# Similarity-Based Wakeup Management for Mobile Systems in Connected Standby

Chun-Hao Kao[1], Sheng-Wei Cheng[1], and Pi-Cheng Hsiu[2]
[1] Department of Computer Science and Information Engineering, National Taiwan University, Taiwan
[2] Research Center for Information Technology Innovation, Academia Sinica, Taiwan
{r02922029, d02922004}@csie.ntu.edu.tw, pchsiu@citi.sinica.edu.tw

## ABSTRACT

Resident applications, which autonomously awaken mobile devices, can gradually and imperceptibly drain device batteries. This paper introduces the concept of *alarm similarity* into wakeup management for mobile systems in connected standby. First, we define hardware similarity to reflect the degree of energy savings and time similarity to reflect the impact on user experience. We then propose a policy that aligns alarms based on their similarity to save standby energy while maintaining the quality of the user experience. Finally, we integrate our design into Android and conduct extensive experiments on a commercial smartphone running popular mobile apps. The results demonstrate that our design can further extend the standby time achieved with Android's native policy by up to one-third.

## CCS Concepts

•**Computer systems organization** → **Embedded systems;** *Embedded software;* •**Software and its engineering** → **Operating systems;** *Power management;*

## Keywords

Wakeup management, standby energy, user experience, mobile systems

## 1. INTRODUCTION

Most popular mobile apps, including those for instant messaging (e.g., Line), social networking (e.g., Facebook), and email (e.g., Gmail), are *resident applications* which have to periodically synchronize their status information with remote servers or other entities [7]. To preserve those apps' semantics, mobile devices in standby mode maintain Internet connectivity via their network interfaces. This *connected standby* state [12], in which the screen is off while the network remains on, gradually and imperceptibly drains device batteries. A six-month study of 25 mobile device users' real traces [9] found that, on average, smartphones stay in standby mode for 89% of the time and standby energy accounts for 46.3% of total energy use in the context of typical daily usage. A recent measurement study [5] also indicated that, with three popular resident apps installed on a smartphone recovered to its factory settings, its standby time was reduced by nearly half because *unperceived activities* regularly and frequently awakened the smartphone. It is expected that increasing the number of resident apps will accelerate battery depletion.

*DAC '16, June 05-09, 2016, Austin, TX, USA*
© 2016 ACM. ISBN 978-1-4503-4236-0/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2897937.2898091

Numerous energy-efficient schemes have been proposed for mobile systems in the *interactive* state, such as those for CPU computation [10] and OLED displays [2]. Recently, some attention has focused on reducing the energy wasted in connected standby. To prolong standby time, mobile systems resort to a paradigm shift in power management, where every hardware component normally stays off or in a dormant mode unless some app explicitly requests the mobile system to keep the component on. This new paradigm imposes a burden on app developers to explicitly manipulate hardware acquisition and release, giving rise to *no-sleep bugs* that would keep mobile devices awake unnecessarily [6]. To mitigate the misuse of power control primitives, various diagnostic tools have been developed to help application programmers identify possible no-sleep bugs in their source code at compile time [6, 11], or to notify the user of the anomaly when no-sleep bugs are automatically detected in some immature apps at runtime [3].

Apart from no-sleep bugs, normal background activities initiated by various apps can also cause mobile devices to awaken frequently. For example, email sync, although short, can significantly reduce a smartphone's standby time because, once activated, the network interface will be kept on for longer than necessary [12]. To reduce energy waste, fast dormancy is exploited to force the interface to sleep adaptively based on email arrival patterns. In contrast, file uploading is typically data-intensive but delay-tolerant. Because achievable data rates depend highly on the environment and can vary widely over time, for such applications, it is possible to delay data transfers until an energy-efficient network connection becomes available to trade off delay for energy [8]. Moreover, to conserve energy against location positioning, one sensible approach is to reduce the number of GPS invocations by piggybacking the location sensing requests from multiple running location-based applications, and adjust the sensing interval adaptively based on the battery level [13].

Although various solutions have been proposed to leverage the specific characteristics of individual applications, this piecemeal approach may result in a mobile device being frequently and continuously awakened by asynchronous wakeup requests from different apps. Actually, it has been demonstrated that even an immediate remedy, which allows a smartphone to be awakened only at a fixed time interval by forcibly aligning background activities within each interval, can mitigate energy dissipation to some extent [5]. This implicitly suggests a more reasonable design paradigm that the mobile system executes centralized wakeup management, rather than forcing individual app developers to create separate solutions. As background activities need not be executed at specific time points [4], the notion of *inexact alarm delivery*, which has been introduced in Android since version 4.4 [1], enables the mobile system to align alarms according to their attributes to reduce the awakening frequency. This, in turn, raises a technical question: which alarms should be aligned together so as to minimize the total energy wasted in connected standby?

