國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

智慧型裝置待機時喚醒事件規劃

Battery Life Extender - Similarity-Based Wakeup Management in Connected Standby

高淳浩

Chun-Hao Kao

指導教授：郭大維博士, 修丕承博士

Advisor: Tei-Wei Kuo, Ph.D., Pi-Cheng Hsiu, Ph.D.

中華民國 104 年 6 月

June, 2015

中文摘要

為了保持與傳統桌上型電腦一樣的功能，一種典型的應用程式類型-常駐程式透過頻繁地喚醒待機中的行動裝置來保持應用程式的正常運行與硬體元件的使用，即使使用者根本沒有使用此應用程式。這使得保持在待機狀態的行動裝置時時保持在喚醒狀態，巨量, 快速地消耗有限的電池，這個問題隨著使用者安裝的應用程式越來越多樣而越來越嚴重。在這篇論文，我們發現目前行動裝置對於應用程式的事件規劃並沒有為待機狀態的行動裝置做最佳化，因此，我們探以事件相似度的觀點來同時表達兩個相反的目標-使用者體驗與節能, 透過合併相似度高的事件並同時達成這兩個目標。越高的硬體使用相似度，代表合併此事件帶來越多的節能；越高的時間相似度，代表合併此事件損失越少的使用者體驗。除此之外，我們將此事件規劃的設計實作在市售的智慧型手機上並結合市面上的應用程式驗證結果，實驗的結果顯示若考量事件相似度的設計平均可以減少 25% 應用程式事件的耗電，最高可達 32% 應用程式事件的耗電而不喪失使用者感受的到的事件的使用者體驗。

關鍵字: 排程, 節能, 行動系統, 待機狀態, 常駐程式

Abstract

Resident applications are a typical type of mobile applications. In order to maintain the same functionality as traditional desktop, the resident applications frequently awake the mobile device for processing and accessing of hardware components when the mobile devices are in sleep mode even if user doesn't use the application. The problem gets worse with the increasing of installed mobile applications. In this paper, we show that existing event scheduler are not optimized for operation in connected standby mode and we exploit the concept of *event similarity*, which expresses two opposite goals, user experience and energy efficiency, simultaneously. With higher hardware similarity, the scheduler gains more energy reduction; with higher time similarity, the scheduler lose less impact of user experience. Furthermore, we integrate our design into the Android operating system and evaluate it with real-world mobile applications. The experimental results shows that the novel event scheduler reduces the average energy consumption by 25% and up to 32% with little impact on user experience.

Key words:Scheduling, Energy Efficiency, Mobile System, Connected Standby, Resident Application

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Resident applications, a typical type of applications, are important for mobile devices to keep the same functions as applications in traditional desktop. Most of famous applications in mobile devices are resident applications. For example, social network applications run in background to receive latest notifications and instant messaging applications offer real-time text transmission over the Internet when devices are in *connected standby* state, in which the screen is off while the network connectivity stays active [1]. In order to keep the resident applications execute as expected, the mobile devices keep in connected standby and waiting for various internal events, such as network probing, connection maintenance, data synchronization and local tasks [2].

How resident applications threaten the battery life of mobile devices? The resident application would register the wake up events that are delivered in connected standby. In the connected standby state, in order to process the registered event, the clock hardware awakes CPU from sleep mode and resumes device's operating system, as well as the system services. Then, the system service checks there are any pending events and deliver these pending events to satisfy application's requirement. The procedure keeps mobile devices in a high power mode, in which average power is more than ten times higher than sleep mode and quickly depletes the battery of mobile devices. The situation gets worse with the increasing of installed resident applications. For example, the procedure of waking up and sending a packet to maintain connection consumes 1.39J, which is approximate 1 minute energy consumption when device in sleep mode. Suppose that user installs one social network application that awakes every 5 minutes and one instant messaging application that awakes every 200 seconds, the device will be awake total 720 times and the battery life decreases from 9 days to 6 days, a reduction of more than 30%.

In this paper, we present the first wake up event management for mobile device in connected standby. The overall design of today's mobile wake up event management doesn't take into account the characteristics of each events and the user's requirements: the event's perception is not considered into the design of scheduler, the resource management and the event alignment are un-optimized. Thus the mobile device wakes up frequently and redundantly accesses the energy-hungry hardware components. These shortcomings drain the mobile device of its battery.

Based on our finding, we develop a similarity-based event management to assure the exact delivery of user-perceivable events and reduce the energy consumption of user-imperceivable events. It includes three design principles for energy-efficient event management on mobile devices in connected standby.

- In current systems, the event scheduler treats events equally. Thus the efficacy of alignment is not maximized. We define the *event importance* and reduce the wake up times by aligning less importance events to improve energy efficiency.

- In connected standby, every hardware access causes significant energy consumption and seriously threaten the battery life. We explore the *hardware similarity* among events to reduce the hardware access times. With the information of hardware usage of each events, our scheduler aligns the events that have the most similar hardware usage first to reduce the changes of power state and the awake time of each hardware components.

- The protocol design and the application's policy gets more flexible on mobile devices. We explore the *time similarity* among events to constrain the level of inexact delivery on less importance events. We provide two properties: average number of events and maximum interval to help developer design their protocol or application's policy in an energy-efficient way.

Collectively, these techniques achieve significant energy reduction of wake up events. Specifically, we have implemented them by modifying system service on commercial mobile devices. Experimental results show that our revised event scheduler is able to improve wake up events' energy efficiency, reducing the average energy consumption by 25% and up to 32% with little impact of user experience.

In summary, the main contributions of this paper are:

- We show that the wake up events in a connected standby state is a major source of energy consumption in today's smartphone, and that existing event scheduler are not optimized for operation in this mode.
- We propose a similarity-based event management to reduce the energy consumption in connected standby with little impact on user experience.
- We implement these techniques on a commercial mobile device and show that it is significantly more energy efficient.

The remainder of this paper is organized as follows. Section 2 provides some background knowledge and a motivated example that shows the energy reduction with specific characteristics of events. Section 3 describes the detail of event scheduler design, includes definition of event importance and similarity design. Section 4 shows the experimental results on commercial mobile devices. Section 5 surveys the related work and section 6 gives a conclusion of this paper.

Chapter 2

BACKGROUND AND MOTIVATION

2.1 Wake Up Events in Mobile Device

Not like the traditional stationary desktops, most of mobile devices apply an *aggressive sleeping policy* to reduce the energy consumption, especially when user doesn't use the mobile devices. By the aggressive sleeping policy, the default mode of all hardware components is OFF. It means that the hardware components would get into sleep mode as soon as possible after finishing the workload to reduce the high power consumption on awake mode. For example, in the Android system, when you turning off screen, the operating system will examine whether any application requests the CPU resource. If there isn't any application requests the CPU resource, the mobile device would freeze the running tasks and enter the sleep mode.

How can a mobile device wake up to execute scheduled events by itself? From Figure 2.1, there is a system service called *alarm manager service* [3] in mobile device. The developer can register the events in the future through alarm manager service. In the current system design, each event, labeled as E_i, has 4 important attributes.

