Sensing Human-Screen Interaction for Energy-Efficient Frame Rate Adaptation on Smartphones

Jiadi Yu, *Member, IEEE*, Haofu Han, Hongzi Zhu, *Member, IEEE*, Yingying Chen, *Senior Member, IEEE*, Jie Yang, *Member, IEEE*, Yanmin Zhu, *Member, IEEE*, Guangtao Xue, *Member, IEEE*, and Minglu Li

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 percent 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 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 60 percent of the energy consumed by CPU and 35 percent of the overall energy consumption.

Index Terms—Energy efficiency, user experience, touch screen, scrolling operation, frame rate

1 Introduction

Smartphones have gained large popularity rapidly over the recent years. Almost one billion smartphones have been sold since 2009 [17]. The huge popularity of smartphones is due to the availability of millions of appealing applications, bringing users new style of information sharing, efficient communications, and plenty of entertainment. In order to provide more functionality, new features and powerful hardwares have been constantly embedded into smartphones. The improvements in battery capacity, however, have not kept the pace with the increasing demand for energy posed by applications [24]. Enhanced features and the complex operations performed by smartphones are rapidly draining the limited onboard battery life. Longer battery life has been considered as the most important feature to smartphone users [7].

Existing studies on extending smartphone battery life mostly focus on saving power consumed by CPU computation [6], [23], radio usage [4], [21], [25], and application activities [18], [20]. Furthermore, power consumption could be reduced by adjusting screen display parameters such as LCD backlight level [2] and OLED color scheme [9]. 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 20 input-events per second, which results in more than 30 percent time for human-screen (i.e., touch-screen) interactions when using applications like browsing and gaming. 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.

To illustrate the significance of human-screen interactions, we measure the power consumption of a smartphone, Nexus S, during one typical web surfing process, as shown in Fig. 1. 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 percent of the total energy consumption, whereas loading the web browser (triggered by a click) and open the webpage only consume 6.3 percent 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

- *J. Yu, H. Han, H. Zhu, Y. Zhu, G. Xue, and M. Li are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: {jiadiyu, hanhaofu, hongzi, yzhu, gt_xue, mlli}@sjtu.edu.cn.*
- *Y. Chen is with the Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ 07030. E-mail: yingying.chen@stevens.edu.*
- *J. Yang is with the Department of Computer Science, Florida State University, Tallahassee, FL 32306. E-mail: jie.yang@cs.fsu.edu.*

Manuscript received 7 Mar. 2014; revised 17 June 2014; accepted 8 Aug. 2014. Date of publication 23 Sept. 2014; date of current version 29 June 2015. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TMC.2014.2352862

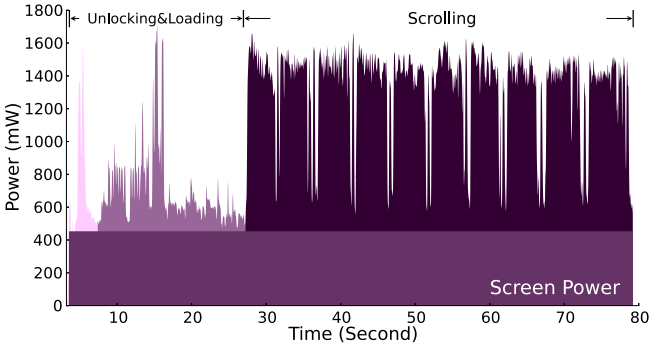

Fig. 1. Energy consumption of screen scrollings during a web surfing on Nexus S.

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 empirically investigate the impact of human-screen interactions to the power consumption on mobile devices including smartphones and tablets through over 300 volunteers. Specifically, 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. Our experiment results show the amount of energy consumed by scrolling can reach up to 50 percent on average over a large range of applications.

Based on our observations, we propose *energy-efficient engine*(E^3), an innovative frame rate adaptation system, which can adaptively adjust the satisfied frame rate without compromising the user experience. 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 find that Logarithmic model can accurately capture the relationship between the screen-scrolling speed and satisfied frame rate based on least-squares regression analysis. We employ an uniform satisfied frame rate model for all users. To build this model, E^3 analyzes the traces of over 300 volunteers and generates a fitting logarithmic curve that satisfies most of users. After a user installs E^3 on a smartphone, the user utilizes the uniform model as his/her user preference model to describe the relationship between the scrolling speed and the satisfied frame rate. Since E^3 generates the uniform model from the traces of around 300 volunteers, the model may not satisfy all the users.

Thus, E^3 employs an cloud server to optimize the user preference model. E^3 provides corresponding mechanism to build a user preference model for each user, i.e., user-specific satisfied frame rate model, and then sends it to E^3 cloud server to accumulate more user preference traces. E^3 cloud server is able to collect these user preference traces continuously to update the uniform satisfied frame rate. When E^3 is installed after a period of time, it can generate a user specific frame rate model based on the scrolling data of the user during this period, and such a model can better fit the user preference.

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 implement our system on five different mobile devices including smartphone and tablet. Our prototype of E^3 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 35 percent of the overall energy consumption while keeping a user satisfaction rate over 88 percent.

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 in Section 7.

2 Related Work

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

Smartphone power model. Many efforts have been made to improve the accuracy of smartphone energy profiling. In the earlier work, the energy estimation relay on external hardware [11], [5]. Dong and Zhong [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 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 recent research work. Balasubramanian 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 [21], [22] on tail energy. Moreover, Athivarapu et al. explored a method to reduce the radio usage by monitoring the program execution pattern [3]. Furthermore, 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

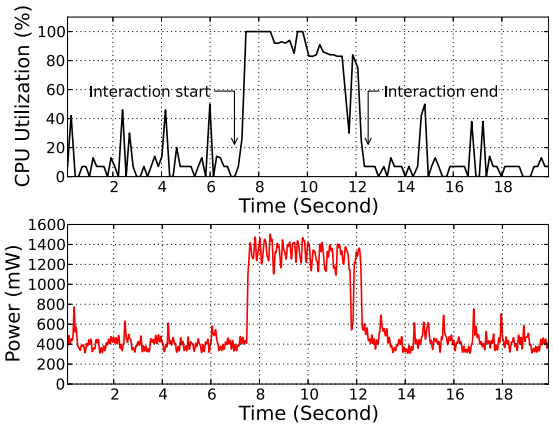

Fig. 2. CPU utilization and energy consumption during one scrolling operation on a webpage.

