

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

行動系統之隱性耗電活動分析

How Is the Energy Wasted -

Exploring Unperceived Activities of Mobile Systems

林季萱

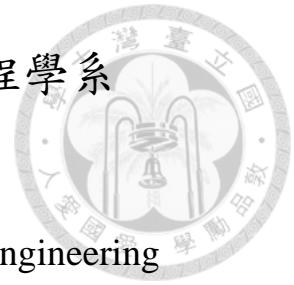
Chi-Hsuan Lin

指導教授：郭大維 博士 / 修丕承 博士

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

中華民國 103 年 6 月

June 2014







中文摘要

如智慧型手機等的行動系統已成為現代人生活中不可或缺的一部份。由於行動裝置上電池容量相當的有限，如何了解系統中的耗電分布並減少不必要的耗電一直是重要的課題。在考慮使用者觀感(User Perception)的情況下，我們發現目前的行動系統很可能存在嚴重的耗電浪費。因此我們提出了隱性活動(Unperceived Activity)的概念，並設計一連串的提驗及分析，證明其嚴重性，並觀察發生的原因和特性。根據分析的結果，我們提出並實作了可能的解決方向，並在驗證實驗中證明這些方法可以達到可觀的省電效果。

關鍵詞：行動系統、行動裝置、耗電分析、能源效率、使用者觀感、應用程式



Abstract

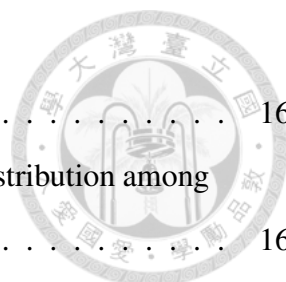
Smartphones have become an essential part of our daily life. As mobile devices and applications keep becoming more powerful and complicated, the demand for energy on mobile devices also rise continuously. Due to the limited battery capacity, power consumption analysis has been an essential study to make the usage of energy more efficient. In this work, we propose an power analysis considering user perception to investigate whether energy consumption is really worthy to users. We propose the concept of Unperceived Activity and waste of energy caused by it. A series of experiments is conducted to prove the seriousness of waste and find out the sources and characteristics of different unperceived activities. Potential directions are suggested to solve the problem. Some simple remedies are also implemented and evaluated to be able to reduce the waste effectively.

Keywords: Power consumption, User perception, Energy efficiency, Mobile applications, Mobile devices.



Contents

Abstract in Chinese	iii
Abstract	iv
Contents	v
List of Figures	viii
List of Tables	ix
1 Introduction	1
2 Background and Motivation	4
2.1 Background	4
2.2 Motivation	6
3 Impact of Unperceived Activities	8
3.1 Unperceived Activities	8
3.2 Exploration of Unperceived Activities	10
3.2.1 Observation Methodology	10
3.2.2 Results	11
3.3 Questions to Answer	14
3.3.1 What causes power impulses in idle session?	14

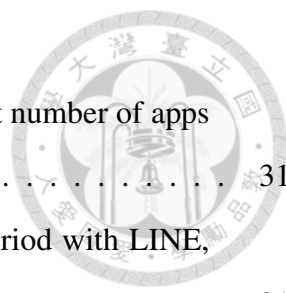


3.3.2	How long are the wake-ups?	16
3.3.3	How many wake-up events occur? How is the distribution among different causes?	16
3.3.4	How is the energy impact of different types of wake-up?	19
3.3.5	How many redundant activities are made?	21
3.3.6	What are the purposes of these redundant activities?	23
3.3.7	How is the energy impact of different redundant activities?	24
4	Insights to Potential Solutions	28
4.1	Directions to Potential Solutions	28
4.1.1	Idling	28
4.1.2	Interactive	29
4.2	Implementation: Alarm Alignment	30
4.3	Implementation: Swap Space	32
5	Conclusion	34
	Bibliography	35
	Curriculum Vitae	39



List of Figures

2.1	Android system power states	5
2.2	Energy consumption of a smartphone after lasting for 30 minutes. The case with UA happens after installing some custom apps.	7
3.1	Energy consumption with and without unperceived activity in idle session	11
3.2	Energy consumption with and without unperceived activity in interactive session	12
3.3	Sample power trace in idle session experiments	12
3.4	Sample power trace in interactive session experiments	13
3.5	Distribution of wake-up durations every 30 minutes.	16
3.6	Number of active and passive events every 30 minutes.	17
3.7	Ratio of different causes of events.	17
3.8	Ratio of energy incurred by active and passive events.	20
3.9	Ratio of energy incurred by different causes of events.	20
3.10	Amount of redundancy in storage access.	21
3.11	Amount of redundancy in communication.	22
3.12	Amount of redundancy in computation.	22
3.13	Energy consumption with different amount of redundant activities.	25
3.14	Content size and computation workload of the websites for real case study.	26
3.15	Loading energy of the websites.	26



4.1	Idling energy consumption per 30 minutes with different number of apps before and after applying alignment.	31
4.2	Idling energy consumption using different alignment period with LINE, Chrome, Facebook installed.)	31
4.3	Energy saving of paging-out(swapping space)	33



List of Tables

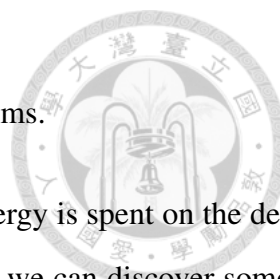
3.1 Types of computation redundancies observed from kmsg and syslog . . . 24



Chapter 1

Introduction

Mobile devices have become an essential part of our daily life, and our need for them keeps expanding. The shipment of mobile device is predicted to reach 1.8 billion by 2018[13], as the number of applications downloaded from Google Play app store have exceeded 60 billions at the end of 2013 [6]. More and more people rely on smartphones to complete various jobs from daily routines like listening musics, receiving emails, and brows the Internet to specific utilities like real-time navigation, word recognition/translation, and health monitoring. To fulfill the needs, diverse hardware components and applications have been developed and applied to mobile devices. The ability and functionality of hardwares keep evolving, and the behavior of applications also becomes more complex to satisfy users demand on performance and convenience. Under this trend, the thirst for energy on mobile devices has grown more and more serious. However, the growth of battery capacity on smartphones is relatively much slower than the demand of energy. From 2000 till now, the factor of increase in battery capacity is no more than 3, while the processing power and screen resolution have increased 12 and 13 times respectively [4]. Under this trends ,how to understand the distribution of energy consumption and reduce

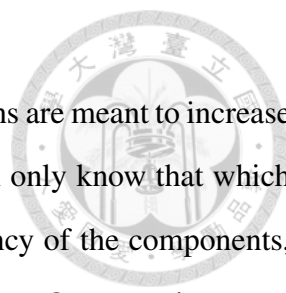


the unnecessary parts have become a crucial issue on mobile systems.

To solve the problem, we have to understand how the energy is spent on the devices. Through analyzing the distribution of power consumption, we can discover some hot-spots of energy drain, find out the causes, and design the corresponding solutions. Various studies in power analysis on smartphones have been proposed from different aspects. Many studies focus on the characteristics of power consumption among different hardware components through direct measurements or estimation to build their power models [18] [9] [27] [22]. Differences among some type of hardware are also analyzed to choose proper components for power saving. For example, since the usage of network resources is prevailing, many studies focus on comparing the energy efficiency between Wi-Fi and 3G[8] [20]. As the number and variety of applications increase, researchers start to analysis the resource usage among different applications or functionalities [23] [25]. Profiling techniques are developed to understand the distribution of energy consumption within a single app [26] [19]. Some are also exploring how user behavior influence the usage of app and hardware resource under various scenarios [22] [23].

Based on these efforts, many power saving strategies have been developed. After recognizing CPU to be a major energy drain, Dynamic Voltage and Frequency Scaling(DVFS) and Dynamic Power Management(DMP) are applied to scale the frequency of cores or shutdown idling cores based on various policies to prevent redundant CPU energy consumption [12][10][11][15]. As backlight is also a main source of drain, many studies suggest dynamic back light scaling and switching policies to dim or shutdown the back light without influence user experience [14][16][11]. For other hardware associated with data transmission, the strategies are mainly to manage the resource usage through traffic scheduling or transmission power adapting [21][17][28].

The analysis studies above only focus on how much energy are consumed on



which hardware or software components, and the proposed solutions are meant to increase the energy efficiency of the energy draining components. We can only know that which parts are the major sources and try to improve the energy efficiency of the components, while the improvements are usually limited and difficult to achieve. One question more important but yet to answer is that **how much energy is consumed necessarily? For various reasons and purposes the power is consumed for, how much of them are worthy?** If we can find out the unnecessary ones and identify their sources, we can simple remove the source to save more power. To achieve the goal, we propose the concept of *Unperceived Activity* in terms of user perception. These activities cannot improve user experience directly, and the energy consumption it cost is then considered to be wasted. Through a series of experiments, we quantify the unnecessary(wasted) energy caused by them and find out their sources and purposes. Based on our observations, potential directions to ease the waste are also suggested and evaluated.