Trigger Time ($T[E_i]$): Alarm manager service will deliver the event as nearly as possible to the trigger time. The trigger time must be after the time $(R[E_i])$ that you register with alarm manager service or the event will be delivered instantly.

Repeating Length ($RL[E_i]$): Lots of resident applications would register repeating event to improve their application's user experience. For example, Facebook will send

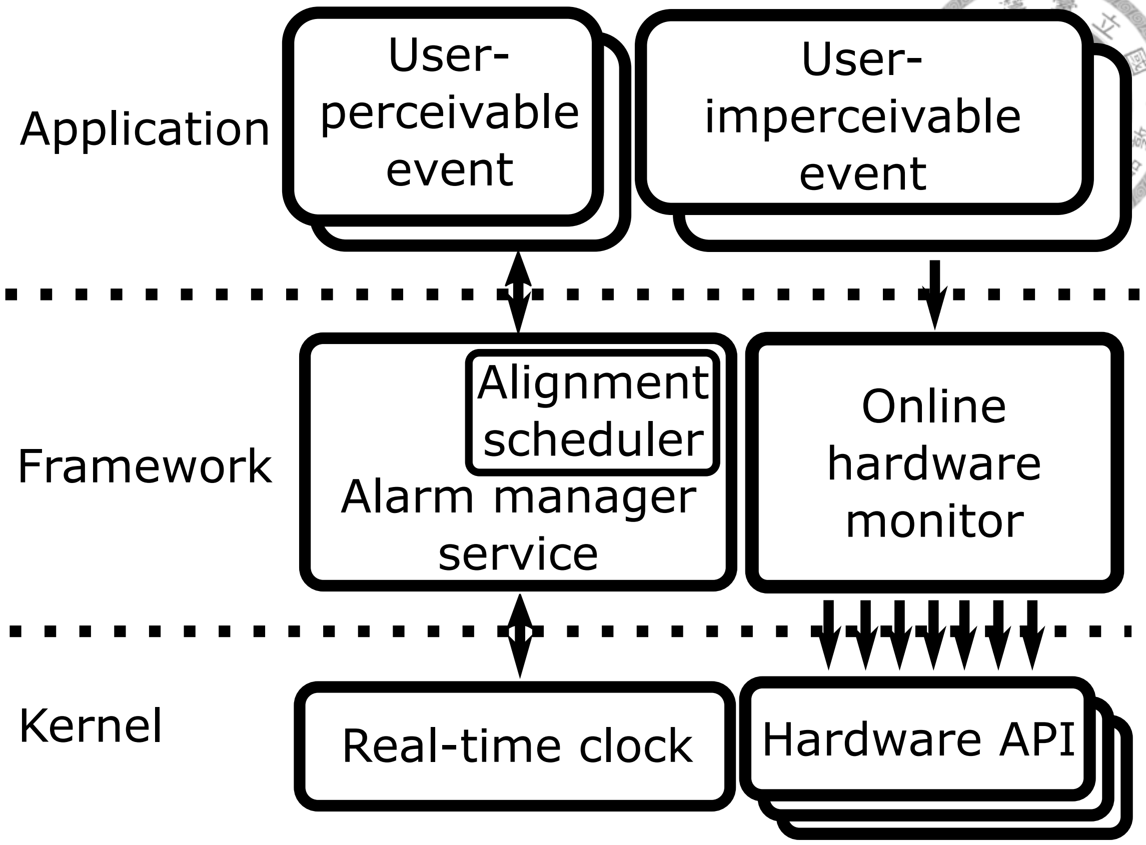


Figure 2.1: System Architecture

a heartbeat every 15 minutes to guarantee the connection between Facebook's server and the mobile device. If an event's repeating length is 0, it means that the event is an one-shot event.

Window Interval ($WI[E_i]$**)**: In order to reduce the number of times of wake up, alarm manager service will try to align events to deliver these events at same time. There is an attribute, window interval, to set the time tolerance. The attribute has two variables, start time and end time. The start time is the earliest time that the event could be delivered and the event cannot be delivered later than end time. If the start time is equal to the end time, it means that the event is an exact event.

Is Wake Up or Not ($WU[E_i]$**)**: Due to the aggressive sleeping policy, mobile device stays in sleep mode most of time. Lots of events don't need to deliver exactly in the sleep mode. For example, the user data collector just needs to upload the data at least once a day. In the sleep mode, only wake up event can awake the device to process the scheduled event. On the other hand, the non-wake up event would be delayed or advanced according the window interval and the neighbor wake up events to reduce the number of times of wake up. When mobile device is in the awake mode, wake up event and non-wake up event have the same behavior.

Alarm manager service runs in an event-driven architecture. It usually maintains two event queues, one is the wake up event queue and another is the non-wake up event queue, both sorted by start time of the window interval in increasing order. Initial, the queues are empty so alarm manager service just wait for new event. There are three important situations would trigger alarm manager service:

Add a New Event: The event would be added to the event queue according its type. There is an *event scheduler* to decide whether the event merges with exist events. We will give a detailed explanation of the procedure in next paragraph. If the event is a wake up event and the event is the first element in the queue, alarm manager service would pass the trigger time of the event to real-time clock module in Linux kernel to awake at trigger time. On the other hand, if the event is a non-wake up event and the event is the first element in the queue, alarm manager service would set a message [4] that would send to itself at trigger time. The message system only operates when the device is in the awake mode so unlike wake up event, it doesn't cause additional power state change.

Get a Message: alarm manager service would traverse the two queues to find the events whose trigger time is advanced or equal to current time and deliver those events.

Real-time Clock Alarm: When alarm manager service pass the trigger time to *real-time clock module*, real-time clock module registers an alarm in Linux kernel. The alarm would be fired by the clock outside the CPU and cause the mobile device awake to resume alarm manager service. The power state change would consume lots of energy and shorten the battery life [5]. After the awaking procedure, alarm manager service would do the same thing like ”get a message”, traversal and deliver the events.

After ”Get a Message” and ”Real-time Clock Alarm”, alarm manager service would determine the next wake up moment, pass the next wake up moment to real-time clock module and wait for the three situations again.

How do alarm manager service aligns an event with another? The decision is made by event scheduler. While adding a new event to the event queue, the event scheduler will check from the head of queue to the end of queue. The new event will align to the exist event whose window interval overlaps with the window interval of the new event to reduce the number of wake up. It is notable that if there are two or more events in the event queue can be aligned with the new event, the new event will be aligned to the most advanced in the event queue. In other words, *the new event would be merged to the first overlapped event in the queue to reduce one time of wake up*.