consumption of different types of Apps in [26]. In addition, [13] discussed the impact on energy consumption of cooperation between Apps and operating system. Pathak et al. [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 interaction 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 AngryBirds 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 Fig. 2. It is clear to see that, once

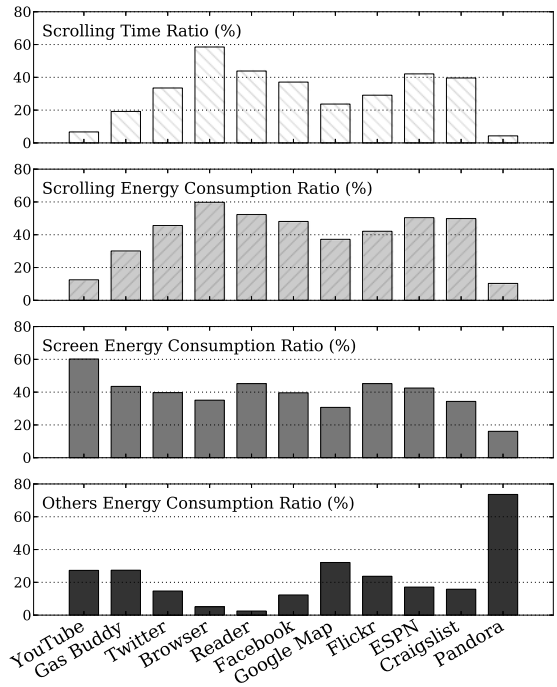

Fig. 3. Scrolling time ratio and energy consumption ratio of several popular smartphone applications.

the scrolling starts, the CPU utilization immediately increases to 100 percent. 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 installed on our test devices (i.e., Nexus One, Nexus S, Nexus Prime, Galaxy S II, Galaxy Tab) for 10 minutes. Fig. 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 percent (For browser, it's almost 60 percent).

We further 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],

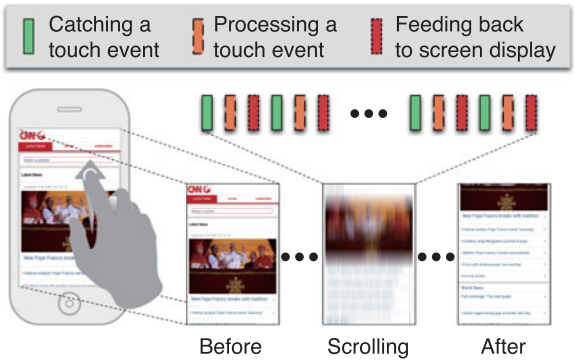

Fig. 4. Illustration of display updates during a scrolling operation.

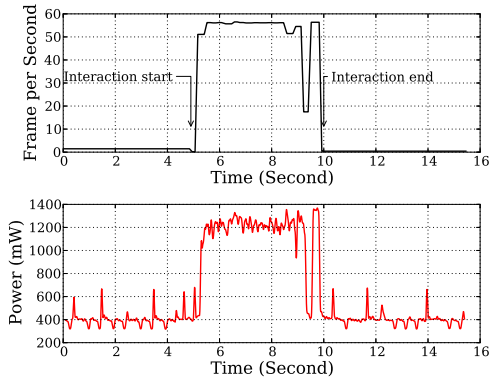

Fig. 5. System frame rate and power consumption during a scrolling operation.

[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 percent 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 percent 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 due to 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 Fig. 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 percent 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 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. Fig. 4 illustrates the workflow of display updates in one scrolling operation. During each scrolling, the three procedures iterate 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. Fig. 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 $60 fps$ (the hardware and software upper bound of the frame rate) or the CPU utilization of 100 percent is reached. The results in Fig. 5 show that the energy consumption has the obvious same changing trend with the frame rate. We also get the same result on our other test devices. The results confirm our inference that the energy consumption is highly related to the frame rate.

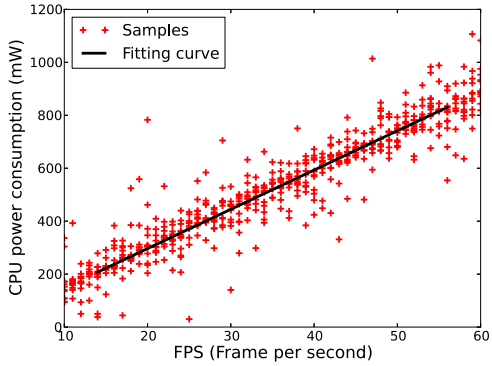

Fig. 6. Relationship between frame rate and CPU power consumption.

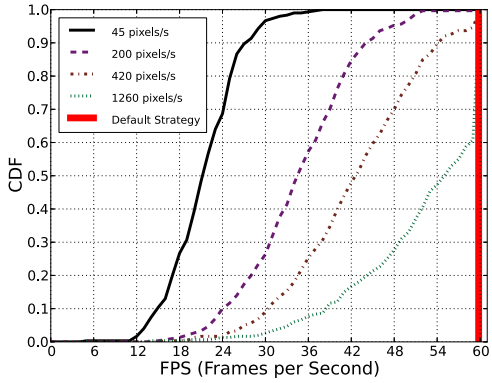

Fig. 7. CDF of the satisfied frame rate in different scrolling speeds.

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. Fig. 6 shows the scatter plot of CPU power consumption with frame rate varying from 10 to 60 fps. We observe that the CPU power consumption is linearly proportional to the frame rate and the power consumption raises from 150 to almost 1,000 mW while frame rate increases to 60 fps. 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 ith 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:

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

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 $\overline{IDAF} \propto \frac{s \times T}{r \times T} \propto \frac{s}{r}$, 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., 5 fps) to the highest value (i.e., 60 fps) 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. Fig. 7 plots the cumulative distribution function (CDF) of satisfied frame rate at four different scrolling speed, i.e., 45, 200, 420, 1,260 pixels per second.

