Sustainable Computing: Informatics and Systems 4 (2014) 183–195

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Power behavior analysis of mobile applications using Bugu

Youhuizi Li*, Hui Chen, Weisong Shi

Department of Computer Science, Wayne State University, Detroit, MI, United States

ARTICLE INFO

Article history:
Received 30 October 2013
Received in revised form 30 May 2014
Accepted 4 July 2014

Keywords:
Application level power
Event analyze
Power efficiency

ABSTRACT

Mobile devices, such as smartphones and tablets, have become an integral part of our daily life. However, the battery drain problem always bothers us. To understand this problem, we design and implement the *Bugu* service which aims to analyzing power and event information and providing users with detailed energy behaviors of applications. We analyzed 100 popular applications' power behavior using Bugu on different platforms. The results showed several interesting observations, including radio service and hardware interrupts, that indicate the potential energy optimization for both applications and systems. We further revealed the underlying reason of different power consumption for several applications in case studies. For example, the low efficient usage of system service in *iHeartRadio*. Finally, lessons learned from software-based power profiling and the ground truth of application level power consumption is discussed.

1. Introduction

Nowadays, mobile devices, such as tablets and smartphones, have become an important part of our daily life. According to a statistical report of Cisco [1], by the end of 2014, the number of mobile-connected devices will exceed the number of people on earth, and there will be nearly 1.4 mobile devices per person in the near future. At the same time, the development of mobile devices also stimulates the application market. The number of Android applications increased 50% in last year, which is over 1,200,000 [2].

There is no doubt that these applications make our life more convenient and colorful, but they are also big energy consumers on mobile devices and significantly influence battery lifetime and user experience [3]. As an end user, we want to know "*For the same functionality, which application is more energy-friendly?*" Except the battery issue, energy efficient applications are more competitive on the market. In a green software awareness survey [4], data shows about 70% people believe that optimizing software is an effective way to save energy and 58% of respondents would select software applications which have energy level labels on them. Application developers often ask the question: "*Why do my applications consume such amount of power?*" especially for mobile devices. System developers focus on the whole system, not just some components or specific applications. Answering the question "*How to save and effectively control system power?*" is the final goal of system developers. However, the first step to answer these questions is to understand the energy consumption of the system and applications.

There are several battery power related mobile applications available at the Google Play store [5–8]. For example, Dr. Power is a tool that presents battery usage for running applications and system [8]. It provides the average power information for sensors, wakelock, data usage and processes in each application. Asides from supporting these data, more detailed component level power information, such as CPU and I/O, will be helpful for developers. Moreover, the average process's power is not enough, the real time power information needs to be exported so that developers can analyze which action/part costs more power. Mittal et al. [9] proposed an energy emulation tool that allows developers to estimate the energy consumption of their apps in a simulator. They considered three components: CPU, network and display in the system, while some useful components are not included (such as DSP and sensors). Trepn Profiler [10] is another power profiling tool which monitors CPU, memory, and network states and supports system battery information. It provides per-rail power usage for latest Snapdragon MDP devices which contain special circuitry. That makes it only suitable for certain types of devices. Besides, all the information is too complicated for normal end users. They prefer to know how much power an application consumes so that they can be guided to choose their applications more effectively.

We design and implement the Bugu service, which is an application level power profiler and analyzer. As Fig. 1 illustrates, the Bugu server returns related applications' power information to

* Corresponding author. Tel.: +1 3134606759.
E-mail addresses: huizi@wayne.edu (Y. Li), huichen@wayne.edu (H. Chen), weisong@wayne.edu (W. Shi).

http://dx.doi.org/10.1016/j.suscom.2014.07.002
2210-5379/

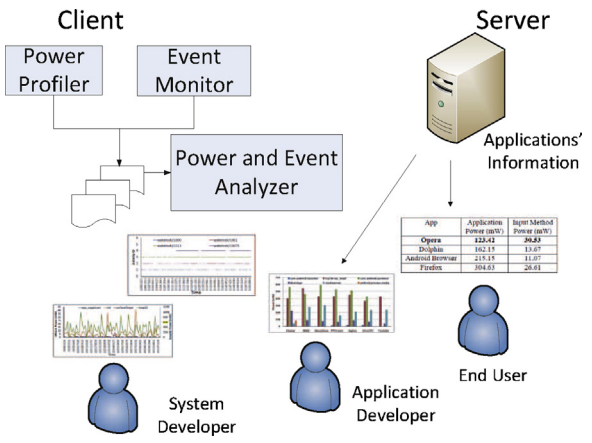


Fig. 1. The overview of Bugu.

end users and gives them more suggestions when they choose applications. For application developers, aside from the similar applications' power information gathered from the server, the Bugu client also shows the event information of their applications, so that application power problems can be easily distinguished. From the viewpoint of system developers, detailed system power information provided by the Bugu client is helpful for them to adopt power saving mechanisms. Leveraging Bugu, we analyzed 100 popular applications and revealed the root causes of high power consumption for some of them in case studies. From the result we observed several aspects that can be improved to save both application and system energy, such as sensors and video module energy efficiency, phone service *rild*.

In this paper, we mainly have three contributions:

- We build the Bugu client which contains profiler, monitor and analyzer. It provides event information that helps application developers optimize their applications and the system-wide power information that assists system developers to analyze background processes.
- With the help of crowdsourcing, the Bugu server provides most applications' power metric for end users. More suggestions are presented to users, and they can make right choice to get the same functionality with less energy consumption.
- We evaluate the Bugu service comprehensively and analyze 100 popular mobile applications on different platforms. Several implications are derived based on the observations and the root causes of large power consumption for several typical mobile applications are analyzed.

In the following sections of this paper, we introduce the Bugu service and the implementation of its components in Sections 2 and 3 respectively. Then, we demonstrate our experiments results and propose four critical implications for energy-efficient mobile-application and system design in Section 4. Following that, we discuss the challenges of software-based power profiler in Section 5. The related work of energy saving approaches and application level energy analyzing is presented in Section 6. Section 7 summarizes the conclusions of our research.

2. System design

The Bugu service is mainly designed for system designers, mobile application developers and end users. Thus, the Bugu service not only presents the application-level power consumption on a single device, but also supplies a group of REST (Representational State Transfer) [11] style APIs for users to share and compare power data. It also supplies event information that may help them to understand the underlying reasons that cause the power consumption.

As Fig. 1 describes, the Bugu service includes two parts: the Bugu server and the Bugu client. The Bugu server collects applications' power information on each device and supports the Bugu client with these data. The Bugu client is used to monitor application power consumption, monitor events and analyze these information. The results are presented in tables and figures for easy understanding and comparison.

2.1. The Bugu server

The Bugu server has two functions: collecting application power information from the Bugu client and supplying these power information to users. Users can contribute their data to the Bugu server by uploading their profiling records, which will help future customers. With the first function, we maintain a large database of application power consumption information on different types of mobile devices. After ranking these applications, users can get better understanding before installing them. The building of this database requires users' contribution so that we can cover as much applications and devices as possible. It is a huge and continuous work. At present, we provide power information for most popular applications on several Android devices that we have.

Based on the type of device and the type of application the user wants to compare, the Bugu server finds the related power information and delivers it to the Bugu client. Then, end users know the comparison results of these applications, they can choose an energy-friendly one to install. For application developers, they can compare the power consumption with the application they developed to evaluate their products.

2.2. The Bugu client

The Bugu client has three main functions: estimating application-level power consumption, monitoring system and application events and displaying the information to the user in a meaningful way. It is composed by power profiler, event monitor, power and event analyzer and user interface module. The procedure is as follows: the power profiler and event monitor record the raw data they need; and the analyzer extracts the data and sends application level power and event information to UI module which displays the data including other applications' information obtained from the Bugu server to users in a meaningful way.

2.2.1. Power profiler

Power profiler is responsible for estimating the system and application power consumption. It uses a group of energy models, which are listed in Table 1, to estimate energy consumption based on how much of each hardware resource was utilized by each application. With the time information, we calculate the average power consumption. The power profiler considers the following components: CPU, Wi-Fi, 3G, GPS, sensors, bluetooth, screen, radio, and so on. We leverage some energy models from our former paper [12], and tune the parameters for mobile platform. For components like sensors, we build the energy models according to their different power states. Aside from application's power, we also record the power of hardware components in the system. So far, we do not consider screen power for each application, while it is available on system level. The reasons are as follows: from research of Dong et al. [13], we know that for OLED screen, different color presented can affect screen power. While the applications' user interface is part of their design style, it will affect user experience if the color is changed. For LCD display, the screen power is determined by brightness level, applications themselves cannot save much on screen

Table 1
The energy models.

