch is applied, while there is a more negative effect with 50% stretch. This means that the user did not observe degradation in the quality of their experience. Our conclusion is that the control can still meet a satisfactory experience for the user when the selected target battery lifetime is extended by 25%.

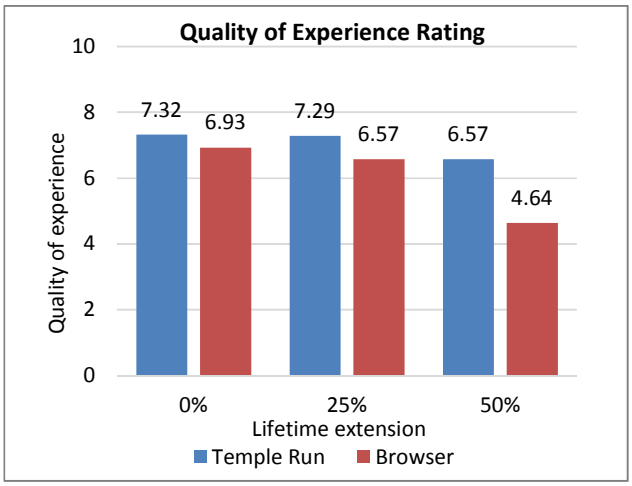


Figure 6. Quality of experience Rating

To further prove this point we refer to the fact that there is a well known relation between Frames per Second (FPS) and user experience in the case of gaming applications [22]. For this reason, we observe the behavior of FPS during the execution of Temple Run in the three cases of 0%, 25% and 50% target extension in our Qualcomm tablet. FPS can be monitored in Qualcomm devices thanks to a built-in feature that enables the system to write values to *logcat*. Figure 7 reports the FPS traces for 30 seconds of execution. In the case of 0% extension, the FPS is close to 60, which is the maximum value allowed by Android, and represent the highest quality of user experience. In case of 25% extension, the FPS is around 50, which represent a lower, but still acceptable user experience (as confirmed by results in Figure 6). Finally, for a 50% extension, the FPS drops even below 40, which starts to represent a source of discomfort for the user. Note that these results match what already shown in Figure 6.

To show more into details how the management policy works, we report in Figure 8 the Nexus 5 execution frequency of the 4 CPU cores and the maximum frequency set by the controller for the case of +25% battery stretch in 200 seconds of the Angrybirds trace. Our controller does not fix the frequency, but only limits the range of the underlying frequency governor (black dotted line).

To show the tradeoff between t_{charge} and performance, we execute a set of experiments on popular Android benchmarks on the Nexus 5: Antutu, Geekbench3, GFX and Vellamo Browser. Antutu is a suite of benchmarks comprising CPU, 2D and 3D evaluation. Geekbench3 is a CPU evaluation tool which tests both single and multicore performance (Gb3_single and Gb3_multi). GFX is a suite of graphics benchmarks from which we selected *t-rex* (GFX_trex). Finally Vellamo Browser is a benchmark testing device performance while using Chrome. All these benchmarks provide a final score which is an indicator of performance quality. Figure 9 shows the scores obtained for the presented benchmarks with varying values of t_{charge} (respectively at 15%, 25% and 50% extension), normalized over the maximum score (which is obtained when the control is not active).

From the results, we can notice that as battery lifetime is extended, the score of the benchmark (e.g. the device performance) decreases. In addition, we can notice that different applications show different sensitivities, which motivates further the choice of differentiating AP-states based on applications.

Finally, in the next experiment we want to demonstrate the compatibility of our framework with different kernel Frequency Governors. For doing this we execute the Vellamo Browser benchmark with a target time t_{charge} of 300 seconds (corresponding to a 25% extension over the

non-controlled scenario) and a budget E_{budget} of 550 Joules. In Figure 10 we show the curve of consumed energy in three different cases, in which respectively are active the *ondemand*, *interactive* and *conservative* governor.