2.2 Motivation

In order to save the energy consumption within standby mode, we try to align the events with additional information of perception, time similarity and hardware similarity. The event will be presented like Figure 2.2. Figure 2.3 shows an example of the effectiveness of these additional information. Figure 2.3a is the initial situation. E_i and E_j are the events in the event queue and we are adding E_k to the queue. E_k could be merged to E_i or E_j because the stripe parts, window interval, are overlapped. Without the alignment, the three events trigger separately and the total energy consumption is 1230 Joules. With the original policy, as shown in Figure 2.3b, E_k would be added to the first overlapped event in the queue so E_i and E_k would be merged. The total energy consumption is 1215 Joules. With the information of perception, E_i must be triggered exactly because it is perceived by user. On the other hand, E_j and E_k will not be perceived by user so we can trigger them outside the window interval with little impact of user experience. As shown in Figure 2.3c, we find that *merge events with same hardware usage would save more energy than events with different hardware usages*. it is more suitable to merge E_j and E_k because both of them needs the GPS information. After the alignment, the mobile device only needs to contact to satellites to get the GPS information once and the total energy consumption reduces to 630 Joules. In other words, E_j and E_k can be aligned because their time similarity and hardware similarity are high. The selection problem would be complex when the number of events and the number of hardware components increase. We will present an algorithm to solve the problem in Chapter 3.

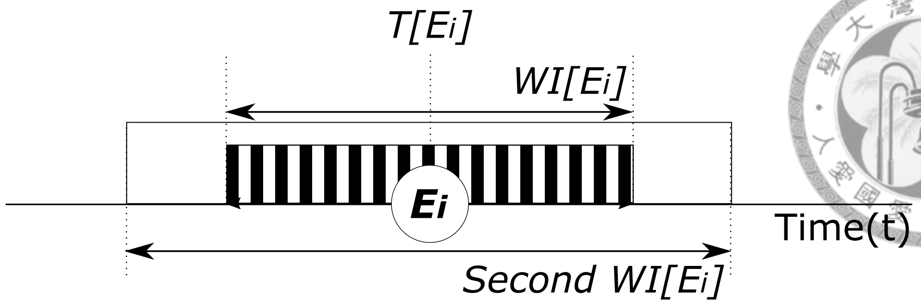
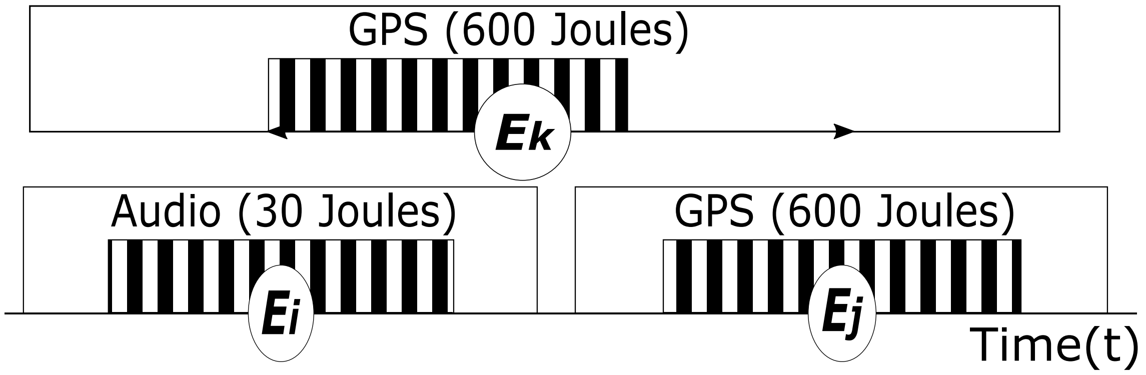
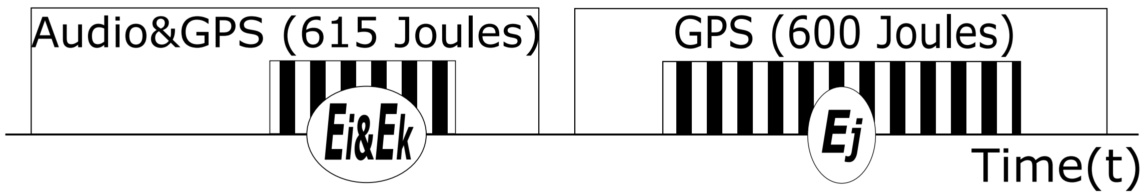


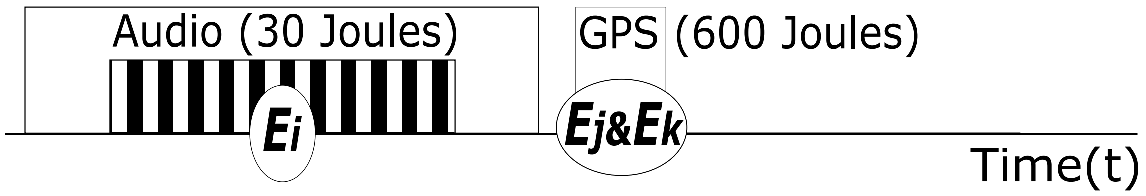
Figure 2.2: Event Representation in Timeline



(a) Events without Alignment



(b) Events with Original Policy



(c) Events with Hardware Information

Figure 2.3: A Motivation of Events with Additional Information

2.3 Design Challenges

Although the previous example shows the benefit of perception and similarity, several design challenges need to be resolved to realize the concept, in additional to the issue of compatibility with existing mobile operating systems.

Determining Event Importance: The first challenge is how to determine the importance of each event to reflect the user's perception of its delay. A straightforward way is to categorize mobile applications based on how they affect user experience and assign them fixed importance. However, an application may have two or more events that have different behavior in connected standby; moreover, the event's behavior changes as the version of applications and devices.

Determining Event Similarity: The second challenge is how to determine the time similarity and the hardware similarity among events to express two opposite goals, user experience and energy efficiency, in a concept, similarity. How many level of time similarity is enough to reflect the time tolerance of each importance level? In addition, the energy reduction after aligning events might vary from device to device. How to design hardware similarity to present the energy reduction after align the events?

Scheduling Based on Importance and Similarity: The last challenge is how to ensure the exact delivery of the user-perceivable events and reduce the energy consumption of the user-imperceivable events in connected standby based on the information of event's importance, time similarity and hardware similarity. We have to provide the worst-case time error of each importance level to let the developer design the application's policy; moreover, with little impact of user experience, how to schedule these events in a energy-efficient way? In other words, the event scheduler needs to reduce the energy consumption with the constraint on user experience.

Chapter 3

SIMILARITY-BASED WAKE UP EVENTS POWER MANAGEMENT

In this chapter, we will introduce a *similarity-based algorithm* to manage the events in an energy-efficient way. First, we will present the definition of event importance. Second, the definitions of importance and the similarity among events in *time* dimension and *hardware usage* dimension are proposed. Third, under the consideration of user experience and energy-efficient, we design an event scheduler. After all, we implement the algorithm on a commercial mobile device and test the result with famous applications.

3.1 Importance of Wake Up Events

We propose that the event scheduler should treat events differentially to reduce energy consumption with little impact of user experience. A trivial solution [5] for wakeup management is to determine a fixed interval to awake mobile devices periodically. The solution treats all events equally so it leads a phenomenon of delay of user-perceivable event. For example, your alarm clock might delay at most five minutes if the fixed interval is five minutes.

3.1.1 Observation

User Perception: Mobile applications with a large number of creative functionalities induce various user habits and behavior patterns. However, Users pay more attention if the event is perceivable. Because the exact delivery of notifications and alarms dominates the user experience when mobile device in connected standby, this observation suggests that the events using user-perceivable hardware components should be differentiated from others.

