$\mathbf{E^3}$: Energy-Efficient Engine for Frame Rate Adaptation on Smartphones

Haofu Han, Jiadi Yu, Hongzi Zhu
Department of Computer Science and Engineering
Shanghai Jiao Tong University
{hanhaofu, jiadiyu, hongzi}@sjtu.edu.cn

Yingying Chen
Department of Electrical and Computer Engineering
Stevens Institute of Technology
yingying.chen@stevens.edu

Jie Yang
Department of Computer Science and Engineering
Oakland University
yang@oakland.edu

Guangtao Xue, Yanmin Zhu, Minglu Li
Department of Computer Science and Engineering
Shanghai Jiao Tong University
{gt_xue, yzhu, mlli}@sjtu.edu.cn

ABSTRACT

Touch-screen technique has gained the large popularity in human-screen interaction with modern smartphones. Due to the limited size of equipped screens, scrolling operations are indispensable in order to display the content of interest on screen. While power consumption caused by hardware and software installed within smartphones is well studied, the energy cost made by human-screen interaction such as scrolling remains unknown. In this paper, we analyze the impact of scrolling operations to the power consumption of smartphones, finding that the state-of-art strategy of smartphones in responding a scrolling operation is to always use the highest frame rate which arouses huge computation burden and can contribute nearly 50% to the total power consumption of smartphones. In recognizing this significance, we further propose a novel system, *Energy-Efficient Engine*(E^3), which automatically tracks the scrolling speed and adaptively adjusts the frame rate according to individual user preference. The goal of E^3 is to guarantee the user experience and minimize the energy consumption caused by scrolling at the same time. Extensive experiment results demonstrate the efficiency of E^3 design. On average, E^3 can save up to 58% of the energy consumed by CPU and 34% of the overall energy consumption.

Categories and Subject Descriptors

H.5.2 [**INFORMATION INTERFACES AND PRESENTATION**]: User Interfaces—*Input devices and strategies*

SenSys'13, November 11–15, 2013, Rome, Italy.

General Terms

Design, Experimentation, Human Factors

Keywords

Energy efficiency, user experience, touch screen, scrolling operation, frame rate

1. INTRODUCTION

Mobile devices such as smartphones and tablets have received increasing popularity over the recent years (e.g., almost one billion smartphones have been sold since 2009[17]). Millions of appealing apps on smartphones bring users new style of information sharing and more efficient communications. New features and powerful hardware have been constantly embedded into smartphones to provide more functionalities. Battery capacity, however, has not kept the pace with the increasing demand for energy posed by all-singing-all-dancing applications[24], and drains quickly by the enhanced features and the complex operations performed by smartphones. Longer battery life has been considered as one of the most important feature to smartphone users[7], and has aroused great attention in both the academia and the industry.

Existing studies on extending smartphone battery life mostly focus on saving power consumed by CPU computation, radio usage, and application activities. For example, through wisely adjusting the CPU frequency[6][23], minimizing the tail effect[4][21][25], and eliminating energy bugs in applications[18][20], considerable energy saving could be achieved. Furthermore, power consumption could be reduced by adjusting screen display parameters such as LCD backlight level[2] and OLED color scheme[9]. Although many efforts have been made to improve the energy-efficiency of mobile devices, the impact of human-screen interactions on the power consumption of mobile devices remains unknown.

With the rapid advancement of touch-screens, most mobile devices are equipped with touch-screens. Due to the limited size of the screen, finger operations such as clicking and scrolling are indispensable in order to display the content of interest on screen. ProfileDroid[27] reveals that the interaction-intensive applications may generate more than

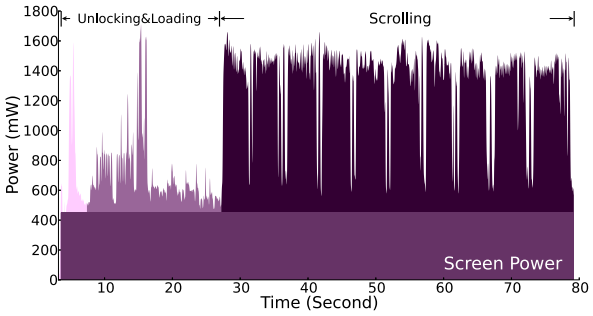


Figure 1: Energy consumption of screen scrollings during a web surfing on Nexus S.

20 input-events per second, which results in more than 30% time for human-screen (i.e., touch-screen) interactions when using applications like browsing and gaming. For example, Figure 1 plots the power consumption of a smartphone, Nexus S, during a typical web surfing process. The surprising finding is that after loading the content of the webpage, the power consumption level jumps three times higher than usual upon each time the user scrolls the screen. During this web surfing process, the scrolling operations consume up to 53.4% of the total energy consumption, whereas loading the web browser (triggered by a click) and open the webpage only consume 6.3% in total. We find that the human-screen interaction causes large energy consumption. The human-screen interaction, however, is essential for touch-screen smartphone usage and dominates the user experience. To save the energy consumption caused by the human-screen interaction, we face the following great challenges. First, the energy consumption should be reduced without compromising user experience. Second, the response to human-screen interaction should be instant and accurate. Finally, the solution should be lightweight and not bring in noticeable overhead. Although there has been work[2][9] on adaptive screen power management of mobile devices, the focus is on saving energy through changing the LCD backlight and OLED color scheme. To the best of knowledge, our work is the first attempt to tackle the problem of high power consumption caused by the human-screen interactions.

In this paper, we first set out to investigate the root cause of the high energy consumption in screen scrolling. Through empirical study of over 300 volunteers, we find that the root cause of the high energy consumption incurred by scrolling is because of the current frame rate adjusting strategy on smartphones that always uses the highest frame rate to display contents. This poses large computation burden and therefore consumes much energy (nearly 50% of the total power consumption). Based on our observations, we propose *Energy-Efficient Engine* (E^3) system, which can significantly reduce the power consumption caused by the scrolling operations while still keeping the user experience of smartphones un-compromised. In E^3, we define a new metric called *satisfied frame rate*, which is the minimum frame rate under which human eyes cannot feel falters on the screen. We calibrate the relationship between the satisfied frame rate and the scrolling speed, and show that the Logarithmic model grounded on the least-square regression analysis can precisely characterize this relationship. Our E^3 system can wisely configure a particular setting most suitable for a specific user, and has the capability to adaptively adjust the satisfied frame rate according to the trained model. The main advantage of E^3 is two-fold. First, E^3 can optimize the frame rate with respect to energy consumption while satisfying the user-experience simultaneously. Second, E^3 is easy to implement and computational feasible on mobile platforms including both smartphones and tablets. Our prototype implementation of E^3 on Android-based mobile devices verifies the feasibility of using E^3 in real environments.

We highlight our main contributions as follows:

- We empirically investigate the impact of human-screen interactions to the power consumption on mobile devices including smartphones and tablets. Specifically, we find that scrolling operations can consume tremendous energy with current frame rate scheme. Our experiment results show the amount of energy consumed by scrolling can reach up to 50% on average over a large range of applications.

- We propose E^3, an innovative frame rate adaptation system, which can adaptively adjust the satisfied frame rate without compromising the user experience. We find that Logarithmic model can accurately capture the relationship between the scrolling speed and satisfied frame rate based on least-squares regression analysis.

- We implement our system on five different mobile devices including smartphone and tablet. Our prototype of E3 verifies the feasibility of the design in real-world scenarios.

- We conduct extensive experiments with 327 volunteers to evaluate the performance of E^3. The results show that, on average, E^3 can save up to 34% of the overall energy consumption while keeping a user satisfaction rate over 93%.

The remainder of this paper is organized as follows. Related work is reviewed in Section 2. We give an in-depth analysis of energy consumption caused by human-screen interaction in Section 3. Section 4 presents the design details of E^3. Section 5 introduces the prototype implementation. In Section 6, we evaluate the performance of E^3 and present the results. Finally, we give conclusive remarks and discuss future directions in Section 7.

2. RELATED WORK

Active work have been done to improve the energy-efficiency of smartphones. We categorize the existing work in the following aspects.

Smartphone Power Model: many efforts have been made to improve the accuracy of smartphone energy profiling. In earlier work, the energy estimation relay on external hardware [11][5]. Dong *et al.*[10] first attempted to design a self-constructive power model of mobile devices. While Pathak *et al.* gave another trial on accounting energy consumption based on the system-call[19] and then improved his work to a fine grained energy accounting system[20].

CPU Power Consumption: Due to the increasingly computing power, the energy consumption of CPU also grows rapidly. Studies on energy consumption of smartphone CPU

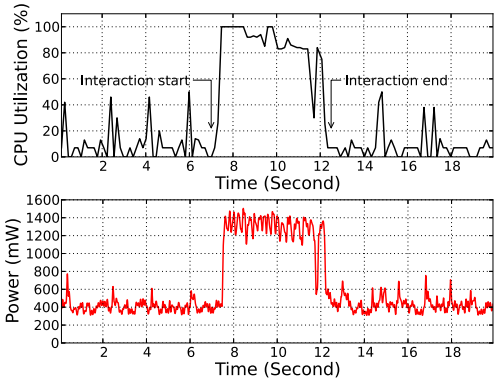


Figure 2: CPU utilization and energy consumption during one scrolling operation on a webpage.

endeavored to find an energy-efficient strategy to dynamically adjusting the CPU frequency and voltage based on the concept of DVFS[6][23].