In this paper, we introduce the concept of *alarm similarity* into wakeup management for mobile systems in connected standby, with the objective of reducing energy waste while maintaining the quality of the user experience. The concept is inspired by the following observations. First, aligning alarms with more similar hardware usage could save more energy. Second, some alarms once delivered will attract the user's attention while some are imperceptible to the user. Because imperceptible alarms could be postponed further than perceptible alarms in the timeline, two time points that are dissimilar for perceptible alarms could be deemed similar for imperceptible alarms. Therefore, we argue that alarms employed in wakeup management should be aligned according to their *hardware similarity* (that reflects the degree of energy savings) and *time similarity* (that reflects the impact on user experience).

Realizing this concept in mobile systems obviously raises several design challenges. First, determining the similarity between alarms is a major challenge. Our design, which is based on some characteristics of background activities [5], classifies similarity into three levels: high, medium, and low. For this classification, we define the hardware similarity between two alarms according to how they share hardware components and their time similarity according to whether they can be delivered together at some later point. Then, another challenge is how to exploit hardware and time similarity during alarm alignment, so as to achieve energy efficiency while maintaining the quality of the user experience. The principle behind our design is to align alarms with the highest possible hardware similarity, provided that the time similarity between the aligned alarms is high enough in terms of our classification. Finally, to validate the concept's practicality, we integrated our design into the Android framework, and conducted extensive experiments on a commercial LG Nexus 5 smartphone with 18 popular mobile apps from Google Play. Compared to the native alignment policy employed in Android [1], the proposed design can prolong the smartphone's standby time by one-fourth to one-third, while preserving the delivery expectation of perceptible activities and the delivery periodicity of imperceptible activities. The experiment results also provide some valuable insights into power management of mobile systems in connected standby.

The remainder of this paper is organized as follows. Section 2 provides some background information and an example to show the potential benefits. In Section 3, we discuss the design philosophy and details. The experiment results are reported in Section4. Section5 presents some concluding remarks.

## 2. BACKGROUND AND MOTIVATION

### 2.1 Wakeup Management in Mobile Systems

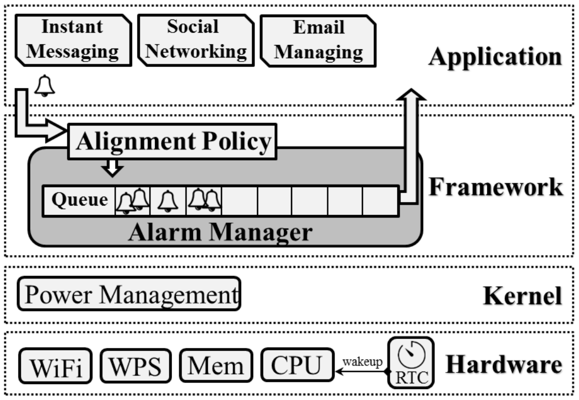

Figure 1: The system architecture

Mobile systems use an *aggressive sleeping philosophy* for energy conservation [6]. In other words, a mobile device, along with its hardware components, is normally kept in a low-power dormant mode unless passively awakened by some application (or system) program. To realize such a philosophy, a mobile system needs to provide an alarm mechanism for apps to register their tasks for execution at some future points. For example, Android, which is considered for our design implementation, employs a mechanism called *AlarmManager*[1] to manage alarm registration and delivery, as shown in Figure 1. Whenever having a task to be run at some future time, an app registers an alarm specifying the *delivery time* to the alarm manager, in which the registered alarms are queued in the increasing order of their delivery times. Then, at the alarm's delivery time, the mobile device (especially the CPU) is awakened (if it is currently asleep) by the real-time clock to deliver the alarm. Finally, the alarm acquires the necessary hardware components, such as the Wi-Fi, to perform the scheduled task. Note that, in Android, alarms can be designated as wakeup or non-wakeup alarms. Wakeup alarms act exactly as mentioned above. In contrast, a non-wakeup alarm can be delivered only when the device is awake and, consequently, its delivery may be postponed to the next time that the device is woken for a wakeup alarm or by an external event (e.g., the user pushing the power button or receiving an instant message).

To prevent the mobile device from being awakened frequently by various apps, the alarm manager will try to align alarms according to their attributes. In the current Android practice (version 4.4 or later), aside from the "nominal" delivery time, an alarm is specified with a *window interval* and a *repeating interval*. The repeating interval indicates the alarm's next nominal delivery time. Accordingly, the alarm manager can reinsert the alarm (with the nominal delivery time) into the queue immediately after its delivery. A repeating alarm is *static* if its repeating interval (which implies the nominal delivery times) is fixed when it is registered, and *dynamic* if its repeating interval is reappointed every time it is delivered. As for a one-shot alarm, its repeating interval is simply set at 0. Moreover, the window interval, which starts at the nominal delivery time, is introduced to allow an alarm to be delivered at any point within the interval, instead of at the nominal delivery time exactly. This time flexibility enables alarm alignment. The native *alignment policy* in Android is as follows. When an alarm is to be inserted into the queue, the alarm manager sequentially examines the queue entries, each of which contains some alarms that can be delivered together, to find an entry in which every alarm's window interval overlaps with that of the new alarm. If such an entry is found, the new alarm is simply placed in the entry; otherwise, a new entry is created for the alarm. However, if the same alarm still exists in the queue when an alarm is to be reinserted, the alarm manager will reinsert all the other alarms, together with the new alarm, into the queue according to their nominal delivery times. Such a realignment design seeks to further reduce the number of wakeups at a cost of slight computation overhead. Note that the above policy is applied to wakeup and non-wakeup alarms separately.

