SEMI: Semi-Online Power Estimates for Smartphone Hardware Components

Ekarat Rattagan, Edward T.-H. Chu, Ying-Dar Lin, *Fellow, IEEE*, and Yuan-Cheng Lai

Abstract—Using the data obtained from a battery monitoring unit (BMU) are a low-cost, easy-to-use way to estimate the power consumption of smartphones. Current online power estimation methods produce significant errors compared to using external power monitors because online methods do not address the three most important factors which affect the efficacy of online power consumption estimates. These are: (1) battery capacity degradation with aging, (2) asynchronous power consumption, and (3) the effect of state-of-charge (SoC) difference, the remaining power level of the battery. This paper presents a semi-online power estimate method which uses charging data to determine actual battery capacity, applies the discrepancy between battery voltage at different workloads for asynchronous power detection, and analyzes the range of SoC (0-100 percent) which causes minimal power estimate errors. The proposed method is validated by conducting a series of experiments on a smartphone and comparing the results with the existing online power estimation methods. Experimental results on the power consumption estimates of real applications indicate that the semi-online method reduced the error rate of power estimates obtained from existing online methods by 27-94 percent. Moreover, this work also shows that battery capacity degradation is the major factor affecting the efficacy of online power estimations.

Index Terms—Power consumption estimation, asynchronous power consumption behavior, battery capacity, state-of-charge

1 Introduction

The power consumption estimates of smartphone hardware components such as CPUs, Screen, Wi-Fi, etc., called power profiles, are useful for various purposes. For example, smartphone researchers can use them to make decisions when to offload computational tasks with low power usage [1], [2]. Smartphone OS developers can use them to implement OS function, such as battery lifetime prediction [3], [4]. Smartphone app developers can use them to create power-related applications such as battery monitoring apps.

In general, the power profiles of smartphones are generated by smartphone vendors embedded in smartphone devices. However, the accuracy of vendor-generated power profiles decreases as smartphones are used over a period of time. The decreasing accuracy is a result of several factors, such as battery capacity degradation [5] and ambient temperature [6]. Mian and Zhong [7] also showed that smartphone power profiles should be regularly rebuilt to mitigate errors in power estimation caused by such factors. It is thus necessary for smartphone researchers or developers to reconstruct the power profiles of smartphones being tested so as to sustain the accuracy of power profiles.

To reconstruct the power profiles of smartphones requires obtaining the power consumption of the hardware components operating in various states. There are typically two state-of-the-art power consumption acquisition methods for smartphones, offline and online methods. Offline methods [6], [8], [9] use external power monitors to measure power consumption, while online methods [6], [10], [11], [12], [13] use a built-in battery monitoring unit (BMU) to estimate power consumption. Clearly, offline methods are more accurate than online methods because the difference in the sampling rates between the external power monitors and BMU, e.g., 5000 Hz versus 1 Hz. However, online methods have more benefits than offline methods in lower cost, ease of use, and scalability. Online methods are particularly suitable for where offline methods are not practical to apply. For instance, smartphones with non-removable batteries or when building power profiles for a lot of smartphones, such as cloud-based testing.

In fact, several existing online methods aim at improving the accuracy of power estimates. However, most of them still do not address the major factors causing large errors of power estimates. The most significant factors that influence estimates are (1) battery capacity degradation, (2) asynchronous power consumption behavior, and (3) the effect of SoC difference in the hardware state training.

1. Battery capacity degradation refers to the decrease in battery capacity (mAh) after several discharge/charge cycles [5]. This causes large errors in existing online methods [6], [10], [12] as most of them still use the

- *E. Rattagan is with the Faculty of Information Technology, Thai-Nichi Institute of Technology 1771/1 Pattanakarn Rd., Suanluang, Bangkok 10250, Thailand. E-mail: ekarat@tni.ac.th.*
- *E.T.-H. Chu is with the Department of Computer Science and Information Engineering, National Yunlin University of Science and Technology, ES 709R 123 University Road, Section 3, Douliou, Yunlin 64002, Taiwan. E-mail: edwardchu@yuntech.edu.tw.*
- *Y.-D. Lin is with the Department of Computer Science, National Chiao Tung University, 1001, Da-xue Rd., Hsinchu City 300, Taiwan. E-mail: ydlin@cs.nctu.edu.tw.*
- *Y.-C. Lai is with the Department of Information Management, National Taiwan University of Science and Technology, 43, Sec. 4, Jilong Rd., Da-An District, Taipei City 106, Taiwan. E-mail: laiyc@cs.ntust.edu.tw.*

Manuscript received 15 Apr. 2016; revised 28 Dec. 2016; accepted 5 Jan. 2017. Date of publication 11 Jan. 2017; date of current version 13 Feb. 2017. Recommended for acceptance by P. Bouvry.
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/TSUSC.2017.2651159

battery capacity value stated on the battery cover to calculate power estimates. The battery capacity on the label is quite different compared with the actual battery capacity value, especially in old batteries.

2. Asynchronous power consumption behavior ('asynchronous power' for short) is the power consumption which is not correlated with the hardware operating state usage. This problem occurs in some hardware components, such as cellular networks [8], [14], and at present no existing online method can deal with it effectively. On the other hand, synchronous power is the power consumption which correlates with the hardware operating state.
3. The effect of SoC difference refers that different SoC (the remaining battery levels) can affect the power consumption estimation obtained form current online methods. E.g., the hardware state trained at the same training time, e.g., 30 mins, but at different SoC, such as 50 and 90 percent, will cause different power consumption estimates. This is due to the nonlinear characteristics of battery data used for calculating SoC [5].

As far as we know, there are no existing online methods that address all three issues elaborated above. This study proposes a semi-online power estimation method, termed SEMI, which aims at reducing the power estimation errors, by addressing these three factors. The central concept of SEMI is based on our observation that all existing online methods are not actually 'online' as they had been termed. In fact, during a power estimation process, these 'online' methods still rely on external equipment, such as a USB cable for charging, to complete their operations. During charging, all existing online methods waste time without achieving anything useful. SEMI thus takes advantage of information derived from the charging process to improve the accuracy of typical online methods.