Radio Power Consumption: since communication is the basic function of smartphones, the power consumption of wireless module draws a lot attention in researching area. Niranjan *et al.* first found the *tail*[4] overhead which is caused by the lingering in high power states after completing a transfer. Then, in 2012, Qian *et al.* elaborately summarized several optimizing schemes[22][21] on tail energy. Besides, Athivarapu *et al.* explored a method to reduce the radio usage by monitoring the program execution pattern[3]. In another direction, Schulman *et al.* discussed the impact of signal strength on communication energy in [25].

Application Power Consumption: with the Explosive growth of smartphone applications, increasingly researchers are interested in investigating the energy consumption in applications. Narseo made a statistical analysis on the energy consumption of different types of Apps in [26]. Besides, [13] discussed the impact on energy consumption of cooperation between Apps and operating system. [18][20] proposed the concept of energy bug in smartphones and gave a first trial on diagnosing energy bugs.

Display Power Consumption: due to the physical characteristics of display hardware, energy saving could be achieved by wisely adjusting LCD and OLED display parameters. Existing work studies screen hardware power consumption through studying the screen power model[16], calibrating the backlight level[2] or the display color scheme[9], while our work explores the power consumption of mobile device made by exorbitant screen frame rate.

Although the battery-life issues of mobile devices have gained much attention, the energy cost made by human-screen interactions such as scrolling remains elusive. There are some studies on frame rate recently, but these works mainly focus on how to improve frame rate through hardware and software[28] or how to use frame rate as a QoS metric[8][15]. Therefore, existing work has not concerned the impact of frame rate to the energy consumption.

3. EMPIRICAL STUDIES & TRACE ANALYSIS

We first conduct an empirical study to show that human-screen interactions may be a new element in the spectrum of power consumption on smartphones. Among all normal in-

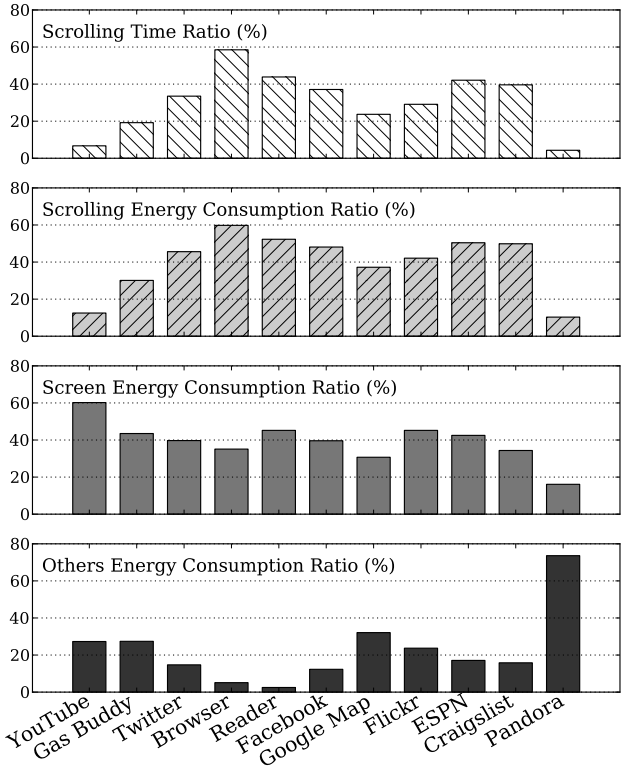


Figure 3: Scrolling time ratio and energy consumption ratio of several popular smartphone applications.

teraction operations, scrolling is a very typical one to display the content of interest on screen. Moreover, the interaction-intensive applications such as browsing and reading are more likely to have a higher ratio of scrolling to click (1.4 in Angry-Birds and 1.1 in CNN)[27]. Besides, the energy consumption caused by click is much lower than scrolling. In this section, we explore the influence of scrolling operations to the energy consumption on smartphones.

3.1 Power Consumption Caused by Scrolling on Screen

To illustrate the relationship between screen scrolling operations and power consumption, we first examine one scrolling operation when browsing a webpage using a typical smartphone (Nexus S). During this operation, we record the CPU utilization and the measurement of energy consumption then plot the results in Figure 2. It is clear to see that, once the scrolling starts, the CPU utilization immediately increases to 100%. Meanwhile, the power jumps up twice higher than usual. Moreover, the CPU utilization and power consumption keep high until the scrolling action ends. Similar results are obtained through more trials of this experiment with different users and mobile devices. Furthermore, we also conduct this experiment with different scrolling speeds and directions, and get the same results.

We then further examine the impact of scrolling operations among different most popular applications on different smartphones. Specially, we randomly select 327 volunteers (i.e., 56 faculty members and 271 students) on campus during lunch time, and let them try at least five applications in-

stalled on our test devices (i.e., Nexus One, Nexus S, Nexus Prime, Galaxy S II, Galaxy Tab) for ten minutes. Figure 3 shows the average scrolling time ratio and the corresponding energy consumption ratio of each tested mobile application. The scrolling time ratio means the proportion of time during scrolling operation to the time of each application usage. It can be seen that for most of the interaction-intensive applications, the scrolling time ratio is higher than 30% (For browser, it's almost 60%).

We also analyze the power consumption caused by scrolling and all other components and factors. The energy consumption is divided into three major factors *Scrolling*, *Screen backlight* and *Others*: *1) Scrolling* represents the part of energy consumption caused by scrolling; This part of energy consumption is measured by separating the increased power that strongly related to scrolling operation. For example, the scrolling operation causes power increment as shown in fig,2, and we classify energy caused in such increments as *Scrolling* energy consumption. *2) Screen backlight* represents energy consumption caused by screen light emitting. It can be measured by employee the mechanisms presented in [9] ,[16]. We note that during the experiments, we use a low frequency to sample the screen display, which only results negligible energy consumption compared to the energy consumption caused by screen emitting. *3) Others* represent the energy consumption caused by various power consumers such as loading applications and contents, radio usage, audio playback, and GPS module. Since we can measure the total energy consumption by using a power meter (as shown in Section 5), the energy consumption of the three components could be obtained.

We find that in most cases, the scrolling energy consumption of the Browser and the Reader could reach up to 59.8% and 52.3% of the total energy consumption, respectively. Regardless of the high energy consumption caused by Radio usage and GPS module, for some applications like Facebook App and Google Map, scrolling is always the most significant factor (Google Map gets a 37.2% share of the total energy consumption) with respect to energy consumption. The scrolling energy of Youtube and Pandora are not as high as the others, because few scrollings are needed during their usage. These results show that scrolling could be the main contributor of the energy consumption on smartphones.

Also, from Figure 3, it is surprising to see that the energy consumption ratios of the *Others* are not very high, which indicates that the network connection and radio transmission may not play a main role in smartphone energy consumption for every daily-used applications.

In general, we find that the average scrolling energy consumption could reach up to 46% of the total energy consumption over all the interaction-intensive applications. The reason that scrolling operations can significantly affect the energy consumption of smartphones is because, with the limited size of the smartphone screen, scrolling is indispensable in human-screen interaction. How to optimize the scrolling operation in term of power consumption is of great importance. In this paper, we focus on minimizing the energy consumption caused by scrolling.

3.2 Impact of Frame Rate Strategy

During the processing of one scrolling, there are three procedures: *catching*, *processing* and *event feedback*. First, in the catching procedure, an interrupt is triggered to inform

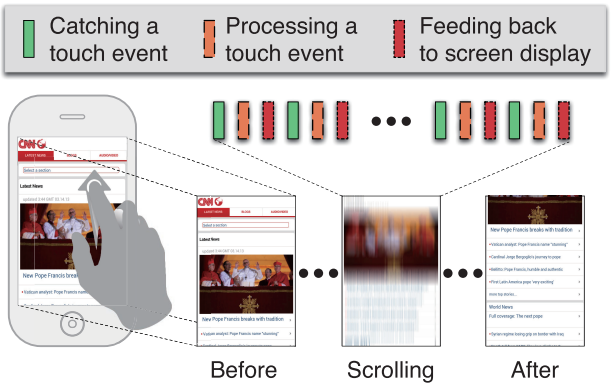


Figure 4: Illustration of display updates during a scrolling operation.

the operating system that there is a user input on the touch screen. Then, the operating system reads arguments of this input and passes them to the relative application. Second, during the processing procedure, the application calls the response functions and generates a respond of this input event. After getting the respond, the application sends requests to the operating system. Finally, in the event feedback procedure, the operating system receives requests from the application, calculates the frame of image to be displayed and refreshes screen to display it. Figure 4 illustrates the workflow of display updates in one scrolling operation. During each scrolling, the three procedures iterates frequently. Any of these three procedures is possible to cause the high energy consumption. Nevertheless, based on the explanation above, since there is little computation in the catching procedure, energy consumption caused by the catching should be very low. Besides, the computation in the processing procedure is related to the input arguments, i.e. the scrolling speed and direction. From the results of our experiment in Section 3.1, however, scrolling speed and direction have no impact on energy consumption. This implies that the energy consumption caused by the processing procedure is low. Based on the above analysis, in order to realize a smooth screen display in the event feedback procedure, the screen updating operation needs to be performed dozens of times per second. Each time the screen updates, CPU resource is consumed to build a new image to display. We thus infer that the screen update in feedback procedure incurs massive CPU resource and therefore high power consumption.