### 2.2 Example of Benefits

To investigate the efficacy of the native alignment policy, we conducted an experiment on an Android smartphone (LG Nexus 5) with two WPS[2] location-based apps and one calendar app installed. Each of the location-based apps registers an alarm to periodically track the smartphone's position based on nearby Wi-Fi and cellular signals. Meanwhile, the calendar app registers an alarm to notify the user of a scheduled appointment via the smartphone's vibrator. We used the Monsoon Solutions power monitor to measure the energy consumed for delivering each alarm. Based on our measurements, each alarm delivery

[1]AlarmManager manages wakeups registered for *internal* tasks, while Google Could Messaging (GCM) deals with wakeups caused by *external* messages. The two mechanism are compatible in Android and orthogonal to each other.

[2]WPS stands for Wi-Fi positioning systems relying on signals from nearby Wi-Fi access points and cellular base stations.

for location positioning consumes 3,650 mJ, while the alarm delivery for calendar notification consumes 400 mJ. In addition, the energy required simply to awaken the smartphone, without wakelocking extra hardware components, is 180 mJ.

Consider a snapshot of the alarm manager, whose queue now contains the calendar alarm and one location-based alarm; meanwhile, the other location-based alarm is just inserted to the queue, as shown in Figure 2(a). According to the native alignment policy, the new alarm will be aligned with the calendar alarm because their window intervals overlap, as shown in Figure 2(b). As a result, the total energy consumed for the three alarms is 7,520 (= $400 + 3650 \times 2 - 180$) mJ. Figure 2(c) shows a more energy-efficient alignment which requires only 4,050 (= $400 + 3650$) mJ. In this alignment, the new alarm tolerates a further postponed delivery, instead of being delivered rigorously within its window interval, so that it can be aligned with the other location-based alarm.

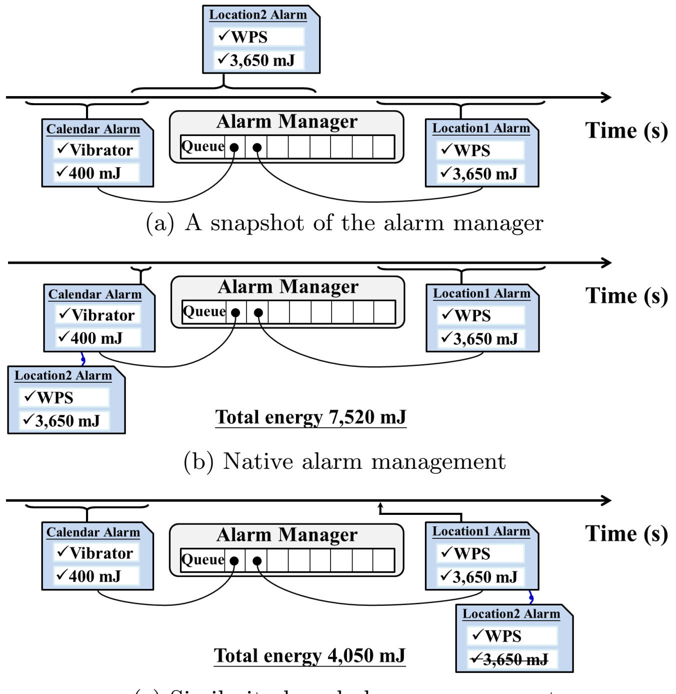

Figure 2: A motivating example

From the experiment, we draw two useful insights that inspire our design. First, alarm alignment may only save a small amount of the energy required to awaken the smartphone, whereas aligning alarms with similar hardware usage could result in significant energy savings. Second, when delivered, some alarms (like those wakelocking the screen or vibrator) will attract the user's attention, while some (like those wakelocking the Wi-Fi or GPS) are not perceptible to the user. Those imperceptible alarms, which account for the majority of alarms registered by resident apps [5], could be postponed beyond their window intervals[3] without significantly affecting user experience. Briefly, the *hardware similarity* between two alarms could reflect the degree of energy savings, while their *time similarity* could reflect the impact on user experience, if they are aligned. Nevertheless, several design challenges must be resolved to realize this concept in alarm management. The first challenge is to determine the hardware and time similarity between two alarms so that the two metrics can appropriately reflect energy savings and the quality of the user experience. Another challenge is to design an alignment policy that exploits the two metrics to achieve energy efficiency while maintaining user experience.