Components	Energy models
CPU	$E_{CPU} = \sum_{i=1}^{NumberOfSteps} Time_i * (IdlePower + MaxPower * U)$ $U = (\Delta T_{sys} + \Delta T_{user})/(\Delta T * CoreNumber)$
Wi-Fi	$E_{wifi} = WifiOnAvgPower * WifiOnTime + WifiActiveAvgPower * WifiActiveTime$
Screen	$E_{Screen} = \sum_{i=1}^{NumOfBrightness} (Time_i * (i/NumOfBrightness * ScreenFullPower))$
Bluetooth	$E_{bluetooth} = BtOnAvgPower * BtOnTime + BtAvgPowerAtCMD * BtPingTime$
Radio	$E_{radio} = \sum_{i=1}^{NumOfSignalBin} (SignalTime_i * SignalAvgPower_i) + RadioScanAvgPower * RadioScanTime + PhoneOnTime * RadioActiveAvgPower$

Table 2
Summary of six types of wakelock.

WAKE_LOCK	CPU	Screen	Keyboard
PARTIAL_WAKE_LOCK	On	Off	Off
FULL_WAKE_LOCK	On	Bright	On
SCREEN_DIM_WAKE_LOCK	On	Dim	Off
SCREEN_BRIGHT_WAKE_LOCK	On	Bright	Off
ACQUIRE_CAUSES_WAKEUP	On	On/Off	On/Off
ON_AFTER_RELEASE	On	On	Off

part. The optimization approaches we want to find are from a functionality aspect, not appearance. Although we can get application level screen power according to the time that an application is in foreground and pixel information, it is not very suitable and it increases overhead of Bugu.

The power profiler saves the power information in a formatted log file, which includes the utilization information of all the active applications on each component. The data is recorded once per second. With this information, system designers could analyze the underlying reasons that cause the energy consumption.

2.2.2. Event monitor

Aside from just monitoring the power consumption, Bugu also monitors the events of system and applications. Those events include: wakelock, Wi-Fi state change, bluetooth state change, audio and video state change and different sensors on/off state. For example, we could know when an application acquired or released a wakelock. In the Android operating system, there are six types of wakelock (showed in Table 2 [14]), which represent the privilege to use several hardware devices. All of them make the processor keep in active state. Many applications drain battery so quickly because of misusing the wakelocks. Thus, those events are helpful for us to understand how the applications cause the power consumption.

For each event, we log the time, type and related information, such as the level of brightness, sensor states. The system developers use the information to deeply analyze the system power consumption problem.

2.2.3. Power and event analyzer

The power and event analyzer is used to process the result recorded by the power profiler and the event monitor. The most important function of this module is to calculate the average power of each application. We write the algorithm to calculate application average power. First, we need to filter the effective data by detecting the longest active period of each application. We define application inactive state as its energy consumption does not change in N successive calculation points. The interval between two calculation points is one second. According to our experiments log, most of the applications are paused or went to background if we cannot detect

```
<?xml version="1.0" encoding="utf-8"?>
<device name="Android">
                . . . . . .
    <item name="screen.on">49</item>
    <item name="bluetooth.active">142</item>
    <item name="bluetooth.on">0.3</item>
    <item name="dsp.video">88</item>
    <item name="radio.active">185</item>
    <item name="gps.on">50</item>
    <item name="cpu.idle">1.4</item>
    <item name="cpu.awake">44</item>
    <array name="cpu.active">
        <value>55.4</value>
        <value>82.1</value>
        <value>113.7</value>
        <value>205.4</value>
        <value>259.0</value>
    </array>
</device>
```

Fig. 2. The resource file in Android system.

their energy variation after three times. Sometimes, energy kept the same because of the sampling delay. Hence, in our experiment, N equals three. Then based on the time period and the logged energy information with usage data, we calculate the average power. The analyzing process is done off-line in order to lower the overhead of the Bugu client.

One of the challenges we faced is that the power monitor cannot accurately run periodically, that's because Android is not a realtime operating system [15]. Thus, we improve our algorithm that when we compute the power during two time intervals, the record will be skipped if the time interval is smaller than the threshold. Otherwise, we may get abnormal power results because there is a delay before we obtain the utilization information.

3. Implementation

To implement the Bugu service, we not only developed the server program and the Android client application, but also modified and compiled the Android system to monitor the events. In this section, we describe how we implement the Bugu server, the power profiler, the event monitor and the Bugu client interfaces.

3.1. The Bugu server

The Bugu server maintains application power information, gathers the information from users and provides comparison results to end users and application developers. When users send a request, the Bugu server returns the same category applications list and each item describes the application name and its power consumption. There are two ways for users to contribute their data to the Bugu server. They can choose the *upload* option on their records, or write results on the submission page. We use REST [16] to implement our server, the request URI describes the parameters of the type of device, the type of application and the limit of returned results. The server interprets the request and send back the corresponding results.

3.2. Power profiler

The power profiler is implemented as a service running in the background periodically. It requires the base power of hardware components and their utilization for each application to estimate the power. We get the base power information from the `PowerProfile` class of Android, which reads power values from a resource file (as Fig. 2 presents). For example, we could get the power of the

CPU when it is working on each power step, and the value under *cpu.active* corresponding to power consumption of different CPU frequencies. For the components that are not reachable from the `PowerProfile` class, we did some experiments that described in Section 4 to get their base power. In addition, we get most the application level resource utilization from the `BatteryStats` class. For each application running in the system, their statistic information can be achieved from *batteryStats.getUidStats()*. Then the components utilization information is obtained by calling corresponding method: *getSensorStats()*, *getProcessStats()*, *getWakelockStats()* and so on. The audio and video time are achieved by modifying Android source code since the logging part have not implemented and the original results in `BatteryStats` are all 0. Some data are read from Linux file system, for example, the transmission packets for each process. All the results are already logged for each process, so it can be used directly in the real scenario. The Listing 1 presents the segment of code that describes how we use the information to estimate the energy consumption of each process.

Listing 1. The example of CPU power calculation.

```
private void processCPUPower(Uid.Proc ps)
{
    long userTime = ps.getUserTime(statsType);
    long systemTime = ps.getSystemTime(statsType)
        ;
    appPowerInfo.foregroundTime += ps.
        getForegroundTime(statsType) / 1000;
    appPowerInfo.cpuTime += (userTime +
        systemTime) * 10;
    int totalTimeAtSpeeds = 0;
for (int step = 0; step < speedSteps; step++)
  {
    cpuSpeedStepTimes[step] = ps.
        getTimeAtCpuSpeedStep(step, statsType
        );
    totalTimeAtSpeeds += cpuSpeedStepTimes[
        step];
}

    if(totalTimeAtSpeeds > 0)
    {
        for (int step = 0; step < speedSteps;
            step++) {
            double ratio = (double)
                cpuSpeedStepTimes[step] * 1.0 /
                totalTimeAtSpeeds;
            appPowerInfo.cpuPower += ratio *
                appPowerInfo.cpuTime *
                speedStepAvgPower[step];

        }
    }
}
```

With the base power and utilization information, the power profiler computes the accumulated power consumption of each hardware component. The application power is the sum of all components' power since the components' usage information is recorded for each process in `BatteryStats`. The detailed power and utilization data are logged for further analysis.

3.3. Event monitor

The implementation of the event monitor requires Android system's support, so that we can monitor all the events information. We implement this function by modifying the `BatteryStatsService` class (as Listing 2 shows), which collects all the system and application events that related with battery usage. For each event noted in `BatteryStatsService`, we log its states and make it visible to users. When `BatteryStatsService` receives an event, it will broadcast an `Intent` message, which could be received and logged by the Bugu client.

Listing 2. The example of logging wakelock event.

```
public void noteStartWakelock(int uid, int pid,
    String name, int type) {
        enforceCallingPermission();
        synchronized (mStats) {
            mStats.noteStartWakeLocked(uid, pid, name
                , type);
        }
       if(enableEventListen){
            synchronized(helper)
            {
                helper.noteStartWakelock(uid, pid,
                    name, type);
            }
        }
    }
```

In addition, we also use the event monitor to trigger energy-optimization actions by sending some special `Intent` messages to energy-aware services of the Android system. One message we sent is Wi-Fi tail, which will be generated when the Wi-Fi enters the tail stage. As far as the message is received, the system could leverage the tail stage to piggyback some asynchronous data, such as a post of twitter cached before.

3.4. User interfaces

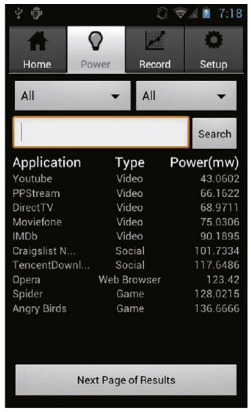
To easily use the Bugu service, we provide friendly user interfaces as Fig. 3 shows. By default, we show all the applications power information in the server as Fig. 3(a) presents, users can search the particular application or category. For each log file, we support six operations as illustrated in Fig. 3(b). Power figure (Fig. 3(d)) shows the most power consumption processes' power variation with time. Event figure (Fig. 3(e)) focuses on one application and shows its event information.