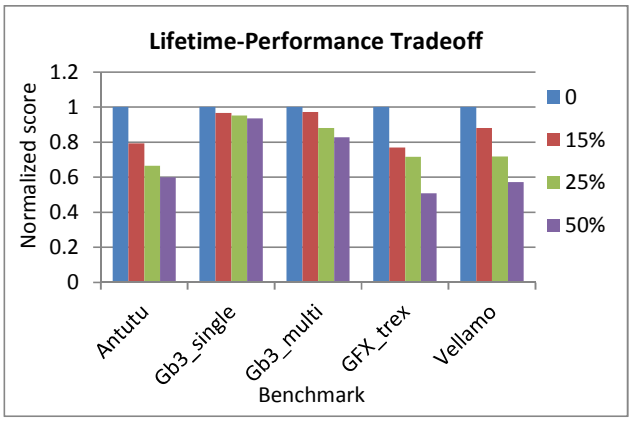


Figure 9. Benchmark evaluation of battery lifetime and performance tradeoff

This comparison shows that our controller is compatible with different governors and in all cases the target on battery lifetime is met. It also shows that in the case of Vellamo benchmark the conservative governor shows a better energy efficiency with respect to the other two governors, as it achieves a higher score with a lower power consumption. This can be justified assuming that for the conservative governor, the Vellamo benchmark controllable power component prevails over the non-controllable one.

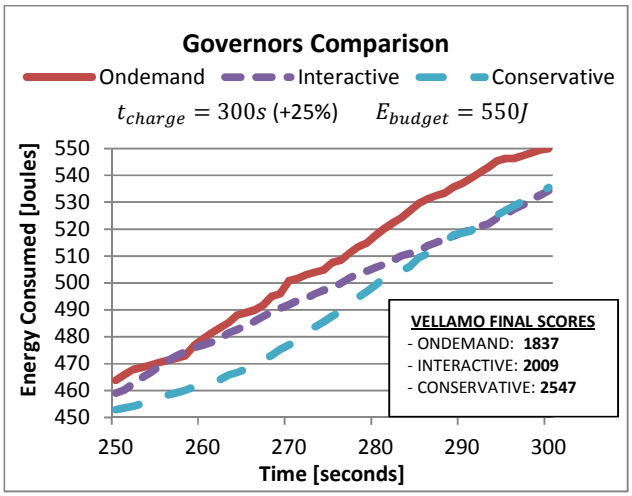


Figure 10. Comparison of different frequency governors

5. CONCLUSION

In this work we presented **BLAST**: **B**attery **L**ifetime-constrained **A**daptation with **S**elected **T**arget. Blast is the first application-aware power management framework for mobile devices which controls operating conditions in order to meet a predefined battery lifetime. We also presented a lightweight and portable implementation on a real Android device, compatible with different Frequency Governors. The experiments show that our solution is effective in guaranteeing the predefined target battery lifetime and that it still meets user experience requirements with a selected battery lifetime extension set to 25%. The average rating of real users is within 5% for a battery lifetime extension set to 25%.

6. ACKNOWLEDGEMENT

This work was sponsored in part by Samsung Research America and NSF grant number 1218666.

7. REFERENCES

[1] Shye, A; Scholbrock, B.; Memik, G., "Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures," Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on , vol., no., pp.168,178, 12-16 Dec. 2009

[2] Ge Bai; Hansi Mou; Yinhong Hou; Yongqiang Lyu; Weikang Yang, "Android Power Management and Analyses of Power Consumption in an Android Smartphone," High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on , vol., no., pp.2347,2353, 13-15 Nov. 2013

[3] Geunsik Lim; Changwoo Min; Dong Hyun Kang; Young Ik Eom, "User-aware power management for mobile devices," Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference on , vol., no., pp.151,152, 1-4 Oct. 2013

[4] Sangwook Kim; Hwanju Kim; Jeaho Hwang; Joonwon Lee; Euiseong Seo, "An event-driven power management scheme for mobile consumer electronics," Consumer Electronics, IEEE Transactions on, vol.59, no.1, pp.259,266, February 2013

[5] Aaron Carroll and Gernot Heiser. 2010. An analysis of power consumption in a smartphone. In Proceedings of the 2010 USENIX conference on USENIX annual technical conference (USENIXATC'10). USENIX Association, Berkeley, CA, USA, 21-21.