[3]An alarm's window interval is empirically determined by the app's developer and thus could be adjustable to some extent.

## 3. SIMILARITY-BASED ALARM MANAGEMENT

In this section, we explain how alarm similarity is determined (Section 3.1) and present our alignment policy along with its properties (Section 3.2).

### 3.1 Similarity Determination

#### *3.1.1 Hardware Similarity*

Before explaining how to determine hardware similarity, we describe two extreme cases that provide useful insights. Aligning two alarms that wakelock two mutually exclusive sets of hardware components can only save the energy required to awaken the mobile device. By contrast, if the two aligned alarms share a completely identical set of hardware components, the energy consumption can almost be reduced by half. It is mainly because the task that follows an alarm's delivery often continues only for a short period of time [5]; consequently, the energy overhead required to activate the components is relatively significant but can be amortized. As for any case between the two extremes, the total energy consumed for the two alarms can be partially reduced.

Based on the above observations, we classify hardware similarity into three levels: high, medium, and low. Note that some hardware components (such as the CPU and memory) are essential when a mobile device stays awake, so only those components that can be wakelocked autonomously by alarms are considered to determine similarity. In other words, the set of hardware components wakelocked[4] by an alarm can be empty. The hardware similarity between two alarms is high if their hardware sets are completely identical and not empty. If the two sets are not empty but partially identical, the hardware similarity is medium. Otherwise, the similarity is deemed low. Although the above classification is justifiable, there are different ways to classify hardware similarity. For example, we can obtain a four-level distinction by further dividing the medium similarity into two levels, depending on whether the identical components are energy hungry or not. In contrast, a two-level distinction, depending simply on whether the two alarms wakelock any identical components, could also be helpful.

#### *3.1.2 Time Similarity*

Similarly, we classify time similarity into three levels. The reasons behind the classification are as follows. To enable alarm alignment, the window interval is introduced into Android to allow for the postponement of an alarm within an "empirically acceptable" interval. We observe that the user's perception of alarms can be leveraged to align alarms more aggressively, without significantly affecting the user experience. This observation is inspired by an interesting finding in a recent study on unperceived activities in mobile systems [5]: most activities which awaken a smartphone are not perceptible to users. Thus, we classify alarms into perceptible and imperceptible alarms, where an alarm is perceptible[5] if it wakelocks user-perceptible hardware components (such as the screen, speaker, and/or vibrator), and is imperceptible otherwise. Because imperceptible alarms could be postponed beyond their window intervals, we associate every alarm with an additional attribute called the *grace interval* to further elevate the chance of aligning alarms. To ensure the delivery of a repeating alarm within every repeating interval, its grace interval must be smaller than its repeating interval. Meanwhile, to reveal the efficacy of the differentiation between

[4]In Android, the hardware wakelocked by an alarm is not specified during alarm registration, but immediately after its delivery. Thus, in our implementation, a newly-registered alarm's hardware set is initialized as empty and updated later.

[5]For completeness, one-shot alarms and newly-registered alarms are deemed perceptible because their hardware sets are unknown until they are delivered.

perceptible and imperceptible alarms, the grace interval should be no smaller than the window interval.

According to the above reasons, the time similarity between two alarms is considered high if their window intervals overlap. If the grace intervals, but not the window intervals, of the two alarms overlap, the time similarity is medium. Otherwise, the time similarity is deemed low. Undoubtedly, there are also different ways to classify time similarity. However, such a classification, along with our alignment policy, can conform to the delivery expectation of perceptible alarms and, meanwhile, preserve the delivery of imperceptible alarms in their specified repeating intervals periodically. More details will be discussed later in Section 3.2.2.

## 3.2 Our Alignment Policy

### 3.2.1 Alignment Rules

Now, we present our alignment policy, which exploits hardware similarity and time similarity, to reduce energy consumption while maintaining the quality of the user experience. The principle behind the policy is to align alarms with similar hardware usage as far as possible, provided that all perceptible (resp. imperceptible) alarms must be delivered within their window (resp. grace) intervals. To this end, our policy contains two phases, the search phase and the selection phase, as follows. Given an alarm to be inserted into the alarm queue, we first remove the same alarm if it is still in the queue. Then, the search phase sequentially examines the queue entries, each of which contains some aligned alarms, to find all the *applicable* entries for the new alarm in terms of user experience. Next, if multiple entries are found, the selection phase picks the most *preferable* entry in terms of energy efficiency. Finally, the new alarm is placed in the first found, most preferable entry if any applicable entries exist; otherwise, if there is no applicable entry or the alarm queue is empty, a new entry is created for the alarm.

More specifically, to identify the applicability of an entry and differentiate the preferability among entries, we define five attributes for each entry. The window (resp. grace) interval of an entry is the overlap of the window (resp. grace) intervals of all alarms in the entry, while the hardware set is the *union* of the hardware sets of all alarms in the entry. An entry is deemed perceptible if it contains any perceptible alarms, and imperceptible otherwise. For a perceptible (resp. an imperceptible) entry, its delivery time is the earliest point of its window (resp. grace) interval. Moreover, the hardware and time similarity defined for an alarm and an entry follow those defined for two alarms in Section 3.1.

