Smartphone Energy Drain in the Wild: Analysis and Implications

Xiaomeng Chen
Ning Ding
Abhilash Jindal
Y. Charlie Hu
Maruti Gupta
Rath Vannithamby

TR-ECE-15-03
March 1, 2015

School of Electrical and Computer Engineering
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285

Smartphone Energy Drain in the Wild: Analysis and Implications

Xiaomeng Chen[†^] Ning Ding[†^] Abiliash Jindal[†^]
Y. Charlie Hu[†^] Maruti Gupta[⋆] Rath Vannithamby[⋆]
[†]Purdue University [^]Mobile Enerlytics [⋆]Intel Corporation

ABSTRACT

The limited battery life of modern smartphones remains a leading factor adversely affecting the mobile experience of millions of smartphone users. In order to extend batter life, it is critical to understand where and how is energy drain happening on users' phones under normal usage, for example, in a one-day cycle.

In this paper, we conduct the first extensive measurement and modeling of energy drain of 1520 smartphone in the wild. We make two primary contributions. First, we develop a hybrid power model that integrates utilization-based model and FSM-based model for different phone components with a novel technique that estimates the triggers for the FSM-based network power model based on network utilization. Second, through analyzing traces collected on 1520 Galaxy S3 and S4 devices in the wild, we present detailed analysis of where the CPU time and energy is spent across the 1520 devices, inside the 800 apps, as well as along several evolution dimensions, including hardware, Android, cellular, and app updates. Our findings of smartphone energy drain in the wild have significant implications to the various key players of the Android phone eco-system, including phone vendors (*e.g.,* Samsung), Android developers, app developers, and ultimately millions of smartphone users, towards the common goal of extending smartphone battery life and improving the user mobile experience.

1. INTRODUCTION

The smartphone market has been growing at a phenomenal rate. eMarketer [2] estimates the smartphone users worldwide surpassed the 1 billion mark in 2012 and will total 1.75 billion in 2014, and expects smartphone adoption to continue on a fast-paced trajectory through 2017.

Despite the phenomenal market penetration of smartphones, the user experience has been severely limited by the phone battery life. For example, a survey in May 2014 by research company GMI [7] of 1000 Britons shows 89% rated long battery life as an "important" factor when buying a new smartphone – long battery life rated higher than all the other features.

To improve the mobile experience of billions of smartphone users, it is critical to study ways of extending smartphone battery life. As the rechargeable battery technology has remained stagnant and seems unlikely to deliver the energy requirements, optimizing the energy drain is the more promising approach. Tackling this problem requires a thorough understanding of where energy drain is occurring inside the phone over the course of a typical battery recharge cycle, for example over a period of a day.

Understanding the energy drain of smartphones in normal users' daily life however cannot be easily done via controlled experiments in the lab, since the energy drain for a given device is affected by external conditions and user behavior in significant ways. First, the energy drain of wireless interfaces such as WiFi or cellular data, which accounts for a significant fraction of the total device energy drain, can be affected by the different signal strength the device experiences throughout the day [9]. Second, different users set very different device configurations (*e.g.,* notification and WiFi settings). Third, different users spend differing amounts of time each day on the phone. Fourth, different users install and play with different apps on their devices, which can have very different energy drain behavior. Fifth, even for the same app, different users can have very different usage patterns, resulting in different energy drain rates.

In this paper, we undertake to our knowledge the first effort towards performing such an energy analysis of smartphones in the wild. Conducting such a study faces two major challenges. First, to enable the measurement of hardware components and apps and services running on the phones in the wild, we need an accurate power model for smartphones that (1) does not rely on triggers that can only be collected by modifying the Android framework or the kernel of the phones, (2) does not rely on packet-level trace (*e.g.,* from tcpdump) which would require users to root their phones, violating the service plans for carriers such as AT&T and Verizon; and (3) can capture details on activities such as WiFi beaconing, cellular paging, and SOC suspension. We overcome this challenge by developing a hybrid model that combines utilization-based and FSM-based model for different phone components and estimates the triggers for the FSM-based network power model based on network utilization. The key insight is to strike a careful balance between the estimation error and granularity of the utilization logging which directly affects the estimation error and logging overhead.

A second challenge is to collect triggers for driving such a power model of real users' phones under the normal usage in their daily lives, while incurring low logging overhead. To this end, we developed a free app that performs logging and released it to Google Play so volunteers can download and contribute to the data collection.

We summarize our contributions and highlights of our findings as follows.

- We developed a hybrid power model for estimating energy drain of the hardware components and apps and services running on the smartphones in the wild. The model relies on light-weight logging that can be performed without modifying the Android framework or rooting the phones and hence is readily deployable.
- Through a carefully designed free app that performs low overhead logging, we collected all the triggers needed to drive our power model from 1520 Galaxy S3 and S4 devices, geographically distributed over 56 countries, covering a total of 49326 days and 199 mobile operators.
- Our in-depth analysis of the CPU usage time on the 1520 devices shows that: (1) During screen-on periods, the CPU on average,

is Idle for 80.0% of time, signifying that user activities do not use a lot of computational power. (2) On average, the total CPU busy time during screen-on and screen-off periods is 10.2% and 14.4% respectively, suggesting a significant portion of the total CPU busy time is spent during screen-off periods running apps and services in the background. (3) Out of the total CPU busy time (24.6%), background services which run on behalf of the apps account for about 6.2% (in absolute), which testifies to the uniqueness of the Android programming environment where much of the common tasks are abstracted and provided by the system as services which can be directly called by apps and hence simplifies app programming. (4) On average, the devices spent 45.5% of the time connected to WiFi, 32.2% of the time connected to Cellular, and 22.5% of the time disconnected from both, showing a significant period of time the user has no network connectivity.