To verify our analysis, we further set up experiments to examine the energy consumption caused by screen updating in the feedback procedure. Here a *frame* refers to the image displayed on screen after one screen update operation and a *frame rate*, denoted as r, refers to the frequency of screen update operation, measured with the unit of *frame per second* (FPS). We first monitor frame rate in real-time via modifying the source code of Android. Figure 5 plots the frame rate and power consumption when surfing a webpage on a Nexus S smartphone. During one scrolling operation, the operating system uses all the necessary CPU resource to promote the frame rate until either $60fps$ (the hardware and software upper bound of the frame rate) or the CPU utilization of 100% is reached. The results in Figure 5 show that the energy consumption has the obvious same changing trend with the frame rate. We also get the same result

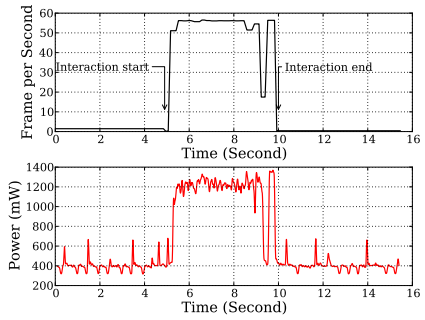


Figure 5: System frame rate and power consumption during a scrolling operation.

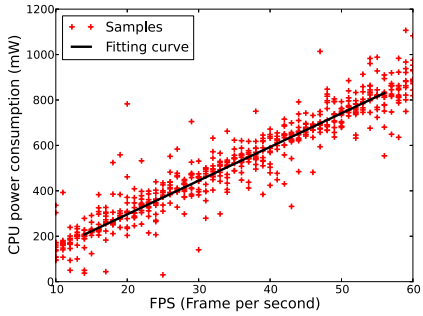


Figure 6: Relationship between frame rate and CPU power consumption.

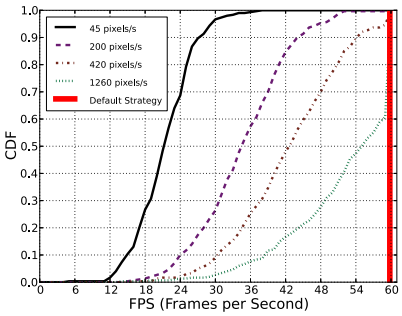


Figure 7: CDF of the satisfied frame rate in different scrolling speeds.

on our other test devices. The results confirm our inference that the energy consumption is highly related to the frame rate.

To capture the relationship between the frame rate and the power consumption more accurately, we collect a trace of frame rate and the corresponding power consumption measurements and analyze the energy cost under different frame rate. Figure 6 shows the scatter plot of CPU power consumption with frame rate varying from $10fps$ to $60fps$. We observe that the CPU power consumption is linearly proportional to the frame rate and the power consumption raises from $150mW$ to almost $1000mW$ while frame rate increases to $60fps$. The strong correlation between the frame rate and the power consumption reveals that the screen update operation is the major cause that incurs the high power consumption while scrolling.

3.3 Exploring User Experience on Frame Rate

Based on the analysis above, we know that higher frame rate will lead to larger energy consumption. One straightforward solution for energy saving is to simply reduce the frame rate no matter how a user operates on the screen. However, blindly carrying this out is not feasible since inappropriate low frame rate will make the user feel faltering when scrolling the screen. To explain this phenomenon, we define the image difference between two adjacent frames as $IDAF = L(bit(f_{i-1}), bit(f_i))$, where $bit(f_i)$ is the binary string representation of the i^{th} frame (i.e. the picture that is displayed on screen) and $L(x_1, x_2)$ is the *Levenshtein Distance* [14] of binary string x_1 and x_2.

Apparently, a higher IDAF value means bigger image difference between two adjacent frames, which indicates a more inconsistent display. More specifically, with frame rate r in a period T, $\overline{IDAF}$ is defined as:

$$\begin{aligned} \overline{IDAF} &= \frac{1}{r \times T} \sum_{i=0}^{r \times T - 1} L(bit(f_i), bit(f_{i+1})), \\ &\approx \frac{1}{r \times T} L(bit(f_{t=0}), bit(f_{t=T})), \end{aligned} \quad (1)$$

where $L(bit(f_{t=0}), bit(f_{t=T}))$ is the display change in T, which is in proportion of scrolling distance S. Moreover, S can be expressed as $S = s \times T$. Thus, the relationship between $\overline{IDAF}$, s, r and T is

$$\overline{IDAF} \propto \frac{s \times T}{r \times T} \propto \frac{s}{r}, \quad (2)$$

when r decreases, the $\overline{IDAF}$ increases. In other words, lower frame rate would result in a bigger difference between two adjacent frames' images. Therefore, if a low frame rate is adopted to respond to a scrolling operation, the user would feel faltering about the display on screen. On the contrary, when scrolling speed s decreases, the $\overline{IDAF}$ decreases, which implies the user may feel more comfortable on the screen display. We denote $r_{min}(s)$ as the minimum frame rate that the user would not feel the display is faltering while scrolling at speed s. To guarantee the user experience, we need to adopt a frame rate r so that $r \geq r_{min}(s)$. On the other hand, we also need to minimize the power consumption caused by scrolling operations. Therefore, the optimal frame rate with respect to user experience and power consumption is $r_{min}(s)$.

To determine the $r_{min}(s)$, we probe the *satisfied frame rate* at which the user would not feel faltering of users at different scrolling speeds by field testing. In our test, given a scrolling speed, we automatically increase the frame rate from the lowest value (i.e., $5fps$) to the highest value (i.e., $60fps$) to display a scrolling webpage and let a user stop the process when she/he satisfies with the current frame rate. We randomly pick 327 volunteers on campus and let each volunteer do our tests for 5 minutes. Figure 7 plots the cumulative distribution function (CDF) of satisfied frame rate at four different scrolling speed, i.e., 45, 200, 420, 1260 pixels per second.

We find that the satisfied frame rate increases as the scrolling speed rises. For example, at the speed of 45 pixels per second, 80% volunteers are satisfied with $23fps$ and the satisfied frame rate value increases to $59fps$ at the speed of 1260 pixels per second. It can also be seen that, comparing with the default frame rate strategy which always uses the maximum frame rate (e.g., typically $60fps$), it is promising to use the satisfied (if not optimal) frame rate so that power consumption caused by higher frame rate can be saved while still satisfying users.

3.4 Modeling the Appropriate Frame Rate

To discover the relationship between the scrolling speed and the satisfied frame rate, we analyze the trace collected in our field tests. Figure 8 depicts the scatter plot of the satisfied frame rate versus the corresponding scrolling speed. The inset in Figure 8 shows the plot on logarithmic-linear scale. From the inset, we find the satisfied frame rate is linear with the scrolling speed, indicating the satisfied frame rate is some form of logarithmic function of the scrolling speed.

To further quantify the relationship between the satisfied

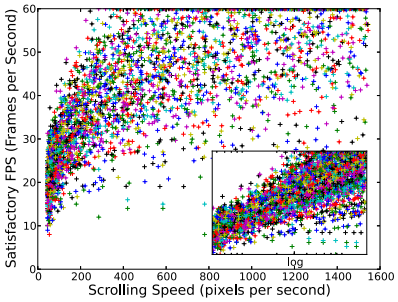


Figure 8: Satisfied frame rate in different scrolling speeds.

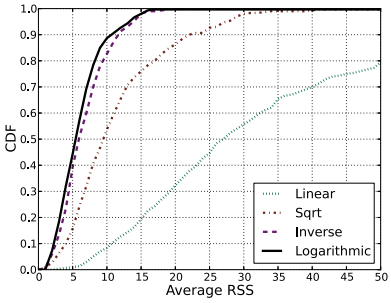
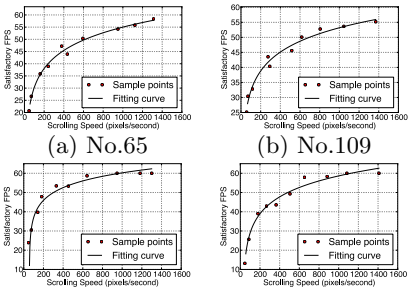


Figure 9: CDF of RSS on Linear, Sqrt, Inverse and Logarithmic formula.



(a) No.65 (b) No.109 (c) No.177 (d) No.272

Figure 10: Curve fitting of user-specific preference with Logarithmic model on 4 randomly chosen volunteers.

frame rate and the scrolling speed, we examine four models, as listed in Table 1, which could generate similar distribution by using the least-square regression analysis. In least-square regression, the estimator of each model is the *Residual sum of squares*, which is defined as follows:

$$RSS = \sum_{i=1}^{n} (r_{min}(s_i) - f(s_i))^2, \quad (3)$$

where $r_{min}(s_i)$ is the optimal frame rate at speed s_i and $f(s_i)$ is predicted frame rate given by the model at speed s_i. A small RSS indicates a tight fit of the model to the data.