Table 1: The applicability and preferability of a queue entry

| Time \ Hardware | High | Medium | Low |
|---|---|---|---|
| **High** | 1 | 3 | 5 |
| **Medium** | 2 | 4 | 6 |
| **Low** | $\infty$ | $\infty$ | $\infty$ |

In the search phase, if either the new alarm or the examined entry is perceptible, to ensure the delivery of every perceptible alarm within its window interval, the entry is considered applicable to the alarm only if the time similarity between them is high. Contrarily, if neither the alarm nor the entry is perceptible, the entry is applicable to the alarm if their time similarity is high or medium. Accordingly, the above rule ensures that no alarm will be delivered outside its grace interval (unless it is a non-wakeup alarm). Then, if multiple applicable entries exist, the selection phase picks the most preferable one based on Table 1, where 1 represents the highest degree of preferability and $\infty$ represents that the entry is not applicable. The reason behind the table's setting is as follows. Because the search phase ensures the quality of the user experience, as far as energy efficiency is concerned, an entry with a higher degree of hardware similarity is preferable. Furthermore, for entries with the same degree of hardware similarity, an entry with a higher degree of time similarity is always preferable. Note that, like the native alignment policy, our policy is applied to wakeup and non-wakeup alarms separately.

### 3.2.2 Properties

Next, we discuss some properties with respect to the delivery behavior of different types of alarms under our alignment policy. In Android, wakeup alarms can be classified into one-shot and repeating alarms. We further classify repeating alarms into perceptible and imperceptible alarms. Because our policy implementation treats one-shot alarms as perceptible alarms which must be delivered within their window intervals, the delivery expectation of these two alarm types is the same as under the native alignment policy. Before discussing the delivery behavior of imperceptible alarms, we define two notations for ease of presentation. Let the window interval and the grace interval of some alarm respectively be $\alpha$[6] and $\beta$ times its repeating interval. Based on the definition of the grace interval introduced in Section 3.1.2, we have $0 \leq \alpha \leq \beta < 1$.

We depict two properties of imperceptible alarms as follows. The maximum interval between adjacent deliveries of any imperceptible alarm is guaranteed to be $(1+\beta)$ times its repeating interval for both static and dynamic repeating alarms. This is because two adjacent deliveries will occur in different repeating intervals; moreover, the former delivery can occur at the nominal delivery time at the earliest, and the latter can occur at the end of the grace interval at the latest. On the other hand, the minimum interval between two adjacent deliveries is guaranteed to be 1 and $(1-\beta)$ times the repeating interval for dynamic and static repeating alarms respectively, because the former delivery can occur at the grace interval at the latest and the latter can occur at the nominal delivery time at the earliest. By a similar argument, the maximum interval under the native policy is $(1+\alpha)$ times the repeating interval for both static and dynamic repeating alarms; meanwhile, the minimum interval is 1 and $(1-\alpha)$ times for dynamic and static repeating alarms respectively. The maximum interval ensures that the alarm will be delivered at least once, while the minimum interval helps avoid bursty deliveries of the same alarm. Accordingly, each imperceptible alarm will be delivered once and only once in every specified repeating interval. Note that the above discussion can be directly applied to non-wakeup alarms when the mobile device stays awake. If the device is asleep, non-wakeup alarms will not be delivered until the device is awakened again, which is the same as under the native policy.

## 4. PERFORMANCE EVALUATION

## 4.1 Experiment Setup

We implemented our design in Android and conducted extensive experiments on an LG Nexus 5 smartphone with 18 popular mobile apps. The smartphone's hardware and software specifications are detailed in Table 2. To reduce the potential influence of other devices in the same mobile network, the smartphone accessed the Internet via a TP-LINK WR841N 802.11n access point dedicated for our experiments. We used the power monitor produced by Monsoon Solutions to measure the smartphone's transient power and energy consumption. In addition, to profile each app's behavior, we inserted several hooks into the hardware WakeLock APIs, as well as AlarmManager, in the Android framework to log every alarm's time attributes and hardware usage at runtime.

For the performance evaluation, we found, from Google Play, 18 resident mobile apps with different time attributes and hard-

[6]In Android, unless otherwise specified by the app developer, the default $\alpha$ is set at 0.75.