We find that the satisfied frame rate increases with the increasing of the scrolling speed. For example, at the speed of 45 pixels per second, 80 percent volunteers are satisfied with 23 fps and the satisfied frame rate value increases to 59 fps at the speed of 1,260 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 60 fps), 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.

To discover the relationship between the scrolling speed and the satisfied frame rate, we analyze the trace collected in our field tests. Fig. 8 depicts the scatter plot of the satisfied frame rate versus the corresponding scrolling speed. The inset in Fig. 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.

3.4 Modeling the Appropriate Frame Rate

To further quantify the relationship between the satisfied frame rate and the scrolling speed, we examine four models, as listed in Table 1, which could generate similar

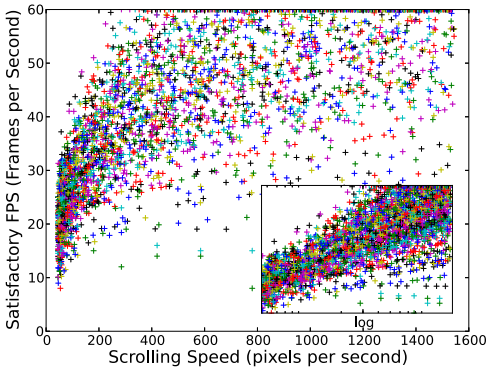

Fig. 8. Satisfied frame rate in different scrolling speeds.

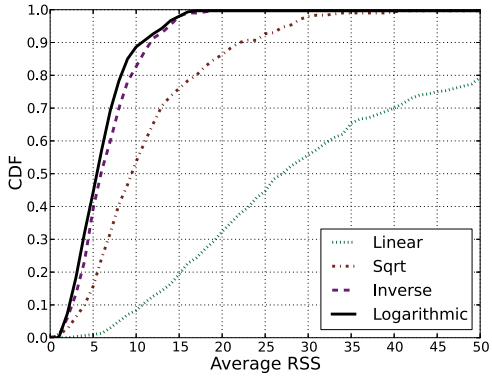

Fig. 9. CDF of RSS on Linear, Sqrt, Inverse, and Logarithmic formula.

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:

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

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 Fig. 9 and list the average RSS value of each model in Table 1. We find that the Linear model 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 screen-scrolling speed and the satisfied frame rate, i.e.,

$$r_{min}(s) = a \times \log(s + b) + c. \tag{3}$$

Based on the Logarithmic model and trace from 327 volunteers, a user preference model, i.e., the satisfied frame rate model, could be achieved.

4 Design of E^3

From the previous investigation, we know that the 60 fps 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., 24 fps in movie), 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 discuss the design details of our system E^3.

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

4.1 Overview

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 could be reduced for energy saving while leaving the user experience un-compromised. In this section, we present an overview of E^3 design. E^3 first starts to monitor user actions performed on the touch-screen and estimates the real-time scrolling speed. After that, each time the screen updates, E^3 adjusts the frame rate based on the real-time scrolling speed and the 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 Fig. 10. It consists of three stages: the *Modeling Stage*, the *Reacting Stage*, and the *Evolving Stage*.

The purpose of the *Modeling Stage* is to build a user preference model in E^3 cloud server. During this stage, E^3 first builds a uniform satisfied frame rate model in E^3 cloud server based on the logarithmic relationship between the screen-scrolling speed and users' satisfied frame rate. After that, the Logarithmic model can be further improved.

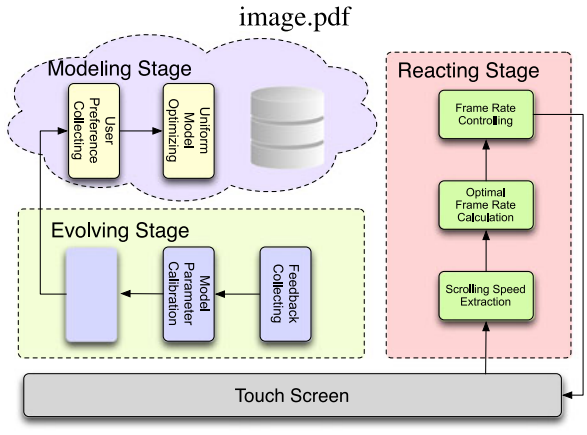

Fig. 10. E^3 architecture.

Finally, the preference model is stored in *Uniform Preference Model*. Additionally, E^3 is able to collect more user preference traces from users' feedbacks to update continuously the uniform satisfied frame rate model in E^3 cloud server. Such an approach enables the uniform model to become more and more accurate and fits all the users better.

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 *Uniform Preference Model* established in the *Modeling 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 users. 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 Improvement* procedure is invoked to regenerate a new specific preference model for the user. After that, the specific model is sent to E^3 cloud server to accumulate more user preference traces, updating the uniform model.

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

4.2 Modeling Stage

In Section 3, we find that the relationship between the screen-scrolling speed and the satisfied frame rate can be described using a Logarithmic model. To build this Logarithmic model, E^3 analyzes the traces of 327 volunteers and generates a fitting logarithmic curve that satisfies most of the users. Specifically, E^3 generates a logarithmic curve which maps a scrolling speed to a frame rate greater than or equal to most users' satisfied frame rate at the scrolling speed.

In the Logarithmic model, the frame rate increases with the increasing of the scrolling speed to guarantee user experience. However, a higher frame rate also causes higher energy consumption. When browsing a webpage, the user could scroll fast on the screen to reach a particular position on the webpage. During this scrolling, the user often skips and pays no attention to the webpage contents before he/she stops scrolling. It is thus unnecessary to adopt a high frame rate during fast scrolling since the frames refreshed are ignored during the scrolling. The Logarithmic model in [29], however, still generates a high frame rate when scrolling fast. Therefore, the Logarithmic model can be further improved to achieve more energy savings based on the scrolling speed.