The rest of this paper is organized as follows: Section 2 presents the background knowledge about mobile system design, user behavior, and our motivation. In Section 3 we define the concept of **unperceived activity** and reveal the waste they cause. Possible directions are proposed to eliminate the waste, and their effectiveness is shown through evaluation experiments. Section 5 will introduce the related work of power analysis on mobile devices. Section 6 is the conclusion.



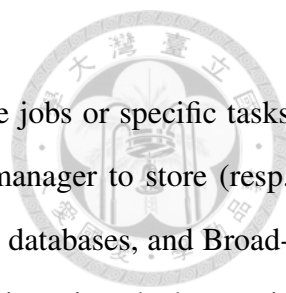
Chapter 2

Background and Motivation

2.1 Background

Reducing energy consumption has been a challenging and essential issue for mobile devices due to its limited battery capacity to sustain daily usage. However, the ever-rich functions of hardware as well as the rapidly-growing number of software (applications) complicate the system behavior and hinder the analysis from easily breaking down the power usage on mobile phones. In this work, we take a well-known mobile platform, e.g., Android system, as a example to study the power issues and explore those power-hungry activities in different points of view.

An Android operating system is actually a Linux-based system, where each application is an unique process that has its own Dalvik Virtual Machine (DVM) to execute application's code in an isolation environment. An typical android application could be constructed out of these four components, *Activity*, *Service*, *Content Provider* and *Broadcast Receiver*. *Activity* is used to provide graphic user interface (GUI) to interact with



users, whereas Service runs in the background to perform routine jobs or specific tasks without interacting with user. Content Provider acts as a data manager to store (resp. load) shared data in (resp. from) different kinds of file systems or databases, and Broadcast Receiver is responsible to respond to the system-wide notifications, i.e., the battery is low. Therefore, various types of applications could be implemented in different way. For example, the gaming applications should adopt Activity to interact with users, the music-playing applications could use Service to play music in background, the news-related applications could utilize Content Provider to organize the daily news, and a downloader application could exploit Broadcast Receivers to notify the users of when a file is successfully downloaded. As any running applications might be suspended to background whenever they are closed or other applications are switched to foreground, some system services, such as Google Cloud Messaging (GCM) manager or AlarmManager, are provided to let the applications register their tasks to execute when needed. Thus, once the system is interrupted by the system events, i.e., timer timeout or the arrival of network packet, Android will send an *INTEND* message to activate the applications that register the corresponding task to handle the events in spite of that the system is currently put into sleep mode or other applications are running.

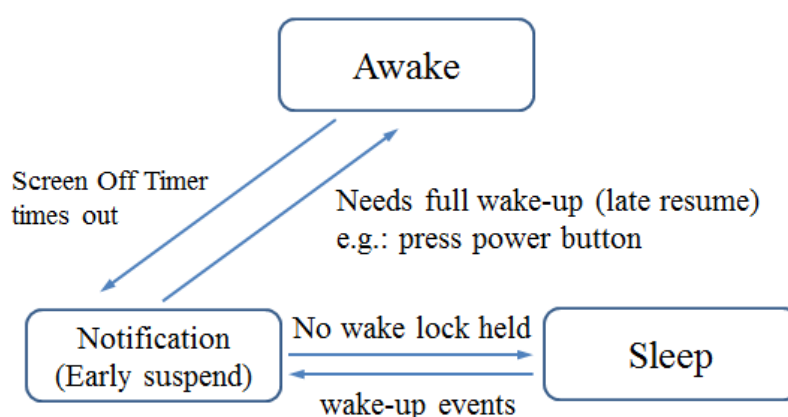
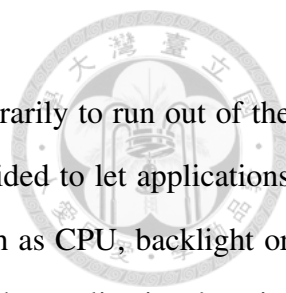


Figure 2.1: Android system power states



In order to prevent the system from being woke up arbitrarily to run out of the battery, an straightforward mechanism, called *wake lock*, is provided to let applications themselves take charge of acquiring the hardware resources, such as CPU, backlight or network, when system is woke up from sleep mode to perform the applications' registered tasks. In general, the android system would switch among three states, e.g., awake, notification and sleep, depended on which type of wake lock is acquired currently as depicted in Figure 2.1. When a *full wake lock* [3] is acquired, the hardware resources, i.e., CPU and backlight, could be utilized by application's own choice, and the system will enter awake state. If the system idles for a certain period of time, the state will transfer to notification state such that those resources that are not locked currently by the wake locks will be turned off at once. Finally, the system will go into sleep state if no wake lock is held for a while. Hence, any misuse of wake lock might lead to long stay in awake or notification state, where the power consumption of these two states is far larger than that of idle state. For example, in Galaxy S3, the power consumption of awake state is about 800–1000 mW, whereas the power consumption of idle state is only 17–30 mW on average.

2.2 Motivation

Even though many excellent works that are proposed for power management, i.e., wake lock on Android, and power analysis have been extensively studied, little work is done to explore or reduce those power-hungry activities which are not directly perceived or needed by users. In particular, such activities might occur more frequently and hard to be identified due to more complicated hardware, software along with unpredictable user behavior. A simple experiment under two scenarios was conducted to observe the existence of energy consumption hidden behind user attention. The first scenario was evaluated

based on a smart phone without installing any application except for the default ones, whereas the phone in the second scenario additionally installed two popular applications, namely GMAIL (email application) and Line (instant messaging application), and connected to a Wi-Fi AP with broadcast packets sent every minute. Both scenarios last for 30 minutes with the screen off, and their energy consumption were reported as shown in Figure 2.2. Even if we already make sure that no meaningful activities, i.e., email synchronization for GMAIL or new message from Line, could be captured during the 30 minutes, the significant difference (70J and 207J) of energy consumption could be observed from the results.

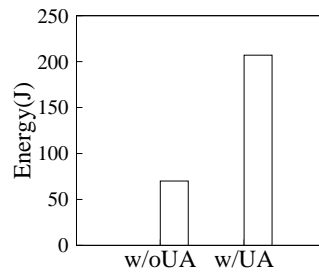


Figure 2.2: Energy consumption of a smartphone after lasting for 30 minutes. The case with UA happens after installing some custom apps.

As a result, the above observations motives us to seek to understand 1) how much energy is actually needed by users, instead of that wasted on each components 2) what unperceived activities drain the energy behind the scenes. Very different from the past works on analyzing the power consumption of either the CPU for software execution or other hardware components such as backlight, we emphasize on the finding of those power-hungry activities that is not directly contributed to users. In this work, a series of experiments was conduct to simultaneously capture the log of the real activities and the instant power on a mobile phone. By means of jointly investigating both traces, more profound and insightful finding could be explored and helpful for system designer to further improve the energy efficiency.



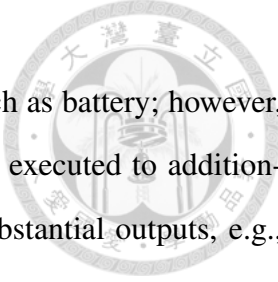
Chapter 3

Impact of Unperceived Activities

3.1 Unperceived Activities

To systematically inspect the impact of *unperceived activities* on energy consumption, the system activities were studied in terms of two scenarios: idle and interactive session. The idle session is defined as the period that users do not pay attention to their mobile devices currently, where the devices will be in sleep mode for most of the time. On the other hand, the interactive session comes about when users are interacting with mobile phones by touch screen or other input devices, where more hardware resources might be turned on in this session. It should be noticed that the unperceived activities in both two sessions will show up in different way.

For the idle session, the unperceived activities are the processes (applications) that work in background without using or triggering any user-perceived hardware such as screen or speaker to draw user attention. For example, some applications might keep in background to periodically perform synchronization with remote server, constantly



backup the system log or repeatedly check the hardware status such as battery; however, most of the time, these applications might be overly and actively executed to additionally create more energy consumption and do not generate any substantial outputs, e.g., incoming message notification, which could be perceived by users.

Even when users are using smartphones, energy cost is not always worthy as well. Redundant activities may occur and cost extra energy. For example, when using a web album application, the app may download the pictures first, decode and layout the pictures, and then draw the UI components to show the content. However, due to some errors like inconsistency of state, the action of downloading, layout, and drawing may be done all over again. Since the final contents are the same no matter whether redundant activities happened or not, users cannot perceive the redundancy directly. Therefore, we define the unperceived activities as redundant activities in interactive session.

In the following sections, we conduct a series of experiments to observe the power consumption in the scenarios of power wastes, reveal the seriousness, and analyze the causes of them to discover possible strategies to eliminate these wastes. For idle sessions, we discover that smartphones are waked up through interrupts passively or actively, which contribute a majority of energy consumption when idling. We also analyze the distribution and purposes of different interrupts, and how they consume energy. As for interactive sessions, we show that there are lots of redundant works due to the unexpected rebuild of apps by reproducing the app switching processes. We then further categorize the redundancies, make a breakdown of energy costs, and point out the main cause of these redundancies.