- Our analysis of the energy drain across the 1520 devices shows that: (1) On average the SOC suspend state, cellular paging, WiFi beacon, and WiFi scanning account for a total of 27.1% of the daily energy drain, of which cellular paging is a significant energy hog using up 14.6%. (2) Additionally, background apps and services during screen-off contribute to 12.6% of the total energy. (3) Overall, on average a whopping 41.2% of the total energy drain in a day occurs during screen-off periods. (4) Out of the 59.8% energy incurred during screen-on periods, a little less than half, 27.4%, is spent on the screen. (5) The energy drain from active networking over cellular (LTE and 3G) and over WiFi are 11.8% and 1.5%, respectively, showing cellular drains significantly more energy than WiFi. Further, a significant portion of the energy drain, 10.6% out of 11.8% for 3G/LTE and 1.38% out of the 1.5% for WiFi, are tail energy.
- We further study the 800 apps running on the 1520 devices which show that (1) Background energy can be significant for apps: accounting for more than 50% of the total energy for 22.5% of the apps, with an average of 27.1% across all 800 apps. (2) Within the foreground app energy, screen energy is the largest portion, at 62.3% on average across the apps. (3) Within either foreground or background periods, CPU and GPU dominate networking energy – the average ratios are 2.7x and 2.8x, respectively, across the 800 apps. (4) We further studied the app energy drain by different Google Play app categories and gained much insights into energy bottlenecks of the apps in different categories.
- Finally, we analyzed the energy drain of the 1520 devices along several evolution dimensions, including hardware evolution, Android evolution, cellular evolution, and app evolution. Our analysis shows that (1) the average CPU time of S4/Jellybean devices is 8.1% longer than S3/Jellybean devices, which translates into 11.3% energy drain increase; (2) the average CPU time of S3 and S4 KitKat devices are 33.2% and 22.5% higher than the corresponding Jellybean S3 and S4 devices, which translates into only 9.7% and 8.0% energy increase. (3) popular apps such as Facebook and Chrome have many different versions running in the wild, and their average power draw during foreground runs fluctuates by up to 150.8%.

The detailed analysis and findings of the smartphone energy drain in the wild from our study have significant implications to the various key players of the Android phone eco-system, including phone vendors (*e.g.,* Samsung), Android developers, app developers, and ultimately millions of smartphone users, towards the common goal of extending smartphone battery life and improving user mobile experience.

The remaining paper is organized as follows. §2 presents the power model we developed for estimating the detailed energy drain of smartphones in the wild. §3 then presents the CPU time and energy analysis of the 1520 Galaxy S3 and S4 devices in the wild using the traces collected from these phones. §4 zooms into the apps and services running on the devices and analyzes their energy drain behavior, and §5 analyzes the energy drain of the 1520 devices along several evolution dimensions, including hardware, Android, cellular, and app evolutions. We discuss related work in §6 and conclude in §7.

2. POWER MODELING FOR PHONES IN THE WILD

The simplest way to measure the battery drain of a smartphone is to use a power meter [5]. However, such an approach suffers two drawbacks: (1) it cannot be used to measure phones in the wild; and (2) it cannot measure the energy drain of individual apps and services concurrently running on the phone since the power meter outputs the total power draw of the phone. To measure the battery drain of apps and services of smartphones in the wild, we developed a hybrid power model that requires no modifications to the Android framework or the kernel or rooting the phone.

2.1 Background

Power models for mobile devices in general and wireless components such as WiFi, 3G and LTE radios have been actively studied in recent years, and the proposed power models fall into two major categories.

The first category of power models known as utilization-based models for smartphones (*e.g.,* [21, 24]) are based on the intuitive assumption that the utilization of a hardware component (*e.g.,* NIC) corresponds to a certain power state and the change of utilization is what triggers the power state change of that component. Consequently, these models all use the utilization of a hardware component as the “trigger” in modeling power states and state transitions. Such models thus do not capture power behavior of modern wireless components that do not lead to active utilization such as the promotion and tail power behavior of 3G and LTE [19, 12], and thus can incur high modeling error.

The second category of power models capture the non-utilization-based power behavior of wireless components using finite state machines (FSMs), (*e.g.,* [8, 15, 18, 16, 17, 9] for WiFi and 3G and [12] for LTE. In a nut shell, the built-in state machine of the wireless radio, *e.g.,* the RRC states and transitions in LTE, is reverse-engineered and represented in the finite state machine which annotates each power state or transition with measured power draw and duration values. The triggers for the state transitions are either packet-level traces [19, 12] or networking system calls [17].

2.2 Challenges

The smartphone power model we need to enable measurement of energy drain of hardware components of and apps and services running on the smartphones in the wild needs to satisfy the following requirements:

- The model cannot rely on triggers (*e.g.,* system calls) that can only be collected by modifying the Android framework or the kernel of the phones.
- Similarly, the model should not rely on packet-level trace (*e.g.,* from tcpdump) which would require users to root their phones. Rooting the phones would void the phone’s warranty and violate the service plans for carriers such as AT&T and Verizon.
- Since we also want to measure the energy drain during screen-off periods, the model needs to incorporate behavior such as WiFi beaconing, cellular paging, and SOC suspension as they add up to significant energy drain over time.

2.3 Modeling Overview

Table 1: Summary of power model.

Hardware component power draw	Model trigger
CPU	frequency + utilization
GPU	frequency + utilization
Screen	brightness level
WiFi	FSM + signal strength
3G/LTE	FSM + signal strength
WiFi beacon	WiFi status
Cellular Paging	cellular status
SOC Suspension	constant

Hardware components modeled. We focus on Galaxy S3 and S4 phones in our measurement study due to their popularity, accounting for 11% of the total Android phone market share [1]. We determined the set of phone components to be modeled by measuring the maximal power draw of all the major components using microbenchmarks one at a time, while keeping the load on other components steady. For example, to gauge the GPU power, we kept the CPU at a fixed frequency, and ran the GPU benchmark app [4] that performs 2D rendering. Based on these initial power measurements, we selected the set of components showing significant power draw, as shown in Table 1. We further confirmed the components are largely independent – our model described below which assumes different components are independent and add up to the total power drain of the phone has an error less than 10%.

When actively used, GPS can drain a non-trivial amount of power. We do not model GPS in our study as logging GPS requires explicit user permission and we found many users feel reluctant to give out this permission for privacy concerns.

A hybrid model. To accurately capture the power behavior of all the identified components, we developed a hybrid utilization-based and FSM-based power model that satisfies the above model requirements yet achieves good modeling accuracy. In particular, we resort to utilization-based modeling to capture power behavior of CPU and GPU whose power draw depend on utilization, we use FSM-based modeling for wireless interfaces such as WiFi/3G/LTE, and we model WiFi beacon and cellular paging as constant power draws by averaging their power spikes over time, and finally we model SOC suspension power as a constant power draw.