Specifically, in the improved model, the frame rate is generated according to the previous Logarithmic model when users scroll at a low speed and read the contents displayed on screen, while the frame rate is set to a low fixed value when users scroll at a high speed and pay no attention to the contents displayed on screen. It is needed to find a threshold speed that a user is regarded as paying no attention to the contents displayed on screen if he/she scrolls the screen at a speed higher than it.

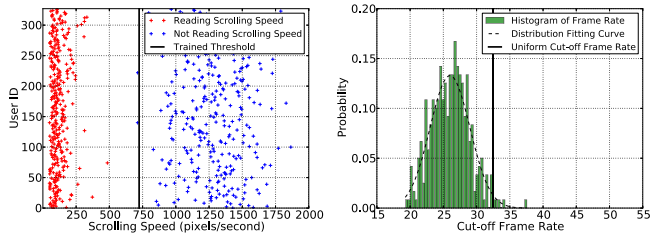

Fig. 11. (a) Scrolling speed while reading or skipping the contents displayed on screen, (b) Volunteers' satisfied frame rate when scrolling and skipping the contents displayed on screen.

In order to find this threshold, 327 volunteers are requested to first scroll the screen when they are reading the contents displayed on screen and next when they just scroll fast and pay no attention to the contents displayed on screen. The scrolling speeds are recorded. Fig. 11a shows the statistical result. It can be seen that there is a large gap of scrolling speeds between the two conditions, and the threshold speed to distinguish the two conditions can be calculated by solving the equation

$$\sum_{i=1}^{n}(x - A_i)^2 = \sum_{i=1}^{n}(x - B_i)^2. \tag{4}$$

Hence,

$$x = \sum_{i=1}^{n}\frac{A_i^2 - B_i^2}{2(A_i - B_i)} = \sum_{i=1}^{n}\frac{A_i + B_i}{2}, \tag{5}$$

where x is the threshold scrolling speed, A is a set containing the scrolling speeds recorded when volunteers are reading the contents displayed on screen, B is a set containing scrolling speeds when volunteers are scrolling fast and paying no attention to the contents displayed on screen, and A_i and B_i is the ith scrolling speed in the set A and B, respectively.

In this experiment, x is solved as 725 pixels/second. So when a user scrolls on screen at a speed higher than 725 pixels/second, the user is regarded as paying no attention to the contents displayed on screen. In that case, the frame rate can be fixed to a low value as long as the user feels satisfied and regards it as a smooth display when fast scrolling.

In order to find a minimum satisfied frame rate (called as a cut-off frame rate), the 327 volunteers are requested to carry out another an experiment. In the experiment, a program is installed into mobile devices detecting the users' cut-off frame rates when they scroll the screen at a high speed and pay no attention to the contents displayed on screen. Fig. 11b shows the distribution of 327 volunteers' cut-off frame rates. Assume that ξ is the random variable representing user's satisfied cut-off frame rate, then ξ is a normal distribution

$$\xi \sim N(\mu, \sigma^2), \tag{6}$$

where μ is the mean or expectation of the distribution and σ is standard deviation. In the normal distribution $N(\mu, \sigma^2)$,

$$P(\xi \leq \mu + 2\sigma) = 97.5\%, \tag{7}$$

which means 97.5 percent of users' satisfied cut-off frame rate is less than or equal to $\mu + 2\sigma$. So the uniform cut-off frame rate in improved model is set to $\mu + 2\sigma = 32.5$ fps (the black solid line in Fig. 11b). Under the frame rate, about 97.5 percent users are satisfied when they just scroll fast and ignore the contents displayed on screen.

Therefore, we could improve the logarithmic model by using the fixed cut-off frame rate (i.e., 32.5 fps) when scrolling speed is greater than 725 pixels/second, while the real-time frame rate is still generated according to the logarithmic model in the case that the scrolling speed is not greater than 725 pixels/second. This improvement is applied in E^3 cloud server, and the improved model is used as an uniform user preference model. When E^3 is installed into a user's mobile device for the first time, it is connected to E^3 cloud server and obtain the latest uniform user preference model. After that, the real-time frame rate is generated according to the uniform user preference model.

Additionally, since E^3 generates the uniform model from the traces of 327 volunteers, the model may not satisfy all the users. Thus, the model needs to be further improved after installation to satisfy user experience. Since different users may have different scrolling habituations, when E^3 is installed after a period of time, it can generate an user specific model based on the scrolling data of the user during this period,and such a model can better fit the user preference. (We provide the detailed information in Section 4.4). Due to this consideration, E^3 keeps tracing user's specific model, and uploads it to E^3 cloud server. According to the original data and the uploaded data, E^3 can update the uniform model automatically in the cloud server so that the model fits all users better. Hence, the uniform model becomes more and more accurate as more user's specific models are collected to E^3 cloud server. The frame rate generated by the uniform model always satisfies most users. Moreover, based on traces from more users, the threshold speed and the cut-off frame rate are updated continuously so that they will also become more and more accurate.

4.3 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 section, we first present the scrolling speed extraction, then discuss the frame rate controlling later.

4.3.1 Scrolling Speed Extraction

In order to dynamically adjust the frame rate in real-time, E^3 needs to monitor the real-time scrolling speed. Assume that there is a pixel line shown on the bottom of the screen at time t, and this pixel line appears in the middle of the screen after scrolling upwards to see the content below. If the distance of these two positions is known, along with the time interval between the two positions, the real-time scrolling speed can be calculated. Specifically, the pixels of an image or a part of an image can be extracted into a pixel sequence line by line. Hence, at time t, all pixels of an image displayed on screen can be expressed as a pixel sequence, and a certain pixel line of the image can be expressed as a subsequence. By matching the subsequence in the pixel sequence of the whole screen, it is easy to obtain the position of the subsequence in the screen pixel sequence at time t. After the scrolling time $\triangle t$, the subsequence has forwarded (or backwarded) for a distance, then its new position in the screen pixel sequence can be obtained at time $t + \triangle t$. Then, the distance that this subsequence has scrolled during $\triangle t$ can be calculated. The scrolling speed during $\triangle t$ is

$$S_{\triangle t} = \frac{|P(l_i, t - \triangle t) - P(l_i, t)|}{\triangle t * R_h}, \tag{8}$$

where S is the scrolling speed during $\triangle t$, l_i is a certain line of an image, $P(m_i, t)$ is the position of l_i in the pixel sequence of the image displayed on screen at time t and R_h is the horizontal resolution (i.e., the number of pixels per line) of the mobile device.