Based on the above observation and the event's attribute, we classify event importance into three levels: high (perceivable), medium (wakeup), low (non-wakeup). However, there are different way to classify importance. For example, we can further divide the wakeup events into two importance levels, depending on how frequent user uses the applications that deliver the events.

3.2 Similarity of Wake Up Events

3.2.1 Observation

We conducted some preliminary experiment to adjust trigger time of the events and did some survey of the events. We find two observations that could provide us insights to determine the similarity.

Time Tolerance: There are lots of events that happen when the mobile device is in connected standby. The events include network probing, connection maintenance and data synchronization.. [2] From the user view, Almost all of those events don't initiate by user so user are not aware that events deliver inexactly. From the developer view, these events are routines. They only need to be delivered once in every 15 minutes or at least 24 times in a day. Therefore, we could adjust the trigger time of these events without the perception by user.

Dynamic Power Management: Hardware components in the mobile device have different power state to reduce the power consumption [6]. There are two benefits of delivering the events with same hardware usage at same time. 1) *Overhead on transition*: The energy consumed by transition between sleep mode and awake mode can be saved. 2) *Awake mode power consumption*: For some specified hardware components, we can reduce the time in the awake mode after alignment. For example, if we merge two events that both want to get location information, we only have to awake the GPS module once and deliver the location information for the two events at same time.

With the two insights, there are two relations between every two events. If the two events have higher similarity in time dimension, it means that we can get a wake up moment that deliver the two events without less loss of user experience; If two events have higher similarity in hardware usage dimension, we can align the two events to gain more energy reduction.

3.2.2 Similarity Determination

So how similar are the two events? We define the similarity between two events as follows.

Time($TIME[E_i, E_j]$): We define the time similarity of two events into three levels: HIGH, MID and LOW. The first gap, HIGH and MID, is designed for HIGH importance event. If the time similarity is HIGH, it wouldn't lose user experience if we align two events. If the time similarity is MID, it means that the delay-sensitive events wouldn't tolerate the time shift after merging. Last, if the time similarity is LOW, it suggests that the two events shouldn't be merged because of the mass time shift unless it is not wakeup event. In our implementation, we set HIGH if the window intervals of the two events are overlapped. It means that after merging the two events, we can find a trigger time in the two window intervals and this is also the same guarantee as original policy. Set MID if the second window intervals are overlapped.[1]The start time of second window interval must be after the trigger time of previous event in the same series and the end time of second window interval must be before the trigger time of next event in the same series. The constraints satisfy the order of events in same series. Last, set LOW if it's not HIGH or

MID. Figure 3.1 shows the three cases of time similarity.

We show the three cases of time similarity in Figure 3.1. E_i1 and E_j1 are in HIGH time similarity so event scheduler can align them without loss of user experience. E_i2 and E_j2 are in MID time similarity. The event scheduler can give much useful guarantee for MID time similarity. We will discuss it at 3.3.2. E_i3 and E_j3 are in LOW time similarity. We cannot give any guarantee of the trigger time if event scheduler aligns two events with LOW time similarity. It is only suitable for the events that are not essential in connected standby.

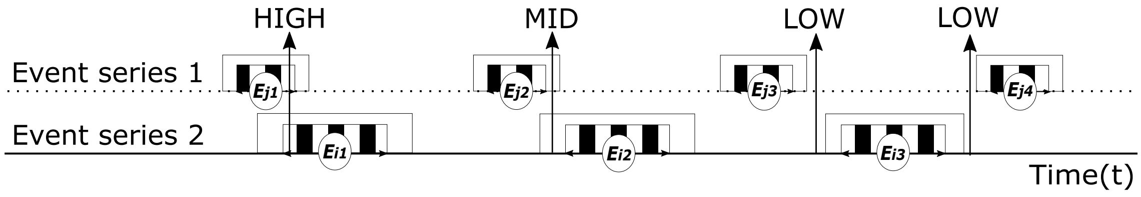


Figure 3.1: Time Similarity Example

Hardware($HARDWARE[E_i, E_j]$): We add a new attribute, hardware usage, for each event. The hardware set includes network, vibrator, audio, screen, accelerometer, network GPS and GPS. We choose the hardware set because they are frequently used in connected standby. For example, health applications usually periodically access accelerometer to measure the walk time and run time and remind you when you have to exercise more [7]. Location-based service would send you notifications about the attractions around you based on the information from GPS. The hardware set used in connected standby has to be manipulated carefully to maintain the standby time. We also define hardware similarity into three levels: HIGH, MID and LOW. HIGH hardware similarity means that the hardware usage of the two events are completely identical. The event scheduler can align the two events to gain the highest energy reduction; MID hardware similarity means that the two events share at least one hardware component. Because lots of events only use one hardware component in the hardware set, the MID level is enough to cover most cases; LOW hardware similarity means that they don't share any hardware components. There are different ways to classify the hardware similarity. We can further divide

[1]In order to demonstrate that the user-imperceivable events can be treated extremely unequally without affecting user experience significantly, the second window interval is $[T[E_i]-0.99*RL[E_i], T[E_i]+0.99*RL[E_i]]$ for repeating events and $[T[E_i], T[E_i]+0.99*(T[E_i]-R[E_i])]$ for singleton.

MID hardware similarity into sharing the most energy-hungry component and sharing the less energy-hungry component.[2]

3.3 Event Scheduler Designs

3.3.1 Design Principles

Before going to the event scheduler design, we discuss the design principles used in our scheduler design to satisfy the user experience and energy-efficient simultaneously.

User Experience Consideration: Different events have different delay tolerance from a user perspective. Most of user attaches great importance to the events that they can perceive and doesn't pay attention to the less importance events [8]. The *user-perceivable* event means that the event uses vibrator, audio or screen to notify important thing to user. For example, a calendar notification that reminds you of a meeting has to be triggered exactly. However, user doesn't care when mobile phone collects the location information in connected standby. Therefore, We can differentiate between HIGH importance events and less importance events and use different criterions to manage two types of event.

Energy Efficient Consideration: Different hardware components have different energy saving after alignment. With two self-made apps, we control the events exactly and request different hardware components at each time. Compare the energy consumption between two situations: 1. Awake device twice and deliver two events separately that access same hardware. 2. Awake only once but deliver two events that access same hardware at the same time. We group the hardware components with similar energy saving and classify all the hardware components to five levels. Figure 3.2 shows the distribution of energy saving on each level. We find that *the energy saving for each level is scattered thus it easily chooses a weight for each level to represent the energy saving*. The Table 3.1 shows that the energy saving for each level and the hardware components in each level. For example, if we merge two events that both access location information through mobile network, it would save 13.6(CPU) + 217.1(Network GPS-3G) = 230.7 (uAh). By

[2]We will use the notation $SIMILARITY[E_i, E_j] =< TIME[E_i, E_j], HARDWARE[E_i, E_j] >$ to representation similarity.

the above steps, we can arrange all the hardware components into fixed level and get a weight for each level. Then, we can use the weights to predict the result after merging. Furthermore, if there is a new hardware component, we could classify it to the level that has the closest energy saving.