In summary, the triggers for modeling all the components are shown in Table 1, and can be collected on unmodified user phones.

2.4 Modeling Details

Before we start, we distinguish two classes of programs consuming CPU time and energy on the smartphones: apps and system services. System services include kernel processes (with UID < 1000) and framework processes (with UID between 1000 and 9999) such as `LocationManagerService` which are exported from the Android framework for use by apps. Apps can be user-level programs or system apps such as `Calendar` and `Clock` and have UIDs > 10000, and an app can potentially invoke services exported by the Android framework or the kernel. In this paper, we simply refer to system services as *services.*

CPU. As specified in previous section, we used CPU microbenchmarks to obtain the relationship between the CPU power draw and CPU operating frequency and also devised a methodology for accounting for multiple cores running at different frequencies. Further details can be found in Appendix A.

Screen. To model the power draw of Galaxy S3/S4 which are both AMOLED screens, we derived a power model based on screen brightness and ignored screen content to reduce our overhead. Details can be found in Appendix A.

GPU. We developed a power model for GPU based on the different power states as well as accounting for the operating frequency during each state. Details can be found in Appendix A.

To use the GPU power model, we log the duration of each GPU frequency and state combination every 1 second, and predict the GPU power draw of each interval based on the GPU model.

SOC during suspension. When the CPU and other hardware components are offline, the entire SOC is suspended and draws a constant current. We turn the screen and WiFi off, set the phone in airplane mode; soon after the SOC is put in suspension by the Linux power manager, and we measure the SOC base power draw in this state. The measured constant power draw of the SOC suspended state are 3.8 mA and 5.1 mA for Galaxy S3 and S4, respectively.

WiFi Beacon. When the WiFi radio is associated with an AP and in power saving mode, the WiFi radio wakes up at fixed intervals to receive beacons from the AP. Each beacon thus results in a power spike of the WiF radio, and we noticed that the width and shape of the spikes are independent of the WiFi version, channel frequency, or the AP. We average the energy of 50 spikes over the duration of the spikes and then model the WiFi beacon power as constant current over time over the base SOC power – 1.1 mA for screen off and 3.3 mA for screen on for both Galaxy S3 and S4.

WiFi scanning. When WiFi is enabled on the phone, whether in connected or disconnected state, it performs scanning, *i.e.,* to try and connect to a suitable network. In associated state, we account for the energy draw using WiFi beacon process above. In disassociated state, the WiFi radio needs to search all possible channels until it finds one to connect to. For example, the 2.4 GHz band (802.11/bg) has 11 channels and the 5GHz (802.11ac) band has 22 channels and all may have to be searched. Galaxy S3 and S4 devices are capable of both bands, though we found most of WiFi hotspots seem to operate at 2.4GHz. Our measurements showed that WiFi scanning on these two phones has a duration of 3.4 seconds and average power draw of 64 mA. When scanning completes, Android generates an event, `android.net.wifi.SCAN_RESULTS` Thus we can find out how many scanning events happened by logging such events.

Cellular paging. On a celluar network, the base station periodically broadcasts a message during the 3G/LTE Idle state to signal incoming downlink data or voice call or SMS. This is called paging. The power meter shows paging results in a power spike on the phone modem every 1.28s, and this happens regardless whether the phone has a SIM or not, and with or without mobile data enabled. As with WiFi beacon, we average the spike energy over time and model the cellular paging as a constant current over time over the base SOC power – 8.3 mA on S3 and 2.3 mA on S4.

WiFi, 3G, LTE State Machine Models. WiFi, 3G, and LTE interfaces have multiple power states (see Appendix A) and the power draw and duration at the Active state is affected by the wireless signal strength [9]. Further, as in [11], we noticed signficant CPU power draw during pure data transfer workload, due to interrupt handling and TCP/IP stack processing, and therefore we need to carefully decouple CPU power draw from the wireless interface power draw in training the model. To develop signal-strength-aware power models for the wireless interfaces for our phone, we connected the phone to the power meter and ran data transfer microbenchmarks. While the power meter collects the power profile, we also recorded the packets via tcpdump alongside signal strength values via Android APIs as well as core frequencies and the CPU utilization. We varied the signal strength received by the phone by adjusting the distance between the phone and the AP for WiFi experiments and changing the location of the phone for 3G/LTE experiments. In post-processing, we synchronized the power profile from the power meter, tcpdump and signal strength traces. We derived the power draw by the radio interface(s) by subtracting the CPU power from the total power. We inferred the different power states of WiFi, 3G, LTE following the procedure in [12, 9] and derived the various parameters of the signal-strength-aware power state machine for each interface. Tables 10 in Appendix A shows the WiFi and LTE power draw for Galaxy S3 and S4 under different signal strength.

Estimating network events from network usage. The above power-state-machine models for wireless interfaces are driven by network events collected in a packet trace or a network system call trace. However, on an unmodified user phone, we can only collect network usage information periodically. Thus we need a way to convert network usage to network events. Specifically, our objective is to convert the number of bytes sent N_{snd} and received N_{rcv} logged at each logging interval T into a sequence of network send and receive system calls, each with a time t_i and the number of bytes sent n^i_{snd} or received n^i_{rcv}.

To decide the rules of convertion, we first found out the distribution, timing and message sizes in popular apps. We played top 20 apps in Google Play on a Galaxy S3 phone many times and logged the network system calls using systemtap which requires modifying the Android framework. We found the average message size is 600 bytes with a variance of 200 bytes for a send call and 6200 bytes with a variance of 2000 bytes for a receive call. The timings of receive system calls in each interval across different apps follow a randomly uniform distribution. Thus in our heuristic, we assume system call timings to follow a random uniformly distribution within each interval, and rely on controlling the logging interval T to be small enough so that the estimation error is acceptable. We further found in the apps we played that a receive call will always be preceded by a send call which agrees with the natural client-server communication model.