3.2 Exploration of Unperceived Activities



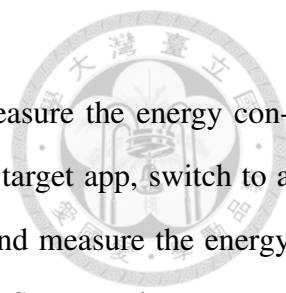
3.2.1 Observation Methodology

Environment and tools

All of the experiments are conducted on Samsung Galaxy S3 I9300 model, with 1.4 GHz Cortex-A9 quad-core CPU, 1G DRAM memory, 32GB internal storage. It supports Wi-fi, 3G for network transmission. The operating system is Android 4.1.2 on Linux 3.1, and all the modifications are based on them. Instant power and accumulated energy consumption of the smartphone is measured through Monsoon Power Monitor [5]. The samples are tagged with timestamps and plotted on the graph. The experiments are conducted under Wi-Fi or 3G network connections to reflect the influence of network usage. To study the impact of app behavior, we chose some popular and representative apps of different types and applied to our experiments. It is reported that text messaging, social network, and web browsing are the top 3 frequently used types of applications [1]. Based on the report, we chose LINE, Facebook, and Chrome respectively to represent the typical behavior of the 3 apps types.

Steps

For each session, we design a scenario respectively to measure and compare the energy consumption with and without unperceived activity. In idle session, all custom apps are removed except one single target app. Before start measuring, we start the app and press HOME button to preempt the target app to background and then turn off the screen. After making sure it has entered the sleep mode, we start the power measuring for 30 minutes to get the power trace and total energy consumption. During the processes of experiments, we also make sure there is no interference that makes screen on. As for



interactive session, we operate a series of app switching and measure the energy consumption during the interaction with each target app. We start a target app, switch to a predefined series of apps, switch back to the target app at last, and measure the energy consumption. Besides the target apps, Gmail, Google Map, and Samsung browser are used in the scenario.

To observe the activity of Linux kernel and Android framework, we capture 2 existing logs: **kmsg** and **syslog** in Linux and Android respectively. Messages from device drivers can be captured through kmsg, while the activities of Android system and applications can be observed in syslog. We use *tcpdump* to monitor Wi-fi or 3G network traffic in idling session. WireShark [7] is utilized to visualize the time line of network traffic. By tracing back the source of packets using IP address, we can also identify which app a packet from. We use *Traceroute* to see the name space of each IP and identify whether it belongs to a server of target apps. For file access activities, we use Linux *inotify* API to monitor read/write requests to their *file* and *cache* folders of each target app.

3.2.2 Results

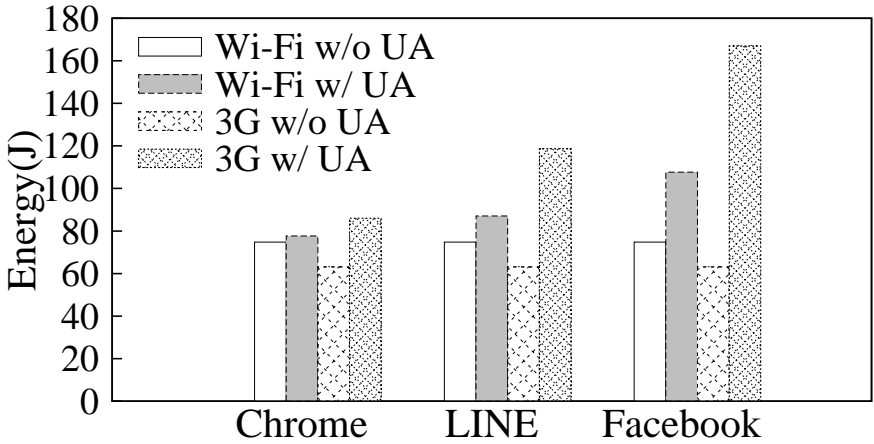


Figure 3.1: Energy consumption with and without unperceived activity in idle session

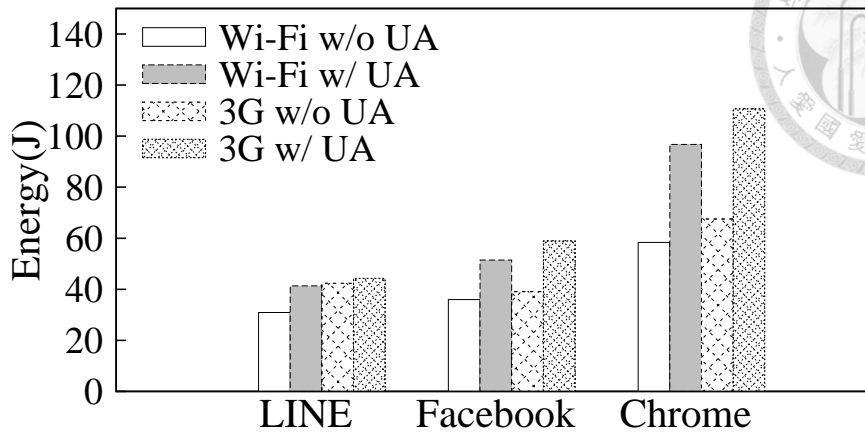


Figure 3.2: Energy consumption with and without unperceived activity in interactive session

Figure 3.1 and figure 3.2 show the total energy consumption with and without unperceived activity of each test case in idle and interactive sessions. As shown in the graphs, the waste of energy do exists in all test cases in varying degrees. Among the cases in idle session, the maximum amount of energy increase is up to 136% if unperceived activity occurs when using 3G and Facebook. It can also reach more than 60% in interactive session in the case of Chrome under Wi-Fi connection. We can also observed that the amount of increase is higher under 3G connection, and the seriousness of energy waste varies among the three apps. Even the same app in different sessions introduces different amount of waste. For instance, LINE and Facebook introduce more waste than Chrome in idle session, while Chrome and Facebook cause more in interactive session.

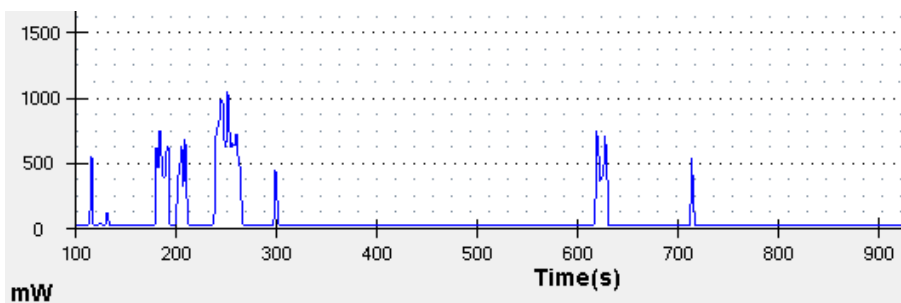
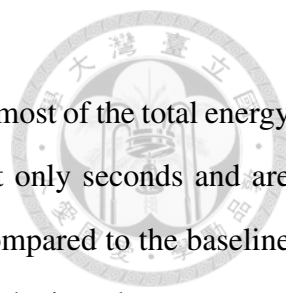


Figure 3.3: Sample power trace in idle session experiments



In idle session, we found that **power impulses** contribute most of the total energy consumption as shown in figure 3.3. Although the impulses last only seconds and are usually sparse, the average power level of them is much higher compared to the baseline power. While the baseline power remains not higher than 40 mW, the impulse power can reach up to 1000 mW. The shape and size of impulses also diverse. Some impulses are longer and higher than others; impulses in 3G environment may have some specific shape, which is due to the power state transitions of 3G RRC.

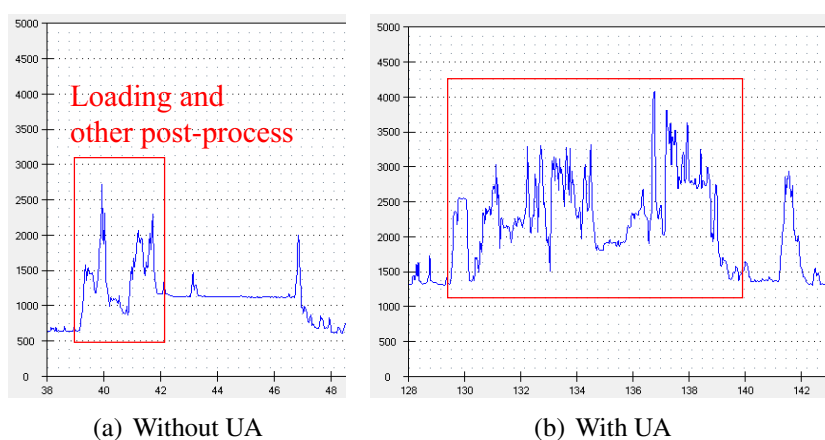
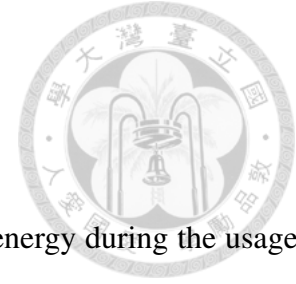


Figure 3.4: Sample power trace in interactive session experiments

In interactive session, as shown in figure 3.4, we found that the power usually remains high when unperceived activity happens. The power will initially remains at baseline, and rises sharply when an app is about to switch back to foreground, causing a series of impulses. Even after completely presenting the content, the power will still goes high for a period of time, causing up to half of the loading energy. For example, after Facebook showed the post list and the first power impulse finished, there was another impulse persisted for about 2 to 4 seconds.