Operating system provides the interface to extract pixels of a frame into a pixel sequence. In our system, the *KMP algorithm* [12] is used to match a subsequence in the whole pixel sequence. The complexity of the algorithm is $O(n + m)$, where n is the length of the whole pixel sequence and m is the length of the subsequence. Since the complexity of other operations is $O(1)$, the complexity of the total process is $O(n + m)$.

However, since an image has many the same lines, it is hard to localize the position of a subsequence in the pixel sequence of the image correctly if the pixels of only one line are used as a subsequence to match. In order to calculate the scrolling distance of the two frames (i.e., two images displayed on screen before and after the scrolling time $\triangle t$), it is necessary to obtain the accurate positions of the matched subsequence in the pixel sequence of both frames. Obviously, the best way is to match the whole common area between the two frames. Nevertheless, since the scrolling speed is calculated in real-time, the time interval $\triangle t$ is very small. So there are just a few lines that have been scrolled out of the screen during $\triangle t$. i.e., the common area between the two frames is almost as large as the whole screen, which results in high cost of matching. For example, given a mobile device with a resolution of 960×640, we have to match a subsequence (common area) whose length is close to $960 \times 640 = 614{,}400$.

In order to decrease this cost, we try to decrease the size of the matching area without influencing the correct rate. To determine the size exactly, experiments are implemented to compute scrolling distance using the above method and compare the computed result with the ground truth. Specifically, the experiments are based on 1,000 random webpages (containing graphs, plain texts, and the mix of both) on three mobile devices (i.e., Nexus One with a processor of QSD8250 1 GHz, Nexus Prime with a processor of OMAP4460 1.2 GHz, and Galaxy Nexus with a processor of Exynos 4210 SOC 2-core 1.2 GHz). The result is shown in Fig. 12. It can be seen from Fig. 12 that the correct rate curves are very close to

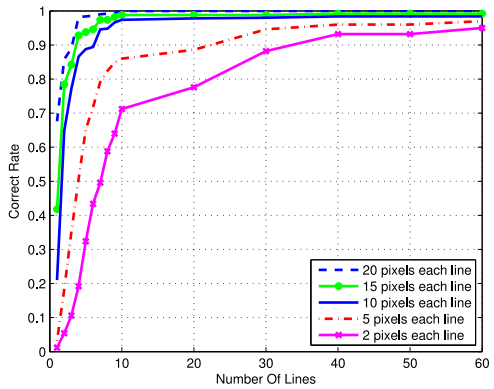

Fig. 12. Correct rate of different area size.

each other with a width (i.e., the number of pixels every line) greater than or equal to 10, while the correct rate curves tend to be stable with a height (i.e., the number of lines) greater than or equal to 10. With this 10×10 pixel^2 as the matched area, we can get an correct rate of 97.4 percent. Hence, an area with the size of 10 pixels every line and 10 lines (i.e., 10×10 pixel^2) is an ideal area to match for computing the scrolling distance. Morever, the total time of computing the real-time scrolling speed once is 1.2 ms, which is fast enough for E^3 since the frame rate is up to 60 fps, and a time interval of 1.2 ms could guarantee the scrolling speed be calculated once as soon as the frame refreshes. The process then calculates the scrolling speed at most 60 times, taking 72 ms in one second. In other words, the CPU overhead is up to 7.2 percent, which is quite small for the overall CPU overhead. Hence, we use a 10×10 pixel^2 area as a matching area to match in the two frames before and after frame refreshing to obtain the accurate scrolling distance during $\triangle t$. After that, the real-time scrolling speed can be calculated according to Equation (8). Furthermore, the real-time scrolling speed is used in the *Optimal Frame Rate Calculation* procedure to obtain the optimal frame rate in accordance with the user preference model.

4.3.2 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 three-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 60 fps 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

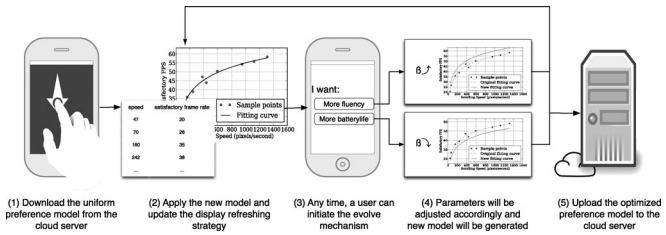

Fig. 13. Illustration of E^3 evolving mechanism.

the time consumed by update as τ, then the sleep time in this iteration should be

$$T_{sleep} = \frac{1}{r_{min}} - \tau. \tag{9}$$

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.

4.4 Evolving Stage

Although E^3 uses an accurate model, the uniform satisfied frame rate model, to describe users preference on scrolling speed, the performance can be improved by considering the preference of a specific user. In order to keep the pace with such evolution, E^3 should provide corresponding mechanism to build a user preference model for each user, i.e., user-specific satisfied frame rate model.

The workflow of E^3's self-evolving is shown in Fig. 13. Based on the user satisfied frame rate model, E^3 adjusts the frame rate based on the real-time scrolling speed. However, after a period of usage, the user may feel unsatisfied with the current configuration, i.e., the user satisfied frame rate model. In that case, the user can always use an evolving program to adjust parameters of preference model. All a user has to do is to choose to have a longer battery life or a better user experience.

In order to guarantee user experience and maximize energy saving simultaneously, it is necessary for the user 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:

$$ExpLoss = \beta \times (r_{min}(s_i) - f(s_i)), \tag{10}$$

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:

$$EngLoss = (1 - \beta) \times (f(s_i) - r_{min}(s_i)). \tag{11}$$

β in Equations (10) and (11) is a weight which represents the balance factor between energy and user experience.