4. Evaluation

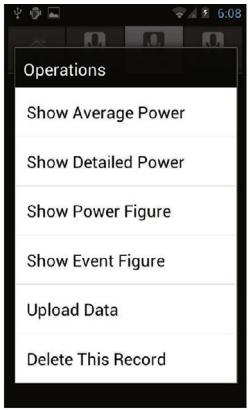
In this section, we evaluate our work on tablet and smartphone. We present how the Bugu service works for three groups of users: end user, application developer and system developer. After collecting 100 applications power data, we did some analysis and found several observations. Moreover, we analyze the overhead of Bugu and summarize the implications we got from the experiments.

4.1. Experiment setup

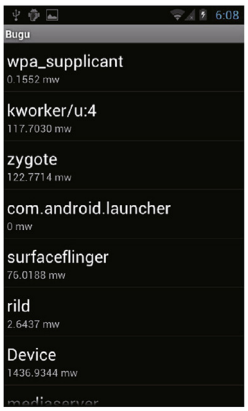
As we described above, Bugu acquires system information from two classes, *PowerProfile* and *BatteryStats*, and calculate application power consumption based on our power models. To verify the accuracy of data read from *PowerProfile* class, we first compared the resource file between different Android OS versions. We found that the file is corresponding to mobile phone models, not the Android operating system. It is the same when we updated from Android 2.3 to Android 4.0. Besides, we wrote our own testing benchmarks to see if the results are consistent with the data recorded in the file. We mainly tested brightness, CPU, socket connection and file input/output. In our experiments, the benchmark applications run foreground and other applications were terminated to keep the accuracy. We connected a resistor between battery and phone, then attached the National Instruments devices [17] to record the voltage of the resistor and the phone. Hence, we got the current of the phone based on the resistor. After that we can calculate power as well as energy information. The resource file contains power data of different state of screen, Wi-Fi, CPU, bluetooth and so on. We run our benchmarks to compare the data we collected with the information in the file. Besides, we calculated file I/O data and put it in



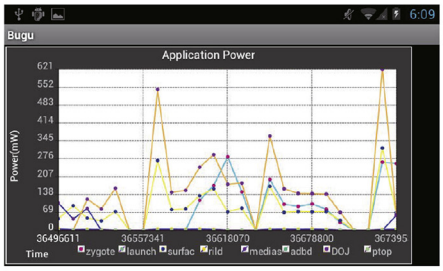
(a) Applications power information achieved from the Begu server.



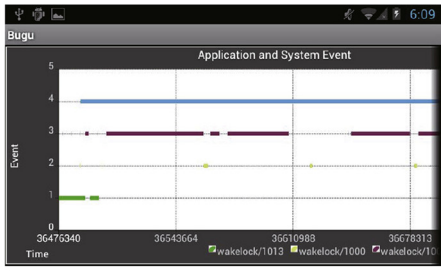
(b) Six operations for recorded log files.



(c) Average power consumption of applications running in the system.



(d) Applications power variation.



(e) Application event information.

Fig. 3. The Bugu client user interfaces.

our power calculating system, which is not supplied in the resource file.

The mobile devices we used for these experiments are Google Nexus S and Motorola Xoom. Their hardware parameters are presented in Table 3 that includes sensors information. Both of these devices use Android 4.0.4 OS. According to AndroLib's statistics [18], there are 640,000 android applications in the market. To ensure representative results, the applications we choose cover most categories. They have over one million installs and ranked in Top-100 as claimed by Google Play. Table 4 lists the information and representative applications.

4.2. Bugu case studies

We introduce three case studies here to illustrate how different kinds of users can take advantage of the Bugu service.

4.2.1. End user scenario

An end user usually wants battery work longer without frequently charging. Aside from saving energy by operating system or shutting down unused devices, this goal can also be achieved through installing energy-friendly applications. The Bugu server maintains a lot of applications' power data with the hardware platform information. End users can request these information and search the category of the application they want to install, the data returned is ranked by the power consumption of applications. Except the application's characteristics, such as UI and special functionality that improve user experience, end users can also take power consumption into consideration.

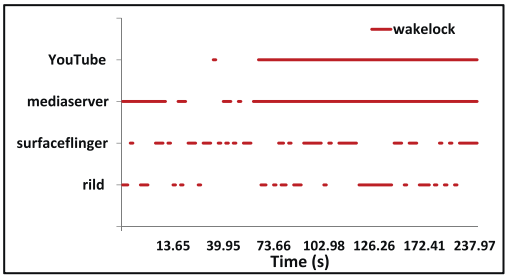
Take browser as an example, assume users want to install *Opera* on their device. They can simply send the type of device and application name *Opera* to the Bugu server. Table 5 lists several applications' data stored in the server, and the browser part will be returned to the users. Noted that the power data is calculated under the general usage situation. In this experiment, we chosen six popular websites including *cnn*, *espn*, *amazon*, opened them one by one and each time scrolled down to see all the information. We can see that *Opera* consumes less power than *Firefox*, which makes it more competitive. To figure out the behind reasons, we analyzed the event information and raw power log data of *Opera* and *Firefox*,

Table 3
Experiment platforms.

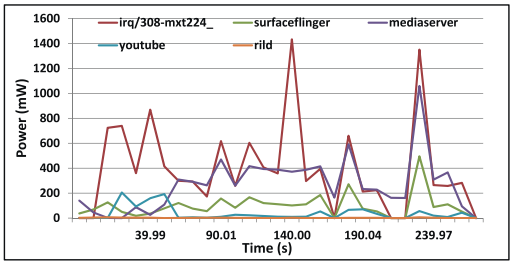
Hardware components	Nexus S	Xoom
CPU	ARMv7 Processor rev 2	ARMv7 Processor rev 0
Frequency (MHz)	100–1000, 5 steps	216–1000, 8 steps
RAM (MB)	335	718
Sensor	KR3DM Accelerometer GP2A Light Sensor AK8973 Magnetic Sensor	KXTF9 Accelerometer Ambient Light Sensor AK8975 Magnetic Sensor

Table 4
Summary of selected applications.

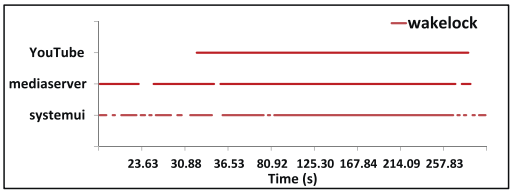
Category	Applications
Business	Documents To Go, UPS Mobile, Pocket Cloud Remote, etc.
Game	Fruit Ninja, Temple Run 2, Talking Tom Cat, etc.
Finance	Google Finance, Expense Manager, TurboTax SnapTax, etc.
Health and Fitness	Instant Heart Rate, Workout Trainer, Lose It, etc.
Media and Video	YouTube, RealPlayer, Movies by Flixster, etc.
Music and Audio	iHeartRadio, Amazon MP3, Google Play Music, etc.
Education	Kids Animal Piano Free, How to Draw, Aldiko Book Reader, etc.
Tools	PicsArt, Barcode Scanner, Tiny Flashlight, etc.



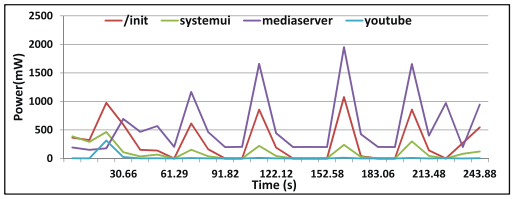
(a) Event information of YouTube running on Nexus S.



(b) The power variation information on Nexus S.



(c) Event information of YouTube running on Motorola Xoom.



(d) The power variation information on Motorola Xoom.

Fig. 4. The comparison of YouTube event and power information.

Table 5
The comparison of applications power consumptions.

Browser	Application power (mW)	Game	Application power (mW)
Opera	123.42	NinJump	141.73
Dolphin	162.15	Temple Run	142.75
Firefox	304.63	Cut the Rope	149.12
Health	**Application power (mW)**	**Reading**	**Application power (mW)**
Instant heart rate	65.96	Kindle	86.34
Lose It	83.55	Daily Bible	131.23
Cardiograph	92.26	Audible	158.95

the results show that their CPU power has big difference. *Firefox* may do more processing and calculation to improve user experience, further analysis about their power behavior can be found in Section 4.3.3.

4.2.2. Application developer scenario

On one hand, the Bugu server provides related applications' power data for application developers to compare. On the other hand, developers can get event information from the Bugu client, which gives the optimization direction from power consumption aspect. In this section, we use video application as an example to show how Bugu works. In our experiment, the new application developed is *YouTube*.