3.3 Questions to Answer

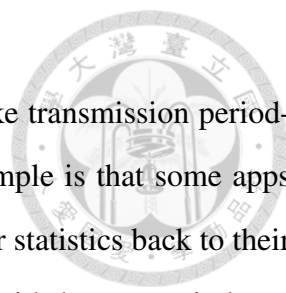
We have shown that unperceived activities could take up lots of energy during the usage process of smartphones. To reduce the waste, we have to understand their purpose and analysis their features to design effective strategies to solve the problem.

3.3.1 What causes power impulses in idle session?

We found that the power impulses are mainly caused by wake-up interrupts. It's reasonable since after the smartphone enters the sleep mode, CPU itself will also be suspended, and then only interrupts can wake the smartphone up. In our trace, we observed 2 types of wake-up interrupts: clock interrupts caused by Android AlarmManager, I/O interrupts from wireless modules and modems. Smartphone apps and Android system will utilize AlarmManager to register timers in order to wake the smartphone up and operate the planned works in some future time. These interrupts are thought to be *Active* since they are requested and caused by apps of system themselves. On the other hand, wake-ups due to the interrupts from network components are caused external events like an arrival of packet. Therefore, they are referred as *Passive* since they are not caused by system or apps them selves. Some common sources of these events are listed below:

Active event:

Some features of apps like instant messaging, data synchronization, and system update need the latest information from servers in time. They usually rely on polling or push notification to know when to fetch the data from servers. Common implementations of push technology usually need to exchange messages with the servers occasionally or periodically in order to maintain the network connections with them. We observed that



Facebook and LINE maintain multiple TCP connections and make transmission periodically, which is a typical behavior of push service. Another example is that some apps will dump user usage patterns, app performance records, and other statistics back to their servers for analysis. The data are often batched and sent back with longer period and larger data size. Facebook will send data back to their server every hour by a background service *AnalyticsEventUploader*, causing bursts up to 6 seconds.

Passive event:

Receiving response packets is one of the main source of passive events. Some are the pushed information from the servers, while some are the delayed responses of earlier requests. After sending the request in the first wake-up, the smartphone may be suspended again due to the expiration of wakelocks [?] before the delayed packets come back. When the response arrived, the smartphone will be waked up again to receive the data, causing passive interrupts.

Besides, other devices in WLAN will perform network probing through broadcast or multicast to maintain certain network information. For example, in wireless network environment, Address Resolution Protocol(ARP) will broadcast to ask the MAC address of devices in the network. We also have observed Internet Control Message Protocol(ICMP), Internet Group Management Protocol(IGMP), and many other network protocol activities that have similar behaviors.

Note that not only the target apps or works that directly associated with interrupts cause extra energy consumption. Once the smartphone is waked up, other suspended jobs will also be triggered and increase both workload and energy consumption before going back to sleep again. For example, we observed that BatteryService will start update the display of battery status everytime the smartphone wakes up, even though users cannot



perceive these updates since the screen is still off.

3.3.2 How long are the wake-ups?

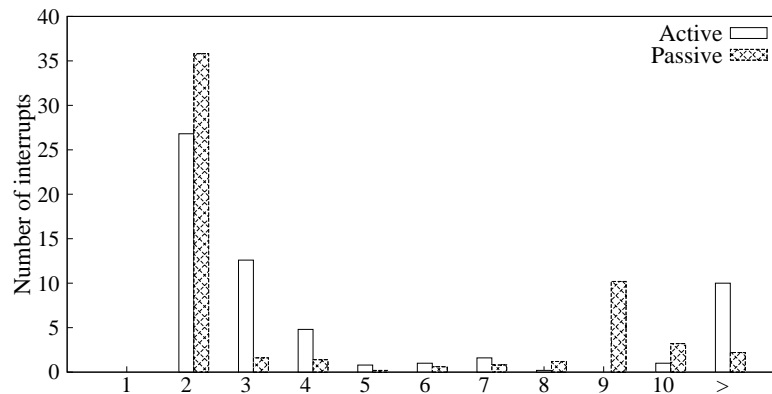


Figure 3.5: Distribution of wake-up durations every 30 minutes.

Figure 3.5 shows the distribution of wake-up durations induced by active or passive events. Most of the events are no longer than 3 seconds, causing short power impulses as we observed in the former section. Some of the passive events last longer than 8 seconds, because RRC inactivity timer may keep smartphones active for a period of time after data transmission using 3G connection. However, most of them from modem are still short because no actual transmission occurs. Despite the short wake-up durations, these interrupts can still induce high power impulses.

3.3.3 How many wake-up events occur? How is the distribution among different causes?

Figure 3.6 shows the number of different wake-up events with each target apps. Basically, more events occur in the cases of LINE and Facebook. Beside, the number of passive events under each target app is much more in 3G than in Wi-Fi and takes up about 50%

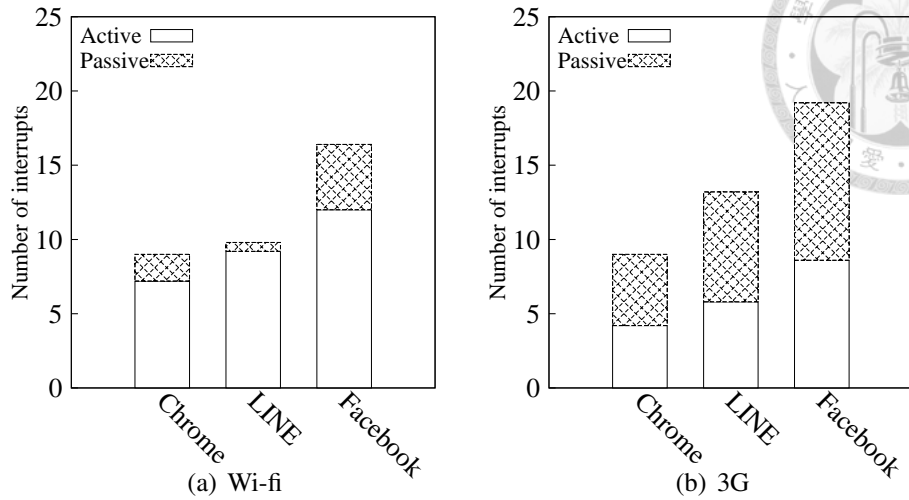


Figure 3.6: Number of active and passive events every 30 minutes.

to 60% of the total events. On the contrary, active events are more frequent in Wi-Fi and take up 70% to 95% of the events.

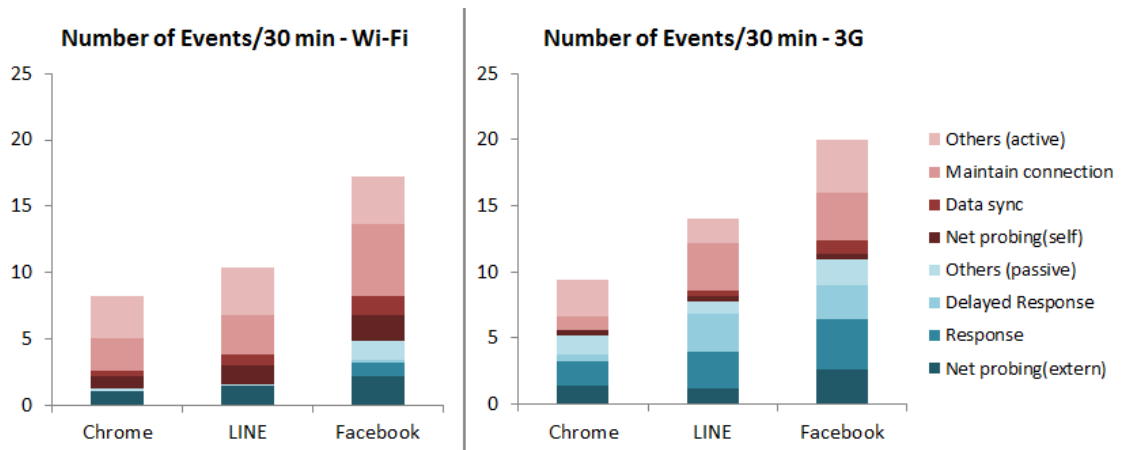
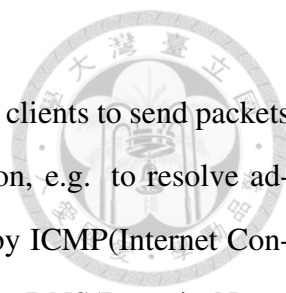
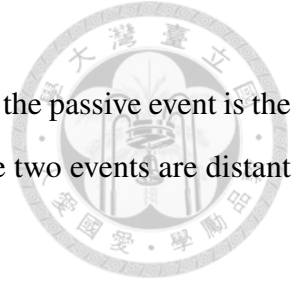


Figure 3.7: Ratio of different causes of events.