BAsed on the above observation, we convert the network usage (N^i_{snd}, N^i_{rcv}) within a interval T_i to a sequence of $K = \lceil N_{rcv}/6200 \rceil$ network system calls $\{(n^i_{snd}, n^i_{rcv}, t^i)\}$, where i = 1, ..., K, $n^i_{rcv} = 6200$ if $i < K$, $n^i_{rcv} = N_{rcv}$ mod 6200 if $i = K$, and $n^i_{snd} = N_{snd}/K$; $t^i = random(0, T)$.

The logging interval T determines the tradeoff between the estimation accuracy and the logging overhead. To determine the network logging interval T, we performed what-if analysis and caculate the network energy estimation error under different values of T against the network energy derived from the actual system call sequence we logged. We found T = 1 second gives an acceptable error of 5.1% during screen-on intervals and T = 5 second gives an acceptable error of 7.4% during screen-off periods. We therefore used these two parameters in the field pexeriment. In future work, we plan to improve the estimation accuracy, by customizing the parameters according to app categories and individual user behavior.

2.5 Logging App Design

We have designed a free android app called Anonymo [1] that when downloaded to a user's phone, performs logging of all the required information needed for driving our power model. All the information collected (summarized below) are anonymized before uploaded to our server.

In principle, the more fine-grained utilization information we collect, the more accurate the power model will be in estimating the energy drain of each app by the CPU, GPU, and wireless interfaces, but also the higher the logging overhead. In designing the app, we carefully chose logging intervals to strike a balance between these two objectives.

[1]The app's actual name is anonymized.

Coarse-grained logging (every 5 minutes). Coarse-grained logging happens every 5 minutes, where the app logs app-wise CPU usage (from `/proc/[pid]/stat`), and the per-core CPU usage (from `/proc/stat`) and the duration staying on different frequencies (from `/sys/devices/system/cpu/cpu[id]/`).

Fine-grained logging (every 1 or 5 seconds). Fine-grained logging happens every 5 seconds during screen-off when CPU is on and every 1 second during scree-on, where the app logs the network usage (in bytes) of all apps that had data transfer during the interval, by reading `/proc/uid_stat/[uid]/`.

Dynamic event logging. Finally, dynamic events are logged on demand. These include WiFi, mobile data, and screen being switched on and off, WiFi being associated and scanning, WiFi and cellular signal strength change, battery level change (1% granulairty), and every app's start and stop.

Logging overhead. We mesured three types of logging overhead of Anonymo, CPU time, network bytes and total energy. On average across the 1520 devices where we collected traces (details in §3), the average overhead of Anonymo per day are: (1) CPU time 214.7s, 2.4% of total. (2) Network bytes 190 KB, 0.3% of total. (3) Energy 7.3 mAh, 0.6% of total.

2.6 Model Validation

We experimentally validated our power model by measuring the model accuracy in both screen-on periods and screen-off periods on a small set of user devices that have Anonymo installed and running. The devices are connected to a Monsoon power meter, whose reading serve as ground truth. We note for our energy study energy estimation accuracy is much more relavent than instantaneous power estimation accuracy.

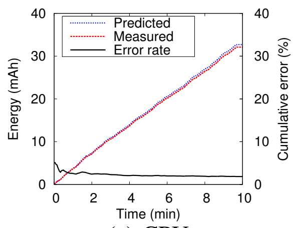
Component model validation. We first validate the component models for Galaxy S3 and S4 by generating synthetic workload on each component and comparing the predicted energy against the power meter reading. We run each experiment for 10 min and repeat 3 times, and show the cumulative energy over time by model prediction and by power meter, as well as the cumulative energy error rate. Due to space limit, we only shows the results for Galaxy S3; results for Galaxy S4 are similar hence are omitted.

Note that for WiFi, LTE and GPU we are comparing the component energy plus CPU energy since running benchmark requires the CPU to be on.

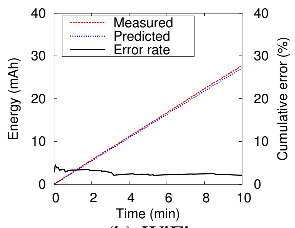
CPU: For CPU validation we turn off the screen, run worker processes on each core and set each core to random frequencies every 10s. We log the frequency and utilization of each core and apply the CPU model to predict the CPU energy consumption. Figure 1(a) shows that the cumulative energy error rate starts at 5.0% and smoothes to 1.9% for 10 min duration.

CPU+Network: For network we place the phone at a medium signal strength location, turn off the screen, and run a simple C socket program to continuously download from a local server for 10 min. To minimize the CPU power in downloading, we only enable one core and set it to lowest frequency, 384MHz. We log the network traffic and CPU frequency and utilization, and apply the network and CPU models to predict the total energy consumption for downloading. The total energy error rate is 1.9% for WiFi and 5.7% for LTE, as shown in Figure 1(b)(c).

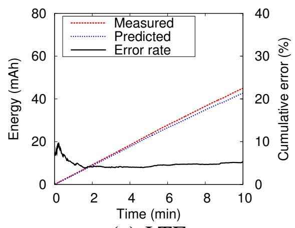
CPU+Screen+GPU: For GPU we leave the display on and run 2 GPU benchmark apps [3] for 10 min. We log the GPU frequency and state, CPU frequency and utilization, and screen brightness, and apply the GPU, CPU and screen models to predict the total energy consumption. The overall energy error rate is 1.2% for benchmark 1, and 11.0% for benchmark 2. The Figure 1(d) shows the result for benchmark 1.



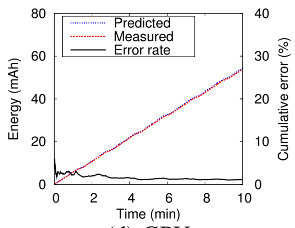
(a) CPU



(b) WiFi



(c) LTE



(d) GPU

Figure 1: Model validation for CPU, WiFi, LTE and GPU – energy drain over time by estimation and by the power meter, and the relative estimation error as a function of duration (right y-axis).

Table 2: Trace statistics.

Devices > 10 days trace	1520
Aggregate trace duration	49326 days
Median trace duration	34.0 days
Countries of origin	56
Mobile operators	191
Unique phone types	2
Rate of mobile RSSI reading	when signal changes, effective: 1/min
Rate of network usage reading	screen-on: every 1 second, screen-off: every 5 seconds

Whole phone energy estimation.