SEMI works by first addressing battery capacity degradation, by using both discharging and charging data, including the charging current (mA), to determine the actual battery capacity. The charging current is obtained by using a low-cost device which monitors the current via a USB cable during charging. Second, by applying the methods developed by [11], where estimated current can be deduced by the difference in battery voltage associated with two different workloads, SEMI is able to detect and estimate asynchronous power consumption in a device. Third, this study assesses the effect of different SoC of the hardware state training process to determine an optimal SoC that would result in minimal power estimation errors. Compared to earlier work [15], this work investigates the power consumption estimates of an additional hardware component, Wi-Fi, and validates the efficacy of SEMI on the real applications.

The remainder of this paper is organized as follows. Section 2 explains the background and related work. Section 3 describes the problem statement. In Section 4 the SEMI method is presented, and Sections 5 and 6 shows the xperiment setup and results. Finally, Section 7 concludes the paper.

2 Background

This section reviews our knowledge of online power estimation methods and some important factors which significantly affect the accuracy of power estimation.

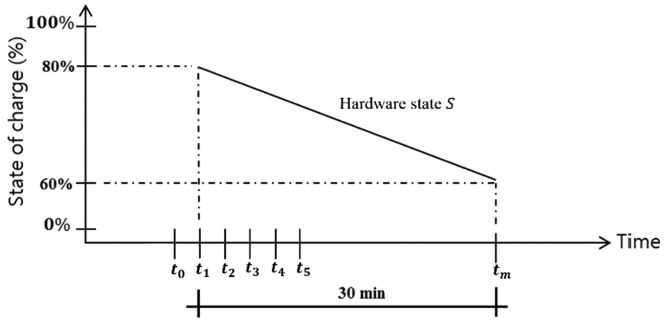


Fig. 1. An example of online power estimation.

2.1 Online Power Estimation Methods

The traditional online power estimation methods are those which use battery data, provided by a BMU, to estimate the power consumption of hardware components of a smartphone. The power consumption of each component refers to the power consumption of the hardware operating state. The hardware operating state ('hardware state' for short) is a combination of various hardware parameters, which include CPU utilization and frequency, screen brightness, and 3G and Wi-Fi packet rates. Each parameter is composed of a range of discrete values, e.g., CPU utilization contains discrete values from 1 to 100 percent, and screen brightness contains discrete values from 0 to 255.

The battery data provided includes battery voltage (mV), state-of-charge (SoC), i.e., the percentage of remaining battery power, battery temperature, etc. Nevertheless, the BMUs of most commercial smartphones do not give current data (mA) because the current sensors have been removed to save cost. Hence, the existing online power estimation methods which use current data to measure the power consumption are workable for only a few smartphones, as shown in [7], [13].

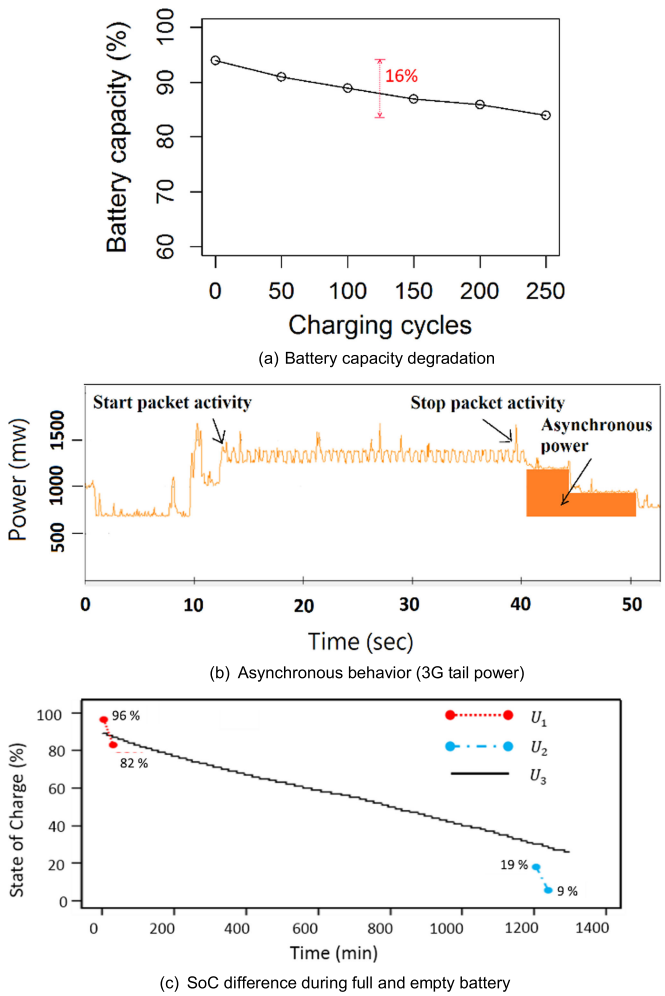
For most smartphones whose BMUs do not provide current data, the power consumption estimates of the target hardware component can be obtained by collecting battery data, while the hardware state is controlled (training) for a period of time during the discharge process. After completing the training process, traditional online methods use the battery data thus obtained to calculate the power consumption estimates of the target hardware state as per the following equation

$$P = \frac{(U \times C \times V)}{T} \tag{1}$$

where P is the power estimate (mW) associated with the target hardware state, U is the difference between the starting SoC and ending SoC, C is the battery capacity value given on the cover of the battery, V is the battery voltage (V), and T is the training time (hour). Fig. 1 illustrates the traditional online power methods to estimate the power consumption of hardware state S trained for 0.5 hour. The starting and ending SoC are 80 and 60 percent, respectively, so that U equals 20 percent (80 − 60 percent). Assuming that the battery capacity is 2,600 mAh and voltage is 3.8 V, then, based on Eq. (1), the power estimates of this hardware state is $(0.2 \times 2{,}600 \times 3.8) \,/\, 0.5 = 3{,}952$ mW.