To include the influence that application may bring to the system, the Bugu server not only provides each applications' information, but also gives other four most power consumption processes of each application and compares the union of them. Thus, there are six processes compared in Fig. 5. The data is collected from Nexus S, and we can see that system processes which support our applications consume much more power than the application itself. For instances, *systemui* is responsible for drawing the user interface, *mediaserver* provides sound and other support for media. From the perspective of the target application, *YouTube* is in a good situation, its power is lower than others. Fig. 4 presents the event information and power variation of *YouTube* on both Nexus S and Xoom. These information helps developers deeply understand the power issue. For event information, the *x*-axis is time; the *y*-axis is the processes that generate these events. The recorded events include wakelock, sensor, screen, etc. We only found wakelock information in this scenario, the *mediaserver* process appeared in both devices, and *rild*, *surfaceflinger* occupy wakelock on Nexus S while *systemui* on Xoom side. *YouTube* also occupies the wakelock for a long time as showed in Fig. 4(a) and (c), developers can analyze their code to improve the wakelock utilization, for instance, release wakelock in app pause state. Fig. 4(b) and (d) demonstrate power variation of processes which occupy the event or has high power consumption, *YouTube* consumes high power when we start the application, while *mediaserver* periodically reached the high point. After analyzed the resource usage information in the beginning of *YouTube*, we found it transmitted network packets and dealt with user inputs (e.g. touch, click). No other abnormal data detected. Another reason for such high power consumption is the preparation system did for starting new activity. So if the developers want to optimize *YouTube*, they should focus more on handling user inputs efficiently and balancing data downloaded.

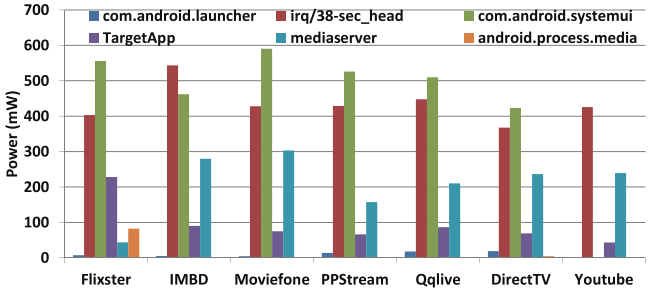


Fig. 5. The power comparison of seven video applications.

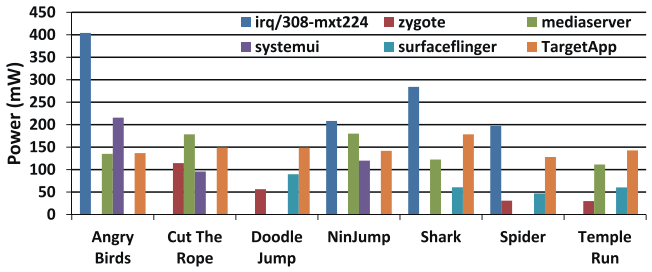


Fig. 6. The power comparison of seven games.

4.2.3. System developer scenario

System developers care more about the whole system power consumption, not a particular application or hardware component. Bugu provides power information of all processes running in the system, which exactly helps them to know the whole picture. From previous experiments, we observed that system processes consume much more power than target application itself. To show it is a common issue, we did another experiment on game applications. We evaluated 7 popular games: Angry Birds, Cut The Rope, NinJump and so forth. Fig. 6 demonstrates the power consumption of each game and several corresponding system processes on Nexus S. We can see that system processes, such as *irq/308-mxt224*, *mediaserver* and *zygote*, consumed much power, they were not negligible comparing with our target applications.

Except active applications, background applications are also a main concern for system designers. We did several experiments to show how applications and the system behave in sleep mode. In the experiments, we first tested the situation that only system processes exist and no applications opened. From Fig. 7 we notice that *surfaceflinger* occupied the wakelock almost all the time on Xoom, while *systemui* and *rild* dominated on the Nexus S. *surfaceflinger* and *systemui* work on the user interface drawing and rendering part, *rild* is responsible for the phone service. These processes acquired and released wakelock continuously, which make

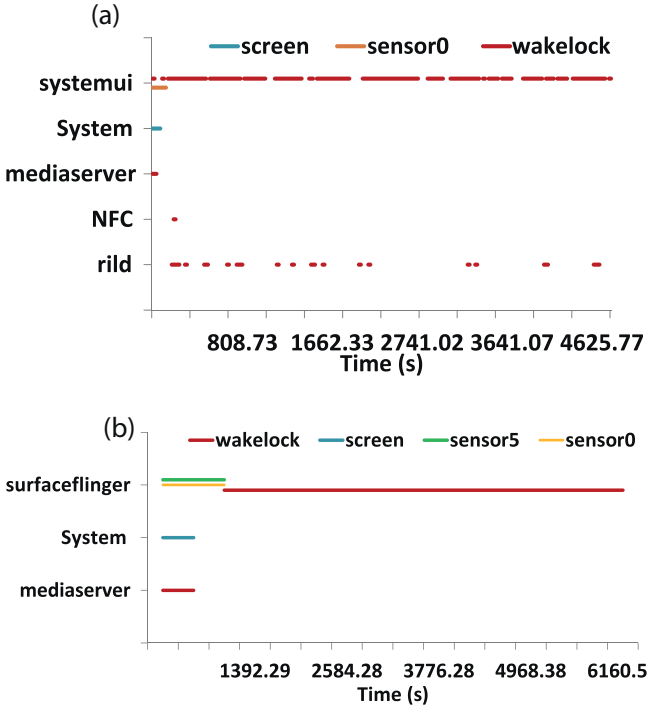


Fig. 7. The comparison of devices event information under "sleep" mode with no application running.

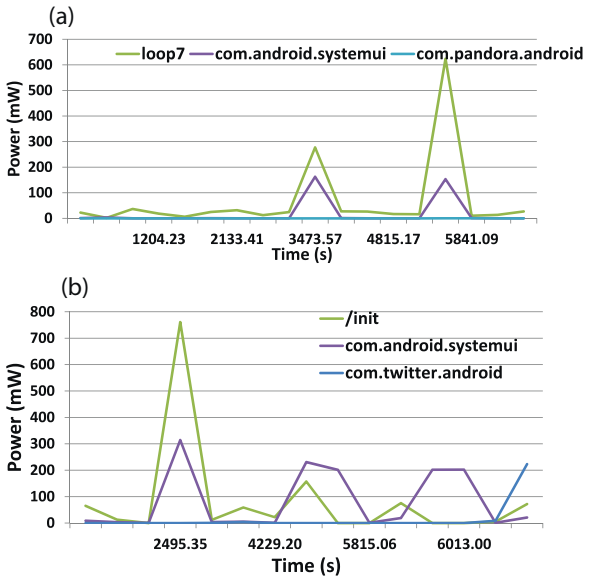


Fig. 8. The comparison of devices power information under "sleep" mode with background applications.

the processor can hardly get the chance to work in C states. Comparing with Nexus S, Xoom can last much more longer after one fully charge. To present the real case when users use these devices, we did the experiments with applications running in the background. The most common situation it represents is when users go to sleep, their mobile devices are in sleep state without exiting all opened applications. Fig. 8 shows the power variation of top three power consuming processes of Nexus S and Xoom under "sleep" mode with unclosed applications. Before we put the devices into "sleep" mode, we opened *facebook*, *twitter*, *youtube*, *angrybird* and *pandora*, and played with each of them for a few minutes. From the figure we know that most applications' power are low, while system processes still consume a lot. Hence, system developers should focus more on optimizing these background processes and services.

4.3. Applications power information analysis

In this section, we analyze mobile applications' power data in both foreground and background situation. For each case, we describe applications' total power and distribution among main hardware components.

4.3.1. Apps run in foreground

We first introduce foreground scenario. In the experiments of 100 applications, their power ranges from 20 mW to over 700 mW. 10% of them consumes less than 50 mW, 50% is less than 200 mW. The average power is 227 mW, and 20% is greater than 335 mW in our dataset. It is reasonable that the power varies so much. Pdf reader will run longer than Angry Bird with the same battery capacity. Intuitively, the power consumption of applications in the same category should be in the same level. To further prove the statement, aside from video and game applications presented above, we also took the Education, Health and Fitness applications' data into consideration. In these four categories, Education apps consume less power than the other three categories; only the power of NYTimes greater than 200 mW. Most of Media and Health apps power are within 300 mW and 200 mW respectively. In a specific category, the applications' power also varies. The power difference between Temple Run 2 and Speed Skater is as high as 300 mW. For applications produced by the same company, the difference is

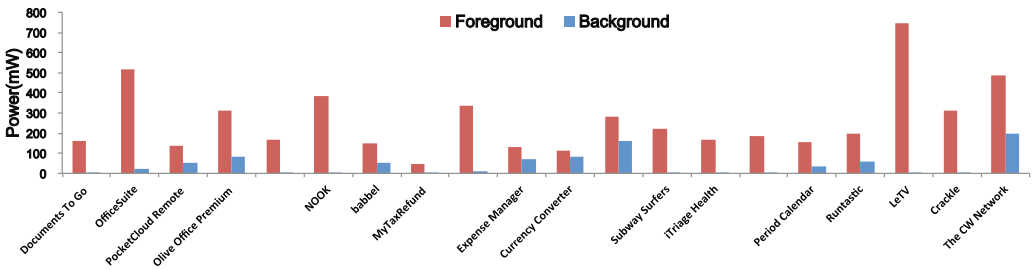


Fig. 9. The comparison of applications background and foreground power consumption.

smaller as "Talking" series (Talking Angela 320 mW, Talking Ben 350 mW, Talking Tom Cat 410 mW) suggest.