Screen-on: In screen-on tests, we installed 25 top apps on Google Play including 11 games, 7 online chat apps, 4 music apps and 3 news apps. The games are CPU-intensive and the rest are network-intensive. A normal user performed similar operations for the same type of apps, 2-3 minutes each, under WiFi and under LTE. Figure 2 shows the cumulative estimated energy drain over time closely matches that of the power meter output, with relative error converges to below 10% under WiFi and 10.3% under LTE beyond 20 min.

Screen-off: In screen-off tests, we installed the same 25 apps phone and logged in to the apps if necessary (*e.g.,* Facebook) using a normal user account. We then left the phone screen-off for 1 hour with either WiFi or LTE connectivity. At the end of each test, we compard the estimated energy from the power model and the power meter output to caculate the model accuracy. Figure 2 shows the cumulative estimated energy drain over time closely match that of the power meter output, with the relative error converges to below 4.1% under WiFi and 5.0% under LTE beyond 20 min.

3. CHARACTERIZING ENERGY DRAIN IN THE WILD

In this section, we first describe the trace collection and then present the energy analysis of the 1520 user phones in the wild using the power model developed in §2.

3.1 Trace Collection

We used the data collected through voluntary and anonymous contributions from users of the Anonymo app [2]. We collected traces from 670 Galaxy S3 and 850 Galaxy S4 devices worldwide. Each user trace ranges from 10 days to 2 months in length, with an average of 32.5 days (median 34.0 days). The detailed characteristics of the trace are shown in Table 2.

3.2 Trace Overview

We start with an overview of the general usage behavior of smartphone users. Figure 3 shows the distribution of the average daily

[2]Anonymo received exemption from the full requirement of 45 CFR 46 or 21 CFR 56 by the IRB of our organization, and was released Anonymo in Google Play on March 12,2014

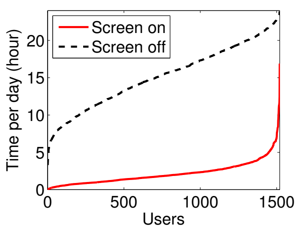


Figure 3: Distribution of total screen-off/screen-on time across all users.

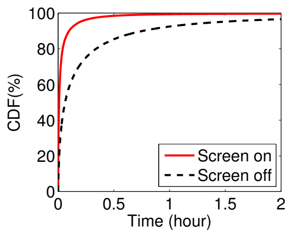


Figure 4: CDF of all screen-off/screen-on intervals for all users and all days.

screen-off and screen-on time across the 1520 users in the sorted order. The average, 10th percentile, and 90th percentile screen-on time are 129.5, 41.5, and 232.5 minutes, respectively. This is consistent with a recent study by Flurry [6], an app analytics firm, which found that users are spending 162 minutes per day on mobile devices, out of which mobile app usage accounts for 139 minutes.

Figure 4 shows the CDF of the duration of individual screen-off and screen-on intervals for all users across all days, truncated at 2 hours. We observe that as expected, the screen-off intervals tend to last much longer than screen-on intervals: the average, 10th percentile, and 90th percentile durations are 23.0 minutes, 15.0 seconds, and 45.6 minutes for screen-off intervals, and 5.2 minutes, 5.5 seconds, and 5.9 minutes for screen-on intervals, respectively.

Next we calculate the time spent by each device connected to different cellular technologies. We find that although it has been 5 years since LTE first started entering the commercial market, a majority of the devices, 8 and 960, respectively, could only connect to 2G and 3G from time to time, but majority of the network traffic, about 90%, are transmitted over 3G and LTE.

Finally, Table 3 shows the breakdown of Android versions on the 1520 phones. We see 47.8% of the phones ran Android 4.2 Jellybean and 52.2% of the phones ran Android 4.4 KitKat, and no phone in our trace ran the latest Android 5.0 Lollypop.

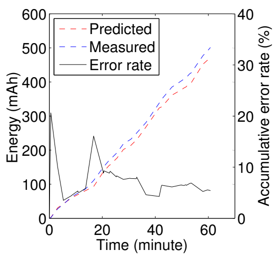
Table 3: Android version breakdown.

Android version	Percentage on S3	Percentage on S4
4.2 Jellybean	77.8%	24.0%
4.4 KitKat	22.2%	76.0%

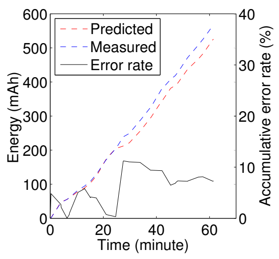
3.3 CPU Time Analysis

Since a primary source of energy drain is the CPU, before we break down the energy drain, we first study the CPU time break down. To help understand the CPU time breakdown, we first briefly explain how a device enters screen-off and screen-on periods.

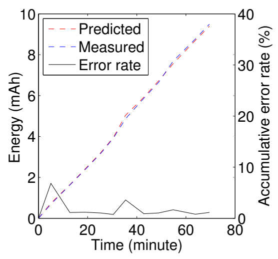
CPU time breakdown. The above discussion suggests that to see the complete picture of how the CPU time is spent, we need to break down the total CPU time, *e.g.,* in a day, into the following seven main components:



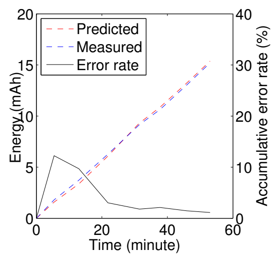
(a) Screen-on, using WiFi



(b) Screen-on, using cellular

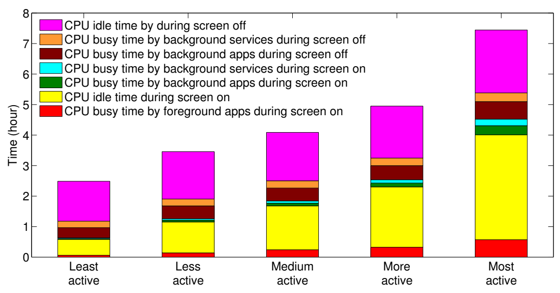


(c) Screen-off, under WiFi

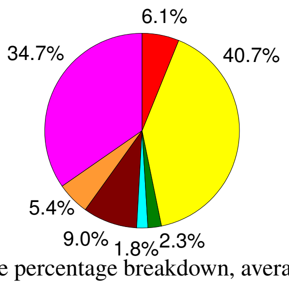


(d) Screen-on, using cellular

Figure 2: Battery drain estimation error during screen-on and screen-off. Shown are energy drain over time by estimation and by the power meter, and the relative estimation error as a function of duration (right y-axis).