2.2 Factors Affecting Power Estimation

Battery Capacity Degradation. Commercial smartphones are generally equipped with removable or non-removable Li-Ion



(a) Battery capacity degradation

(b) Asynchronous behavior (3G tail power)

(c) SoC difference during full and empty battery

Fig. 2. Three factors affecting the online power estimation.

batteries, with a capacity of 1,500-3,000 mAh, shown on the battery cover. With such a limit to battery capacity and a power-hungry application usage, smartphone batteries discharge very fast, and need to be recharged more and more frequently. The increase in discharge/charge cycles results in a decrease in battery capacity caused by their internal chemical reaction [5]. Consequently, the actual capacity of old batteries is lower than the capacity stated on the cover. Fig. 2a shows that about 16 percent of battery capacity is lost after 250 discharge/charge cycles [5]. Existing online methods use the battery capacity given on the battery cover to estimate the power consumption of smartphones. However, without determining the actual battery capacity, the power consumption based on the labeled battery capacity leads to large errors in power consumption estimates. And the older the battery, the larger the error becomes.

Asynchronous Power refers to the consumed power of a hardware component is not correlated with its operating state. For example, Fig. 2b shows the asynchronous power in 3G (tail power) of between 40 to 50 seconds, even though the 3G utilization during asynchronous power is similar to the 3G utilization of between 0 to 10 seconds. An alternative way to estimate asynchronous power was proposed by Pathak et al. [8], a system call-based and offline method to determine asynchronous power. While the system call-based and offline method is accurate, it is labour-intensive and requires intimate knowledge of the smartphone operating system (OS). It is thus not applicable to most commercial smartphones whose OS is a closed source. Also, the system call functions of some hardware components, such as GPU, are hidden from OS.

Different State-of-Charge. The SoC is the value that indicates how much battery capacity (percent) remains. Those commercial smartphones equip with BMU can provide battery data such as SoC, voltage, and temperature. Most of the existing online methods are based on the assumption that the same hardware state trained under the same duration consumes the same SoC difference in all ranges of SoC (0-100 percent), as shown as line U_3 in Fig 2c. As a result, the existing online methods which train each hardware state on any SoC without being aware of the effect of SoC differences, would cause large power estimation errors. What has been observed in this work, however, is that the hardware state will consume a significantly larger SoC difference when it is trained during an almost full or an empty SoC. As shown in Fig. 2c, the target hardware state trained at SoC $= 96$ percent consumes a SoC difference $U_1 = 14$ percent; whereas it consumes SoC difference $U_2 = 10$ percent at SoC $= 19$ percent. This is due to the nonlinear characteristics of SoC when it is closed to 0 or 100 percent [5]. Therefore, without addressing the optimal range of the SoC during hardware state training, the power estimates constitute a large error rate compared to the actual power measurement.

3 Problem Statement

Assume a device under test (DUT) equipped with a battery whose capacity is C (mAh) and voltage is V (V). The DUT comprises several hardware components, and each component has various parameters, e.g., the CPU has two parameters, utilization and frequencies. Each parameter has discrete values: CPU utilization is between 1 to 100 percent, and CPU frequencies are 250, 350, ... 1,600 MHz. A hardware state S_i is defined as a vector that contains the parameters of all target hardware components. For example, the hardware state S_0, the idle state, contains parameters that include CPU utilization $= 1$ percent, CPU frequency $= 250$ MHz, screen brightness $= 0$, etc.

Let the DUT includes $N + 1$ hardware states, denoted as $S_0, \ldots, S_n$. Each state S_i was trained to estimate the associated power consumption. In the training process, during battery discharge, the training app controls the target hardware state S_i to be active for a specific period of time, while periodically collecting related battery data provided by BMU. The sampling interval of the training app is d seconds, while the update interval of BMU is r seconds, where $d < r$. Table 1 summarizes all parameters and their definitions. Given the battery data associated with the hardware state S_i, the objective is to use that data to estimate synchronous power p_i, asynchronous power p_i^{async}, and asynchronous duration T_i^{async}, associated with the hardware state S_i, so that the error rate between the power estimates of the hardware state S_i, including p_i^{sync} and p_i^{async}, and the measured power Y_i, obtained from an external power monitor, is minimized.

4 SEMI

The goal of SEMI is to improve the online methods in terms of accuracy by dealing with the three factors mentioned

TABLE 1
Parameter and Definition

Unit	Unit Symbol
I^{CH}	Battery charging current (mA)
T^{CH}	Battery charging time (hour)
I^{DCH}	Battery discharging current (mA)
T^{DCH}	Battery discharging time (hour)
C	Battery capacity (mAh)
V	Battery voltage (V)
C'	Current battery capacity (mAh)
r	Update interval of the BMU
d	Sampling interval of the training app
Q	A ratio of r and d
S_i	Hardware operating state
$t_{\mathrm{i,x}}$	Training time slot x of the hardware state S_i
$\mathrm{u}(\mathrm{t_{i,x}})$	SoC at the time slot $t_{i,x}$
$\mathrm{v}(\mathrm{t_{i,x}})$	Battery voltage (μv) at the time slot $t_{\mathrm{i,x}}$
p_i^{sync}	Synchronous power of hardware state S_i (mW)
p_i^{async}	Asynchronous porver of hardware state S_i (mW)
T_i^{async}	Asynchronous duration of hardware state S_i (mW)
Y_i	Porver measurement of hardware state S_i (mW)

above. Generally, SEMI operates in three steps. The first step (battery capacity correction) aims at reducing the error rates caused by the battery capacity degradation. This step determines actual battery capacity C' by using both discharging and charging data. Next, the second step (optimal SoC detection) aims at finding the range of SoC which causes the least errors in SoC difference. Finally, the third step (power consumption estimates) aims at estimating the synchronous power p_i^{sync} and asynchronous power p_i^{async} of each hardware operating state S_i, $where\ i = 0, .., n$. In particular, the third step employs actual battery capacity C' and optimal range of SoC obtained from the first and second steps, respectively, in their operations. Fig. 3 depicts all operation steps of SEMI.

4.1 Battery Capacity Correction

As the manufactured battery capacity C decreases to C' after several discharge/charge cycles, this section aims to determine the actual battery capacity C' so as to reduce the error rate of the power estimates. Typically, C' is simply

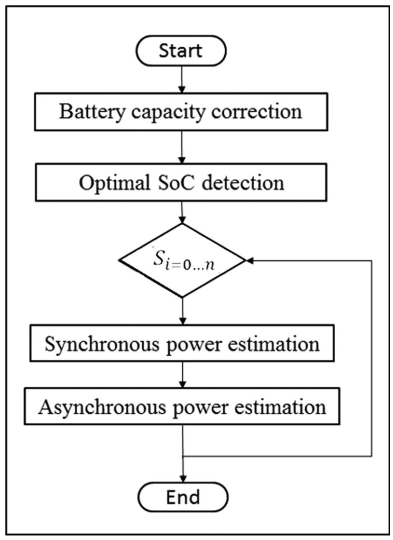


Fig. 3. The operation steps of SEMI.