To figure out where the power goes, we analyzed the detailed power information logged by the Bugu client which contains main hardware components' power dissipation for each application. We summarize two metrics which are important factors to reflect component power information: *NumberofAppearance*, it is defined as number of applications use the component over total number of applications, and *PowerRatio*, which equals the percentage of the component consumed power over total application power. According to our experiment results, CPU is used in all applications and its average *PowerRatio* is the highest in the components we considered. This means most of the time the CPU dominates the applications' power. 13% and 20% applications use GPS and Audio respectively, and they contribute nearly 20% power to the applications. Although there are only 14% applications in our dataset play with Video module, the average *PowerRatio* is as high as 61%. After focusing on high video power ration application, we found that video power is much higher than CPU power except two applications that their video power and CPU power are almost equal. Hence, when applications play video, its main power dissipation very likely transfers from CPU to Video.

4.3.2. Apps run in background

For background applications, we classify them as two categories: idle background and active background. The former represents the applications that will stop working and enter suspend status when in background. Active background means the applications that still have activities even in background, such as download applications and music applications.

Idle Background: The idle background situation is common for most applications, especially for media apps and games. Fig. 9 demonstrates background power and foreground power of 21 applications in our dataset. 60% of the applications, their background power is less than 50 mW, and two of them are over 100 mW. The background power varies from 1.5 mW to 190 mW. The applications are listed by their category. Similar with the foreground case, the background power of applications which are in the same category also varies; Office Suite background power is 20 mW while Olive Office Premium reaches 80 mW. For applications with high ratio of background power to foreground power, like Expense Manager, we found their power is dominated by CPU power consumption.

When applications go to idle background state, users move their focus to the new foreground application. Except maintaining the status in case they will run again in a short time, they should occupy resources as less as possible. Hence, an energy efficient application should reduce their background power consumption and maintain the ratio of background power to foreground power in a relatively small range.

Active Background: Some applications still active and function normally when they are in background. For example, we open Pandora to listen to music and at the same time we check emails or read news in foreground. In that situation, we claim that Pandora is in *ActiveBackground* state. For this kind of applications, they complete most of their work in active background situation.

In the experiments, we choose five popular applications from Music and Audio category: Pandora, iHeart Radio, Amazon MP3, TuneIn Radio and Spotify, and four download applications: Download Manager, tTorrent Lite, uTorrent and aTorrent. Amazon MP3 randomly played local songs and radio applications played several stations, four download applications downloaded a 325 M video file. Fig. 10 describes their power dissipation in foreground, active background and idle background situations. When the applications enter active background situation, their power dissipation is less than foreground case and most of them only decrease a little. The power consumption of uTorrent in the two cases are almost the same, Pandora's power reduce 20 mW which is about 5% of total power. For idle background situation, six of their power are less than 5 mW, aTorrent's idle background power is also less than 6% of total foreground case power. Spotify and aTorrent decrease around half of the power when enter active background case, the user experience of the two applications did not change, there was no visible delay to play music and download the video. The possible explanation may relate to other functionality suspended in background.

4.3.3. Real Apps case studies

We analyzed several applications' power information, which includes: *Pandora*, *iHeartRadio*, *Facebook*, *Firefox*, by tracking resource usage information through Bugu. With the consideration of system processes and the comparison of similar applications' data, we revealed some underlying reasons of high power situations. In the experiments, we installed target applications in a clean OS and the logged power is the whole system power. Since we only run the target application in foreground and suspend all background processes (e.g. Google Services), any system power variation is mainly caused by the target application.

In the Music and Audio category, the most popular applications are *Pandora* and *iHeartRadio*. Figs. 11 and 12 demonstrate their CPU and system power variation when listen to music. By

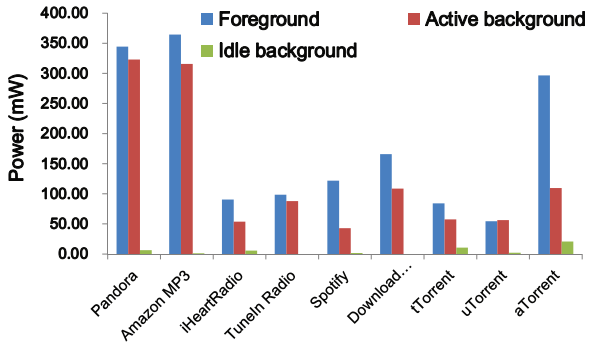


Fig. 10. The comparison of applications power consumption in foreground, active background and idle background.

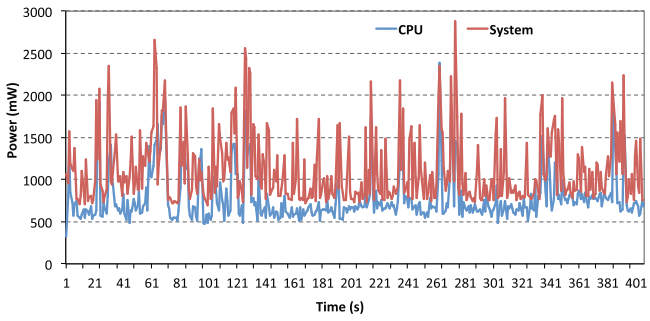


Fig. 11. The system and CPU power information of Pandora.

comparing the two applications' system power, we noticed that the power of *iHeartRadio* was higher than *Pandora*'s when playing music. The CPU power in the two applications was reasonable as the high power situations were caused by handling user input, such as changing channel/song, and network transmission. To find the root cause that lead to high system power of *iHeartRadio*, we first detailed compared the resource usage information which includes audio, video, wakelock, CPU time and network packets of the two applications. Fig. 13 shows their audio and wakelock time, the audio time of *iHeartRadio* was all 0. Hence, when we calculate the application power, the result of *iHeartRadio* was less than *Pandora*'s (showed in Section 4.3.2) as the audio power and wakelock power of *iHeartRadio* were almost 0. However, it did not illustrate the high system power of *iHeartRadio*. Next, we analyzed the information of all processes running in the system. Aside from Bugu, system (uid:1000) and the target application, the active process was mediaserver (uid:1013) in both cases. Because the audio time is logged in *MediaPlayer.java* in Android OS, we think *iHeartRadio* did not use build-in player program to communicate with mediaserver, which causes high system power consumption when playing music. The resource usage information of *Douban Artists* further proved the statement since the trend of audio time and system power were similar with *Pandora*'s. The wakelock data of *iHeartRadio* and *Douban Artists* was almost the same, and the high

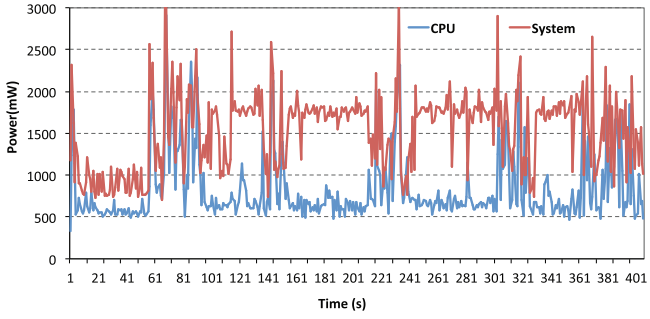


Fig. 12. The system and CPU power information of iHeartRadio.

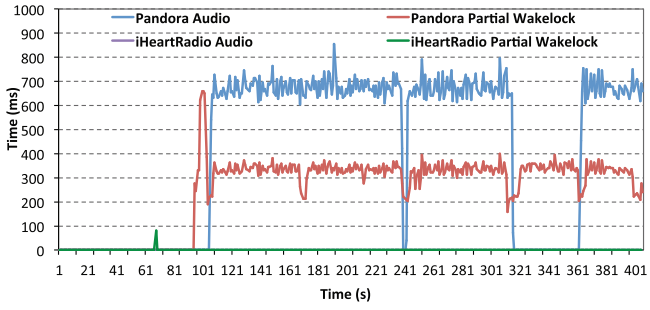


Fig. 13. The information of wakelock and audio time for Pandora and iHeartRadio.