(a) Average daily CPU time breakdown of 5 groups of the 1520 users.



(b) Daily CPU time percentage breakdown, average over all users.

Figure 5: Daily CPU time breakdown.

- CPU busy time by background services during screen-off;
- CPU busy time by background apps during screen-off;
- CPU busy time by background services during screen-on;
- CPU busy time by background apps during screen-on;
- CPU busy time for foreground apps during screen on;
- CPU idle time during screen-on;
- CPU idle time during screen-off.

Note the rest of the time in a day, where the CPU is neither busy nor idle, is when the CPU is in suspended state or the phone is powered off.

Figure 5(b) shows the average percentage breakdown of daily CPU time across all the users (the legend for different colors are in Figure 5(a)). We make the following observations about Figure 5(b). (1) **CPU idle:** Out of the total CPU time in a day, CPU is idle for 40.7% during screen-on and 34.7% during screen-off. In foreground, except for games that can be keeping CPU busy while the user is idle, most apps are not using the CPU when the user is not directly interacting with the app (*e.g.,* touch screen activities.) Thus the large idle CPU time suggests during screen-on periods, the users are idle for a significant portion of the time, *e.g.,* reading web pages, emails, or thinking. In screen-off periods, CPU idle time should ideally be close to zero as apps should keep the CPU on (*e.g.,* by holding wakelocks) only when they are actively computing something. However, we find that a huge portion of screen-off CPU time(49.1%) is spent idle(34.7%). (2) **Screen-on vs. screen-off:** On average, the total CPU busy time during screen-on and screen-off periods are 10.2% to 14.4%, suggesting a significant portion of the total CPU busy time is spent during screen-off periods running apps and services in the background. (3) **Services vs. apps:** Out of the total CPU busy time (25.6%), background services which run on behalf of the apps account for about 28.1% (7.2% in absolute). This testifies to the uniqueness of the Android programming environment where much of the common tasks are abstracted and provided by the system as services which can be directly called by apps and hence simplifies app programming. We note that during screen-on periods, we cannot easily infer how much of the background service CPU time is due to foreground apps and how much is due to background apps; inferring such causality would require changing the Android.

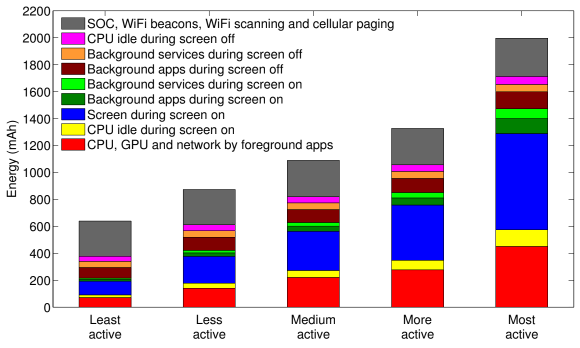
3.4 Energy Analysis

Energy breakdown by activities. We first break down the total energy per day per device among different activities as follows:

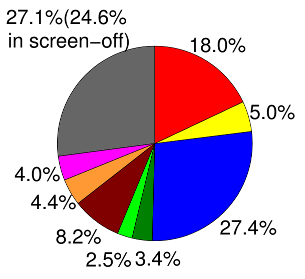
- Energy by WiFi beacon, WiFi scanning, cellular paging, and SOC base power during screen-off;
- Energy by background services and apps, respectively, during screen-off;
- Energy by background services and apps, respectively, during screen-on;
- CPU, GPU and network energy by foreground apps during screen-on during CPU busy and CPU idle, respectively;
- CPU idle energy during screen-on and screen-off respectively;
- Screen energy by foreground apps during screen-on.

Note each app and service energy component includes both CPU and networking energy. The reason we separate screen energy from other components is that they only happen for foreground apps during screen-on, and depend on non-app factors such as the brightness level.

For each device, we calculate the average daily energy of the 5 groups of users as before (on left), as well as the average energy percentage breakdown across all users (on right). We make the following observations about Figure 6(b). (1) **Overall screen-on vs. screen-off:** Overall, on average a whopping 41.2% of the total energy drain in a day occurs during screen-off periods. This is rather significant, and countering the expectation that when a phone is turned off and not used, it should consume little energy. (2) **Suspended state energy:** During screen-off periods, on average, the energy drain while the phone is suspended, *i.e.,* due to SOC

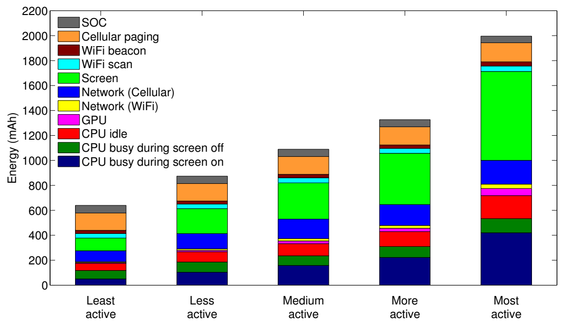


(a) Average daily energy drain breakdown of 5 groups of the 1520 users.

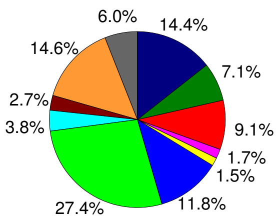


(b) Daily energy percentage breakdown, averaged over all users.

Figure 6: Daily energy breakdown by activities.



(a) Average daily energy drain breakdown of 5 groups of the 1520 users.



(b) Daily energy percentage breakdown, averaged over all users.

Figure 7: Daily energy breakdown by components.

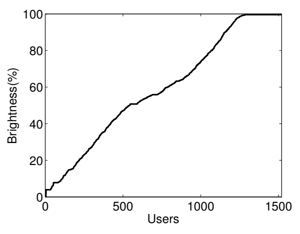


Figure 8: Distribution of average screen brightness level.