We then further breakdown active and passive events based on their causes as shown in figure 3.7. Through mapping packet bursts to the wake-up events using timestamps, we can know that if an event triggers a series of packet transmissions, say traffic bursts. Through examining the source IP of the first packet in every traffic burst, we can determine if the burst is initiated by remote servers or the smartphone itself. The other details about how we identify the causes are listed below:

- 
- **Active - Network Probing** Some network protocols require clients to send packets actively in order to obtain network environment information, e.g. to resolve addresses. Here we filter out the events with bursts initiated by ICMP(Internet Control Message Protocol), ARP(Address Resolution Protocol), DNS(Domain Name Service), and other similar protocols as network probing events.
 - **Active - Connection maintenance** Usually, the processes of maintaining connection are simply sending acknowledgment messages between clients and servers. Therefore, the number of packets in a burst is usually small. We take the events with burst that is 1) in TCP, SSL, or other stateful connections 2) smaller than 10 packets (but not network probing) as this category.
 - **Active - Data Synchronization** Unlike maintaining connection, the size of data is usually larger that requires more packets to carry, causing larger traffic bursts. In our cases, filtering the events with burst larger than 10 packets can properly identify most of the data synchronizations.
 - **Active - Others** They are active events are not mapped to traffic bursts.
 - **Passive - Network Probing** Similar to active network probing events, they are caused by network protocols that aim to probe network environment.
 - **Passive - Response** As mentioned, if there exists connections between clients and servers, servers can actively push data back to the clients. We simply take the events with traffic bursts that sent from remote servers with stateful protocols as response events.
 - **Passive - Delayed Response** If a passive event whose traffic burst is the response of the former active event, e.g. ack from server for maintaining maintenance, it is viewed as delayed response. To be more specific, if the source IP of the active event



is the destination IP of the passive event and vice versa, then the passive event is the delayed response of the former active event. However, if the two events are distant in time, they are not considered to have any relationship.

- **Passive - Others** Active events are not mapped to traffic bursts. This might be due to some control signals sent by modems to communicate with the system.

Under Wi-Fi environment, most of the active events belong to *connection maintenance*, which take up about 30% of the total events. They become more frequent after installing Facebook. The number of *others* active events is also about 20%-40% of the total events, which means that there are still lots of events caused by apps for unknown purposes. In 3G, passive events of *response* and *delayed response* increase obviously compared with Wi-Fi in all cases. They take about 25%-40% of the total events in 3G, while they almost do not exist in Wi-Fi. Compared with Chrome, the amount of increase is higher in LINE and Facebook because they tend to have more network transmission actions in nature, and thus more packets will be possibly delayed. The numbers of active events in 3G decrease slightly, but their distribution is almost the same. This may be due to that apps will adjust their strategy of network usage. For example, the period of polling for connection maintenance may be shorter to prevent losing connections and provide better services.

3.3.4 How is the energy impact of different types of wake-up?

Figure 3.8 shows the energy breakdown among different types of events. Baseline denotes the baseline energy to support the smartphone, which is consumed by neither passive nor active events. The energy cost by these wake-up events is about 20%-33% in the cases of Wi-Fi, and it can even reach up to 58%-68% in 3G. The total energy consumption is much

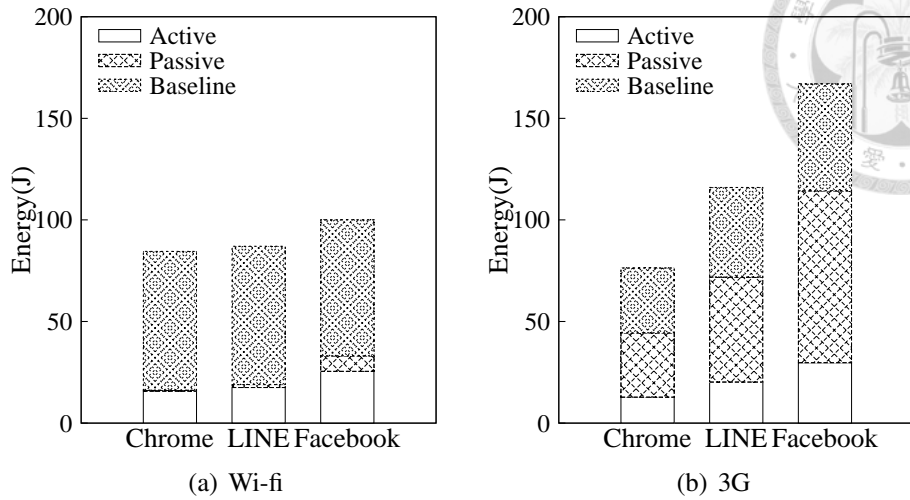


Figure 3.8: Ratio of energy incurred by active and passive events.

higher in 3G since it consumes more power in nature when transmitting packets. This implies that the impact of these UA is more serious under 3G environment. Consistent with the distribution of events, active events induce most of the energy consumption (70%-87% among the energy cost by events) under Wi-fi, while passive events contributes the most (74%-76%) under 3G environment.

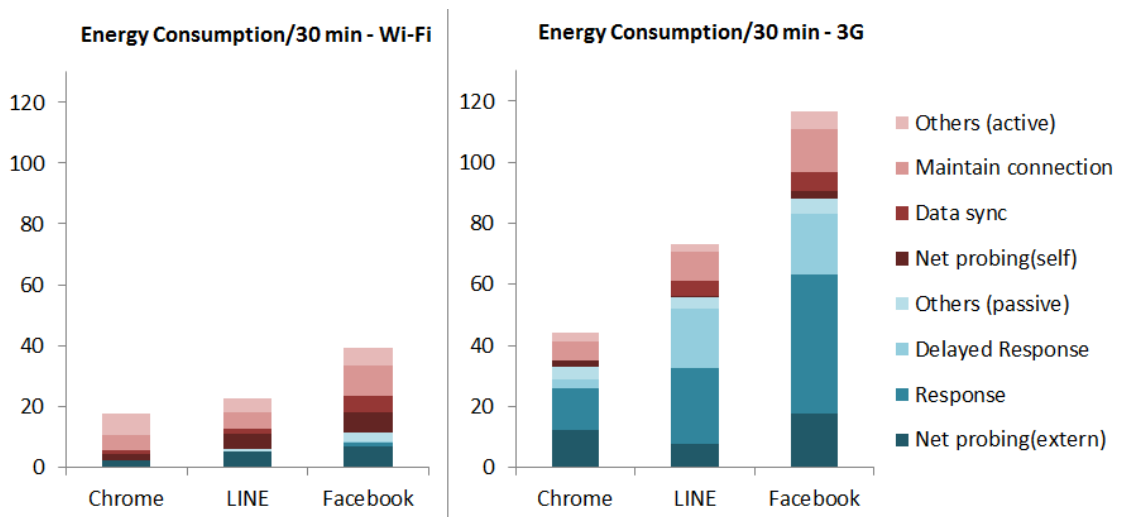
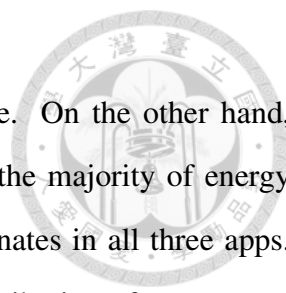


Figure 3.9: Ratio of energy incurred by different causes of events.

Figure 3.9 shows the detail distribution among different causes of events. It is also similar to the distribution of the occurrence of events in each case. In Wi-Fi en-



vironment, *others* dominates the energy consumption in Chrome. On the other hand, *connection maintenance*, *network probing*, and *others* consume the majority of energy consumption in LINE and Facebook. As for 3G, *response* dominates in all three apps. *Delayed response* takes more in LINE and Facebook like the distribution of event numbers. This means that **the frequency of interrupt may be the main factor the power waste** during the idle session.

3.3.5 How many redundant activities are made?

To evaluate the amount of redundant activities, we choose the number of file accesses, number of network packets transmitted, and CPU utilization as general matrices. File accesses include read and write actions, and packets sent and received are counted altogether. CPU cycles are normalized to the highest CPU frequency. We think they are fairly enough to reflect the increase of both computation-intensive and data-intensive activities.

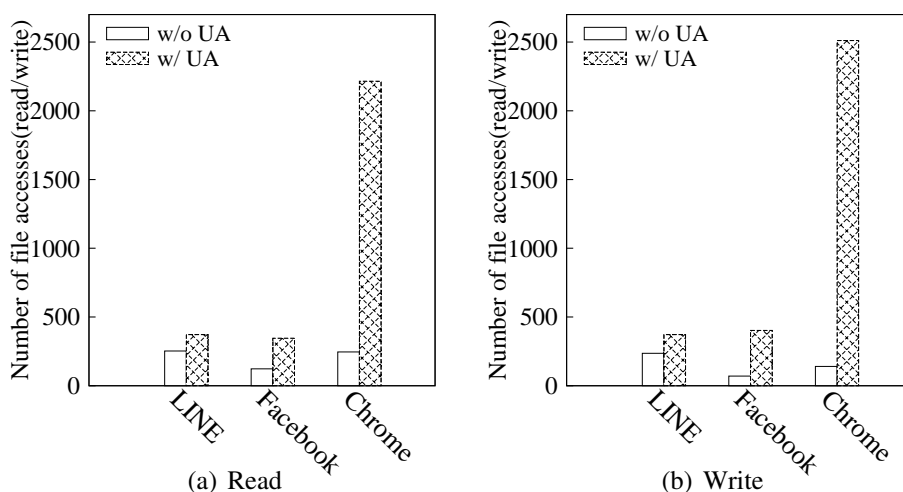


Figure 3.10: Amount of redundancy in storage access.