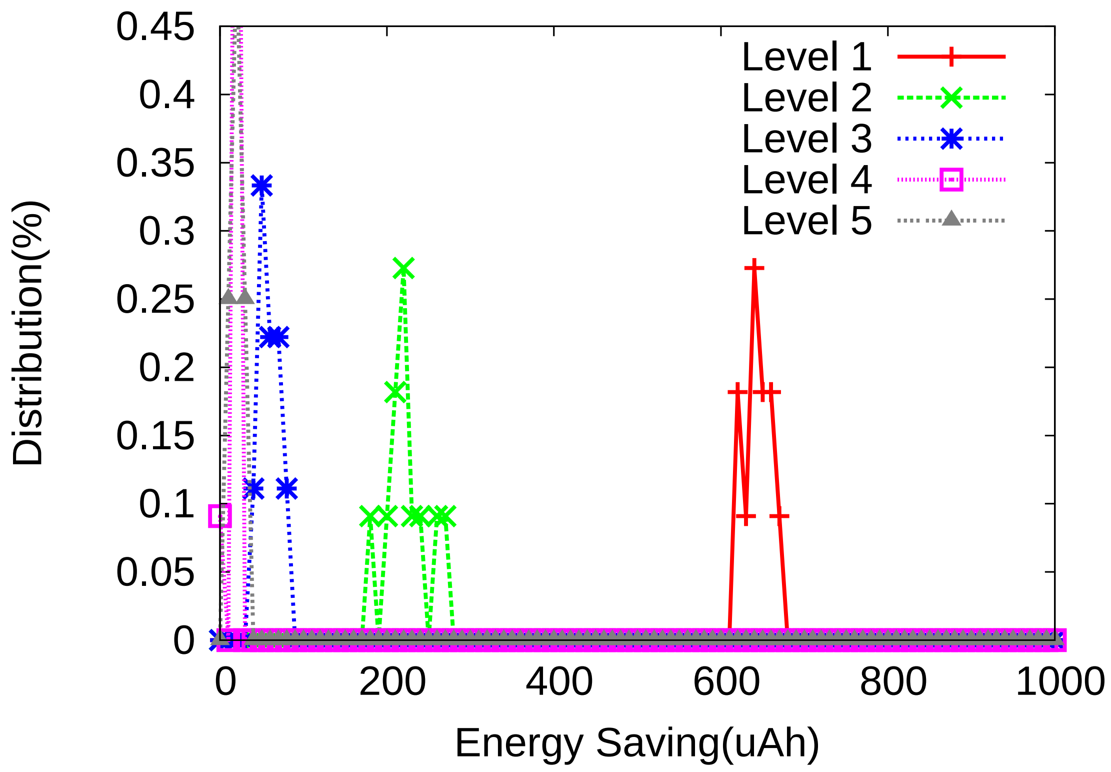


Figure 3.2: Energy Reduction Distribution for Each Hardware Level

Level	Energy Reduction(Joules)	Hardware Component(s)
1	634.927	Screen(, GPS)
2	217.0795	Network GPS-WIFI, Network GPS-3G
3	47.46836	Network-3G, Sensor-Acc(, Audio)
4	13.60363636	CPU, Vibrator
5	0.416212	Network-WIFI

Table 3.1: Energy Reduction for Each Hardware Level

Based on the above design principles, the event scheduler must ensure the exact delivery of user-perceivable events and reduce the energy consumption of user-imperceivable events by aligning same hardware usage event. The event importance and the time similarity among events can be used to ensure the exact delivery of user-perceivable events and the hardware similarity can be used to reduce the energy consumption.

3.3.2 Event Scheduler Design

The event scheduler will align events when add a new event to the event queue. We will explain the detail of alignment rules that determine whether a new event can be merged to an exist event by the event importance and time similarity. There might be two or more exist events can be aligned but only one event can aligned because the time similarity among events in the event queue must be LOW. The new event have to choose which exist event is better by the hardware similarity.

Alignment Rule: The first step, the scheduler has to determine whether two events can be merged. The operation executes when add a new event to event queue. The event scheduler traverses the event queue to find whether any events can merged with the new event. For each event in the event queue, the event scheduler determines that new event can be aligned to the exist event based on the following information: the importance of the new event, the importance of the exist event and the time similarity between the two events. In order to ensure the exact delivery of user-perceivable event, if one of the two events is HIGH importance, the two events only can be aligned if the time similarity is HIGH. In other words, the user-perceivable event will trigger in the window interval after alignment.

On the other hand, if none of the two events is HIGH importance, it means that both of the events aren't user-perceivable. The event scheduler will align the two events if the time similarity is equal or greater than MID. The alignment rules aren't limited. We choose the user-perception because it can be directly sensed by user and we treat user-perceivable event in a strict way to satisfy the user experience. On the other hand, we only align user-imperceivable events whose time similarity is greater than or equal to MID. It provides two good properties: 1) *The average number of events*: for repeating event, we can guarantee that the average number of events approaches original policy in the long term. It is suitable for the data collector. 2) *The maximum of interval between two events in same series*: let's take a look at any two continuous events, the original interval between any two continuous events is $1 * repeatinglength$. The advanced event could be shifted to approximate $1 * repeatinglength$ earlier and the last event could be shifted to $1 * repeatinglength$ later so the maximum of interval can be limited to $3 * repeatinglength$. The properties can be used in protocol design or the policy of data synchronization. For example, for connection maintenance , Facebook needs to send a heartbeat every 15 minutes, at least once every 60 minutes. It can register an inexactly repeating event whose repeating interval is 15 minutes and the operating system could help to decide when should wake up the device to process the event.

The alignment rules satisfy the use experience. For user-perceivable events, it assures the exact delivery so user don't aware the event scheduler. For user-imperceivable events, it provides two properties that are help developer to design their applications. Next, the event scheduler has to reduce the energy consumption with the constrains of user experience.

3.3.3 Selection Algorithm

In this section, after alignment rules determining how many events could be aligned to the new event, the event scheduler, however, has to choose which one is better for energy efficiency. Because the events in the event queue have been arranged by the alignment rules and selection algorithm, none of events in the event queue can be further aligned due to the time similarity. The scheduler has to choose the best one from energy-efficient perspective.

Selection Algorithm: How to align the events to reduce more energy consumption based on the information of similarity? From the second observation in section 3.2.1, it suggests that it is better to align the events with same hardware usage to gain more energy reduction. The problem becomes to find the most similar event that can be aligned to the new event. The input is the similarity between the new event and the mergeable events from the alignment rules. From the table 3.2, for each pair of similarities, we can find a rank value. The event scheduler has to find the highest rank and align the new event to the event belongs to the highest rank. Because the scheduler has satisfied the user experience by the alignment rules, you can find that the table gives higher hardware similarity higher rank value. The design would let event find the highest hardware similarity first to gain more energy reduction. If there are two or more events have highest hardware similarity, it checks the time similarity to reduce the impact of user experience.