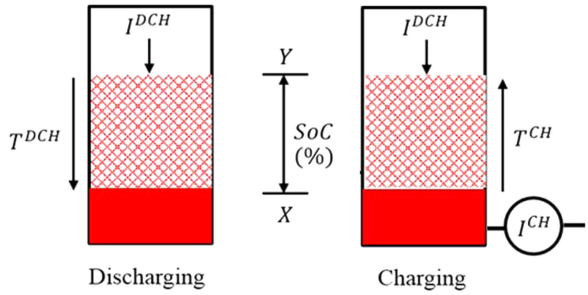


Fig. 4. Charging and discharging data of the same hardware state associating with the same SoC difference.

estimated by multiplying the averaging discharging current I^{DCH}, associating with a controlled hardware state, and total discharging time T^{DCH}, the time for discharging battery capacity from 100 to 0 percent of SoC: $C' = I^{DCH} \times T^{DCH}$. However, since the discharging current I^{DCH} cannot be measured directly, this work proposed a technique to obtain I^{DCH} from the discharging and charging data, based on the similarity of the amount of battery energy usage (SoC difference) when a DUT is put into the same controlled hardware sate during discharging, and charging process. Fig. 4 gives the discharging and charging data. The discharging data includes the total discharging time T^{DCH} for discharging battery capacity from Y to X percent SoC, U_{yx}; whereas the charging data includes the total charging time T^{CH} for increasing battery capacity from X to Y percent SoC, U_{xy}, and the average charging current I^{CH} obtained by using a low cost device via a USB cable.

Based on the similarity of SoC difference U_{yx} and U_{xy}, the discharging current I^{DCH} is calculated as

$$\begin{aligned} SoC_{discharge} &= SoC_{charge} \\ I^{DCH} \times T^{DCH} &= (I^{CH} - I^{DCH}) \times T^{CH} \\ I^{DCH} &= \frac{I^{CH} \times T^{CH}}{T^{DCH} \times T^{CH}}. \end{aligned} \tag{2}$$

Finally, C' can be calculated by using the charging and discharging data as follows:

$$C' = \left(\frac{I^{CH} \times T^{CH}}{T^{DCH} \times T^{CH}} \right) \times T^{DCH}. \tag{3}$$

Note that Eq. (3) is workable for a battery system whose charging and discharging paths are the same. If not, the charging time T^{CH} of all hardware states are similar and the discharging current I^{DCH} associated with each hardware state cannot be differentiated.

4.2 Optimal SoC Detection

This section aim to discover the optimal range of SoC for training a hardware state which causes the smallest power estimation errors, we process as follows. First, a battery is fully charged. Next, the battery is allowed to discharge from full to empty, while a DUT is put into a hardware state S_i for a certain period of time. Then, for each training period, the power esitmation of the associated range of SoC is estimated, by using the traditional online power estimation (synchronous power) and the actual battery capacity C'

obtained from the previous step. Finally, the accuracy of the power estimates of each range of SoC is compared with the external power monitor in order to find the range of SoC with the highest accuracy in order to be used in the next step.

4.3 Synchronous Power Estimation

Algorithm 1 estimates synchronous power, p_i^{sync} by using the training app to periodically sample battery data, starting from time $t_{i,0}$ to $t_{i,m}$. For each time slot $t_{i,x}$, the battery voltage $v(t_{i,x})$ is collected and inserted into list L. At the time slot $t_{i,1}$, the hardware state of DUT is changed from the idle state S_0 to the hardware state S_i, and the $u(t_{i,m})\%$ of SoC is collected and inserted into a list U. Finally, at the time $t_{i,m}$, the $U(t_{i,m})\%$ of SoC is collected and inserted into list U, while the DUT is forced into the idle state S_0. After finishing collecting data, all battery voltages in the list L are averaged to V, and the synchronous power P_i^{sync} is then calculated as

$$p_i^{sync} = \frac{\big(u(t_{i,1}) - u(t_{i,m})\big) \times C' \times V}{t_{i,m} - t_{i,1}} \tag{4}$$

Algorithm 1. Synchronous Power Estimation

input: Actual battery capacity C':
vars: An empty list L and U, $V = 0$;
for $x = 0, 1, \ldots, m$ **do**
  $L \leftarrow$ insert($v(t_{i,x})$);
  if $x == 1$ **then**
    Put a DUT is in the hardware state S_i;
    $U \leftarrow$ insert($u(t_{i,x})$);
  end if
  if $x == m$ **then**
    Put a DUT is in the hardware state S_0;
    $U \leftarrow$ insert($u(t_{i,x})$);
  end if
end for
$V = mean(L)$
Return $p_i^{sync} = \frac{(U_0 U_1) \times C' \times V}{t_{i,m} - t_{i,1}}$;

4.4 Asynchronous Power Estimation

After estimating synchronous power P_i^{sync} in the previous step, the next step is to estimate asynchronous power P_i^{async}, based on the knowledge that the current value (mA) is linear related to the size of voltage difference [11]. The asynchronous power consumption of each hardware state S_i is thus deduced from the voltage difference v_{diff}. After time $t_{i,m}$, the training app continues checking whether a hardware state S_i includes asynchronous power or not, as shown in Algorithm 1, by finding that $v_{diff} = |v_{drop} - v_{rec}|$, where $v_{drop} = v(t_{i,0}) - v(t_{i,2})$ and $v(t_{i,0}) > v(t_{i,2})$, is the decrease in battery voltage when a hardware state is changed from S_0 to S_i and $v_{rec} = v(t_{i,m+2}) - v(t_{i,m})$, where $v(t_{i,m+2}) > v(t_{i,m})$, is the amount of battery voltage recovery when a hardware state is changed from S_i to S_0. To detect asynchronous power, the sample slot of the training app needs to straddle the BMU update time slot, i.e., between m, $m+2$, $m+4$, and so on, to detect the change in battery voltage. Then, v_{diff} is compared with a defined threshold to determine asynchronous power of S_i.