Table 2: Specifications of LG Nexus 5

| Hardware | |
|---|---|
| CPU | Quad-core 2.26 GHz Krait 400 |
| Memory | 2GB LPDDR3 RAM |
| Cellular | 3G WCDMA UMTS/HSPA/HSPA+ |
| WLAN | 2x2 MIMO Wi-Fi 802.11 a/b/g/n/ac |
| Screen | 4.95in Full HD 1920x1080 IPS LCD |
| Peripheral | Speaker, Vibrator, Accelerometer, etc. |
| Battery | 3.8V 2300 mAh |
| **Software** | |
| OS | Android 4.4.4 |
| | Linux kernel 3.4.0 |

ware usage, as listed in Table 3. However, five apps (marked with *) behaved irregularly and their duration of hardware wakelocking was not reproducible. For a fair comparison, we developed an imitated app to simulate each of these five apps based on the time and hardware patterns of their alarms logged in advance. To reduce the potential impact of human intervention, we wrote an Alarm Clock app that automatically turns off the speaker and vibrator immediately after it launches a perceptible notification for one second. Moreover, we used WPS, instead of GPS, because the time required by GPS to locate a position varied significantly. In the table, ReIn denotes the repeating interval (in seconds) of the major alarm in the app; the window interval was $\alpha$ times the repeating interval; and S/D indicates whether the alarm is static or dynamic repeating. We investigated two different scenarios: light and heavy workloads. The light workload comprised Alarm Clock (which launched a perceptible notification every 30 minutes) and the first 11 apps (whose alarms wakelocked the Wi-Fi only). Because all alarms (except for the perceptible one) wakelocked the same hardware, this scenario evaluated our design's efficacy when time similarity was applied only. By contrast, the heavy workload comprised all 18 apps, including the last six apps whose alarms would individually wakelock the WPS, the accelerometer, or the speaker and vibrator. This scenario demonstrated the efficacy when hardware similarity was exploited as well.

Table 3: Mobile apps used in the experiments

| H | L | Apps | ReIn | $\alpha$ | S/D | HW Usage |
|---|---|---|---|---|---|---|
| ● | ● | Facebook | 60 | 0 | D | Wi-Fi |
| ● | ● | imo.im | 180 | 0 | D | Wi-Fi |
| ● | ● | Line | 200 | 0.75 | D | Wi-Fi |
| ● | ● | BAND | 202 | 0 | D | Wi-Fi |
| ● | ● | YeeCall | 270 | 0 | S | Wi-Fi |
| ● | ● | JusTalk | 300 | 0 | S | Wi-Fi |
| ● | ● | Weibo | 300 | 0 | D | Wi-Fi |
| ● | ● | KakaoTalk | 600 | 0.75 | D | Wi-Fi |
| ● | ● | Viber | 600 | 0.75 | D | Wi-Fi |
| ● | ● | WeChat | 900 | 0.75 | D | Wi-Fi |
| ● | ● | Messenger | 900 | 0.75 | S | Wi-Fi |
| ● | ● | Alarm Clock | 1,800 | 0 | S | Speaker & Vibrator |
| ● | | Drink Water | 900 | 0.75 | S | Speaker & Vibrator |
| ● | | Noom Walk* | 60 | 0.75 | S | Accelerometer |
| ● | | Moves* | 90 | 0.75 | S | Accelerometer |
| ● | | FollowMee* | 180 | 0.75 | S | WPS |
| ● | | Family Locator* | 300 | 0.75 | S | WPS |
| ● | | Cell Tracker* | 300 | 0.75 | S | WPS |

We compared our design, named SIMTY, with Android's native alignment policy [1], denoted as NATIVE and discussed in detail in Section 2.1. In SIMTY, the grace interval of every alarm approached its repeating interval, i.e., $\beta = 0.96$, to demonstrate that perceptible and imperceptible alarms can be treated extremely unequally (while keeping the minimum interval between adjacent deliveries to prevent the same alarm from being delivered again before the smartphone went back to sleep). To show the energy efficiency, the adopted metric was the total energy consumed in connected standby. Furthermore, to evaluate the impact on user experience, the adopted metric was the average of the *normalized delivery delays* of alarms, where an alarm's normalized delivery delay is defined as 0 if it is delivered within its window interval, and otherwise the delay time (behind its window interval) normalized by its repeating interval. Note that the smartphone was recovered to its factory settings before each experiment. We then installed and launched the pre-selected apps and left the smartphone in connected standby for 3 hours (which was sufficient to observe some regular patterns). We conducted each experiment three times to reduce the potential influence of uncontrollable factors (like instant network speeds and system alarms) and reported the average value.

## 4.2 Experiment Results

Figure 3 shows the total energy required by NATIVE and SIMTY with the light and heavy workloads. In terms of the energy consumed to keep the smartphone awake, SIMTY produces energy savings greater than 33% of the energy required by NATIVE for both scenarios. The result clearly demonstrates the benefits of the grace interval introduced for imperceptible alarms, as well as the preference of aligning alarms with similar hardware usage. On top of the hardware similarity, we observed that the duration of wakelocking a hardware component is another decisive factor in reducing energy consumption. That is, SIMTY could be further extended to first align alarms that wakelock the same hardware for a "similar amount of time". However, this requires that the duration of hardware wakelocking be specified during alarm registration in Android's future practice. Moreover, the sleep mode alone accounts for a significant proportion of the total energy consumption in connected standby. This portion of energy usage cannot be reduced by alarm alignment, and should motivate further investigation of low-power hardware designs. Overall, compared with NATIVE, SIMTY saves standby energy by 20% and 25% under the light and heavy workloads respectively. The saved energy is sufficient for SIMTY to prolong the smartphone's standby time by one-fourth to one-third.