We apply these four models listed in Table 1 to the data of each individual volunteer. The model parameters are selected to minimize RSS value. We plot the CDF of the average RSS in Figure 9 and list the average RSS value of each model in Table 1. We find that the Linear model's has a larger RSS value of 32.15 which indicates the average deviation on each predict point of this model is as high as $\sqrt{32.15} \approx 5.67 fps$.

Both the Logarithmic and the Inversely proportional model have a much lower average RSS value of 6.14 and 6.61, respectively. In summary, the Logarithmic model has the minimum average deviation on each predict point. Thus, the Logarithmic model is a better description of the relationship between the scrolling speed and the satisfied frame rate, i.e.,

$$r_{min}(s) = a \times \log(s + b) + c. \quad (4)$$

With the Logarithmic model, a user-specific preference model of each single user could be achieved. Figure 10 shows 4 sets (randomly chosen from 327 volunteers' sets) of sample points and their fitting curve with Logarithmic model, each set represents the satisfied frame rate on varied scrolling speed of one volunteer. Despite the diversity of user preferences, the Logarithmic model fits well with all sample sets. Thus, the Logarithmic model is compatible with different users.

Table 1: Possible Formulations

Name	Formulation	Average RSS
Linear	$r_{min}(s) = a \times s + b$	32.15
Sqrt	$r_{min}(s) = a \times \sqrt{s + b} + c$	11.40
Inverse	$r_{min}(s) = \frac{a}{s+b} + c$	6.61
Log	$r_{min}(s) = a \times \log(s + b) + c$	6.14

4. DESIGN OF E^3

From the previous investigation, we know that the 60fps frame rate strategy costs excessive CPU resource, which leads to the high energy consumption. However, in many circumstances, a much lower frame rate is proven to be optimal (e.g., 24fps in movie and 30fps in TV), which indicates that it is possible to relieve the stress on both CPU and battery of smartphones by changing the frame rate strategy and reducing the frequency of display update. In this section, we first present the key design goals of E^3, then discuss the design details of our system E^3.

4.1 Key Design Goals

Building an energy-efficient system for frame rate adaptation on mobile devices involves the following key design goals.

User Specific Preference Model: As we discussed in Section 3.4, although the Logarithmic model precisely describes user preference on frame rate, different users may have distinctive model parameters. To make sure the preference model is optimized for every user, E^3 should build a user-specific preference model for each single user.

Online Optimal Frame Rate Adaptation: Taking over the control of frame rate may cause delays in screen display, which goes against the user experience. In order to realize an instant and accurate respond to human-screen interaction, E^3 should calculate and filter the optimal frame rate in real time.

Flexible User Options: In case that the user preference changes over time or the preference model deviates from the expectation of user, E^3 should allow users to send feedback messages so that it can recalibrate the preference model and correct the errors.

Low Overhead: The target of the system is to save energy, thus E^3 should be lightweight. In particular, the computational complexity of scrolling speed estimation and the optimal frame rate calculation should be as low as possible.

Good User Friendliness: In order to provide a good user friendliness, the preference learning process of E^3 should complete in short time. Besides, operations of user should not annoy or distract users from their regular activities.

4.2 System Architecture

From the analysis in Section 3.4, we know that the Logarithmic model can precisely describe a user's preference on frame rate. Based on the preference model, the frame rate

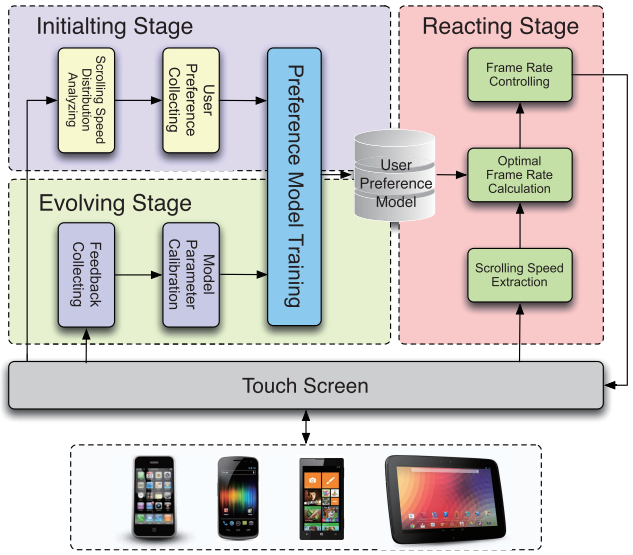


Figure 11: E^3 architecture.

could be reduced for energy saving while leaving the user experience un-compromised. In this subsection, we present an overview of E^3 design. E^3 first takes samples of the satisfied frame rate on a series of scrolling speeds of a specific user. Then, it applies the least-squares regression on the sampled data to calculate parameters of the Logarithmic model in Equation (4). After that, E^3 starts to monitor user actions performed on the touch-screen and estimates the real-time scrolling speed. Finally, each time the screen updates, E^3 adjusts the frame rate based on the real-time scrolling speed and the user-specific Logarithmic model. In addition, E^3 could also take user feedbacks to evolve the user preference model over time. The architecture of E^3 is shown in Figure 11. It consists of three stages: the *Initiating Stage*, the *Reacting Stage* and the *Evolving Stage*.

The purpose of the *Initiating Stage* is to build a user specific preference model in a fast and convenient way. During this stage, the system will first run the *Scrolling Speed Distribution Analyzing* procedure to analyze the scrolling speed distribution of the user. With the knowledge of the scrolling speed distribution, a highly-efficient sampling on the user preference curve could be achieved, which releases the further burden of preference learning process for users. After that, in the procedure of *User Preference Collecting*, a preference learning program will ascertain the satisfied frame rate for each sample point. Then, the least-squares regression is applied on the sampled data to generate the parameters of the Logarithmic model within the*Preference Model Training* procedure, which establishes a user-specific preference model.

In the *Reacting Stage*, the *Scrolling Speed Extraction* procedure works in the background to constantly monitor the touch screen events and estimate the real-time scrolling speed. With the *User Preference Model* established in the *Initiating Stage*, the *Optimal Frame Rate Calculation* procedure can be used to calculate the optimal frame rate from the real-time scrolling speed. Finally, in the procedure of *Frame Rate Controlling*, the optimal frame rate is applied to the operating system for energy saving.

In order to keep the pace with the changing of the preference of users, we design the *Evolving Stage*, which allows E^3 to dynamically update the preference model based on the feedback from the user. In particular, the user's feedback is first collected in the *Feedback Collecting* procedure. Then, E^3 adjusts the parameters of preference model in the *Model Parameter Calibration* procedure according to the feedback. Finally, the *Preference Model Training* procedure is invoked to regenerate the preference model.

In the rest of this section, we will elaborate the details of each stage accordingly.

4.3 Initiating Stage

The preference model of a specific user should be established by a preference learning program, before any actual change on the frame rate strategy. In this subsection, we first present the scrolling speed distribution based user preference collecting, then discuss the preference model generation details.

4.3.1 *Scrolling Speed Distribution based User Preference Collecting*

We know that the Logarithmic model precisely describes the user preference on frame rate. To accurately estimate parameters of the Logarithmic model for a particular user, ideally, it requires a learning program to sample the satisfied frame rates of the user under a large variety of scrolling speeds for a sufficient long period. However, this will cause the user to experience a long un-pleasant model-training period. It would be more user-friendly to probe the satisfied frame rate on a small sampling set.

To this end, E^3 first calculates the probability distribution of the user scrolling speed by logging the interactive operations of the user, as shown in Figure 12. We find that the highest probability density appears with the scrolling speed in range $60 \sim 130$ pixels/second. Based on the observation shown in Figure 12, the probability density of the scrolling speed approximately obeys the log-normal distribution, i.e.,

$$P(s|\mu,\sigma^2) = \frac{1}{s\sqrt{2\pi\sigma^2}} e^{-\frac{(ln(s)-\mu)^2}{2\sigma^2}}. \quad (5)$$

Then, the maximum-likelihood estimation is applied to estimate the value of parameter μ and σ.

With the log-normal distribution, sampling points $\{x_{1..n}\}$ are chosen according to the principle that the higher the probability density, the denser sampling. More specially, all x_i that satisfies

$$\int_0^{x_i} P(s|\mu,\sigma^2) = \frac{i}{n+1} \quad (6)$$

are chosen to be sampling points. Finally, Algorithm 1 is used to collect $\{r_{min}\}$ corresponding to $\{x_{1..n}\}$.

The rationale behind this principle is that higher probability of a scrolling speed means that the user prefers to scroll under such speed. Therefore, the model should be trained more biased towards that speed in order to more accurately capture the behavior of the user. With this sampling strategy, E^3 can efficiently reduce the required number of sampling points and obtain more precise model configurations.