Table 3.2: The Rank of Similarity

Hardware Time	HIGH	MID	LOW
HIGH	1	3	5
MID	2	4	6
LOW	x	x	x

After adding a new event to the event queue, *Get a Message* and *Real-time Clock Alarm* are same to the original policy because add more calculation might cause the wake up period longer than original to increase the energy consumption of each wake up moment. The alignment rules and selection algorithm has reduced the number of wake up times as much as possible.

3.4 Implementation Detail

In this section, we consider some technical issues that arise when implementing our scheduler in the Android operating system.

3.4.1 Online Hardware Monitor

In order to measure the hardware similarity among events, we need a online hardware monitor to know the hardware usage of each event. Because almost all of events are routines, we can use the last hardware usage to predict the current hardware usage. For network usage, the proc [9] provides the network usage for each application id so we only have to monitor the value after processing the event. For other hardware, the application's hardware requests are via the framework's api so we add 6 hooks to the hardware-related api to pass an event while application requests vibrator, audio, screen, accelerometer and

GPS. We identify who requests the hardware resource through *Binder* [10]. The binder provides the application id that calls the function. With the information of application id of running events, we can compare the application id to know the hardware usage of each event and store the hardware usage into a hashmap. The key of hashmap is the application id and alarm id because an application might have two or more event series; the value of hashmap is a bit string that represents the hardware usage. Each bit of the string represents a hardware component. 1 means that the alarm uses the hardware component and 0 means not. The bit strings can be quickly compared to get the hardware similarity. For example, an ”and” operation can identify whether the hardware usage is MID. The hashmap provides quick update and get functions that are frequently used in our algorithm.

3.4.2 Modification for Event Scheduler

For implementation of scheduler, we modify the source code of alarm manager service. We modify the part of finding an index of the new event in the event queue. Our scheduler traverses the event queue to find the mergeable events and return the index of the most similar event. After aligning the new event to the exist event, the window interval will be trimmed to the overlapping of window intervals of the two events and the trigger time is changed to the start time of the new window interval. In order to avoid misusing of wakelock [11] after rescheduling the events, we add a timeout for each wakelock that requests when screen off. The timeout is set to 20 seconds that larger than the execution time of all events that we observe.

Chapter 4

PERFORMANCE EVALUATION

4.1 Experiment Setup

To gain further insights into similarity-based wakeup management, we conducted extensive experiments on a commercial Android smartphone with some real-world mobile applications. We used a LG Nexus 5 smartphone, one of the most popular Android smartphones when this study was conducted. The specifications of the related hardware and software are detailed in Table 4.1. If necessary, the smartphone can access the Internet via a TP-LINK WR841N 802.11b/g/n access point dedicated for our experiments. We set the location access via network provider to reduce the potential influence of signal strength between GPS satellites and mobile device. We used the power monitor produced by Monsoon Solutions [12] to measure the smartphone's transient power and energy consumption. In addition, we save the history of events to measure their hardware usage, time error and trigger time.

We studied possible combinations of applications with different characteristics, namely, user-perceivable, high energy reduction and low energy reduction. Simulated alarm clock, simulated location-based applications(LBA) and network-based applications, the table 4.2 lists the all of applications were chosen for the performance evaluation. We write a simulated application that simulates the alarm clock because the alarm clock needs to be stopped automatically. For the LBA, the workload of Family Locator and FollowMee is not stable so we also write a simulated application that request the location informa-

Table 4.1: Specifications of LG Nexus 5

Hardware	
Processor	Krait 400 quad-core 2.26 GHz
Memory	2GB RAM
3G Network	3G WCDMA - bands 1/2/4/5/6/8
Screen	IPS LCD Full HD 1920x1080 px
Connectivity	2x2 MIMO WiFi 802.11 a/b/g/n/ac
Battery	3.8V 2300 mAh, 8Wh
Software	
OS	Android 4.4.4_r2 Linux kernel 3.4.0

Table 4.2: Mobile Applications Used in the Experiments

	Weather	IM	Social	LBA	Other
Network	LifeWeather GoWeather	Messenger WeChat Line Viber KakaoTalk	Facebook Twitter Weibo		TTPOD LINE Webtoon Bilibili anime
GPS				Simulated application	

tion periodically. The period of simulated applications refers to the Family Locator and FollowMee. All of the accounts that log in the applications are testing account in order to reduce the effect of notifications. We conducted the experiment under three scenarios, light user, casual user and heavy user. Light user and casual user only use the network-based applications and the number of events in casual user is approximate twice in light user. The heavy user uses network-based applications and LBAs simultaneous. The actual applications used in three scenarios as follow. 1) Light user: Alarm clock + 1 Weather + 2 IM + 2 Social + 2 Others. 2) Casual user: Alarm clock + 2 Weather + 5 IM + 3 Social + 3 Others. 3) Heavy user: Alarm clock + 2 Weather + 5 IM + 3 Social + 3 Others + 2 LBA.

We compared our similarity-based scheduler (denoted as SB) with a conventional design (denoted as Android) and a trivial solution, fixed interval - 5 minutes (denoted as FI5). Recall that we discussed the implementation and tunable parameters of SB in Section 3. Android employs the FCFS policy that aligns the new event to the first mergeable event to achieve less calculation and shorten the wake up period as described in Section 2.1. FI5 is proposed as a trivial solution. It awakes device every 5 minutes and delivers the events before the wake up moment. To show the efficacy of SB, Android and FI5 in energy reduction, the adopted metric was the total energy consumption of the wake up events required for a scenario. Furthermore, for the performance comparison, because user only aware the user-perceivable events, the metrics adopted were the time error of user-perceivable events. The experiment flow is as follows. 1) Log in all experiment applications under the scenario. 2) Restart the smartphone. 3) Wait for three minutes to complete the boot operation 4) Collect the log every 30 minutes.