Figure 3.12, 3.10 and 3.11 show the increase ratio of file access, network packets, and CPU cycles required to load target apps with and without redundant activities. When

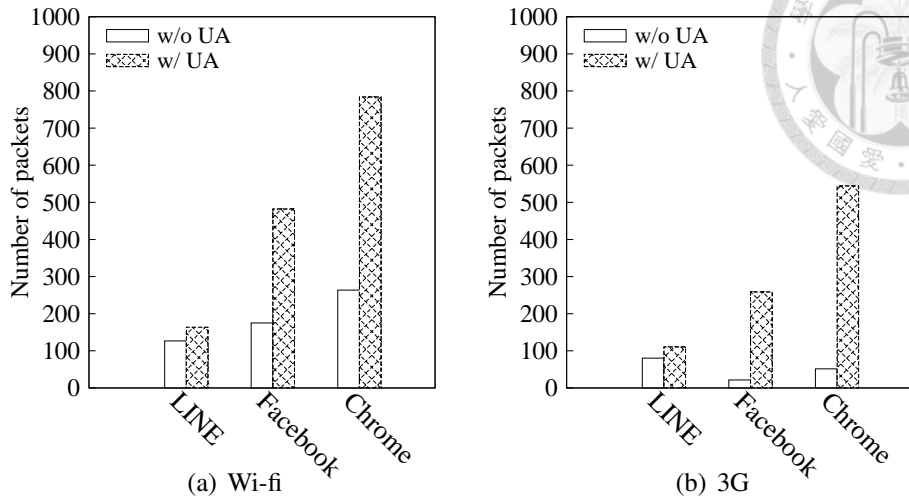


Figure 3.11: Amount of redundancy in communication.

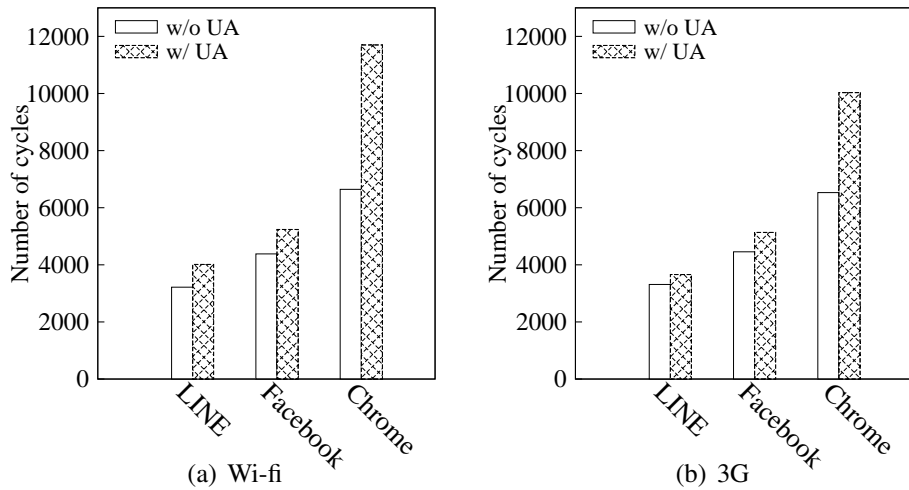


Figure 3.12: Amount of redundancy in computation.

there are redundant activities, all the three indexes increase with all target apps. The diversity of apps again makes the ratio of increase differs among target apps. For Chrome, the number of file accesses even expands up to 200 times higher under the waste scenario, and the packets transmission also expands 10 to 100 times higher. This is because Chrome will reload data from local cache or even from the servers. Facebook also has similar behaviors that will cause extra file accesses and network usage. CPU utilization also rises in all target apps. The growth rate of Chrome is higher since the rendering of websites requires lots of CPU workload.



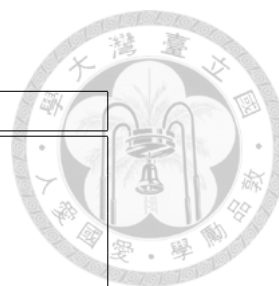
3.3.6 What are the purposes of these redundant activities?

To analyze the cause of the wastes during app reloading, we group the kmsg and syslog traces based on their scenario and use CHI square to select feature events of normal and waste scenario respectively.

Table 3.1 shows some typical feature events of the waste scenario. We found that most of the feature events are for building user interfaces, loading libraries, system services registration, and other works related to application initializations. Various customized tasks are also observed in different apps. For example, repeated rendering of web pages also exists in our logs. Garbage collection, Android memory trimming events, and process killing by Low-memory Killer also becomes more frequent.

Based on these observations, we suggest that these redundancies may be caused by aggressive memory recycling. For example, rebuilding of processes and virtual machines happens when a app which has been killed by the Low Memory Killer is opened again. It is usually followed by the redrawing of GUI and request of system services. Even not killed by LMK, some apps will still release some GUI components or data structures themselves to save heap spaces, causing rebuilding of GUI and re-computation of data. For instance, Chrome will release opened tabs(pages) when smartphones are near low-memory, and the web pages will need to be loaded and rendered again when opened.

The increase of network transmission and file access also supports our suggestion. Many apps, like browser or map, need to download contents from the Internet and therefore use network interfaces intensively. If those contents cannot be preserved in memory or cached in local storage, they will need to be re-downloaded from the Internet, causing extra network transmissions. Multimedia contents with larger size like images or videos make downloading even more expensive since the transmission time will become



Category	Item
GUI drawing	OpenGLRenderer WindowManager AbsListView TextLayoutCache libEGL, ManagedEGLContext Choreographer
Service Requesting	MotionRecognitionService LocationManagerService MediaPlayerService, MediaPlayer MediaPlayer-JNI
Process initialization	Bundle MtpDeviceJNI dalvikvm

Table 3.1: Types of computation redundancies observed from kmsg and syslog

longer.

Further more, apps may store static data like user accounts, images, history, and other statistics in the local storage and load them into memory as needed. For example, Facebook and LINE will cache the thumbnail of user profiles or other pictures as local files, and user accounts are also saved in local databases. If these contents are recycled, all the data must be reloaded from the files.

3.3.7 How is the energy impact of different redundant activities?

Unlike idle session, it's very difficult to break down the energy consumption to different redundant activities since they could operate asynchronously. For instance, smartphones can be playing a video while the later part of the video is still being downloaded. Therefore, we made a real case study on Chrome by utilizing a customized website to control the amount of redundant activities of computation and data acquisition. We utilized the amount of Javascript loops to simulate different computation workload of redundant ac-



tivities. As for data acquisition, we used the number of pictures to control the content size. We then went through the former experiment scenario for interactive session to compare the energy consumption with and without redundancy.

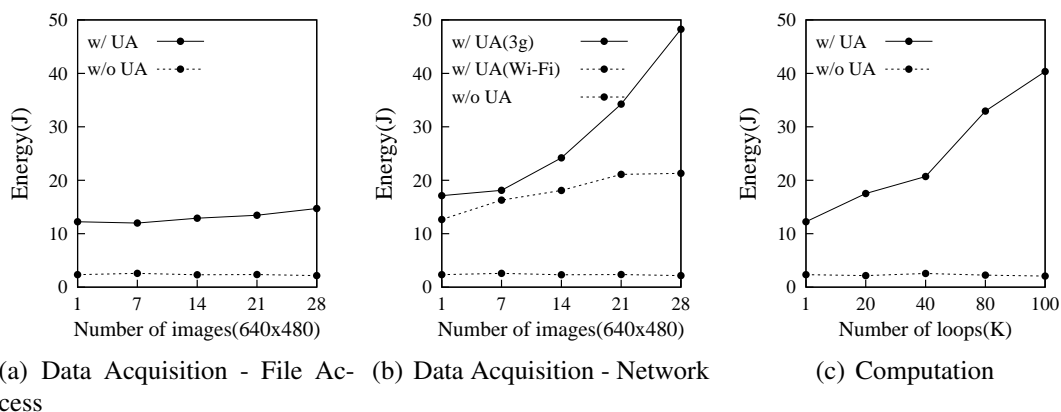
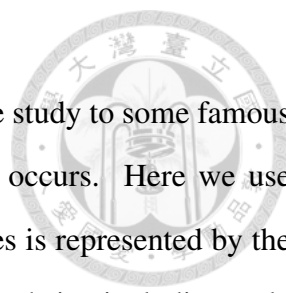


Figure 3.13: Energy consumption with different amount of redundant activities.