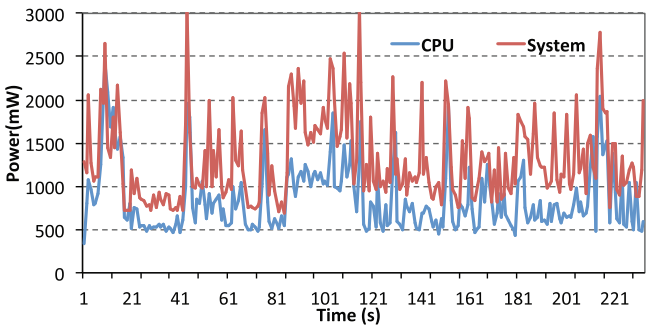


Fig. 14. The system and CPU power variation of Facebook.

wakelock usage in *Pandora* was mainly the result of frequent advertising. Social network application becomes the main platform that keeps people in touch in today's society. In our experiments, we checked the latest news of friends and posted the status with and without photo. The power variation of *Facebook* and resource usage information are demonstrated in Figs. 14 and 15 respectively. At the starting of the application and dealing with the user inputs, CPU power dominated the whole system power. There is a high power period from 81s to 115s, and it is caused by taking a photo as shown in Fig. 15 (*Facebook Full Wakelock* and *Facebook Accelerometer* overlapped). When we prepared to post status, the location process became active for several seconds and it used *Particial Wakelock* and GPS. Users may share their location in the posted status. As the result, the corresponding system power was increased a little bit. The same situation can be also found in *Twitter*, the location process appeared and the system power increased. When we posted a status with photo, *Twitter* delegated the job to Android default application *Gallery* while *Facebook* handled by itself. On the aspect of the system power, the two approaches are similar although the wakelock and accelerometer were used by different processes.

For browser applications, we analyzed *Firefox* and compared its data with *Opera*'s as the high power consumption of *Firefox* demonstrated in the previous section. We opened several popular webpages. On the perspective of system processes, the situations of the two applications were similar. The particial wakelock's time of mediaserver and location process occasionally increased, they were not actually in active state. The wakelock time of *Google Search Box* also increased, which was more frequent than in other applications, such as *YouTube* and *TempleRun*. For application itself, CPU power dominated the whole power. Aside from user inputs, network activity also causes high CPU usage. Fig. 16 illustrates the system power and packets information when *Firefox* was in foreground. The peak points of high packets transmission correspond to high power consumption. There are a lot of times that packets were over 10,000, while the situation happened much less in *Opera*'s case. Hence, we think it is the main reason for high power consumption of

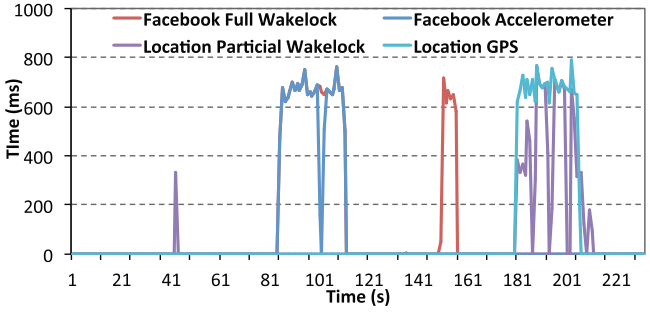


Fig. 15. The part of the system resource usage information when playing with Facebook.

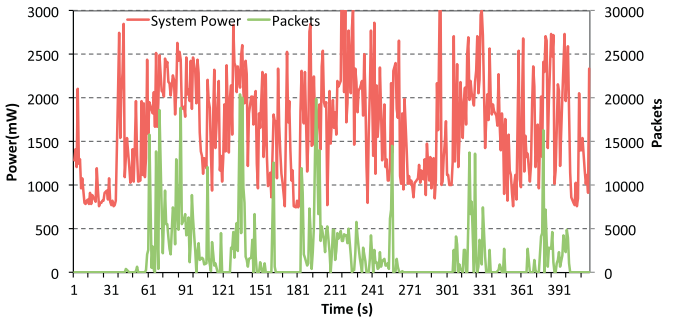


Fig. 16. The system power and packets information of Firefox browser.

Firefox. For download applications, high packets transmission may help save energy since the system can go to sleep state after the job is done. However, it is not hold for frequent user interactive applications as the interval time between two tasks (user inputs) is not always longer enough for system to switch to the sleep state.

4.4. Bugu accuracy

As discussed in Section 5, there is no ground truth for application level power consumption. Hence, to analyze the accuracy of Bugu, we focused on the whole system power. Fig. 17 demonstrates the measured power and estimated power for several popular applications. We used a BK Precision programmable power supply [19] to power up the smartphone, which provides a constant voltage of 3.8 V and records current data. We calculated the system power, which is listed as measured power, based on the current information. The estimated power is calculated and logged by Bugu. For game, music and video applications, the estimated power is greater than hardware measured power; for social and utility applications, the most results from Bugu is equal or less than the measured power. The average error rate of Bugu for total system power is 5%.

4.5. Bugu overhead

The overhead of Bugu is mainly caused by the power profiler and event monitor. The data processing is done when the user wants to read an experiment record. When we did experiments described above, we also recorded the power consumption of Bugu. The power consumption of Bugu is around 5–10 mw, which accounts for 2.52% of the foreground application power consumption on average. Moreover, we compared the system power with and without Bugu. The power results were calculated by attaching the power meter to the battery. For the situation that no active foreground application exists, Bugu causes 200 mW extra system power. Because Bugu samples resource usage information once per second, it stops the CPU and system to stay in a low power state and lead to such amount of system power overhead. In real measuring cases, there is always a "target" application running, the average extra power Bugu generated on the system level is around 100 mW. Compare with 1000–1500 mW whole system power, the overhead is acceptable.

4.6. Implications

4.6.1. Radio service

Our experiments show that *rild*, which is the daemon of Android radio service, generates a lot of wakelocks even when the device is not active. Even though the power consumption of this service is not high, it keeps the processor active and consumes a large amount of energy. These wakelocks are generated during processing unsolicited commands, such as network status change, SMS notify, USSD (Unstructured Supplementary Service Data) notify and signal strength or time changed. Among these unsolicited commands, some of them, such as SMS notify, are important to users. However, we do notice that a large amount of unsolicited commands, such as signal strength change, received are not highly required. To design an energy-efficient Android radio layer interface, we should reevaluate the structure of unsolicited command processing part by filtering part of the commands. In this way, we can increase the chance of making the device work in the "real sleep mode". Another approach is putting the long lasting service to a low power coprocessor, so that the coprocessor can handle part of the data processing without waking up whole system.

4.6.2. Hardware interrupts

In addition, we observed that several processes, such as *irq/308-mx224_* and *irq/38-sec_head* consumes a large amount of energy when we ran several applications. *irq/308-mxt224_* is the threaded interrupt handler for the touchscreen controller. Different with traditional cell phones and normal computer systems, current mobile devices have much more sensors to supply various functionality to users. These sensors generate a large amount of hardware interrupts and consume a large amount of energy. Aside from *rild*, the sensor related processing can also be delegated to the low power coprocessor. We argue that we should revisit the design of interrupt handling part for current mobile operating systems since the design of hardware platform is totally different now.

4.6.3. Energy-efficient applications

As our experiments in Section 4.3 show, applications' power consumption varies a lot. In low battery status, users can stop some unnecessary applications to save energy for phone service. Besides, the power consumption of applications in the same category can be very different even though the functionality of them are the same. That means, it is possible to develop energy efficient

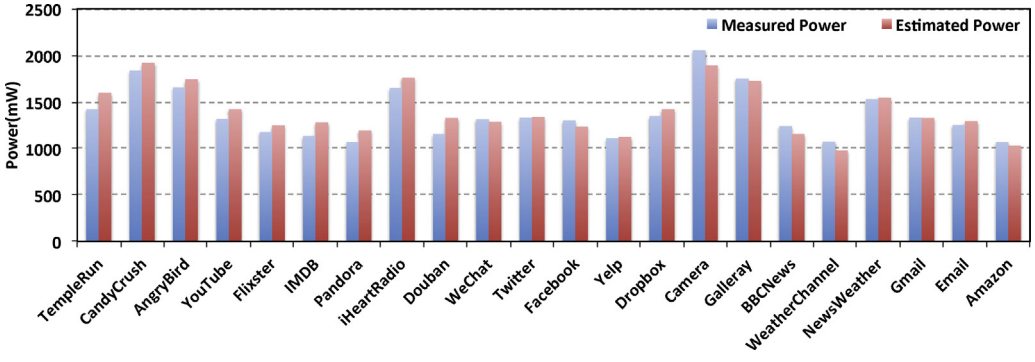


Fig. 17. The comparison of measured power and estimated power for popular applications.

applications without influencing user experience. In detail, there are two directions that we can look into. Aside from the main hardware components, other parts such as DSP, sensors are also needed to be used in an energy-efficient way. For example, video applications may use video module more than CPU since video module power consumption dominates the whole application's power. Another direction is to improve applications' background situation. From the experiment results, some applications in the background are not really suspended. Considering the user behavior that they usually put applications in background rather than kill them, these applications may generate big influence on the system energy. Hence, when the developers implement applications, they should reevaluate the background case and decrease the power consumption as much as possible.