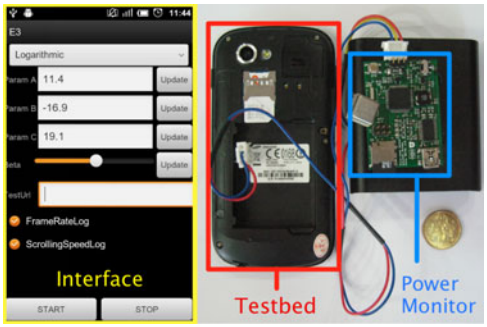

Fig. 14. User interface, measurement tools, and testbeds.

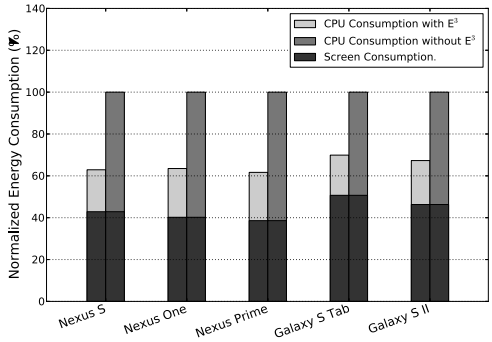

Fig. 15. Energy consumption on multiple devices during a web surfing with and without E^3.

Based on Equations (10) and (11), we define *Energy-Experience-Residual* (E^2R) to evaluate the accuracy and efficient of a user-specific model at one specified scrolling speed. E^2R is defined as:

$$E^2R = \begin{cases} ExpLoss^2 & if\ r_{min}(s_i) > f(s_i) \\ EngLoss^2 & if\ r_{min}(s_i) < f(s_i), \end{cases} \quad (12)$$

E^2R 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

$$E^2RSS = \frac{1}{n}\sum_{i=1}^{n} E^2R_i. \quad (13)$$

E^2RSS produces a more precise and user-aware model than RSS. 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. Meanwhile, the specific user preference model is sent to E^3 cloud server, so that E^3 can update the uniform model based on more user preference traces.

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 Fig. 14. 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 when using different devices. Fig. 15 shows the results when using different devices. The energy consumption without E^3 on each device is normalized to 100 percent 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 (38.2 percent on Nexus One and 39.5 percent on Nexus Prime) and gets a good overall energy saving on the tablet (29 percent 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 Fig. 16. 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 57.3 percent 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 33 percent. 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 percent of the CPU energy consumption while browsing Twitter.

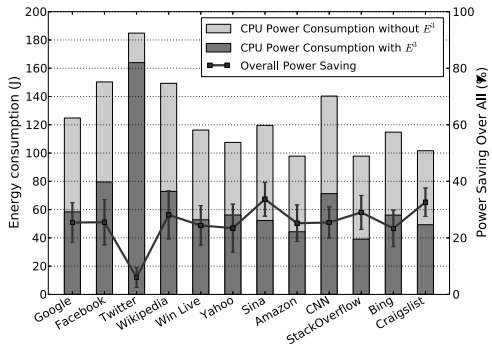

Fig. 16. Energy-saving of E^3 while surfing on a series of websites.

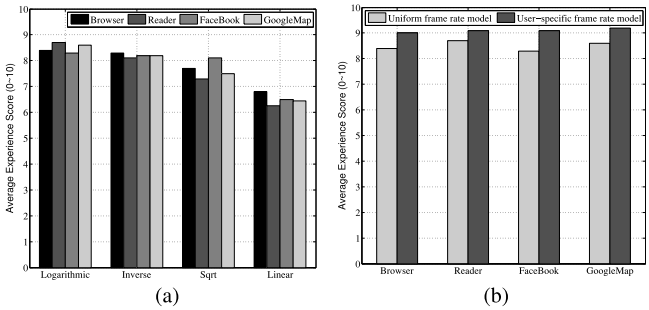

Fig. 17. (a) Average grades of user experience in different models and applications, (b) Average grades of user experience in uniform and user-specific frame rate models.

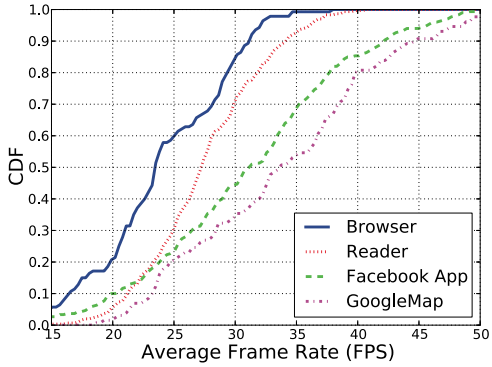

Fig. 18. CDF of frame rate in four popular applications with E^3.

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. 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 balance factor β. Finally, the overhead of E^3 is analyzed.

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 Fig. 17a. We observe that in each model and application, E^3 with the Logarithmic model has the highest user experience grade for each application. For example, the Reader application with the Logarithmic model gets 8.85 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 is 6.2 in Reader. 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. Similar results can be found in other applications, such as Browser, Facebook App, and Google Map, indicating that E^3 achieves a good user experience across different applications. 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.

We next compare the performance of the uniform model to that of the user-specific models [29]. The results are shown in Fig. 17b. From this figure, we observe that although the user-specific models get slightly higher scores than the uniform models, the performance difference between our uniform model and the user-specific models is very small. Since the uniform models are approximated to the "average" of all the user-specific models, the uniform model can satisfy most of users without going through a model-training period, thus providing better user experience. Compared with the uniform model, the user-specific models [29] require the user to go through a model-training period, which could impose an impact on user experience while using E^3-enabled smartphones.

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. Fig. 18 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 Fig. 3), E^3 successfully reduces the frame rate to 27 fps for more than 70 percent volunteers. For Reader, frame rates in 70 percent cases are reduced to less than 30 fps. The results tell us that E^3 could reduce $1/3 \sim 1/2$ display updates for most users in daily-used applications.