From figure 3.13, we could see that the amount of computation redundancy is proportional to the amount of waste. Data acquisition also increases the amount of waste. However, data acquisition activities through storage accesses have less influence than through network. For data acquisition activities through network transmission, it causes more waste through 3G than through Wi-Fi.

This can also help us explain the variances of the waste induced by Facebook and Chrome in the former experiment. The fast changing content in Facebook and the diversity of multimedia files(like thumbnail, film, etc.) make the energy vary along test cases. As for Chrome, because the page could be either loaded from cache or re-downloaded from network, whether a page will be cached depends on the current app status, causing consumption distribution varies among tests. Larger variance is also caused by the memory management implementation within Chrome itself. Chrome will recover some of the preserved pages under different degrees of low-memory situation to save memory and prevent itself from being killed by LMK. Despite not being killed, the pages will be abandoned by Chrome itself, making extra consumptions.



To support the former observation, we also apply the case study to some famous existing websites to observe the energy consumption when UA occurs. Here we use Wi-Fi as network interface. Computation workload of the websites is represented by the Javascript execution time, and content size means the total size of website, including style sheets and other multimedia components. Data size of the websites are calculated through accumulating all data received throughout loading process. These statistics are measured through Chrome Development Tool.

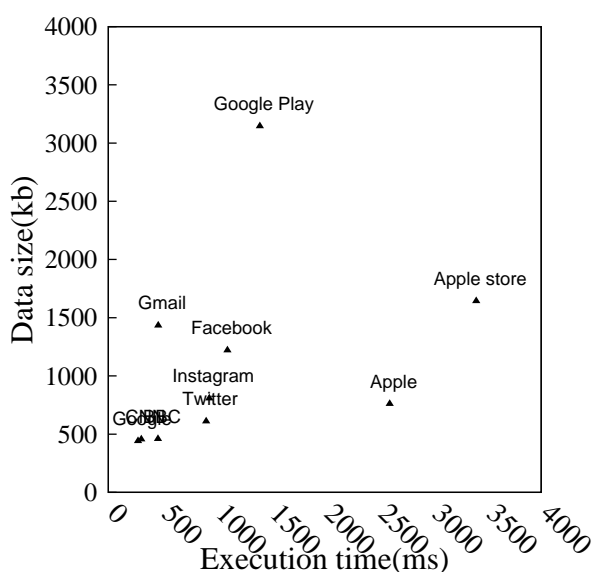


Figure 3.14: Content size and computation workload of the websites for real case study.

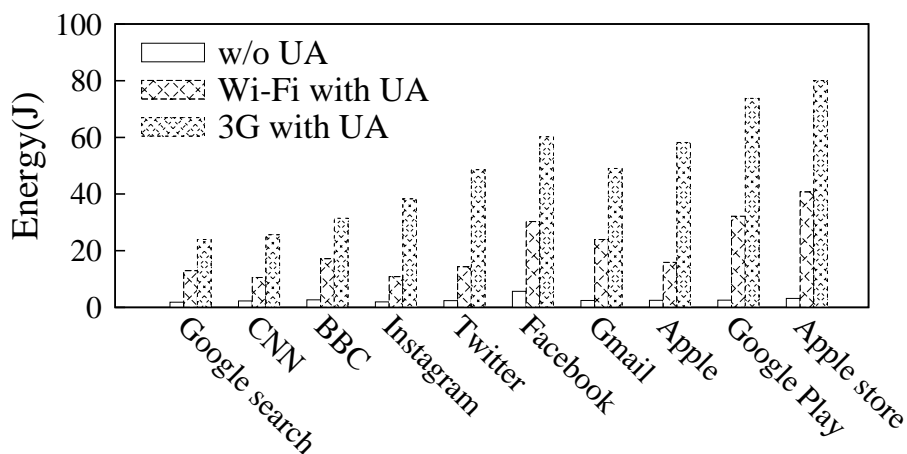
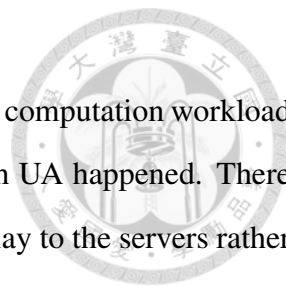


Figure 3.15: Loading energy of the websites.

Similar with the controlled case study, as the data size and computation workload of website increase, more energy are cost to load the website with UA happened. There are some exceptions(e.g.) which are mainly due to the routing delay to the servers rather than the delay of wireless connection.





Chapter 4

Insights to Potential Solutions

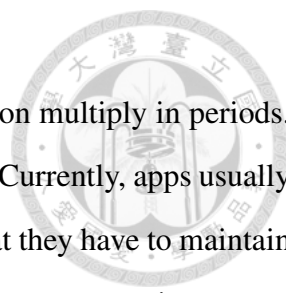
In this section, we proposed several potential directions to reduce unperceived activities we observed in both idle and interactive sections.

4.1 Directions to Potential Solutions

4.1.1 Idling

- **Wake-up Alignment**

To reduce the frequency of active events, the usage of alarm should be managed. If systems simply grant every request from every app, the number of active events will increase and raise energy consumption as more and more apps that utilize alarms are installed on smartphones. Therefore, we think the fire of alarms should be aligned together as much as possible to reduce the frequency of active wake-up events. This can be achieved on smartphones through the scheduling of alarms from apps. We can



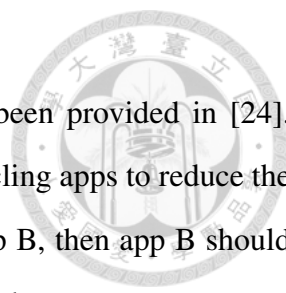
align delay-tolerable alarms or alarms that have least common multiply in periods. Another way is to provide a centralized notification service. Currently, apps usually implements their own notification services, which means that they have to maintain one or even multiple connections to their servers and thus incur more active events. Therefore, we think that notification services should be provided by a single service so that there will be only one connection needed to be maintain on a smartphone. Google Cloud Service has provide a similar mechanism.

- **Network Probing Filtering** The main problem of network probing is that some broadcasts are actually useless to the system and appse, but they are still able to force the device to wake up. If we can filter out those unimportant broadcast packets in Wi-fi or 3G modems before they wake up the system, the number of wake-up events can be reduced effectively.

4.1.2 Interactive

- **Finer Memory Management Strategy Considering Energy Cost**

One of the main source of redundant activities is aggressive memory management with insufficient memory space. Because Android will try to keep background apps in memory as long as possible, the space occupied will expand as users switch between different apps. Once memory usage exceed the specific threshold, Android Low-Memory Killer(LMK) will start killing background apps. If the user switches back to an app that just has been killed, it will need to be rebuilt throughout. This means the whole process of app initialization, including process building, GUI drawing, and other specific jobs of different apps, must be done all over again. To reduce the number of app reload actions, we could utilize the history of app usage to predict what apps are unlikely to be opened next time and choose these



apps to kill when low-memory. Similar mechanism has been provided in [24]. The cost of app rebuilding should be considered when recycling apps to reduce the overhead. If the cost to rebuild an app A is higher than app B, then app B should have higher priority of being killed than app A. We could also use swap space to make a fine-grain memory management, and reduce the possibility for smartphone to enter low-memory scenario. When system needs more memory space, it can be obtained by swapping some pages out temporarily rather than killing a whole app.

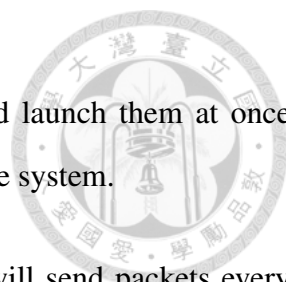
- **Prevent Redundant Resource Usage in Apps**

Another cause of redundancy is the inefficient usage of GUI components. For instance, programmer may use multiple Android Activities to show different part of contents. However, some of them can be shown with partial update of an Activity without building a whole new Activity; When switching between pages presented as Activities, some of the loaded Activities may be released after users pressing the BACK button, and the whole Activity must be rebuild once the page is opened again.

Therefore, programmers should prevent from using too much redundant GUI components to deliver little information. For example, using Android Fragment [2] for partial update of Activities; During the switching of pages in an app, the loaded Activity should be preserved rather than released after users pressing the BACK button.

4.2 Implementation: Alarm Alignment

We modify the Linux module managing real time clock(RTC) to schedule the alarm requests from apps. We simply buffer the alarms that can wake up the system(alarms with



type ELAPSED_REALTIME_WAKEUP and RTC_WAKEUP) and launch them at once every specific period. The period can be specified through proc file system.

In our evaluation, we first consider multiple apps that will send packets every 60 seconds in background even when smartphones are idling. Figure 4.1 shows the total energy consumption after idling for 30 minutes with different number of the testing apps installed. As the number of apps increases, the total energy consumption without alignment rises continuously. After applying alignment, the energy consumption can be reduced up to 50% under Wi-Fi and 63% under 3G network when installing 6 apps; more apps installed, more energy it can save.