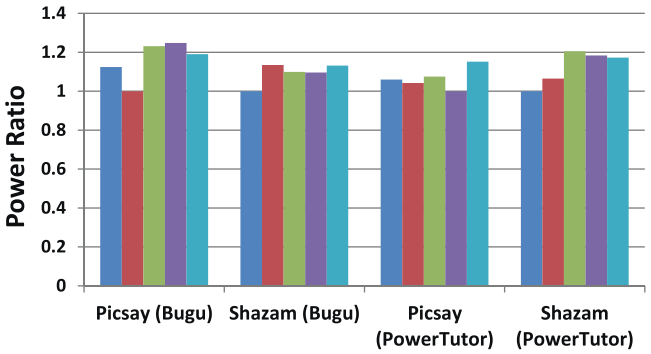


Fig. 18. The comparison of different power results.

4.6.4. System power management design

One important goal that we design operating system is to protect the hardware from misuse by applications. However, some power management APIs, like wakelock, are not used efficiently and they can cause big energy issue. We think the APIs for application design should be reevaluated from energy saving angle.

The energy consumption of screen, processor, radio and Wi-Fi accounts for about 95% of the whole system energy consumption. Among these devices, it is hard to decrease the power of screen, which accounts for about 50%, through many kinds of system level energy-efficient strategies. In addition, the space to decrease the power of radio and Wi-Fi is low if the users need to use them. Even if we could filter some of the unsolicited commands, we cannot make radio work in low power mode. Thus, it is nearly inevitable to design an energy-efficient strategy that can drop the energy consumption of the system significantly. So, we claim that there is no chance to solve the power problem for mobile devices with a single energy-efficient strategy. The mobile operating system needs a group of energy-efficient design strategies to work together to accomplish this goal.

5. Discussions

Throughout the performance evaluation of Bugu, we noticed that there are two fundamental issues about power profiling that need to be answered, i.e. *Accuracy of software based power measurement tool* and *Ground truth for power measurement*.

Is software based power measurement tool a good choice? There are two basic approaches to measure power/energy: hardware measurement using power meters and software based measurement tool leveraging power models. Different power models are applied in the software measurement tools. For instances, hardware components utilization power model in PowerTutor [20] and finite state machine of components' states transition in Eprof [21]. Since the software approach is easier to use and no additional device is required, more and more people prefer to choose software tools to measure power/energy consumption. However, in our experiments, we found the software-based results are not stable at least for some applications. In Fig. 18, we listed two applications' power information measured by Bugu and PowerTutor. In our experiments, each application ran 5 times and each time the operations we did are identical. In order to eliminate application interference, only the target application and the measurement tool are opened. We normalized the power consumption to the minimum value in the total 5 results to emphasize the power variation. The data shows that the variation can be as large as 20%, in that case, what result should we use? It is hard to tell the correct value and it is possible that if we continue measure, there will be more options come out. The situation may be caused by the operating system process, the communication between application and OS and the complicated hardware/software interactions. Hence, the software-based result is reasonable after repeated experiments and a single experiment data is not very useful. In other words, the results measured by the software-based tools are accurate at some extent.

Ground truth? How to evaluate the accuracy of a power measurement tool? The common approach is comparing the calculated results with the ground truth. However, the ground truth for application level power is not easy to get. As we described above, comparing with other software-based tools is not a good choice. From the hardware measurement point of view, a simple approach is using power meter to get the whole system data. If we want to get each component's power information, the measurement should be did on the testbed where the circuit information is provided. While the two approaches are for system power, they cannot give an accurate power information for each application if there are multiple application running in the system. So, most researchers only run one application to match the results in training stage. The big challenge is how to get the ground true of allocating system power to each application and considering the application interference.

6. Related work

Nowadays energy becomes more and more important. There are many researchers working on the energy saving issue. As early as 2002, energy has been treated as a first-class resource in ECOSystem [22] which contains a currentcy model and allocate the energy to different tasks according to user preferences to extend battery lifetime. Besides, Koala [23] predicts the performance and energy consumption and dynamically control frequency to save energy. Since mobile devices become more and more popular, researchers are also interested in saving energy in mobile field [24,25]. For example, Cinder [26] also leverage the idea and treat energy as a resource, but it allocates energy directly to each processes and uses hierarchical structure to control the resource, which avoiding the competing between parent and child processes. Roy et al. [27] implemented Cinder and showed the good performance even with malicious applications. However, all these researches are from system viewpoint and they need to change original operating system, which is not easy to implement and spread. The Bugu service can benefit three groups of people, for system developers, it is a tool to analyze existing problems and then modify system accordingly.

To save power, we first need to measure the power. The easiest way is using multimeters to read data and calculate power directly, so we need extra device to calculate power in the early stage. The prototype version of PowerScope [28] uses a digital multimeter to sample the current drawn of the profiling computer and records system activity at the same time. As a result, it generates an energy profile for later analysis. PowerPack [29] also uses digital meter

to measure, while it focuses on each hardware components (CPU, disk, memory, etc.). For convenience, researchers use power model to calculate power consumption. Dempsey [30] extracts power parameters to model the disk drive power consumption. Bertran et al. [31] took advantage of performance counters to build power model and provided per component power consumption. Quanto [32] system addresses network communication power model in embedded system, the key parameter is network event. Dong and Zhong [33] provided a self-constructive approach to build system energy model for mobile systems by using the smart battery interface to get enough information. Bugu system provides application level energy information by monitoring hardware status. Although AppScope [34] also applies that approach, they get time information by monitoring kernel events. Besides, their goal is to measure energy and build energy metering. Our work pays more attention to analyzing the results we get and use them to bring benefits for clients.

Speaking of saving energy for mobile devices, researchers pay more attention to specific components. For instances, EnTracked [35] uses dynamically changing context to schedule GPS so that it is energy-efficient as well as keep the performance. MAUI [36]saves energy through fine-grained code offload after evaluating energy consumption under connectivity constrains. Duan and Bi [37] proposed a hybrid approach which leverages mobile RAM and phase change memory to achieve memory energy optimization. Our work does not care particular part of mobile devices or specific application. We focus on general applications running in the system, give users application-level energy information and help them save energy from both application and system aspects. Pathak et al. [21] presented an energy accounting approach on application level and proposed saving energy by optimizing I/O bundle. The Bugu service illustrates the event information and application power consumption, and gives the whole system power picture. It benefits both application developer and system developer. Carat [38] also compares application's power consumption by collecting from a community of mobile devices. They provide hog report describes the energy hungry applications among all clients and bug report which shows the energy hungry applications only on your machine. However, without information of applications in the same category, they cannot detect if the hog application has an energy bug. Besides, they cannot provide more analysis for developers to optimize their applications.

7. Conclusions

In this paper, we analyzed mobile applications' power behavior using Bugu which is an application level power profiler and analyzer for mobile phones. Bugu is composed of a server side which provides power information of different applications, and a client side that analyzes power and event information for specific applications. We implemented Bugu on Android platform and evaluated its accuracy (95%) and overhead. We showed the case studies of finding the root causes of large power consumption. The analysis of 100 applications' power information is useful for many energy/power related researches on mobile devices, and the implications derived from the observation point out several potential optimization directions.

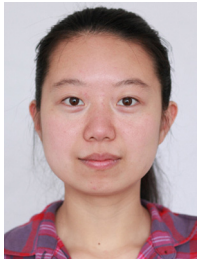
Acknowledgements

The authors would like to thank the anonymous reviewers for their comments and kindly suggestions. This work is in part supported by NSF grant CNS-1205338, the Introduction of Innovative R&D team program of Guangdong Province (No. 201001D0104726115), and Wayne State University Office of Vice President for Research. This material is based upon work supporting while serving at the National Science Foundation.