Fig. 19 plots complementary cumulative distribution function (CCDF) of energy saving in four applications achieved by E^3. In this figure, take Reader for example, E^3 with the optimized model can save more than 47 percent CPU energy consumption and more than 26 percent overall energy consumption (CPU, Radio, Screen, and Sensors energy consumption) for 70 percent volunteers. Also, E^3 without the optimized model saved over 45 percent of CPU energy saving and more than 23 percent of overall energy consumption for 70 percent 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.

Moreover, we compare the energy saving of four applications achieved by E^3 with the optimized model to that of the

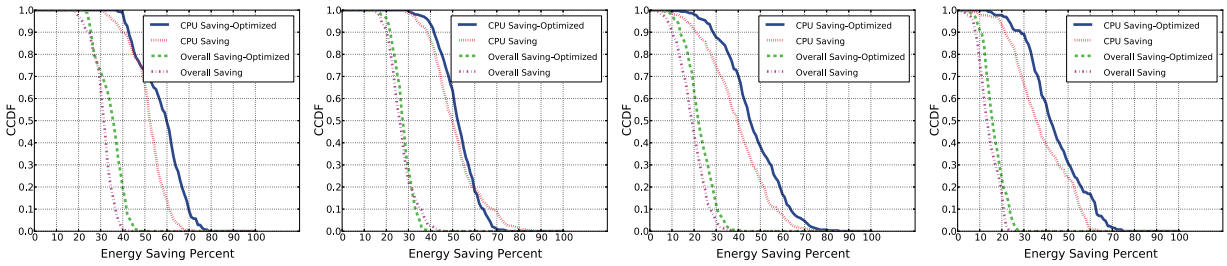

Fig. 19. Complementary cumulative distribution function of energy saving in CPU and overall in four popular applications with E^3.

model in [29]. The results are shown in Fig. 20 and the energy consumption of E^3 with the model in [29] is normalized to 100 percent. It can be seen that E^3 with the optimized model can save more CPU and overall energy consumption than E^3 with the model in [29].

6.3 Impact of Balance Factor β

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 Fig. 17. The normalized energy saving and user experience of four applications are shown in Fig. 21. 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.

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 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 and optimal frame rate calculation have only $O(1)$ computational complexity, while the CPU energy consumption can be reduced up to 60 percent by E^3. As a result, E^3 is lightweight and the overhead could be ignored.

7 Conclusion

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.

Acknowledgments

This work was supported in part by National Science Foundation CNS1217387 and CCF1018270, the 973 Program 2014CB340303, NSFC 61170238, NSFC 61373157, Changjiang Scholar, and Innovative Research Team in University (IRT1158, PCSIRT) China.

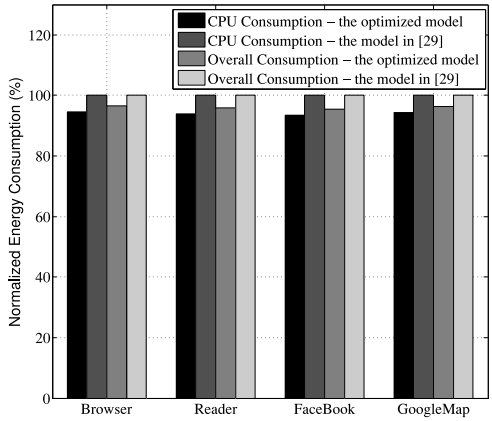

Fig. 20. Energy saving in four applications achieved by E^3 with the optimized model and the model in [29].

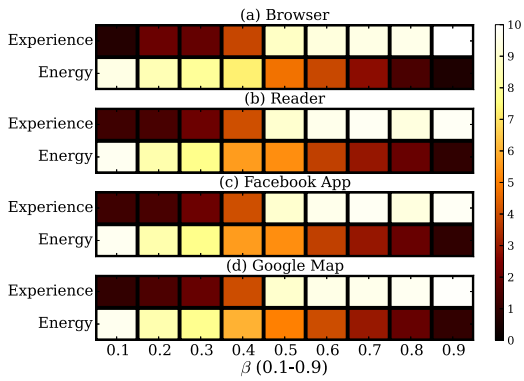

Fig. 21. Impact of β on energy saving and user experience in different applications.

REFERENCES

[1] E^3 testbeds and source code. (2013). [Online] 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 *Proc. 9th Int. Conf. Mobile Syst., Appl., Services*, 2011, pp. 57–70.

[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 *Proc. 18th Annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 101–112.

[4] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas. Conf.*, 2009, pp. 280–293.

[5] F. Bellosa, A. Weissel, M. Waitz, and S. Kellner, "Event-driven energy accounting for dynamic thermal management," presented at the Workshop Compilers Oper. Syst. Low Power, New Orleans, LA, USA, Sep. 27, 2003.

[6] K. Choi, R. Soma, and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times," in *Proc. Conf. Des., Autom. Test Eur.*, 2004, pp. 4–9.

[7] CNN.com. Battery life concerns mobile users. (2005). [Online] Available: http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/

[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 *Proc. 8th Int. Conf. Mobile Syst., Appl., Services*, 2010, pp. 49–62.

[9] M. Dong and L. Zhong, "Chameleon: A color-adaptive web browser for mobile OLED displays," in *Proc. 9th Int. Conf. Mobile Syst., Appl., Services*, 2011, pp. 85–98.

[10] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proc. 9th Int. Conf. Mobile Syst., Appl., Services*, 2011, pp. 335–348.

[11] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proc. 34th Annu. Int. Symp. Comput. Archit.*, 2007, pp. 13–23.

[12] D. Knuth, J. H. Morris, and V. Pratt, "Fast pattern matching in strings," *SIAM J. Comput.*, vol. 6, no. 2, pp. 323–350, Aug. 1977.

[13] D. Le and H. Wang, "An effective feedback-driven approach for energy saving in battery powered systems," in *Proc. 18th Int. Workshop Quality Service*, 2010, pp. 1–9.

[14] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Phys. Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[15] J. McCarthy, M. Sasse, and D. Miras, "Sharp or smooth?: Comparing the effects of quantization vs. frame rate for streamed video," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2004, pp. 535–542.