Algorithm 2. Asynchronous Power Estimation

Input: p_i^{sync}, L;
vars: An empty list L^{async}; v_{diff}, v_{rec}, v_{drop}, T_i^{async};
$v(t_{i,0}) \leftarrow L$
$v(t_{i,0}) \leftarrow L$
$v(t_{i,2}) \leftarrow L$
$v(t_{i,m}) \leftarrow L$
$v_{drop} = v(t_{i,0},) - v(t_{i,2})$
for $x = m + 2, m + 4, \ldots$ **do**
  $v_{rec} = v(t_{i,x}) - \nu(t_{i,x-2})$
  $v_{diff} = |v_{drop} - v_{rec}|$
  if $v_{diff} > threshold$ **then**
    $L^{async} \leftarrow$ **insert**$(p_i^{sync} \times (\frac{v_{diff}}{v_{drop}}))$
    $T_i^{async} \leftarrow T_i^{async} + x$
  else
    break
  end if
end for
$p_i{}^{async} = mean(L^{async})$
Return $p_i{}^{async}, T_i^{async}$;

The criteria to detect asynchronous power is, if v_{diff} is smaller than the threshold, it concludes that the hardware state S_i does not include asynchronous power, and Algorithm 2 then stops working. By contrast, if v_{diff} is larger than the threshold, it concludes that the hardware state S_i includes asynchronous power. If asynchronous power exists, the training app then determines the amount of asynchronous power p_i^{async} and duration T_i^{async}. The amount of asynchronous power can be obtained by

$$p_i^{async} = p_i^{sync} \times \left(\frac{v_{diff}}{v_{drop}}\right) \tag{5}$$

and is then inserted into a list L^{async}, while the duration is accumulated. Finally, when asynchronous power has ended, Algorithm 2 returns the asynchronous power p_i^{async}, estimated by averaging the list L^{async} and the duration T_i^{async}.

Fig. 5 shows an example of asynchronous power estimate. Suppose synchronous power is 1,000 mW, v_{drop} = 20,000, and v_{rec} = 8,000. Thus, v_{diff} = 20,000 – 8,000 = 12,000. By applying Eq. (5) with the data example, the asynchronous power is calculated as $1{,}000 \times \frac{12{,}000}{20{,}000} = 600$ mW.

5 Experimental Setup and Results

5.1 Experimental Setup

SEMI was validated by experiment on a commercial smartphone, Galaxy S4, as the DUT, using a Monsoon power monitor [18], which supplies stable voltage and sample rates at 5 KHz, to measure the actual power consumption. The experimental setup is shown in Fig. 6.

The target hardware components included the CPU, screen, 3G, and Wi-Fi. Each hardware state was trained for 30 minutes to obtain the synchronous power estimates, with additional time for any hardware state associated with asynchronous power. The experiment on the GPU was omitted because of its complexity and that a large number of its parameters are difficult to control for long enough until the SoC has changed. The detail of each hardware component is as follows.

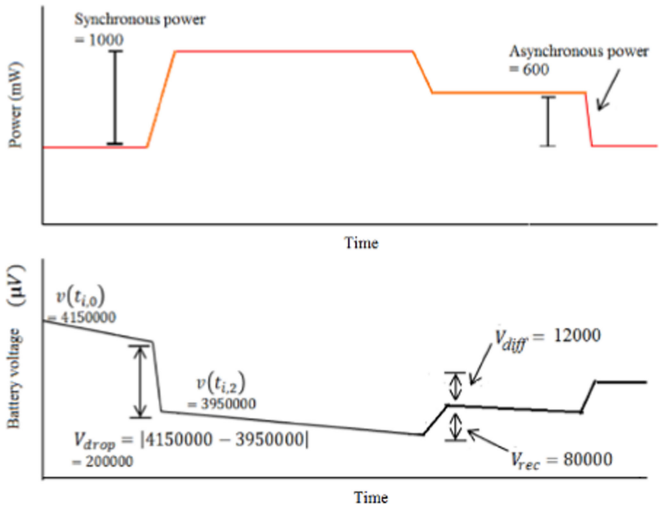


Fig. 5. An example of asynchronous power estimation.

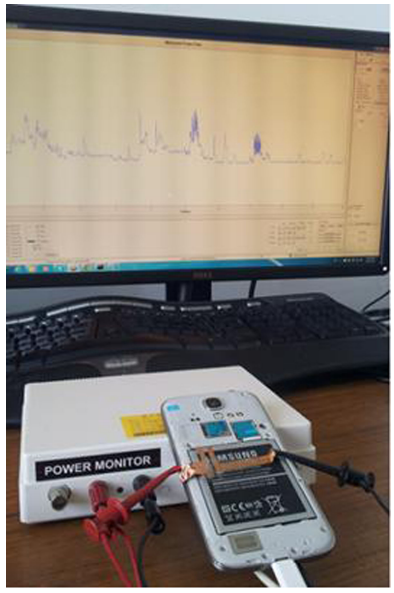


Fig. 6. The experimental setup includes DUT and monsoon.

CPU: CPU power consumption relies on CPU utilization from 1 to 100 percent in a step of 10 percent, and a range of frequencies from 200, 300, . . ., 1,600 MHz. The CPU of the DUT is a Big.Little technology with four Big and four Little cores. However, not all eight cores can operate simultaneously. In fact, the eight cores are separated into four clusters, each cluster consists of one Big and one Little. With the limitation of the OS scheduling policy, only four cores of different clusters can operate simultaneously. In this work, the power consumption of one cluster was evaluated, and the power consumption of the other clusters were deduced from the first one.

AMOLED screen: the proposed method considered only the screen's brightness, i.e., from 0 to 255 in steps of 30, but pixel colors, even though the pixel colors significantly affect the power consumption of the AMOLED screen [19]. This is because there is a high overhead for capturing the screen's pixel colors by a training app.

3G and Wi-Fi: the power consumption of these two network interfaces varies by a number of packets (Tx and Rx) per second, i.e., the DUT communicates with a server by sending and receiving files with various sizes.