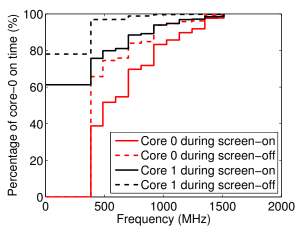


Figure 9: CDF of core-0 and core-1 time spent in different frequencies on S3.

base power, WiFi beacon, WiFi scanning and cellular paging activities, account for 24.6% of the total energy drain throughout a day. This is rather significant considering the energy drain does not contribute to any useful work. (3) **Screen energy:** Out of the 58.8% energy incurred during screen-on periods, a little less than half, 27.4% in absolute, is spent in screen energy. (4) **Useful energy in Screen-on vs. screen-off:** The background apps and services during screen-off together contribute to 12.6% of the total energy drain, in contrast to the 23.9% by background apps and services and foreground apps during screen-on. (5) **CPU idle energy:** Although on average the CPU spends 75.4% of the total CPU time in idle (Figure 5), it only drains on average 9.0% of the total energy. Two factors contributed to this contrast: (1) in entering the idle state, the CPU frequency quickly drops to the lowest possible, *e.g.,* 384MHz on Galaxy S3, and thus draws minimum power. (2) There are no networking activities during CPU idle while there can be networking activities during CPU busy which adds to the energy drain during CPU busy.

Energy breakdown by components. To dissect energy drain by different phone components, we replot in Figure 7(b) the average percentage breakdown of daily energy drain among all users, this time among phone components. We make the following observations. (1) **SOC:** On average the SOC accounts for 6.0% of the daily total energy drain. (2) **Cellular paging vs. WiFi beacon:** The energy drain from cellular paging and WiFi beacon are 14.6% and 2.7%, respectively, showing cellular paging is a significant energy hogger. (3) **Screen:** The screen energy accounts for 27.4% of the total phone energy. (4) **Cellular vs. WiFi:** The energy drain from active networking over cellular (LTE and 3G) and over WiFi are 11.8% and 1.5%, respectively, showing cellular drains significantly more energy than WiFi. (Their correlation with traffic volume is discussed in §3.5.) Furthermore, a significant portion of wireless interface energy drain are tail energy, 89.3% out of total cellular energy is 3G/LTE tail energy, and 90.4% out of the total WiFi energy is WiFi tail energy. (5) **CPU:** The busy CPU energy during screen-on executing services and apps is twice that during screen-off, at 14.4% and 7.1%, respectively. (6) **GPU:** Finally, the GPU accounts for 1.7% of the total energy drain, used by foreground apps during screen-on periods.

3.5 Component Analysis

In this section, we take a close look at the power behavior of some individual phone components to gain insight into their overall energy drain in the previous section.

Screen. To understand the 27.4% of the total energy drain by the screen of the devices on average, we plot in Figure 8 the distribution of the devices' screen brightness level during screen-on intervals. Since a single device can experience multiple brightness levels, for each device we calculate and plot the average brightness level, weighted by the duration spent in each level. We observe an almost uniform distribution, with the average and median levels being 59.0% and 58.3%, respectively. Interestingly, about 16.0% of the devices appear to stay at the 100% brightness level all the time.

CPU. To understand the discrepancy between the CPU busy time ratio of 10.2% to 14.4% and CPU busy energy ratio of 23.9% and 12.6%, between screen-on and screen-off periods, we plot in Figure 9 the distribution of core-0 and core-1 CPU time spent in different frequencies during screen-on or screen-off periods on the 670 S3 devices. Note the fraction of core-1 time in different states and frequencies is normalized to the total core-0 on time, in screen-on and in screen-off, respectively. We make several observations. (1) Core-0 is idle at the lowest 384 MHz for 38.8% of the time during screen-on but 65.7% of the time during screen-off. This high idle time in screen-off happens because often in screen-off background apps/services wake up and acquire some wakelocks and then wait for responses from remote severs during which time the CPU cannot sleep but to stay idle. (2) Core-1 is turned off during 61.3% of the core-0 on time during screen-on, but 78.0% of the core-0 on time during screen-off, *i.e.,* during screen-on the chance of core-1 running when core-0 is running is higher than during screen-off. (3) Both core-0 and core-1 tend to be busy at higher frequencies during screen-on than during screen-off periods. For example, core-0 stays on less than 486 and 1242 MHz for 51.7% and 91.9% of the time during screen-on, and 74.5% and 96.2% during screen-off periods, respectively. Similarly, core-1 stays on less than 486 and 1242 MHz for 18.6% and 35.8% of the time during screen-on, and 18.9% and 21.8% during screen-off periods, respectively.

GPU. The GPU active power and average screen power (at brightness level 50%) are comparable, yet the energy breakdown shows that the GPU drains only 1.7% while the screen drains 27.4%. This is mainly due the fact that most non-game apps use little GPU.

Networking. To understand the high energy drain ratio of cellular data over WiFi across the devices, we compare the time spent and bytes transmitted over the two types of wireless technologies. Figure 10(a) shows the distribution of time each device is spent in each of the three states: connected to WiFi, connected to mobile data, and connected to neither, across the 1520 devices. We observed an almost uniform distribution in terms of the percentage time spent in WiFi between 0% to 100%. Figure 10(b) then shows the average percentage breakdown of time in the three states across the devices: on average, the devices spent 45.5% of the time connected to WiFi, 32.2% of the time connected to mobile data, and 22.5% of the time disconnected from both. In all three states, WiFi rarely performed scanning, accounting for 1.5%, 2.0%, and 1.0% of the total time, respectively. In other words, WiFi did not perform much more scanning when connected compared to when disconnected. Figure 10(c) shows on average 63.6% of total traffic is transmitted over WiFi compared to 36.4% in cellular. Finally, Figure 10(d) shows on average each device spent 1.4x more time in WiFi and transmitting 1.8x more bytes in WiFi, but drains 4.2x less energy in WiFi, excluding WiFi scanning energy. We note this ratio is lower than the cellular to WiFi energy ratio of 7.6x (11.8% over 1.5%) in Figure 7(b). This is an artifact of averaging percentages; the percentage breakdowns per device in the two scenarios have different denominators, *i.e.,* total wireless energy drain versus total device energy drain. WiFi scanning, however, drains 37.2% of the total wireless energy, bringing the average total WiFi percentage energy drain (49.3%) to be almost the same as mobile data (50.7%) Finally, including WiFi beacon, WiFi scanning and cellular paging energy, the split between the total energy due to WiFi and due to cellular are 8.2% and 26.4% (Figure 7).