4.2 Energy Consumption

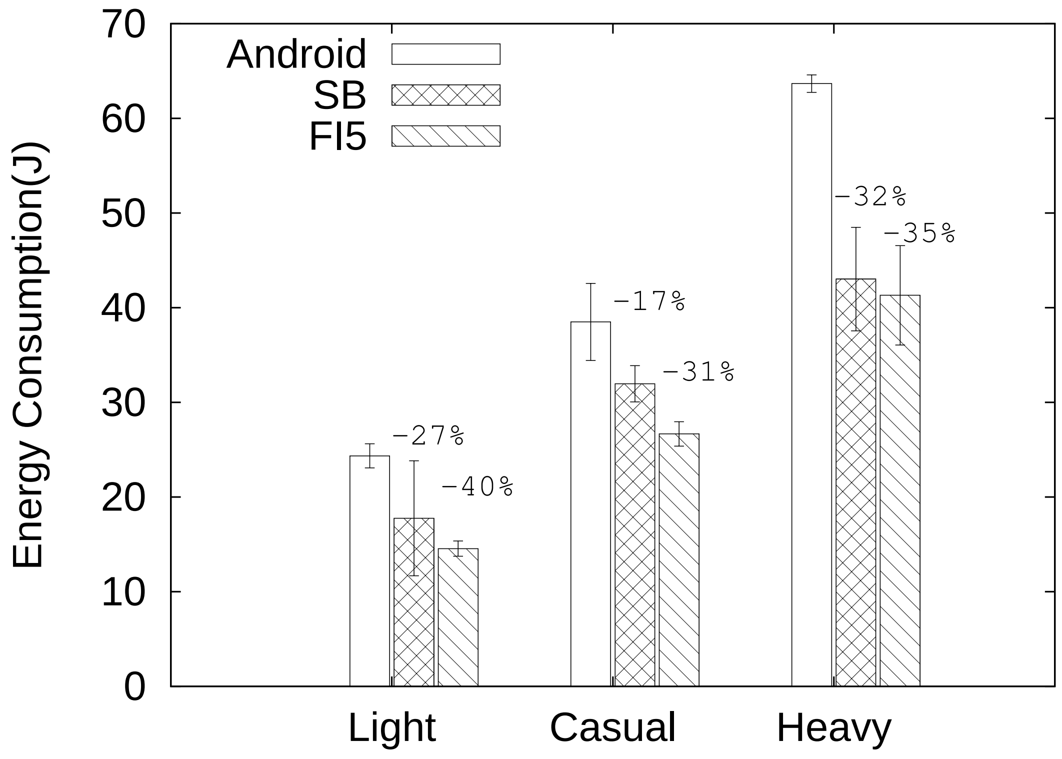


Figure 4.1: Energy Consumption under Three Connected Standby Scenarios

Figure 4.1 shows the energy consumption by Android, SB and FI5 under the three scenarios. In general, SB requires significantly less energy than Android. This result is as expected because SB tries to deliver events with similar hardware usage at the same time. However, in the scenarios when almost all events only access the network resources, SB can still reduce the energy consumption of Android by 27% under the light user scenario

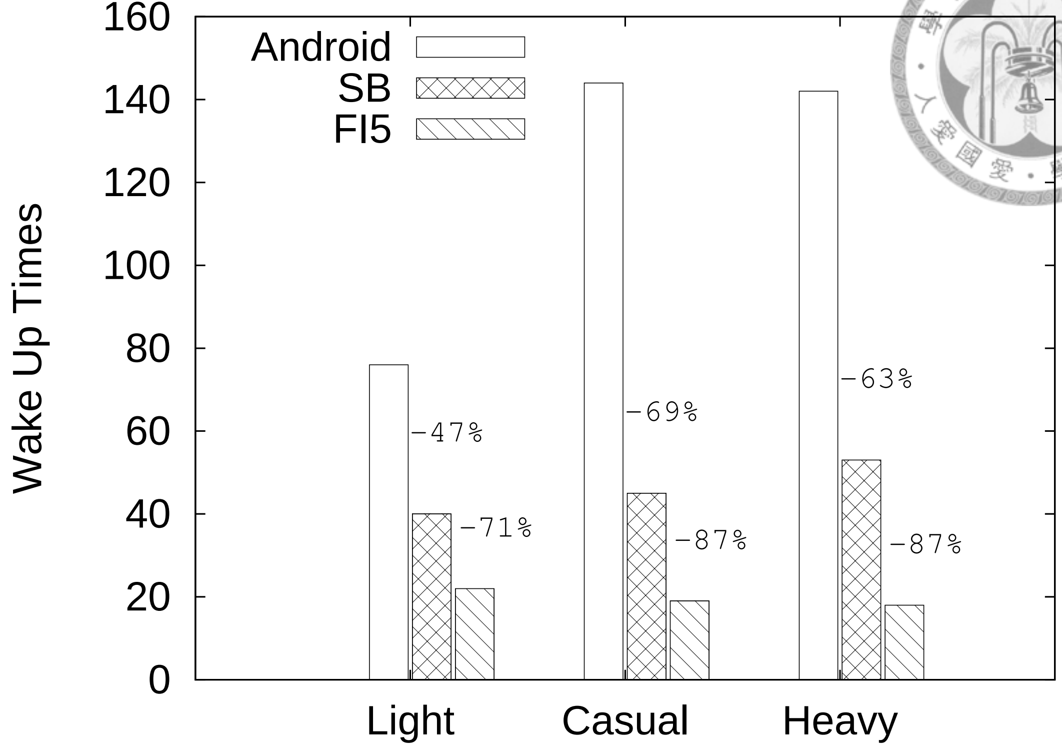


Figure 4.2: Wake Up Times under Three Connected Standby Scenarios

and 31% under the casual user scenario because the identification of the importance of events. Figure 4.2 shows the wake up times by Android, SB and FI5 under the three scenarios. The identification reduces the wake up times of Android by 47% under the light user scenario and 69% under the casual user scenario. The efficacy of SB becomes clearer, in general, when more number of hardware resources are accessed by the events. The result shows that SB reduce the energy consumption under Android by 32% under the heavy user scenario. The reason is that approximate 40% of energy consumption under heavy user scenario is depleted by GPS access and SB reduces the number of GPS access by 50% through aligning the events that are accessing GPS.

For FI5, the energy consumption reduces under Android by 40%, 31%, and 35% respectively under the light, casual, and heavy user scenarios. The reason is that FI5 forces the repeating interval under 5 minutes to adjust to 5 minutes and the number of events greatly decreases. However, in Section 4.3, FI5 provides a worse user experience due to the design. Interesting, the wake up times of SB doesn't increase as the increment of number of the events because the wake up times are controlled by the minimum repeating interval over the event set. The situation also happens in FI5 but it is because of the fixed interval of 5 minutes. The property makes the SB can provide more functionality for developer. For example, a LBA that collects the location of mobile device every 1 minute cannot execute as expected because of the fixed interval of 5 minutes but execute perfectly with little impact of time error in SB.