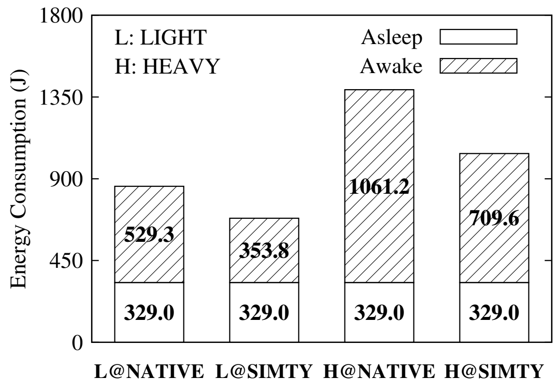

Figure 3: Energy consumption under NATIVE and SIMTY

Since SIMTY trades off potential delay for energy, provided that the user experience is not compromised, we proceed to investigate the average of the normalized delivery delay of perceptible and imperceptible alarms, as shown in Figure 4. For perceptible alarms, both the normalized delivery delays under NATIVE and SIMTY are zero. The result is as expected because, like NATIVE, SIMTY must deliver a perceptible alarm within its window interval. On the other hand, for an imperceptible alarm, SIMTY incurs an extra delivery delay of 17.9% and (resp. 13.9%) relative to its repeating interval under the light (resp. heavy) workload on average. The normalized delivery delay is shorter under the heavy workload than under the light workload, because finding a queue entry with a higher degree of time similarity is generally easier when more alarms are registered and involved in alignment. Note that the user experience is not compromised since these alarms are imperceptible to the user. Interestingly, some imperceptible alarms, which are supposed to be delivered within their window intervals under NATIVE, are slightly delayed. This phenomenon is indicated by the nonzero normalized delivery delays (about 0.4% to 0.6%). The reason is that the smartphone requires some time to awaken from sleep once the real-time clock triggers a hardware interrupt. This, in turn, causes the alarm manager to sometimes deliver alarms with $\alpha = 0$ slightly later than expected.

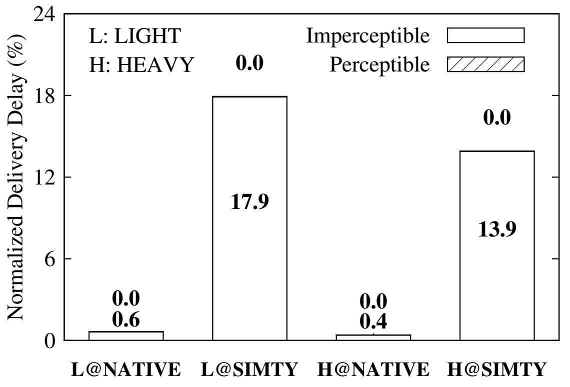

Figure 4: Normalized delivery delay under NATIVE and SIMTY

To gain more insights, we examine the wakeup breakdown detailed in Table 4. In each table entry, the numerator indicates the number of wakeups (under NATIVE or SIMTY) for the alarms acquiring the corresponding hardware, while the denominator indicates the expected number if no alignment policy is applied. To some extent, the smaller the ratio, the more effective the alignment policy. Since the CPU is essential for every wakeup, the corresponding numbers can be considered the total times that the smartphone was awakened. Note that the numbers with respect to the CPU also count one-shot and system alarms, while the numbers with respect to other hardware components only count the major alarms listed in Table 3, so that we can understand how the alarms were aligned. Moreover, the expected numbers of total wakeups are always smaller under SIMTY than under NATIVE. The reduction is due to the dynamic repeating alarms, whose numbers of deliveries depended dynamically on the time points at which they were delivered. Under SIMTY, the number of deliveries of a dynamic repeating alarm could at most be reduced by half as $\beta$ was close to 1. As can be seen in those entries with respect to the CPU, SIMTY can dramatically reduce the total number of wakeups required under NATIVE, from 733 to 193 and from 981 to 259 for the light and heavy workloads respectively.