References

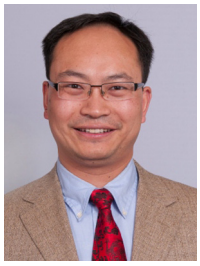
[1] Global mobile data traffic forecast update, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html
[2] Number of android applications, http://www.appbrain.com/stats/number-of-android-apps
[3] A. Pathak, Y.C. Hu, M. Zhang, Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices, in: Proceedings of the 10th ACM Workshop on Hot Topics in Networks, ser. HotNets-X, New York, NY, USA, ACM, 2011, pp. 5:1–5:6.
[4] Green software awareness survey, http://www.sig.eu/blobs/Nieuws/2011/Results%20Survey-201109.pdf
[5] Battery doctor, https://play.google.com/store/apps/details?id=com.ijinshan.kbatterydoctor_en
[6] Juicedefender, https://play.google.com/store/apps/details?id=com.latedroid.juicedefender
[7] Battery stats plus, https://play.google.com/store/apps/details?id=com.rootuninstaller.bstats
[8] Dr.power, http://tawkon.com/blog/en/dr_power
[9] R. Mittal, A. Kansal, R. Chandra, Empowering developers to estimate app energy consumption, in: Proceedings of the 18th Annual International Conference on Mobile Computing and Networking, ser. Mobicom'12, New York, NY, USA, ACM, 2012, pp. 317–328, http://dx.doi.org/10.1145/2348543.2348583 (Online), Available from:.
[10] Trepn profiler, qualcomm Developer Network, https://developer.qualcomm.com/mobile-development/increase-app-performance/trepn-profiler
[11] R.T. Fielding, Architectural styles and the design of network-based software architectures (Ph.D. dissertation), 2000.
[12] H. Chen, Y. Li, W. Shi, Fine-grained power management using process-level profiling, in: Sustainable Computing: Informatics and Systems, ser. SUSCOM, January, 2012.
[13] M. Dong, L. Zhong, Power modeling and optimization for OLED displays, IEEE Trans. Mobile Comput. 11 (2012) 1587–1599.
[14] Google, Wakelock, android power management, http://developer.android.com/reference/android/os/Pow-erManager.html
[15] C. Maia, L.M. Nogueira, L.M. Pinho, Evaluating android OS for embedded real-time systems, in: Operating Systems Platforms for Embedded Real-Time Applications, ser. OSPERT 10, July, 2010.
[16] S. Tilkov, A brief introduction to rest, 2007, December http://www.infoq.com/articles/rest-introduction
[17] Pxi platform, Industry leading, pc-based platform for test, measurement, and control, National Instruments, http://www.ni.com/pxi/
[18] Accumulated number of application and games in the android market, http://www.androlib.com/appstats.aspx
[19] BP Precision, b&K Precision Corp. Model 1785B, http://www.bkprecision.com/
[20] M. Gordon, L. Zhang, B. Tiwana, R. Dick, Z.M. Mao, L. Yang, PowerTutor: a power monitor for android-based mobile platforms, http://ziyang.eecs.umich.edu/projects/powertutor/
[21] A. Pathak, Y. Hu, M. Zhang, Where is the energy spent inside my app?: fine grained energy accounting on smartphones with Eprof, in: Proceedings of the 7th ACM European Conference on Computer Systems, ser. EuroSys'12, New York, NY, USA, ACM, 2012, pp. 29–42.
[22] H. Zeng, C.S. Ellis, A.R. Lebeck, A. Vahdat, Ecosystem: managing energy as a first class operating system resource, in: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS-X, New York, NY, USA, ACM, 2002, pp. 123–132, http://dx.doi.org/10.1145/605397.605411 (Online), Available from:.
[23] D.C. Snowdon, E. Le Sueur, S.M. Petters, G. Heiser, Koala: a platform for OS-level power management, in: Proceedings of the 4th ACM European Conference on Computer Systems, ser. EuroSys'09, New York, NY, USA, ACM, 2009, pp. 289–302, http://dx.doi.org/10.1145/1519065.1519097 (Online), Available from:.
[24] J. Flinn, M. Satyanarayanan, Energy-aware adaptation for mobile applications, in: Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles, ser. SOSP'99, New York, NY, USA, ACM, 1999, pp. 48–63, http://dx.doi.org/10.1145/319151.319155 (Online), Available from:.
[25] N. Vallina-Rodriguez, J. Crowcroft, ErdOS: achieving energy savings in mobile OS, in: Proceedings of the Sixth International Workshop on MobiArch, ser. MobiArch'11, New York, NY, USA, ACM, 2011, pp. 37–42, http://dx.doi.org/10.1145/1999916.199926 (Online), Available from:.
[26] S.M. Rumble, R. Stutsman, P. Levis, D. Mazières, N. Zeldovich, Apprehending joule thieves with cinder, in: Proceedings of the 1st ACM Workshop on Networking, Systems, and Applications for Mobile Handhelds, ser. MobiHeld'09, New York, NY, USA, ACM, 2009, pp. 49–54, http://dx.doi.org/10.1145/1592606.192618 (Online), Available from:.
[27] A. Roy, S.M. Rumble, R. Stutsman, P. Levis, D. Mazières, N. Zeldovich, Energy management in mobile devices with the cinder operating system, in: Proceedings of the Sixth Conference on Computer Systems, ser. EuroSys'11, New York, NY, USA, ACM, 2011, pp. 139–152, http://dx.doi.org/10.1145/1966445.1966459 (Online), Available from:.

[28] J. Flinn, M. Satyanarayanan, PowerScope: a tool for profiling the energy usage of mobile applications, in: Second IEEE Workshop on Mobile Computing Systems and Applications, WMCSA'99, February, 1999, pp. 2–10.
[29] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, K.W. Cameron, PowerPack: energy profiling and analysis of high-performance systems and applications, IEEE Trans. Parallel Distrib. Syst. 21 (5) (2010) 658–671.
[30] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, R. Wang, Modeling hard-disk power consumption, in: Proceedings of the 2nd USENIX Conference on File and Storage Technologies, FAST'03, Berkeley, CA, USA, USENIX Association, 2003, pp. 217–230.
[31] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, E. Ayguade, Decomposable and responsive power models for multicore processors using performance counters, in: Proceedings of the 24th ACM International Conference on Supercomputing, ACM Press, 2010, pp. 147–158.
[32] R. Fonseca, P. Dutta, P. Levis, I. Stoica, Quanto: tracking energy in networked embedded systems, in: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI'08, Berkeley, CA, USA, USENIX Association, 2008, pp. 323–338 (Online), Available from: http://dl.acm.org/citation.cfm?id=1855741.1855764
[33] M. Dong, L. Zhong, Self-constructive high-rate system energy modeling for battery-powered mobile systems, in: Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, ser. MobiSys'11, New York, NY, USA, ACM, 2011, pp. 335–348, http://dx.doi.org/10.1145/1999995.2000027 (Online), Available from:.
[34] C. Yoon, D. Kim, W. Jung, C. Kang, H. Cha, AppScope: application energy metering framework for android smartphone using kernel activity monitoring, in: USENIX Annual Technical Conference, ser. USENIX ATC'12, Boston, MA, USA, June, 2012.
[35] M.B. Kjaergaard, J. Langdal, T. Godsk, T. Toftkjaer, Demonstrating entracked a system for energy-efficient position tracking for mobile devices, in: Proceedings of the 12th ACM International Conference Adjunct Papers on Ubiquitous Computing, ser. Ubicomp'10 Adjunct, New York, NY, USA, ACM, 2010, pp. 367–368, http://dx.doi.org/10.1145/1864431.1864439 (Online), Available from:.
[36] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, MAUI: making smartphones last longer with code offload, in: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, ser. MobiSys'10, New York, NY, USA, ACM, 2010, pp. 49–62.
[37] R. Duan, M. Bi, C. Gniady, Exploring memory energy optimizations in smartphones, in: 2011 International Green Computing Conference and Workshops (IGCC), July, 2011, pp. 1–8.
[38] A. Oliner, A. Iyer, E. Lagerspetz, S. Tarkoma, I. Stoica, Carat: collaborative energy debugging, https://www.wattsupmeters.com



Youhuizi Li is currently a Ph.D. candidate at Wayne State University. She received her B.S. degree from Xidian University in 2010, and M.S. degree from Wayne State University in 2013, both in Computer Science. Her research interests include sustainable computing, mobile system power management, energy-friendly software design and optimization. She has co-authored a couple of journals and international conference papers.

Hui Chen is a Ph.D. candidate in computer science at Wayne State University. His major research interests include computer systems and sustainable computing. Currently, he is working on user-centric power management for mobile operating systems. He received his B.S. degree in school of Mechano-electronic engineering from Xidian University in 2004, and M.S. degree in computer science and engineering from Beijing Institute of Technology in 2008.



Weisong Shi is a professor of computer science at Wayne State University. He received his B.S. from Xidian University in 1995, and Ph.D. degree from the Chinese Academy of Sciences in 2000, both in Computer Engineering. His current research focuses on computer systems, sustainable computing, mobile and cloud computing, smart health. Dr. Shi has published more than 140 peer reviewed journal and conference papers. He has served the program chairs and technical program committee members of several international conferences. He is a recipient of the NSF CAREER award, one of 100 outstanding Ph.D. dissertations (China) in 2002, Career Development Chair Award of Wayne State University in 2009, and the "Best Paper Award" of ICWE'04, IPDPS'05, HPCChina'12 and IISWC'12.