For the hardware training process, we created a training app which automatically varied the workloads of every hardware component. The first trained component was CPU. While the CPU was being trained, the 3G and Wi-Fi components were off, and the screen was set to one color (white) and brightness. After we obtained the power model of CPU (in this work, we used linear regression to build a power model), the screen (brightness) was next trained, and its power consumption was calculated by subtracting the total power from the CPU power consumption. Finally, the 3G and Wi-Fi were trained.

Each hardware state was trained several times and then averaged. The generated power models were validated on four real applications running with six different scenarios. The Mean Absolute Percentage Error (MAPE) was used as the error metric for evaluating the power estimates of individual hardware states. The Root Mean Square Error (RMSE) was used as the error metric to assess the power estimates of real applications. To compare the efficacy of the proposed SEMI method, another version of the semi-online method, SEMI-B, similar to SEMI, except that it used the stated battery capacity instead of the actual battery capacity. The accuracy of the power estimates from both semi-online versions were also compared with the power estimates obtained from other online power estimation methods, i.e., SoD [6] and CABLI [10], and the offline method, which used an external power monitor, was used as the baseline.

It is worth noting that the deeper investigation of hardware operating states will significantly impact on the accuracy of the power estimated models, e.g., different CPU idle time consume different power consumption [16], [17]. However, since CPU idle time can be controlled for a short time, it requires offline methods which use a high sampling rate power monitor (e.g., 5K Hz) to determine its power characteristics. This is very challenging for online methods which use a low sampling rate (like BMU at, e.g., 1 Hz) to capture the power characteristics during a small time window. In fact, such online methods are suitable for the hardware states which we can control long enough, until the SoC is different. For this reason, we thus focused on all possible hardware states which we can control long enough for the BMU to capture the SoC differences.

5.2 The Effect of Battery Capacity Degradation

Fig. 7 gives the comparisons of synchronous power estimate associated with the same hardware state but different battery capacities. The stated battery capacity of traditional online, SEMI, and offline methods are 2,600, 2,000, and 1,670 mAh, respectively. The power consumption estimate obtained from the traditional online methods, SoD, CABLI, and SEMI-B, resulted in the highest errors of 79.3, 59.9 and 59.9 percent, respectively, compared with the offline method. However, by using a battery capacity correction, SEMI can result in error rate of 22.9 percent, compared with the offline method. As this result, battery capacity degradation shows very large errors for the power consumption estimates obtained from traditional online methods.

5.3 The Effect of SoC

Fig. 8 shows the error variance resulting from training the same hardware state S_i (10 percent CPU utilization, 800

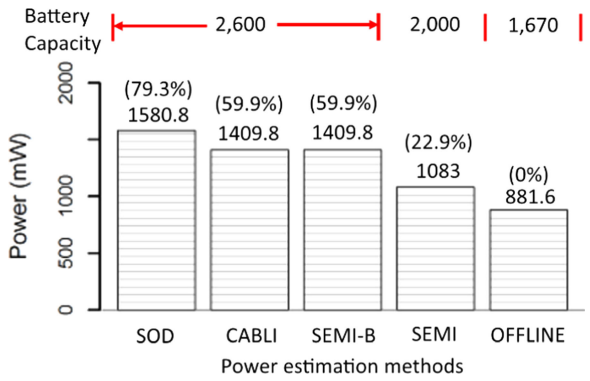


Fig. 7. Battery capacity degradation.

MHz frequency) with a training time of 30 minutes on different SoC. By comparison with the offline method, the results shows that 50-70 percent of SoC had the lowest impact on the accuracy of power estimates, whereas 30-40 percent of SoC shows the highest impact on the accuracy of power estimates. Thus, the hardware state S_i consumed 9 percent of SoC trained at the middle of SoC. On the other hand, it consumed 12-14 percent of SoC when trained at an 0 or 100 percent of SoC. The reasons why all methods performed well during the middle of SoC was because of the relation between the SoC curve and voltage method is linear, whereas the relationship is nonlinear when SoC is closed to 0 or 100 percent [5]. The results also suggest that the hardware states should be trained at the middle of SoC, at 50-70 percent to give lowest error rate. Moreover, the optimal SoC difference is important for obtaining an accurate asynchronous power estimate.

5.4 The Effect of Asynchronous Power Behavior

Fig. 9 gives the comparisons of power estimates of hangout with 3G obtained from SEMI (with and without applying asynchronous power detection) and the offline method. Based on our calculation, Fig. 9 shows that SEMI (with asynchronous power detection) gave an error (MAPE) of 18.4 percent, whereas SEMI (without asynchronous power detection) gave an error of about 31.4 percent. It is worth noting that the MAPE can be further reduced if the power consumption of GPU and screen pixel data are addressed.

5.5 Power Estimate Errors

Fig. 10 shows the error rate, (we used the MAPE as the error metric), of the power consumption estimates obtained from SEMI-B, SEMI, CABLI, and SoD, on the CPU, screen, 3G, and Wi-Fi. In comparison with the offline method, the error

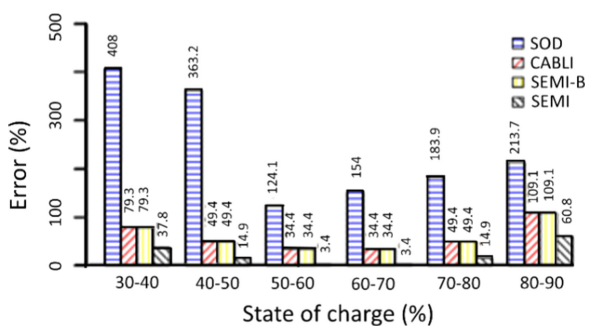


Fig. 8. The effect of SoC difference in the training process.