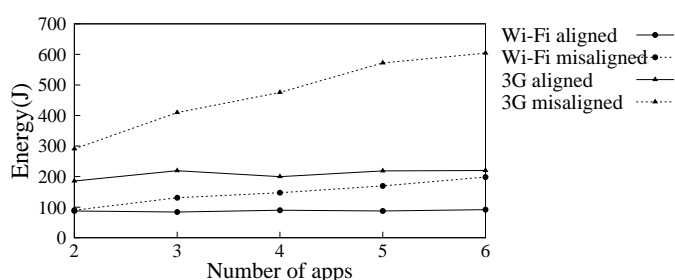


Figure 4.1: Idling energy consumption per 30 minutes with different number of apps before and after applying alignment.

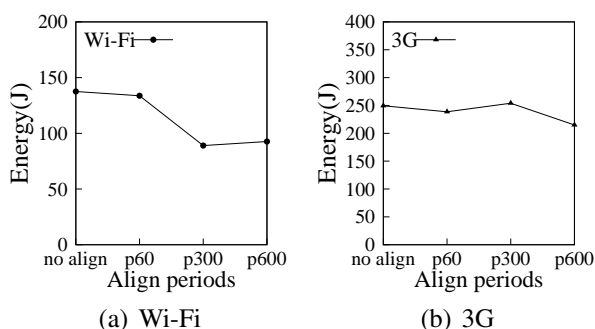
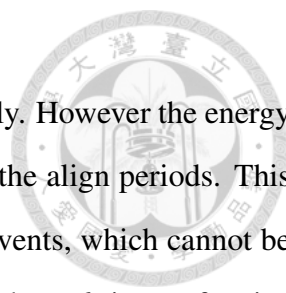


Figure 4.2: Idling energy consumption using different alignment period with LINE, Chrome, Facebook installed.)

Real-case evaluation is also made using our target apps. Figure shows the total energy consumption before and after applying alignment with all 3 target apps installed. Under Wi-Fi network, the energy consumption drops as the align period increases since



the number of active events is limited by the align period effectively. However the energy consumption under 3g network seems to have no dependency on the align periods. This is because that the energy consumption is dominated by passive events, which cannot be controlled through alignment. The number of passive events are about 6 times of active events. Therefore, alignment is still ineffective with 3G although the number of active event has been reduced.

4.3 Implementation: Swap Space

Since Android disabled swap space by default, we modified the configurations of Galaxy S3 stock kernel, rebuild, and flash it into our machine. Then we make a swap file with 512MB in the inner eMMC storage with ext4 file system.

Here we repeat the experiment of interactive session in section 3.2 with swap space enabled. In Figure 4.3, We compare the energy consumption of target apps with UAs before and after applying swap space. In the cases with less waste in the former experiment like LINE, the amount of saving is unapparent; the case with 3G connection consumes even more after using swap space due to the overhead of swapping pages. Nevertheless, the energy consumption can still be cut down effectively in other cases. The ratio of saving can reach up to 32% in the case of Chrome.

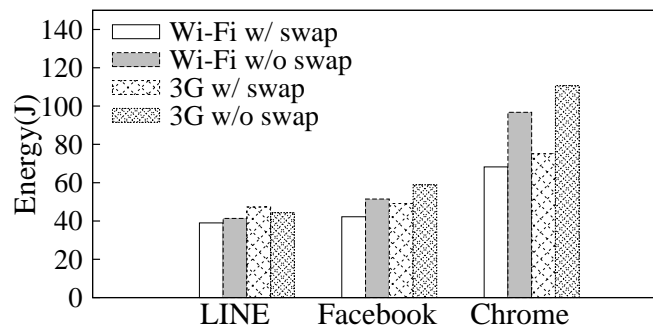


Figure 4.3: Energy saving of paging-out(swapping space)



Chapter 5

Conclusion

In this work, we propose an power analysis considering user perception to investigate whether energy consumption is really worthy to users. We propose the concept of Unperceived Activity and waste of energy caused by it. A series of experiments is conducted to prove the serious of waste and find out the sources and characteristics of different unperceived activities. We found out that these unperceived activities could lead to energy increase up to 136%. Potential directions are suggested to solve the problem. Some simple remedies are also implemented, and energy saved can reach up to 63% in our evaluations.

In our future research, we shall apply this analysis to more applications and different mobile devices, say wearable devices. Based on their characteristics and the observations from the analysis, we can develop more delicate solutions to these devices.



Bibliography


- [1] Americans spend 58 minutes a day on their smartphones. <http://www.experian.com/blogs/marketing-forward/2013/05/28/americans-spend-58-minutes-a-day-on-their-smartphones/>.
- [2] Android fragment <http://developer.android.com/guide/components/fragments.html>.
- [3] Android power manager. <http://developer.android.com/reference/android/os/PowerManager.html>.
- [4] Making smartphones brilliant: ten trends. <http://community.arm.com/groups/arm-mali-graphics/blog/2014/03/25/gpu-compute-dealing-with-the-elephant-in-the-room>.
- [5] Monsoon solutions inc. <http://www.msoon.com/>.
- [6] Number of applications downloaded from google play. <http://www.statista.com/statistics/281106/number-of-android-app-downloads-from-google-play/>.
- [7] Wireshark. <http://www.wireshark.org/>.
- [8] Niranjana Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference, IMC '09*, 2009.



- [9] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, 2010.
- [10] Yu-Ming Chang, Pi-Cheng Hsiu, Yuan-Hao Chang, and Che-Wei Chang. A resource-driven dvfs scheme for smart handheld devices. *ACM Trans. Embed. Comput. Syst.*, 2013.
- [11] Brad K. Donohoo, Chris Ohlsen, and Sudeep Pasricha. Aura: An application and user interaction aware middleware framework for energy optimization in mobile devices. In *Proceedings of the 2011 IEEE 29th International Conference on Computer Design*, ICCD '11, 2011.
- [12] Selim Gurun and Chandra Krintz. Autodvs: An automatic, general-purpose, dynamic clock scheduling system for hand-held devices, 2005.
- [13] IDC. Worldwide smartphone market growth in the first quarter of 2014. <http://www.idc.com/getdoc.jsp?containerId=prUS24823414>.
- [14] Ali Iranli, Wonbok Lee, and Massoud Pedram. Backlight dimming in power-aware mobile displays. In *Proceedings of the 43rd Annual Design Automation Conference*, DAC '06.
- [15] J. Kim, Y. Kim, and S. Chung. Stabilizing cpu frequency and voltage for temperature-aware dvfs in mobile devices. *Computers, IEEE Transactions on*, 2013.
- [16] Chun-Han Lin, Pi-Cheng Hsiu, and Cheng-Kang Hsieh. Dynamic backlight scaling optimization: A cloud-based energy-saving service for mobile streaming applications. *Computers, IEEE Transactions on*, 2014.
- [17] Kaisen Lin, Aman Kansal, Dimitrios LyMBERopoulos, and Feng Zhao. Energy-accuracy trade-off for continuous mobile device location. In *ACM Mobisys*, 2010.



- [18] Aqeel Mahesri and Vibhore Vardhan. Power consumption breakdown on a modern laptop. In *Proceedings of the 4th International Conference on Power-Aware Computer Systems, PACS'04*, 2005.
- [19] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof. In *In Proceedings of the 7th ACM European Conference on Computer Systems*, 2012.
- [20] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. Profiling resource usage for mobile applications: A cross-layer approach. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, 2011.
- [21] Aaron Schulman, Vishnu Navda, Ran Ramjee, Neil Spring, Pralhad Deshp, Calvin Grunewald, Kamal Jain, and Venkata N. Padmanabhan. Bartendr: A practical approach to energy-aware cellular data scheduling. In *In Mobicom*, 2010.
- [22] Alex Shye, Benjamin Scholbrock, and Gokhan Memik. Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 42*, 2009.
- [23] Alex Shye, Benjamin Scholbrock, Gokhan Memik, and Peter A. Dinda. Characterizing and Modeling User Activity on Smartphones: Summary. In *Proc. of ACM SIGMETRICS*, pages 375–376, 2010.
- [24] Wook Song, Yeseong Kim, Hakbong Kim, Jehun Lim, and Jihong Kim. Personalized optimization for android smartphones. *ACM Transaction on Embedded Computing Systems*, 13(2s), January 2014.

- 
- [25] Narendran Thiagarajan, Gaurav Aggarwal, Angela Nicoara, Dan Boneh, and Jatinder Pal Singh. Who killed my battery?: Analyzing mobile browser energy consumption. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12.
- [26] Chanmin Yoon, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, and Hojung Cha. Appscope: Application energy metering framework for android smartphones using kernel activity monitoring. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, 2012.
- [27] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Code-sign and System Synthesis*, CODES/ISSS '10, 2010.
- [28] Xinyu Zhang and Kang G. Shin. E-mili: Energy-minimizing idle listening in wireless networks. In *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*, MobiCom '11.



Curriculum Vitae

Chi-Hsuan Lin was born in 1990 in Taipei, Taiwan. She received his B.S. degree in Department of Computer Science from National Tsing-Hua University, Hsinchu, Taiwan, in 2012. Her primary research interests include low-power strategies on embedded mobile devices and applications, information retrieval, and data visualization.