4.3.2 *Preference Model Training*

With samples on the satisfied frame rate curve from the *User Preference Collecting* procedure, E^3 conducts the least-square regression analysis to calculate the preference model

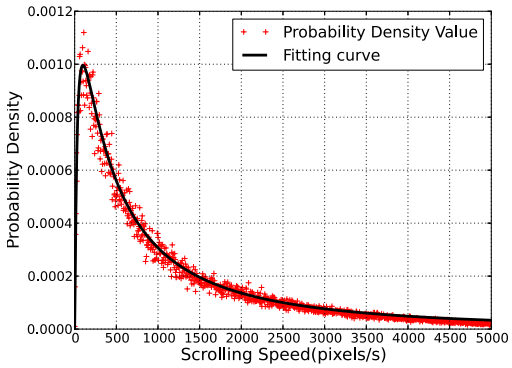


Figure 12: Probability density distribution of scrolling speeds.

Algorithm 1 PREFERENCE COLLECTING($\{x_{1..n}\}$)

Require: A sequence of scrolling speeds $\{x_{1..n}\} > 0$.
Ensure: The satisfied frame rates $\{r_{min}\}$ corresponding to $\{x_{1..n}\}$.

for each $x_i \in \{x_{1..n}\}$ **do**
  $F_{max} \leftarrow 60, F_{min} \leftarrow 0$
  while $F_{max} - F_{min} > 1$ **do**
    $F_{curr} \leftarrow (F_{max} + F_{min})/2$
    if ACCEPTABLE(F_{curr}) **then**
      $F_{max} \leftarrow F_{curr}$
    else
      $F_{min} \leftarrow F_{curr}$
    end if
  end while
  $r_{min}(x_i) \leftarrow F_{max}$
end for
return r_{min}

parameters. Getting the preference model, the model parameters are stored in *User Preference Model* for further usage.

In order to maximize energy saving and guarantee user experience simultaneously, it is necessary to evaluate the impact of frame rate on both user experience and energy saving. We thus design a new least-square regression estimator, called E^2RSS, which can be used to adjust the trained Logarithmic model.

More specifically, when the predicted satisfied frame rate from a model is lower than the frame rate that user demands, there is a user experience loss, which is defined as:

$$ExpLoss = \beta \times (r_{min}(s_i) - f(s_i)), \quad (7)$$

where $r_{min}(s_i)$ is the user satisfied frame rate at scrolling speed s_i and $f(s_i)$ is the predicted satisfied frame rate at speed s_i. Similarly, when the predicted satisfied frame rate is higher than the frame rate that user demands, the extra frame rate will be an energy waste, which is defined as:

$$EngLoss = (1 - \beta) \times (f(s_i) - r_{min}(s_i)). \quad (8)$$

β in Equation (7) and Equation (8) is a weight which represents the balance factor between energy and user experience, we will further discuss it in Section 4.5.

Based on Equation (7) and Equation (8), we define *Energy-Experience-Residual* (E^2R) to evaluate the accuracy and efficient of a model at one specified scrolling speed. E^2R is defined as:

$$E^2R = \alpha P_j(s_i) \times \begin{cases} ExpLoss^2 & if\ r_{min}(s_i) > f(s_i) \\ EngLoss^2 & if\ r_{min}(s_i) < f(s_i), \end{cases} \quad (9)$$

where $P_j(s_i)$ is the probability density of user j's scrolling speed at s_i, α is an amplification coefficient constant that used to neutralize the attenuation of $P_j(s_i)$. A higher $P(s_i)$ indicates a user uses s_i for scrolling more frequently. Thus, $r_{min}(s_i)$ at the scrolling speed that has a higher $P(s_i)$ is obviously more significant than the one has a lower $P(s_i)$. With $P(s_i)$, E^2RSS gives estimations that are more precise on $ExpLoss$ and $EngLoss$.

Energy-Experience-Residual sum of squares (E^2RSS) is an energy-aware and experience-aware estimator to evaluate the accuracy and efficiency of regression, which is defined as:

$$E^2RSS = \frac{1}{n}\sum_{i=1}^{n} E^2R_i. \quad (10)$$

E^2RSS produces a more precise and user-aware model than RSS. Therefore, we take E^2RSS as E^3's default estimator to replace RSS. In order to keep the user experience un-compromised during frame rate adjustment, E^3 only needs a few samples (usually less than 10) to establish the user-specific preference model. Each sampling takes about $20 \sim 30$ seconds and the whole process helps to establish the model precisely, which benefits to both user experience and energy saving.

4.4 Reacting Stage

After obtained the user preference model, E^3 uses it to generate the optimal frame rate based on the user's scrolling speed in real time. Then, the optimal frame rate is used to replace the default system frame rate for energy saving. In this subsection, we first present the scrolling speed extraction, then discuss the frame rate controlling later.

4.4.1 Scrolling Speed Extraction

In order to dynamically adjusting the frame rate in real-time, E^3 needs to monitor the real-time scrolling speed and calculates the corresponding optimal frame rate constantly. The real-time finger position on touch screen could be obtained, via the interface provided by the operating system. Then, we can get the real-time scrolling speed via the distance of two adjacent finger positions and the time interval.

Due to the sampling errors, the captured scrolling speed may fluctuate significantly. Using the frame rates calculated from such unstable scrolling speed samples will cause fluctuations in frame rate. To deal with this problem, the *Kalman Filter* [12] is used to smooth the sampled value so that the current scrolling speed can be estimated more accurately.

Particularly, in order to use the *Kalman Filter* to estimate the scrolling speed on series of finger position samples, we model the finger position and scrolling speed using the framework of the *Kalman Filter*. Thus the following matrices need to be specified: the state-transition model(F_k), the observation model(H_k), the covariance of the process noise(Q_k), the covariance of the observation noise(R_k), and the control-input model (B_k). In our scheme, suppose e_k is the estimation of finger position and scrolling speed at the

k^{th} sample, which can be denoted as

$$e_k = \begin{bmatrix} p_{xk} & p_{yk} \\ v_{xk} & v_{yk} \end{bmatrix}, \quad (11)$$

where p_{xk} and p_{yk} are the k^{th} estimated finger position value on x and y coordinates. Let t_k be the time that the k^{th} sample is taken, v_{xk} and v_{yk} are the k^{th} estimated scrolling speed on x and y coordinates that are calculated from

$$\begin{aligned} \triangle t_k &= t_k - t_{k-1}, & (12) \\ v_{xk} &= (p_{xk} - p_{x(k-1)}) / \triangle t_k, & (13) \\ v_{yk} &= (p_{yk} - p_{y(k-1)}) / \triangle t_k. & (14) \end{aligned}$$

Base on the *Kalman Filter* framework, the true state of time k can be obtained from the $(k-1)^{th}$ state with

$$e_k = F_k e_{k-1} + B_k u_k + q_k, \quad (15)$$

where F_k is the state-transition model matrix, B_k is the control input model matrix which is applied to the control vector u_k, q_k is the process noise which satisfies $q_k \sim N(0, Q_k)$.

With H_k is the observation model matrix and v_k is the observation noise which is assumed to be zero mean Gaussian white noise with covariance R_k i.e. $r_k \sim N(0, R_k)$. The k^{th} sample value z_k can be expressed as

$$z_k = H_k e_k + r_k. \quad (16)$$

To estimate the position and speed in the k^{th} sample from the $(k-1)^{th}$ sample, it is not hard to get an estimation equation like

$$\begin{aligned} e_k &= \begin{bmatrix} p_{x(k-1)} & p_{y(k-1)} \\ v_{x(k-1)} & v_{y(k-1)} \end{bmatrix} + \triangle t_k \begin{bmatrix} v_{x(k-1)} & v_{y(k-1)} \\ 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 & \triangle t_k \\ 0 & 1 \end{bmatrix} e_{k-1}. \quad (17) \end{aligned}$$

With Equation (17) and (15), we can get the state transition matrix

$$F_k = \begin{bmatrix} 1 & \triangle t_k \\ 0 & 1 \end{bmatrix}. \quad (18)$$

Since there is no certain control-input in estimating scrolling speed, so the control-input model (B_k) is a zero matrix. We also set the observation model(H_k) to a first order identity matrix, because the finger position and scrolling speed can be obtained directly by monitoring the touch screen event. Moreover, the covariance of the process noise(Q_k) and the covariance of the observation noise(R_k) can be obtained from empirical measurements. As a result, the procedure of the smoothing scrolling speed can be described as the two following phases:

Predict phase: Using the state estimate from the previous timestamp to produce an estimate of the state at the current timestamp, which can be expressed as:

$$\begin{aligned} \hat{e}_{k|k-1} &= F_k \hat{e}_{k-1|k-1}, & (19) \\ P_{k|k-1} &= F_k P_{k-1|k-1} F_k^T + Q_k, & (20) \end{aligned}$$

where $\hat{e}_{k|k-1}$ is the posteriori state estimate at the k^{th} input event given observations before and including the k^{th} input event, $P_{k|k}$ is the posteriori error covariance matrix, used to estimate the accuracy of prediction, Equation (19) is the predicted value of scrolling speed in the k^{th} input event, and Equation (20) is the predicted estimate covariance in the k^{th} input event.