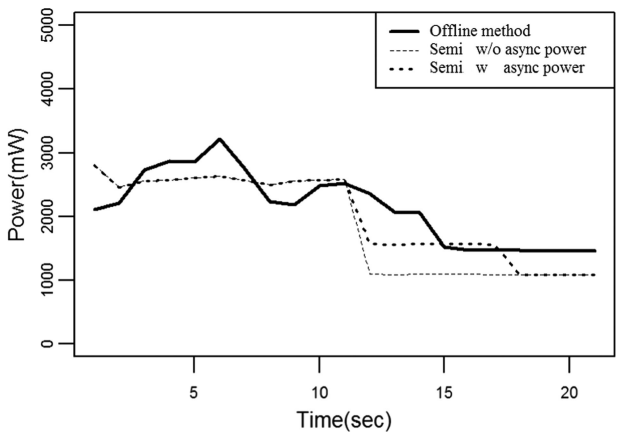


Fig. 9. Comparisons of SEMI (with and without asynchronous power detection) and OFFLINE method.

rates obtained from SEMI were 3, 28, 22 and 25 percent, respectively. The error rates obtained from SEMI-B and CABLI were similar, around 31, 75, and 62 percent for CPU, screen, and Wi-Fi, respectively. However, their error rates were significantly different from that of 3G as SEMI-B improved CABLI by about 36 percent, because CABLI does not address asynchronous power, leading to higher error rates than SEMI. Fig. 9 also shows that SEMI improved SEMI-B by 90 percent on CPU, and 63 percent on screen and 3G, and 59 percent on Wi-Fi. SEMI includes the battery capacity correction, whereas SEMI-B uses the stated battery capacity. Overall, the error rate produced by SoD for four components were highest, at 129, 125, 120, and 115 percent, respectively, because the SoC obtained from the SoD curve is not accurate as showed in [6].

5.6 Real Application Evaluation

The results of real application evaluation are shown in Table 2. SEMI gives the most accurate results compared to the other online methods, improving on those methods by 26.92-93.72 percent. SoD gave the largest power estimation error because of the high error of SoC obtained from the SoD curve generated. The results of SEMI-B and CABLI are similar for apps using Wi-Fi since this hardware component has less effect on asynchronous power. However, their results are quite different from the apps using 3G, which resulted in asynchronous power. Moreover, SEMI showed a larger improvement on the apps which do not intensively

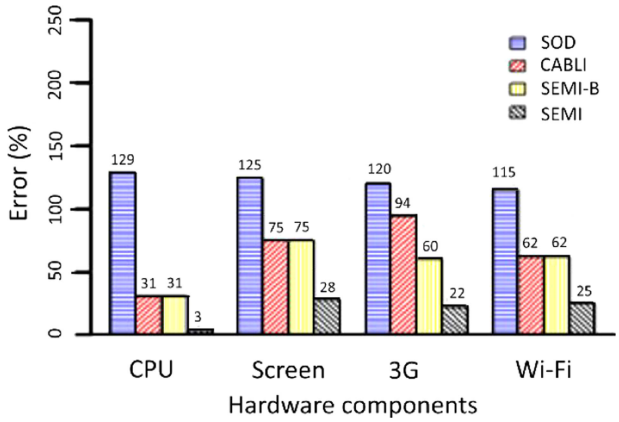


Fig. 10. Power consumption estimates by online methods and SEMI.

TABLE 2
Comparisons of Real Applications Power Estimation (RMSE)

Application	SoD	Cabli	Semi-B	Semi
Chrome Wi-Fi	1690.6	181.5	181.5	172.4
Chrome 3G	2564.6	247.9	227.5	219.5
Gallery Wi-Fi	1868.8	358.5	358.5	217.9
Gallery 3G	6647.4	222.9	238.3	176.1
Hangout 3G	2305.8	214.8	190.9	129.7
Line 3G	2152.3	272.1	282.7	165.4
Average Improvement of SEMI (%)	93.72	27.82	26.92	-

use GPU components, such as Hangout and Line. However, SEMI showed some large errors for apps which use GPU component, such as Chrome and Gallery. Further details of analyzed power estimation errors of the real application are presented in the following sections.

6 Related Work

Existing online methods used in [6], [10], [12] are different from the traditional online methods, in the way in which the SoC difference, U, and training time, T, was determined. In Zhang et al. [6] rather than using the SoC data provided by the BMU, derived an SoC curve which gave the state of discharge (SoD), the association between discharge battery voltage and discharge time. However, the SoD gives significant power estimation errors as it did not address asynchronous power and SoC difference. Zhao et al. [10] derived CABLI which uses the SoC data obtained from BMU and the battery capacity as given on the battery cover. Without addressing battery capacity degradation, asynchronous power, and SoC difference, the power estimates obtained from CABLI also results in large errors. Xu et al. [11] derived V-edge, which showed that the estimated current value of the training hardware state could be inferred from the discrepancy in battery voltages between two different hardware states. The effect of SoC difference causes significant errors in the power estimated obtained by V-edge, as a result of the large variety of battery voltage differences in different SoC. Table 3 shows the comparisons between the existing online methods and the proposed method for the problem areas identified above.

7 Conclusion

This paper proposes a semi-online power estimate method, termed SEMI, for estimating the power consumption of smartphone hardware components without relying on external power monitors. SEMI addresses three major factors which significantly impact on the accuracy of power estimates obtained from existing online power estimate methods, including battery capacity degradation, asynchronous power consumption behavior, and SOC.

The experimental results demonstrate that SEMI reduced error rates caused by battery capacity degradation from 59.9-79.3 to 22.9 percent, by using the actual battery capacity estimated from the charging information, and the asynchronous power consumption factor, from around 31.4 to 18.4 percent, by using the battery voltage difference. In addition, the experimental results suggest that 50-70 percent of SoC is the optimal range at which to perform the online power estimates with minimum error rate. Experimental results on the test of various real applications also showed that the accuracy of power estimates obtained from SEMI significantly improved the existing online power estimate methods from 26.92-93.72 percent. This improvement shows that SEMI can be applied to various power-rererelated tasks on smartphones, such as offloading decision and battery life prediction.

TABLE 3
Comparisons of Online and Semi-Online Power Estimation Methods

Method	Current (mA) estimation method	Battery Capacity Degrade handling	Async behavior handling	SoC analysis
SoD [6]	SoD and battery capacity label	Discharging data	No	No
CABLI [10]	BMU and battery capacity label	Discharging data	No	No
V-edge [11]	Voltage differences	No	No	No
SEMI	BMU and actual battery capacity	Charging data	Voltage difference	Yes

Acknowledgments

This work was done when E. Rattagan was with National Chiao Tung University.

References

[1] K. Kumar, Y. Nimmagadda, and Y.-H. Lu, "Energy conservation for image retrieval on mobile systems," *ACM Trans. Embed. Comput. Syst.*, vol. 11, no. 3, 2012, Art. no. 66.