[16] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in *Proc. 18th Annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 317–328.

[17] MobiThinking.com. Global mobile statistics 2012 part A: Mobile subscribers; handset market share; mobile operators. (2012). [Online] 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 *Proc. 10th ACM Workshop Hot Topics Netw.*, 2011, pp. 5:1–5:6.

[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 *Proc. 6th Conf. Comput. Syst.*, 2011, pp. 153–168.

[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 *Proc. 10th Int. Conf. Mobile Syst., Appl., Services*, 2012, pp. 267–280.

[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 *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 51–60.

[22] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Top: Tail optimization protocol for cellular radio resource allocation," in *Proc. 18th IEEE Int. Conf. Netw. Protocols*, 2010, pp. 285–294.

[23] D. Rajan, R. Zuck, and C. Poellabauer, "Workload-aware dual-speed dynamic voltage scaling," in *Proc. IEEE 12th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2006, pp. 251–256.

[24] R. Rao, S. Vrudhula, and D. Rakhmatov, "Battery modeling for energy aware system design," *Computer*, vol. 36, no. 12, pp. 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 *Proc. 16th Annu. Int. Conf. Mobile Comput. Netw.*, 2010, pp. 85–96.

[26] N. Vallina-Rodriguez, P. Hui, J. Crowcroft, and A. Rice, "Exhausting battery statistics: Understanding the energy demands on mobile handsets," in *Proc. 2nd ACM SIGCOMM Workshop Netw., Syst., Appl. Mobile Handhelds*, 2010, pp. 9–14.

[27] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "ProfileDroid: Multi-layer profiling of android applications," in *Proc. 18th Annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 137–148.

[28] W. Yuan, K. Nahrstedt, S. Adve, D. Jones, and R. Kravets, "GRACE-1: Cross-layer adaptation for multimedia quality and battery energy," *IEEE Trans. Mobile Comput.*, vol. 5, no. 7, pp. 799–815, Jul. 2006.

[29] H. Han, J. Yu, H. Zhu, Y. Chen, J. Yang, and Y. Zhu, "E3: Energy-efficient engine for frame rate adaptation on smartphones," in *Proc. 11th ACM Conf. Embedded Netw. Sensor Syst.*, 2013, pp. 15–28.

Jiadi Yu received the PhD degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2007. He is currently an assistant professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. Prior to joining Shanghai Jiao Tong University, he was a postdoctoral fellow in the Data Analysis and Information Security (DAISY) Laboratory at Stevens Institute of Technology from 2009 to 2011. His research interests include cyber security and privacy, mobile and pervasive computing, cloud computing, and wireless sensor networks. He is a member of the IEEE and the IEEE Communication Society.

Haofu Han received the master's degree in the Department of Computer Science and Engineering, Shanghai Jiao Tong University in 2014. He is currently a software engineer in Tencent. His research interest include mobile computing.

Hongzi Zhu received the PhD degree from the Department of Computer Science and Engineering at the Shanghai Jiao Tong University in 2009. He is currently an associate professor in the Department of Computer Science and Engineering at Shanghai Jiao Tong University, China. His research interests include vehicular ad hoc networks, wireless networks, mobile computing, and network security. He is a member of the IEEE and the IEEE Communication Society.

Yingying Chen received the PhD degree in computer science from Rutgers University. She is an associate professor in the Department of Electrical and Computer Engineering at Stevens Institute of Technology. Her research interests include cyber security and privacy, mobile and pervasive computing, and mobile healthcare. She has published more than 80 journals and referred conference papers in these areas. Prior to joining Stevens, she was with Alcatel-Lucent. She received the US NSF CAREER Award, the Google Faculty Research Award, the NJ Inventors Hall of Fame Innovator Award, the Best Paper Award from ACM International Conference on Mobile Computing and Networking (MobiCom) 2011, and the IEEE Outstanding Contribution Award from IEEE New Jersey Coast Section each year 2005-2009. Her research has been reported in numerous media outlets including MIT Technology Review, Wall Street Journal, and National Public Radio. She is on the editorial boards of *IEEE Transactions on Mobile Computing (IEEE TMC)*, *IEEE Transactions on Wireless Communications (IEEE TWireless)*, and *IEEE Network Magazine*. She is a senior member of the IEEE.

Jie Yang received the PhD degree in computer engineering from the Stevens Institute of Technology in 2011. He is currently an assistant professor in the Department of Computer Science at Florida State University. His research interests include cyber security and privacy, and mobile and pervasive computing, with an emphasis on network security, smartphone security and applications, security in cognitive radio and smart grid, location systems, and vehicular applications. His research is supported by the US National Science Foundation (NSF) and the Army Research Office (ARO). He received the Best Paper Runner-Up Award from IEEE Conference on Communications and Network Security (CNS) 2013 and the Best Paper Award from ACM MobiCom 2011. His research has received wide press coverage including *MIT Technology Review*, *Wall Street Journal*, NPR, CNET News, and Yahoo News. He is a member of the IEEE.

Yanmin Zhu received the PhD degree from the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology in 2007. He is currently an associate professor with the Department of Computer Science and Engineering at Shanghai Jiao Tong University. His research interests include wireless sensor networks and mobile computing. Before that, he was a research associate with the Department of Computing at Imperial College London. He is a member of the IEEE and the IEEE Communication Society.

Guangtao Xue received the PhD degree from the Department of Computer Science and Engineering at the Shanghai Jiao Tong University in 2004. He is currently an associate professor in the Department of Computer Science and Engineering at Shanghai Jiao Tong University, China. His research interests include vehicular ad hoc networks, wireless networks, mobile computing, and distributed computing. He is a member of the IEEE and the IEEE Communication Society.

Minglu Li graduated from the School of Electronic Technology, University of Information Engineering, and received the PhD degree in computer software from Shanghai Jiao Tong University (SJTU) in 1985 and 1996, respectively. He is a full professor and the vice chair of the Department of Computer Science and Engineering and the director of Grid Computing Center of SJTU. His current research interests include grid computing, services computing, and sensor networks.