Update phase: the current priori prediction is combined with current observation information to refine the state estimation can be expressed as:

$$\begin{aligned} \tilde{y}_k &= z_k - \hat{e}_{k|k-1}, & (21) \\ S_k &= P_{k|k-1} k^T + R_k, & (22) \\ K_k &= P_{k|k-1}^T S_k^{-1}, & (23) \\ \hat{e}_{k|k} &= \hat{e}_{k|k-1} + K_k \tilde{y}_k, & (24) \\ P_{k|k} &= (I - K_k) P_{k|k-1}, & (25) \end{aligned}$$

where $\tilde{y}_k$ is the measurement residual, z_k is the observation of finger position and scrolling speed at the k^{th} input event, S_k is the covariance of residual, I is the identity matrix, and K_k is the *Optimal Kalman Gain*. Equation (24)(25) represent the updating of estimation and estimate covariance. Finally, we use $\hat{e}_{k|k}$ as the scrolling speed in our scheme.

Due to the autoregression characteristic of *Kalman Filter*, E^3 is able to estimate the real scrolling speed more precisely, and further achieve a smooth frame rate changing and fluent display animation. Moreover, the complexity of the *Kalman Filter* algorithm is $O(1)$ for inputting one scrolling speed, so the algorithm does not lead to high overhead.

Furthermore, the real-time scrolling speed is used in the *Optimal Frame Rate Calculation* procedure to get the optimal frame rate in accordance with the *User Preference Model*.

4.4.2 Frame Rate Controlling

The *Frame Rate Controlling* can dynamically adjust the display frame rate to the optimal frame rate. Generally, the process of display update is a 3-steps loop, i.e., *Wake up, Update* and *Sleep*. On each iteration, the control thread *wakes up* and then *update* an image to the screen. After the display is updated, the control thread *sleeps* 1/60 seconds to make sure the frame rate does not exceed the 60fps hardware limit.

As we discussed in Section 3.2, the frequently executed image building procedure is the root cause of high energy consumption during scrolling operations. In order to save energy, the frequency of image building should be reduced. Before the control thread falls asleep, E^3 calculates the optimal frame rate according to the current scrolling speed and preference model, then converts the optimal frame rate to a corresponding time interval and takes it as the sleep time.

More specifically, with E^3 employed, a sleep time changing step is added into the iteration. We take note of the time consumed by update as τ, then the sleep time in this iteration should be

$$T_{sleep} = \frac{1}{r_{min}} - \tau. \quad (26)$$

Instead of just using the $\frac{1}{r_{min}}$ as sleep time, we introduce τ to improve the accuracy of frame rate controlling. Additionally, the changing of frame rate will not exceed 60 times per second. Thus, considering only simple calculations are involved here, we could also infer that the overhead of E^3 is negligible.

4.5 Evolving Stage

Although E^3 uses an accurate model to describe the user preference on scrolling speed, the performance can be improved by considering that the preference of users may change over time. In order to keep the pace with such evolution,

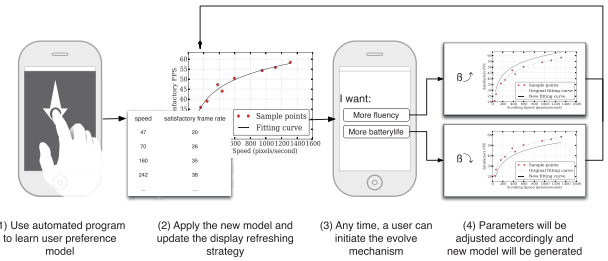


Figure 13: Illustration of E^3 evolving mechanism.

E^3 should provide corresponding mechanism. In particular, we add a balance factor β between energy-efficient and user experience in Equation (7) and (8).

The workflow of E^3's self-evolving is shown in Figure 13. Once the *Initiating Stage* is finished, the preference model that fits the user's demands is established and E^3 starts to adjust the frame rate refer to the real-time scrolling speed. However, after a period of usage, the user may feel unsatisfied with the current configuration. In that case, the user can always use an evolving program to adjust parameters of preference model. All a user has to do is choose more battery life or more user experience.

The functionality of β in E^2RSS is to balance the preference between user experience and energy saving. After the user makes his/her decision, a feedback message (*Energy-Prior* or *User-Experience-Prior*) is sent from the *Feedback Collecting* to the *Model Parameter Calibration*. In the case that a *User-Experience-Prior* feedback is received, the *Preference Model Generator* will adjust β (initially, set to 0.5) to give more weight on user experience. E^3 then updates model parameters according to the message context and finally uses the new estimator E^2RSS to regenerate a new preference model in *Preference Model Training*. Thus, a biased model according to the user feedback is obtained. As time goes by, the user preference model will evolve and eventually, a better model fits the specific user can be achieved.

5. PROTOTYPE IMPLEMENTATION

To test the feasibility of E^3, we build prototypes using different types of android-based smartphones and a tablet. Specifically, we implement E^3 on a Nexus One with Android 2.3, a Nexus S with Android 4.1, a Galaxy S II with Android 4.0, a Nexus Prime with Android 4.0 and a Galaxy Tablet with Android 3.2. We choose Android smartphones/tablet to implement our prototype because Android is an open-source operating system which facilitates the research on it. Our prototypes mainly consist of two parts, i.e., a power meter and E^3 software, as illustrated in Figure 14.

The voltage of smartphones/tablet is measured by an AD convertor and the current is measured by amplifying and observing the voltage drop over a 0.018Ω resistor. The sample rate of the power monitor is set to $1KHz$. We compare the power meter to a high-precision oscilloscope and the result shows that with a sample rate at $1KHz$, the power meter can capture most slight jitters of the smartphone voltage and current. We implement E^3 via developing applications and system modules on Android. The source code of E^3 and tools are available at [1].

We first conduct an experiment to measure the power consumption during browsing a website with and without E^3 using a Nexus S smartphone. The results are shown in Figure 15, where the curves from $0s$ to $10s$ show the energy consumption caused by loading of the browser and the webpage, and curves from $10s$ to $60s$ show the energy consumption during the surfing with ten scrolling operations. The bottom area of both curves represents the energy consumption for screen display. Clearly, it can be seen that, without E^3, the energy consumption caused by scrolling is much higher than that of loading, taking up most of the overall energy consumption. In contrast, with E^3 enabled, the energy consumption caused by scrolling is significantly reduced. Further analysis on the data of this plot shows that E^3 saves up to 63.2% of the energy consumed by CPU and achieves a 36.1% energy-saving in the overall energy consumption. Moreover, it can be seen that there is a close match between the two curves during the first 10 seconds, which indicates there is no difference in the energy consumption with or without E^3 enabled during the loading. This states that the overhead of monitoring the scrolling speed in E^3 is negligible.

Figure 16 shows the results when using different devices. The energy consumption without E^3 on each device is normalized to 100% and the bars with deepest color represent energy consumed by screen. It is clear that E^3 can achieve considerable overall energy savings on multiple smartphones (35.8% on Nexus One and 37.5% on Nexus Prime) and gets a good overall energy saving on the tablet (29% on GALAXY Tab). The reason that E^3 gets higher energy saving ratio on smartphones than the tablet is because tablets have bigger screens that consume much more energy.

We then present the energy saving of E^3 while surfing on a series of websites, as shown in Figure 17. The bars represent the CPU energy consumption with and without E^3 while browsing these websites, we observe that E^3 significantly reduces the CPU power to 55% in average. Furthermore, the line shows the overall energy saving achieved by E^3. From this line, we observe the overall energy saving is up to 32%. One interesting observation we have is that the energy saving while browsing Twitter is not as high as the other websites. Through experiments, we find that the frame rate while browsing Twitter is extremely low, which leaves little room for E^3 to improve the energy-efficiency. Even though, E^3 still can save 9.8% of the CPU energy consumption while browsing Twitter.

The results of the above experiments show that E^3 significantly reduces the power consumption caused by scrolling for different types of smartphones/tablet. We will thoroughly evaluate E^3 via extensive experiments in the following section.

6. PERFORMANCE EVALUATION

In this section, we evaluate the performance of E^3 by analyzing real traces collected from 327 volunteers randomly selected on college campus. During the experiments, each volunteer received two smartphones with and without E^3, and power meters attached to both phones. We note that the volunteers are not aware of which smartphone is embedded with E^3. Then, each volunteer is asked to use different applications (around 10 minutes for each App) on the two smartphones.

We first evaluate the impact on both user experience and energy efficiency of E^3 in various smartphone applications. After that, we discuss the impact of the E^2RSS. Finally, the overhead of E^3 is analyzed.

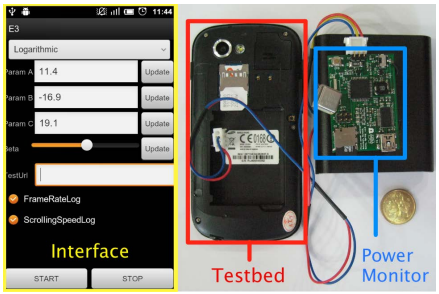


Figure 14: User interface of E^3, measurement tools and testbeds.

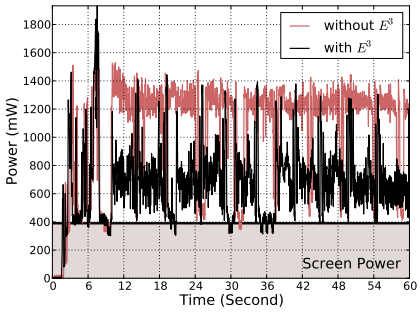


Figure 15: Energy consumption during one web surfing with and without E^3.