[2] Y. D. Lin, E. T.-H. Chu, Y.-C. Lai, and T. J. Huang, "Time and-energy-aware computation offloading in handheld devices to coprocessors and clouds," *IEEE Syst. J.*, vol. 9, no. 2, pp. 393–405, Jun. 2015.

[3] Y. Wen, R. Wolski, C. Krintz, and R. Krintz, "Online prediction of battery lifetime for embedded and mobile devices," in *Issue on Embedded Systems*. Berlin, Germany: Springer, 2004, p. 2004.

[4] J. Cho, Y. Woo, S. Kim, and E. Seo, "A battery lifetime guarantee scheme for selective applications in smart mobile devices," *IEEE Trans. Consum. Electron.*, vol. 60, no. 1, pp. 155–163, Feb. 2014.

[5] I. Buchmann, *Batteries in a Portable World - A Handbook on Rechargeable Batteries for Non-Engineers*, 3rd ed. Richmond, BC, Canada: Cadex Electronics, pp. 80–82, 2011.

[6] L. Zhang, et al., "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proc. 8th IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codes. Syst. Synthesis*, 2010, pp. 105–114.

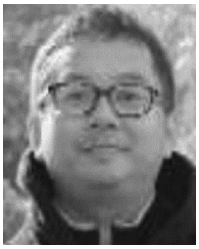
[7] M. Dong and Z. Lin, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proc. 9th Int. Conf. Mobile Syst., Appl., Serv.*, 2011, pp. 335–348.

[8] 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.

[9] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2009, pp. 168–178.

[10] X. Zhao, Y. Guo, Q. Feng, and X. Q. Chen, "A system context-aware approach for battery lifetime prediction in smart phones," in *Proc. ACM Symp. Appl. Comput.*, 2011, pp. 641–646.

[11] F. Y. Xu, Y. X. Liu, Q. Li, and Y. Q. Zhang, "V-edge: Fast self-constructive power modeling of smartphones based on battery voltage dynamics," in *Proc. 10th USENIX Conf. Networked Syst. Des. Implementation*, 2013, pp. 43–56.
[12] J. Lee, H. Joe, and H. Kim, "Automated power model generation method for smartphones," *IEEE Trans. Consum. Electron.*, vol. 60, no. 2, pp. 190–197, May 2014.
[13] W. W. Jung, C. K. Kang, C. M. Yoon, D. W. Kim, and H. J. Cha, "DevScope: A nonintrusive and online power analysis tool for smartphone hardware components," in *Proc. 8th IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codes. Syst. Synthesis*, 2012, pp. 353–362.
[14] 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.
[15] E. Rattagan, E. T.-H. Chu, Y. D. Lin, and Y. C. Lai, "Semi-online power estimation for smartphone hardware components," in *Proc. 10th IEEE Int. Symp. Ind. Embedded Syst.*, 8-10 Jun. 2015, pp. 1–4.
[16] Y. Zhang, X. Wang, X. Liu, Y. Liu, L. Zhuang, and F. Zhao, "Towards better CPU power management on multicore smartphones," in *Proc. Workshop Power-Aware Comput. Syst.*, 2013, Art. no. 11.
[17] M. Guzek, S. Varrette, V. Plugaru, J. E. Pecero, and P. Bouvry, "A holistic model of the performance and the energy efficiency of hypervisors in a high-performance computing environment," *Concurrency Comput.: Practice Experience*, vol. 26, no. 15, pp. 2569–2590, 2014.
[18] Monsoon power monitor, A power meter. (2015). [Online]. Available: http: //www.msoon.com/LabEquipment/PowerMonitor



Ekarat Rattagan received the MS degree in information technology from King Mongkut's University of Technology Thonburi (KMUTT), Bangkok, Thailand, in 2003, and the PhD degree in electrical engineering and computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2016. He is currently a lecturer in the Faculty of Information Technology, Thai-Nichi Institute of Technology, Bangkok, Thailand. His research interests include mobile embedded systems and video game technology.

Edward T.-H. Chu received the PhD degree in computer science from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2010. He has more than 4 years work-experience in the industry, where he worked on embedded software and owns a Chinese patent. He was a visiting scholar at Purdue University in 2009. He joined the Department of Electronic and Computer Science Information Engineering at the National Yunlin University of Science and Technology, Taiwan, as an assistant professor in 2010 and was promoted to an associate professor in 2015. His research interests include embedded systems and real-time operating systems.

Ying-Dar Lin received the PhD degree in computer science from the University of California at Los Angeles (UCLA), in 1993. He is a distinguished professor of computer science with the National Chiao Tung University (NCTU), Taiwan. He was a visiting scholar at Cisco Systems in San Jose, California, during 2007–2008, and the CEO at Telecom Technology Center, Taipei, Taiwan, during 2010-2011. Since 2002, he has been the founder and director of the Network Benchmarking Lab (NBL, www.nbl.org.tw), which reviews network products with real traffic and has been an approved test lab of the Open Networking Foundation (ONF) since July 2014. He also cofounded L7 Networks Inc. in 2002, which was later acquired by D-Link Corp. His research interests include network security, wireless communications, and network cloudification. His work on multihop cellular was the first along this line, and has been cited more than 750 times and standardized into IEEE 802.11s, IEEE 802.15.5, IEEE 802.16j, and 3GPP LTE-Advanced. He currently serves on the editorial boards of several IEEE journals and magazines, and is the editor-in-chief of *IEEE Communications Surveys and Tutorials (COMST)*. He published a textbook, *Computer Networks: An Open Source Approach* (www.mhhe.com/lin), with Ren-Hung Hwang and Fred Baker (McGraw-Hill, 2011). He is a fellow of the IEEE (class of 2013), an IEEE Distinguished Lecturer (2014-2017), and an ONF Research Associate.

Yuan-Cheng Lai received the PhD degree from National Chiao Tung University, Hsinchu, Taiwan, in 1997. In August 1998, he joined the faculty of the Department of Computer Science and Information Science, National Cheng Kung University, Tainan, Taiwan. In August 2001, he joined the faculty of the Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan, where he has been a professor since February 2008. His research interests include performance analysis, protocol design, wireless networks, and web-based applications.