4.3 Application Performance

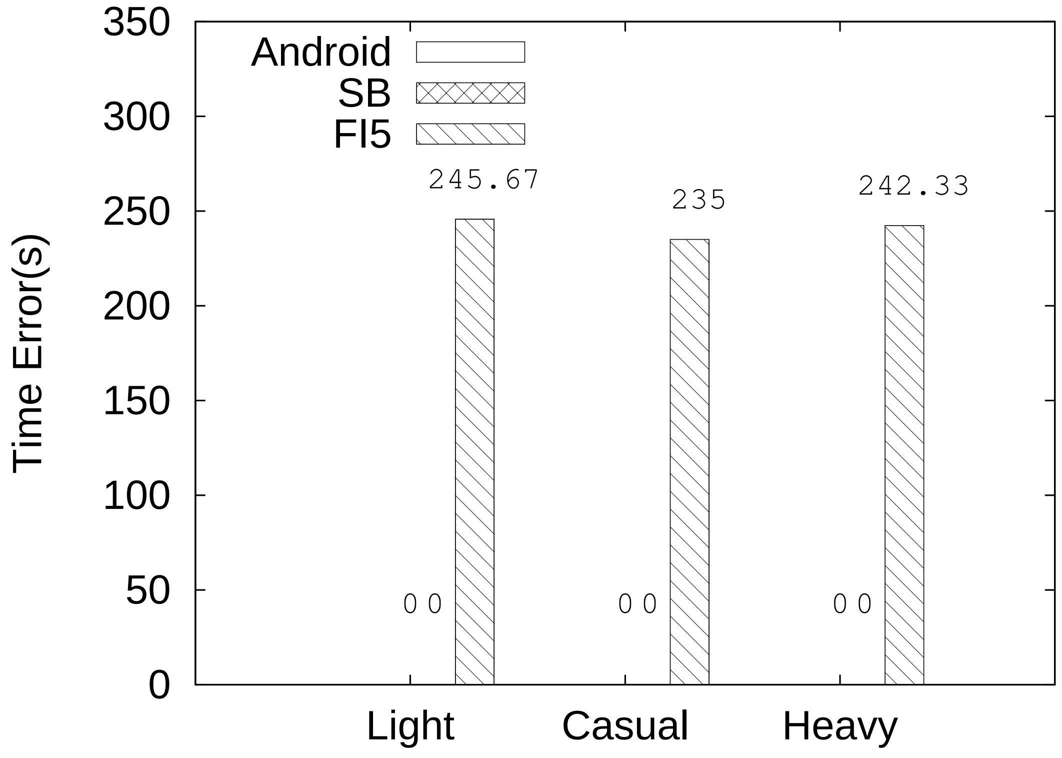


Figure 4.3: Time Error of User-Imperceivable Events under Three Connected Standby Scenarios

Figure 4.3 shows the time error of user-perceivable events by Android, SB, and FI5 under the three scenarios. SB maintains the same user experience of user-perceivable events with Android so user doesn't aware of the execution of SB. On the other hand, FI5 makes the delivery of user-perceivable events inexact and the time error could be from 0 to 5 minutes. In our experiment, because the workload is consistent so the time error keeps around 240 seconds.

Chapter 5

RELATED WORK

Application-Specific Power Management in Connected Standby: Based on the specific characteristics of mobile applications, Xu et al. [1] proposed 5 techniques to optimize the energy efficiency of background email sync on smartphones, such as reducing 3G tail time, decoupling data transmission from data processing. Ra et al. [13] explored the energy-delay trade-off in delay-tolerant, but data-intensive, smartphone applications. They formulated the problem as a link selection problem to minimize the total energy expenditure subject to keeping the average queue length finite. However, a mobile device has multiple applications with different characteristics at the same time so with a system-level power management, the battery on mobile device can be used more efficient by simultaneously manipulating the requests from different applications.

Hardware-Specific Power Management in Connected Standby: Benini et al. [6] first explored the system-level dynamic power management (DPM) for reducing power consumption in electronic systems. In order to minimize the energy consumption in connected standby, most system components stay in the sleep state. Therefore, the energy cost of accessing system components becomes tremendous due to the transition cost. For radio components, Qian et al. [14] studied the effectiveness of optimization strategies on periodic transfers to massively reduce the impact of tail energy. For WiFi components, Pyles et al. [15] present a Smart Adaptive PSM solution that prioritizes network traffic based on application priority to aggregate WiFi traffic. For GPS components, Paek et al. [16] presented RAPS, rate-adaptive positioning system for smartphone applications based on the observation that GPS is generally less accurate in urban areas. Zhuang et al. [17] proposed 4 design principles, such as use of alternative location-sensing mechanisms that consumes lower power than GPS, suppressing unnecessary GPS sensing when the user is in static state. These techniques are complimentary with our design and the concept of event similarity can be used for other DPM-driven hardware components.

System-side Power Management in Connected Standby without consideration of user: CH Lin et al. [2] and Lin, L. et al. [5] implemented an system-side exemplary remedy to reduce the energy consumption in the connected standby. The design, however, doesn't take account of user's perception. We propose that there should be a system component to schedule wakeup events in a user-centric way. The concept of similarity let two opposite goals, user experience and energy-efficiency, can be satisfied simultaneously.

Chapter 6

CONCLUSION

We have presented the first wake up event management for mobile device in connected standby. The definition of event importance shows the efficacy of differentiation between user-perceivable events and user-imperceivable events. The exploration of event similarity provides a general design guidelines for DPM-driven hardware components to satisfy the user experience and energy efficiency. Moreover, the revised event scheduler reduces the energy consumption without loss of user experience of user-perceivable events. In addition, we conducted a series of experiments on a LG Nexus 5 smartphone with some mobile applications found on Google Play. The experimental results show that average energy reduction reaches 25% and up to 32% in heavy user scenario.

Bibliography

[1] Fengyuan Xu, Yunxin Liu, Thomas Moscibroda, Ranveer Chandra, Long Jin, Yongguang Zhang, and Qun Li. Optimizing background email sync on smartphones. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 55–68, New York, NY, USA, 2013. ACM.

[2] Chi-Hsuan Lin, Yu-Ming Chang, Pi-Cheng Hsiu, and Yuan-Hao Chang. Energy stealing-an exploration into unperceived activities on mobile systems.

[3] Alarm Manager. http://developer.android.com/reference/android/app/AlarmManager.html.

[4] Android Message. http://developer.android.com/reference/android/os/Message.html.

[5] L. Lin, J. Ma, H. Yu, J. Qiu, L. Mi, S. Dave, Z. Zu, K. Samynathan, and R. Clark. Saving power in a mobile terminal, August 7 2014. US Patent App. 14/094,494.

[6] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(3):299–316, 2000.

[7] Google Fit. https://fit.google.com/.

[8] Po-Hsien Tseng, Pi-Cheng Hsiu, Chin-Chiang Pan, and Tei-Wei Kuo. User-centric energy-efficient scheduling on multi-core mobile devices. In *Proceedings of the 51st*

Annual Design Automation Conference, DAC '14, pages 85:1–85:6, New York, NY, USA, 2014. ACM.

[9] Proc Filesystem. http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html.

[10] Android Binder. http://developer.android.com/reference/android/os/Binder.html.

[11] Android WakeLock. http://developer.android.com/reference/android/os/PowerManager.WakeLock.html.

[12] Monsoon Solutions, Inc. http://www.msoon.com/.

[13] Moo-Ryong Ra, Jeongyeup Paek, Abhishek B. Sharma, Ramesh Govindan, Martin H. Krieger, and Michael J. Neely. Energy-delay tradeoffs in smartphone applications. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 255–270, New York, NY, USA, 2010. ACM.

[14] Feng Qian, Zhaoguang Wang, Yudong Gao, Junxian Huang, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. Periodic transfers in mobile applications: Network-wide origin, impact, and optimization. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 51–60, New York, NY, USA, 2012. ACM.

[15] Andrew J. Pyles, Xin Qi, Gang Zhou, Matthew Keally, and Xue Liu. Sapsm: Smart adaptive 802.11 psm for smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 11–20, New York, NY, USA, 2012. ACM.

[16] Jeongyeup Paek, Joongheon Kim, and Ramesh Govindan. Energy-efficient rate-adaptive gps-based positioning for smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 299–314, New York, NY, USA, 2010. ACM.

[17] Zhenyun Zhuang, Kyu-Han Kim, and Jatinder Pal Singh. Improving energy efficiency of location sensing on smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 315–330, New York, NY, USA, 2010. ACM.