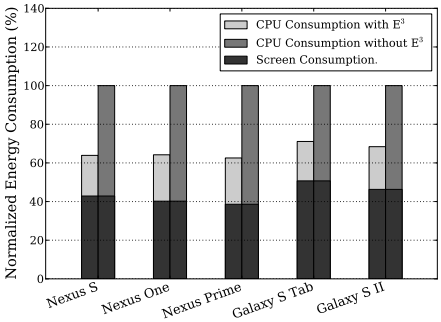


Figure 16: Energy consumption on multiple devices during a web surfing with and without E^3.

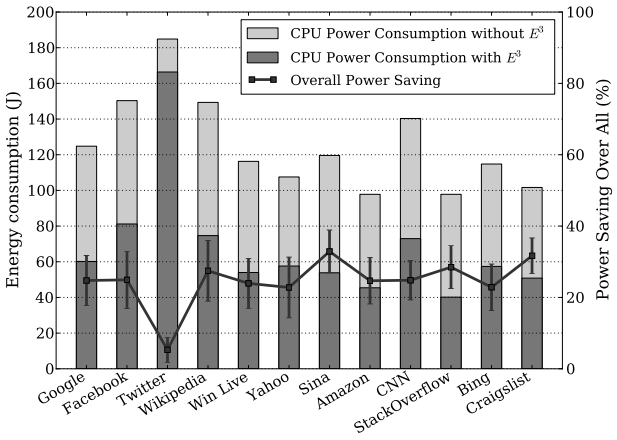


Figure 17: Energy-saving of E^3 while surfing on a series of websites.

6.1 Impact on User Experience of E^3

The volunteers are requested to grade the experience difference of the two smartphones. Our grading system has ten levels, in which the level ten indicates that there is no user experience difference between the two kinds of smartphones and grade level declines as the user experience difference increases.

From the collected traces, the average grades of user experience under the different applications and models are shown in Figure 18. We observe that in each model and application, E^3 with *Kalman Filter* and E^2RSS has the highest user experience grade. For example, the Browser application with the Logarithmic model gets 9.3 in average. In contrast, we notice that the Linear model has a worse user experience, e.g., the average user experience grade on the Linear model with *Kalman Filter* and E^2RSS is 7.2 in Browser. This is because the curve of the Linear model is far from the true user preference. Besides, the Inverse and the Sqrt model also have lower user experience grades than the Logarithmic model, with the same reason as the Linear model. Therefore, Logarithmic model has the best performance on user experience among these four models.

We also find that the Logarithmic model with *Kalman Filter* and E^2RSS has higher grade than the Logarithmic model without *Kalman Filter* or E^2RSS. The user experience is distinctively reduced for all the models without *Kalman Filter* and E^2RSS. Moreover, we observe that, in browser application, there are huge disparities (almost 6 levels) on experience grades in the Linear model with or without E^2RSS. This is due to the convex distribution of user preference pattern and a mismatching model from the standard least-squares regression will produce lower predictions on frame rates than the user demands. On the contrary, E^2RSS introduces the probability distribution of scrolling speed, thus E^3 with E^2RSS achieves more accurate prediction on user preferences. Similar results can be found in other applications, such as Facebook App, Reader and Google Map, indicating that E^3 with E^2RSS achieves a good user experience across different applications.

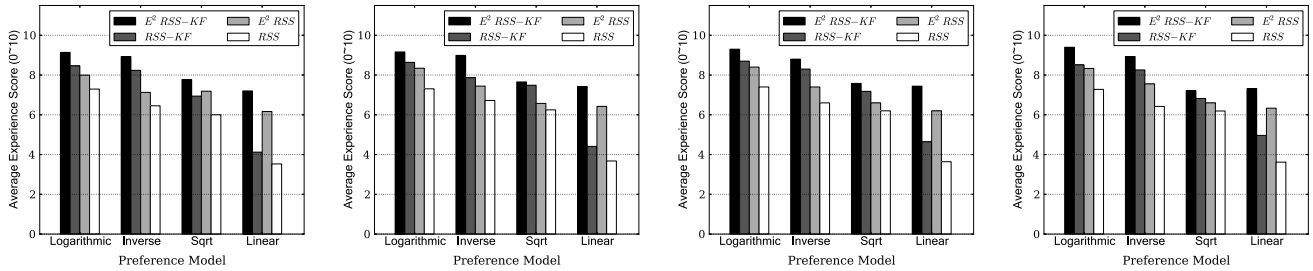
We next compare the performance of our user-specific models and the uniform models, which utilizes all users' samples to calibrate the parameters of a universal model. We note that both the user-specific and uniform models are trained with the E^2RSS and applied with KF. The results are shown in Figure 19. From this figure, we observe that the user-specific models always get higher scores than the uniform models. This is because the uniform models are approximate to the "average" of all the user-specific models, thus nearly half of volunteers may feel that the display becomes not fluent. As a result, the experience grades of uniform models are lower than the user-specific models, which indicates it is necessary to use the user-specific so as to keep the user experience un-compromised.

Based on the results and analysis above, we conclude that E^3 can still give consideration to the user experience while adjusting the frame rate.

6.2 Impact on Frame Rate and Energy-Efficiency of E^3

We further evaluate the impact on frame rate and energy efficiency of E^3 over different applications. Note that without specification, the user preference model is the Logarithmic model, the estimator of E^3 is E^2RSS and *Kalman Filter* is enabled.

Figure 20 shows CDF of frame rate while using four popular applications with E^3. For the most scrolling-intensive application, Browser (according to the results in Figure 3), E^3 successfully reduces the frame rate to 30fps for more than 70% volunteers. For Google Map, frame rates in 80% cases are reduced to less than 40fps. The results tell us that E^3 could reduce $1/3 \sim 1/2$ display updates for most users in daily-used applications.



(a) Browser (b) Reader (c) Facebook (d) Google Map

Figure 18: Average grades of user experience in different models and applications.

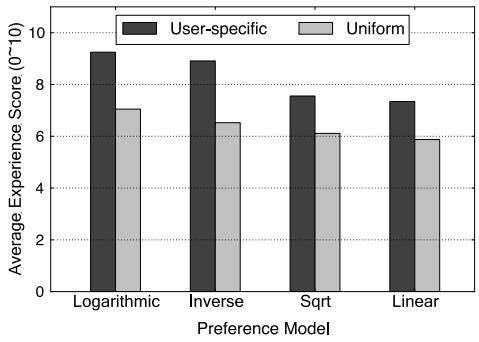


Figure 19: Average grades of user experience in user-specific and uniform models

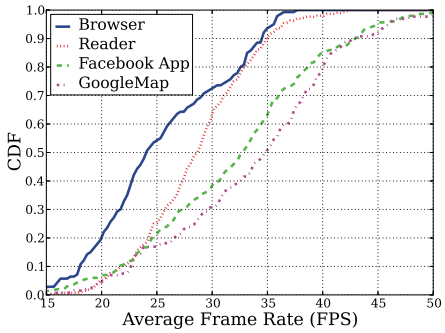


Figure 20: CDF of frame fate in four popular applications with E^3.

Figure 21 plots CCDF (complementary cumulative distribution function) of energy saving in four applications achieved by E^3. In this figure, take Reader for example, E^3 can save more than 45% CPU energy consumption and more than 25% overall energy consumption (CPU, Radio, Screen and Sensors energy consumption) for 70% volunteers. Also, E^3 saved over 30% of CPU energy saving and more than 15% of overall energy consumption for 70% users. Similarly, E^3 achieved significant energy saving in the other applications. In summary, E^3 realize a remarkable energy saving on both CPU and overall energy consumption across multiple popular applications.

6.3 Impact of E^2RSS

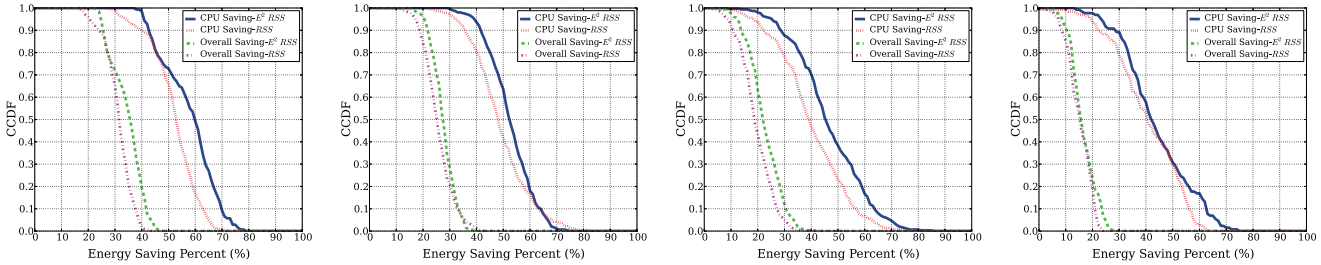
In order to evaluate the impact of E^2RSS, we conduct experiments to compare the energy saving of E^3 with or without E^2RSS in Figure 21. For most of the cases, E^3 with E^2RSS can achieve a higher energy saving than E^3 with RSS. For example, 50% volunteers in Browser application, E^3 with E^2RSS saves more than 60% CPU energy consumption, in contrast, only about 53% energy saving for E^3 with RSS. This indicates that the introduction of E^2RSS improves the energy efficiency of E^3.