[6] Pallipadi, V. & Starikovskiy, A. (2006), The ondemand governor: past, present and future, in 'Proceedings of Linux Symposium, vol. 2, pp. 223-238' .

[7] Benini, L.; Bogliolo, A; De Micheli, G., "A survey of design techniques for system-level dynamic power management," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.8, no.3, pp.299,316, June 2000

[8] Chanmin Yoon, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, and Hojung Cha. 2012. AppScope: application energy metering framework for android smartphones using kernel activity monitoring. In Proceedings of the 2012 USENIX conference on Annual Technical Conference (USENIX ATC'12). USENIX Association, Berkeley, CA, USA, 36-36.

[9] Chengke Wang; Fengrun Yan; Yao Guo; Xiangqun Chen, "Power estimation for mobile applications with profile-driven battery traces," Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on , vol., no., pp.120,125, 4-6 Sept. 2013

[10] Alawnah, S.; Sagahyroon, A, "Modeling smartphones power," EUROCON, 2013 IEEE , vol., no., pp.369,374, 1-4 July 2013

[11] Bonetto, A; Ferroni, M.; Matteo, D.; Nacci, AA; Mazzucchelli, M.; Sciuto, D.; Santambrogio, M.D., "MPower: Towards an Adaptive Power Management System for Mobile

Devices," Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on , vol., no., pp.318,325, 5-7 Dec. 2012

[12] A. A. Nacci, F. Trovò, F. Maggi, M. Ferroni, A. Cazzola, D. Sciuto, and M. D. Santambrogio. 2013. Adaptive and Flexible Smartphone Power Modeling. Mob. Netw. Appl. 18, 5 (October 2013), 600-609.

[13] Nachi K. Nithi and Adriaan J. de Lind van Wijngaarden. 2011. Smart power management for mobile handsets. Bell Lab. Tech. J. 15, 4 (March 2011), 149-168.

[14] Marcelo Martins and Rodrigo Fonseca. 2013. Application modes: a narrow interface for end-user power management in mobile devices. In Proceedings of the 14th Workshop on Mobile Computing Systems and Applications (HotMobile '13). ACM, New York, NY, USA.

[15] Wei, J.; Juarez, E.; Garrido, M.J.; Pescador, F., "Maximizing the user experience with energy-based fair sharing in battery limited mobile systems," Consumer Electronics, IEEE Transactions on , vol.59, no.3, pp.690,698, August 2013

[16] Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan, and Deborah Estrin. 2010. Diversity in smartphone usage. In Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10). ACM, New York, NY, USA, 179-194.

[17] Pathania, Anuj; Qing Jiao; Prakash, Alok; Mitra, Tulika, "Integrated CPU-GPU power management for 3D mobile games," Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE , vol., no., pp.1,6, 1-5 June 2014

[18] Xin Li, Mian Dong, Zhan Ma, and Felix C.A. Fernandes. 2012. GreenTube: power optimization for mobile videostreaming via dynamic cache management. In Proceedings of the 20th ACM international conference on Multimedia (MM '12). ACM, New York, USA, 279-288.

[19] Yiran Chen; Xiang Chen; Mengying Zhao; Xue, C.J., "Mobile devices user — The subscriber and also the publisher of real-time OLED display power management plan," Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on , vol., no., pp.687,690, 5-8 Nov. 2012

[20] Mercati P, Hanumaiah V., Kulkarni J., Bloch S. and Rosing T. “User-centric Joint Power and Thermal Management for Smartphones” MOBICASE 2014, IEEE international conference, Austin, TX, USA, Nov 2014.

[21] Li, Xueliang; Yan, Guihai; Han, Yinhe; Li, Xiaowei, "SmartCap: User experience-oriented power adaptation for smartphone's application processor," Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013 , vol., no., pp.57,60, 18-22 March 2013

[22] Anuj Pathania, Qing Jiao, Alok Prakash, and Tulika Mitra. 2014. Integrated CPU-GPU Power Management for 3D Mobile Games. In *Proceedings of the 51st Annual Design Automation Conference* (DAC '14). ACM, New York, NY, USA, , Article 40 , 6 pages.

[23] https://play.google.com/store/apps/details?id=com.imangi.templerun&