Table 4: The wakeup breakdown

| Scenario / Hardware | Light Workload | | Heavy Workload | |
|---|---|---|---|---|
| | NATIVE | SIMTY | NATIVE | SIMTY |
| CPU | 733/983 | 193/830 | 981/1,726 | 259/1,370 |
| Speaker&Vibrator | 6/6 | 6/6 | 18/18 | 12/18 |
| Wi-Fi | 443/548 | 170/484 | 465/565 | 158/433 |
| WPS | 0/0 | 0/0 | 125/132 | 64/131 |
| Accelerometer | 0/0 | 0/0 | 227/300 | 186/300 |

In addition, we observed that the number of wakeups under SIMTY for each hardware component already approaches the least required number of wakeups. The reason that backs up the observation is as follows. For each hardware component, the number of wakeups is bounded by the experiment duration (i.e., 3 hours) divided by the smallest repeating interval of the static repeating alarms that wakelock the hardware (i.e, 60, 180, and 900 seconds for the accelerometer, WPS, and speaker & vibrator respectively). Under the heavy workload, the alarms were aligned by SIMTY to only wakelock the accelerometer 186 ($\approx 180 = 10800/60$) times, the WPS 64 ($\approx 60 = 10800/180$) times, and the speaker & vibrator 12 ($= 12 = 10800/900$) times. The wakelocking numbers of the Wi-Fi (i.e, 170 and 158 under the two workloads) are even smaller than 180 ($= 10800/60$), because the alarm that wakelocks the Wi-Fi with the smallest interval of 60 seconds was dynamic repeating, not static repeating. The statistical results demonstrate that the time and hardware similarity employed by SIMTY are, indeed, effective in reducing unnecessary wakeups and aligning alarms with similar hardware usage.

## 5. CONCLUDING REMARKS

We advocate that the mobile systems should be responsible for centralized wakeup management, rather than delegating responsibility to individual application developers. This paper presents an alarm manager called SIMTY, which exploits the hardware and time similarity among asynchronous alarms registered by different apps, to save standby energy while maintaining the quality of the user experience. SIMTY is implemented in Android but its rationale is also applicable to other mobile systems. The results of experiments based on an Android smartphone with 18 mobile apps installed show that SIMTY is very effective in avoiding unnecessary wakeups, while preserving the delivery expectation of perceptible alarms and the delivery periodicity of imperceptible alarms. In addition, the number of wakeups under SIMTY for each hardware component already approaches the least required number. To further reduce standby energy consumption, a sensible extension of SIMTY is to align alarms that wakelock the same hardware with the highest possible "duration similarity", if the duration of hardware wakelocking is specified during alarm registration in Android's future practices.

## Acknowledgement

This work was supported in part by the Ministry of Science and Technology under grant 104-2628-E-001-003-MY3.

## REFERENCES

[1] Android 4.4 APIs. https://developer.android.com/about/versions/android-4.4.html.

[2] C.-H. L. C.-K. Kang and P.-C. Hsiu. Catch Your Attention: Quality-retaining Power Saving on Mobile OLED Displays. In *Proc. of IEEE/ACM DAC*, pages 1–6, 2014.

[3] K. Kim and H. Cha. WakeScope: Runtime WakeLock Anomaly Management Scheme for Android Platform. In *Proc. of ACM EMSOFT*, pages 1–10, 2013.

[4] V. Könönen and P. Pääkkönen. Optimizing Power Consumption of Always-On Applications Based on Timer Alignment. In *Proc. of IEEE COMSNETS*, pages 1–8, 2011.

[5] C.-H. Lin, Y.-M. Chang, P.-C. Hsiu, and Y.-H. Chang. Energy Stealing - An Exploration into Unperceived Activities on Mobile Systems. In *Proc. of IEEE/ACM ISLPED*, pages 261–266, 2015.

[6] 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. of ACM MobiSys*, pages 267–280, 2012.

[7] F. Qian, Z. Wang, Y. Gao, J. Huang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Periodic Transfers in Mobile Applications: Network-wide Origin, Impact, and Optimization. In *Proc. of ACM WWW*, pages 51–60, 2012.

[8] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely. Energy-delay Tradeoffs in Smartphone Applications. In *Proc. of ACM MobiSys*, pages 255–270, 2010.

[9] A. Shye, B. Scholbrock, G. Memik, and P. A. Dinda. Characterizing and Modeling User Activity on Smartphones: Summary. In *Proc. of ACM SIGMETRICS*, pages 375–376, 2010.

[10] P.-H. Tseng, P.-C. Hsiu, C.-C. Pan, and T.-W. Kuo. User-Centric Energy-Efficient Scheduling on Multi-Core Mobile Devices. In *Proc. of IEEE/ACM DAC*, pages 1–6, 2014.

[11] P. Vekris, R. Jhala, S. Lerner, and Y. Agarwal. Towards Verifying Android Apps for the Absence of No-Sleep Energy Bugs. In *Proc. of USENIX HotPower*, pages 3:1–3:5, 2012.

[12] F. Xu, Y. Liu, T. Moscibroda, R. Chandra, L. Jin, Y. Zhang, and Q. Li. Optimizing Background Email Sync on Smartphones. In *Proc. of ACM MobiSys*, pages 55–68, 2013.

[13] Z. Zhuang, K.-H. Kim, and J. P. Singh. Improving Energy Efficiency of Location Sensing on Smartphones. In *Proceedings of ACM MobiSys*, pages 315–330, 2010.