We use the normalized energy saving here, to compare the impact of the balance factor β under the same criterion. The normalization is done by uniformly mapping the energy saving interval into [0..10], while β changes from 0.1 to 0.9. The minimum energy saving is mapped to 0 and the maximum is mapped to 10. The meaning of user experience scores are the same with the grades in Figure 18. The normalized energy saving and user experience of four applications are shown in Figure 22. It can be seen that, in all cases, with β changes from 0.1 to 0.9, the user experience shows an uptrend while the energy saving shows a downtrend. In E^2RSS, a higher β value means that more weight is given to the user experience and vice versa. We can also see that, after β exceeded 0.5, the growth of user experience is limited. In contrast, the energy-efficiency downward trend is relatively well-distributed with the growth of β. This result shows that when $\beta = 0.5$, we could get an acceptable user experience while keeping a good energy-saving. Therefore it is better to set the initial value of β to 0.5.

To further investigate the impact of E^2RSS on E^3, we collect traces on both energy saving and user experience grades. Figure 23 illustrates the energy saving on both CPU and overall energy consumption, as well as the user experience grades, under four different models with RSS or E^2RSS. In Facebook App, we find that the Logarithmic model with E^2RSS realizes considerable energy saving on both CPU (56%) and overall (27%) energy consumption while keeping a high user experience grades (almost level ten) at the same time. Compared to the Logarithmic model with E^2RSS, no other model can achieve a higher energy saving while keeping an acceptable user experience. For example, although the Linear model with RSS achieves a good energy saving, the user experience is too low to be accepted. Through E^2RSS estimator, the Linear model remarkably improves the user experience grades, but simultaneously, the energy saving is declined significantly. Except the Linear model, the other models with E^2RSS can achieve both a higher energy saving and a better user experience than the same models with RSS. Also, we get similar results from the other applications. Therefore, E^2RSS is an ideal model estimator for both energy-efficiency and user experience.

6.4 Overheads Analysis

Although E^3 monitors the user input in real-time and adjusts the frame rate according to the scrolling speed, it



(a) Browser (b) Reader (c) Facebook (d) Google Map

Figure 21: CCDF(complementary cumulative distribution function) of energy saving in CPU and overall in four popular applications with E^3.

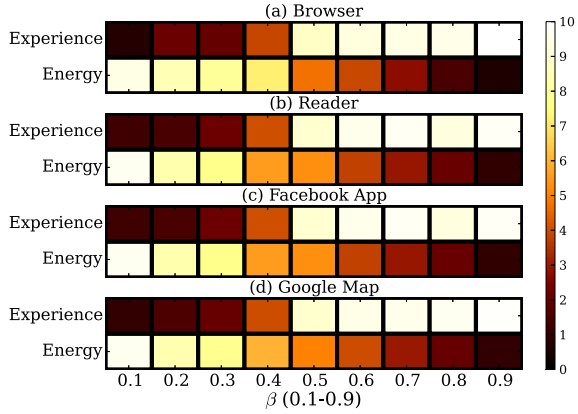


Figure 22: Impact of β on energy saving and user experience in different applications.

causes only negligible overheads. First, when there is no interaction on touch-screen, E^3 causes no computation so that no energy is consumed by E^3. Second, when there is scrolling operations, the event catching, *Kalman Filter* and optimal frame rate calculation have only $O(1)$ computational complexity, while the CPU energy consumption can be reduced up to 58% by E^3. As a result, E^3 is lightweight and the overhead could be ignored.

7. CONCLUSION & FUTURE WORK

In this paper, by analyzing the real traces, we have found that scrolling operation consumes a great amount of energy on smartphones. By further investigation, we have found that the satisfied frame rate is far less than the system default frame rate and the Logarithmic model precisely describes the relationship between the satisfied frame rate and the scrolling speed. We have proposed a scrolling-speed-adaptive frame rate controlling system, E^3, which significantly reduces the power consumption caused by the scrolling operations while keeping the user experience un-compromised. We have implemented E^3 on several types of smartphones and a tablet. Extensive experiment results demonstrated the efficiency of E^3 design.

Following the current research, there are two possible directions for future work on E^3. First, more real traces can be collected with E^3 App to improve the design and implementation of E^3. Second, a self-constructive user preference learning can be designed to automatically extract the user preference model from the daily operations on smartphones.

8. REFERENCES

[1] E^3 testbeds and source code. Available: `http://www.cs.sjtu.edu.cn/~jdyu/research/E3/index.html`

[2] B. Anand, K. Thirugnanam, J. Sebastian, P. G. Kannan, A. L. Ananda, M. C. Chan, and R. K. Balan. Adaptive display power management for mobile games. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 57–70, Bethesda, Maryland, USA, 2011.

[3] P. K. Athivarapu, R. Bhagwan, S. Guha, V. Navda, R. Ramjee, D. Arora, V. N. Padmanabhan, and G. Varghese. Radiojockey: mining program execution to optimize cellular radio usage. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, Mobicom '12, pages 101–112, Istanbul, Turkey, 2012.

[4] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 280–293, Chicago, Illinois, USA, 2009.

[5] F. Bellosa, A. Weissel, M. Waitz, and S. Kellner. Event-driven energy accounting for dynamic thermal management. In *Proceedings of COLP'03*, New Orleans, USA, Sept. 27 2003.

[6] K. Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. In *Proceedings of the conference on Design, automation and test in Europe - Volume 1*, DATE '04, pages 10004–, Washington, DC, USA, 2004.

[7] CNN.com. Battery life concerns mobile users. Available: `http://www.cnn.com/2005/TECH/ptech/09/22/phone.study`

[8] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of MobiSys'10*, pages 49–62, San Francisco, USA, 2010.

[9] M. Dong and L. Zhong. Chameleon: a color-adaptive web browser for mobile oled displays. In *Proceedings of the 9th international conference on Mobile systems,*

(a) Browser (b) Reader

(c) Facebook (d) Google Map

Figure 23: Energy saving of CPU and overall, user experience grades under different models with RSS or E^2RSS in four popular applications.

applications, and services, MobiSys '11, pages 85–98, Bethesda, Maryland, USA, 2011.

[10] M. Dong and L. Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 335–348, Bethesda, Maryland, USA, 2011.

[11] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of ISCA '07*, pages 13–23, San Diego, USA, 2007.

[12] J. Hamilton. *Time series analysis*, volume 2. Cambridge Univ Press, 1994.

[13] D. Le and H. Wang. An effective feedback-driven approach for energy saving in battery powered systems. In *Proceedings of Quality of Service (IWQoS), 2010 18th International Workshop on*, pages 1–9, 2010.

[14] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

[15] J. McCarthy, M. Sasse, and D. Miras. Sharp or smooth?: comparing the effects of quantization vs. frame rate for streamed video. In *Proceedings of the SIGCHI '04*, pages 535–542, Vienna, Austria, 2004.

[16] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, Mobicom '12, pages 317–328, Istanbul, Turkey, 2012.

[17] MobiThinking.com. Global mobile statistics 2012 part a: Mobile subscribers; handset market share; mobile operators. Available: `http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/`

[18] A. Pathak, Y. C. Hu, and M. Zhang. Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pages 5:1–5:6, Cambridge, Massachusetts, 2011.

[19] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of EuroSys'11*, pages 153–168, Salzburg, Austria, 2011.

[20] A. Pathak, A. Jindal, Y. C. Hu, and S. P. Midkiff. What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps. In *Proceedings of MobiSys'12*, pages 267–280, Low Wood Bay, Lake District, United Kingdom, 2012.

[21] F. Qian, Z. Wang, Y. Gao, J. Huang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Periodic transfers in mobile applications: network-wide origin, impact, and optimization. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 51–60, Lyon, France, 2012.

[22] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Top: Tail optimization protocol for cellular radio resource allocation. In *Proceedings of Network Protocols (ICNP), 2010 18th IEEE International Conference on*, pages 285–294, Kyoto, Janpan, 2010.

[23] D. Rajan, R. Zuck, and C. Poellabauer. Workload-aware dual-speed dynamic voltage scaling. In *Proceedings of Embedded and Real-Time Computing Systems and Applications*, pages 251–256, Sydney, Australia, 2006.

[24] R. Rao, S. Vrudhula, and D. Rakhmatov. Battery modeling for energy aware system design. *Computer*, 36(12):77 – 87, Dec. 2003.

[25] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. N. Padmanabhan. Bartendr: a practical approach to energy-aware cellular data scheduling. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, MobiCom '10, pages 85–96, Chicago, Illinois, USA, 2010.

[26] N. Vallina-Rodriguez, P. Hui, J. Crowcroft, and A. Rice. Exhausting battery statistics: understanding the energy demands on mobile handsets. In *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*, MobiHeld '10, pages 9–14, New Delhi, India, 2010.

[27] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Profiledroid: multi-layer profiling of android applications. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, Mobicom '12, pages 137–148, Istanbul, Turkey, 2012.

[28] W. Yuan, K. Nahrstedt, S. Adve, D. Jones, and R. Kravets. Grace-1: cross-layer adaptation for multimedia quality and battery energy. *IEEE Transactions on Mobile Computing*, 5(7):799–815, 2006.