3.6 Light Users vs. Heavy Users

We have seen the screen-on time and hence usage level across the 1520 users in Figure 3 almost follows a uniform distribution except an exponential upward slope towards the end. To gain insight into the energy drain of users with different phone usage intensities, we divide up the users into 5 groups corresponding to the 5 20-percentiles in total screen-on time, and denote them as least active, less active, medium active, more active, and most active groups, respectively. We next compare the CPU time breakdown and energy breakdown of these 5 groups.

Figure 5(a) shows the average CPU time breakdown of users in each of the 5 groups. We observe that in moving from the least active group to the most active group, (1) the time spent in screen-on periods, in particular, the total as well as time in the 4 different categories, grow more or less proportional to the total screen-on time, but the time spent in screen-off periods barely increased. For example, the total average screen-on time for the most active group is 7.1x that of the least active group, the corresponding increase in the average CPU busy time by foreground apps and the average CPU idle time during screen-on are 9.2x and 5.7x, but the increase in the total CPU time in screen-off is only 1.6x. (2) As a result, the ratio of total CPU time during screen-off over screen-on goes down significantly, from 3.1x for the least active users to 0.7x for the most active users.

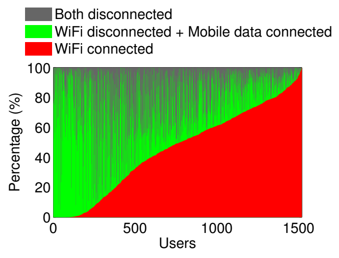
Figure 6(a) and Figure 7(a) shows the average energy breakdown of users in each of the 5 groups, by activities and by components, respectively. Here we see while the different portions of energy during screen-on grows more or less proportional to the screen-on CPU time as we move from the least to the most active groups, all the portions of screen-off energy by SOC, WiFi beacon, cellular paging, WiFi scan, and by the background apps and services during screen-off, remained largely constant, at 419.3mAh mAh. As a result, the relative energy drain during screen-off is quite different for different user groups – for the least active users, on average 65.1% of total device energy drain happens during screen-off, while for the most active users, on average only 24.2% of total device energy drain happens during screen-off. Similarly, the constant energy drain accounts for a higher percentage for less active users than for more active users. For example, the absolute SOC energy drain is a constant, and hence its percentage is high for light phone users, and low for heavy phone users – the percentage is 10.2% for the 20% least active devices, but only 2.4% for the 20% most active devices.

3.7 Implications to Phone Vendors

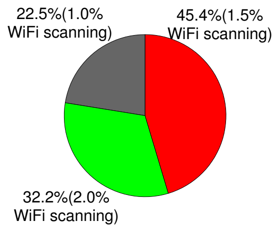
Table 4 summarizes the highlights from the above energy drain analysis of the 1520 devices in the wild, which have a number of implications to the phone vendors, SOC vendors, and cellular carriers. As can be seen from the data, the energy spent in performing no direct useful work such as SoC suspension, cellular paging, WiFi scanning receiving beacons *etc.* is significant at 27.1%. For the cellular network components, cellular designers need to develop better air-interface protocols that consume less energy during activities designed to maintain connectivity such as paging by coming up with more efficient ideas on managing an always-on connectivity. In addition, since WiFi is much more energy-efficient at transporting data, than cellular interfaces, a lot more work needs to go into better energy optimized cellular data transfer, some of it can include using WiFi more aggressively and others may be a more rigorous implementation of power management techniques such as DRX in the cellular domain. Given that the most energy consuming component in the smartphone is the screen at 27.4%, a context-aware algorithm to apply the right brightness level that can be applied for each user would save a lot of energy. Our data shows a great deal of variability in how power is consumed for each user, thus simple heuristics may not be enough, algorithms that take into account user context are essential. Another major area of optimization is the amount of energy spent in running background apps and services by gathering and utilizing user-context information.

4. APP ENERGY ANALYSIS

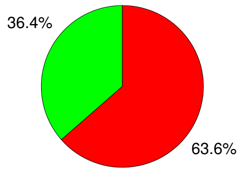
The whole-device energy analysis in the previous section shows



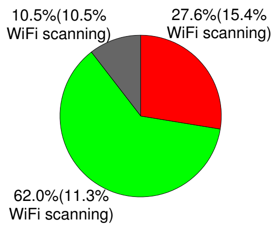
(a) Distribution of breakdown of time spent in WiFi and cellular states.



(b) Average percentage breakdown of time spent in WiFi and cellular.



(c) Average percentage breakdown of bytes transmitted in WiFi and cellular.



(d) Average percentage breakdown of energy drain by WiFi and cellular.

Figure 10: Time and traffic comparison between WiFi and cellular.

Table 4: Highlights of average percentage energy breakdown across the 1520 devices.

Component	Energy Drain
Cellular paging	14.6%
WiFi scanning	3.8%
SOC suspended	6.0%
Screen	27.4%
CPU idle energy (screen-on)	5.0%
CPU idle energy (screen-off)	4.0%
3G/4G energy	11.8%
3G/4G tail energy	10.62%
Apps/services in screen-off	12.6%

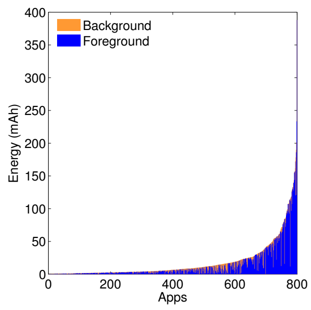
across the 1520 devices, on average 38.1% of the daily energy drain of a device is by apps and services running in the screen-on and screen-off periods. Since the rest of the energy drain such as SOC, WiFi beacon and scanning, cellular paging, and screen energy are largely fixed for a given hardware, in this section, we zoom into these apps and services running on the devices to study their energy drain behavior. Understanding app energy drain is important as it can keep users informed of power-hungry apps and give app developers hints on where to tighten app energy drain.

Our trace from the 1520 devices consists of a total of 800 apps with no less than 10 users and no less than 10 minutes total foreground time, which cover 67% of top 100 apps in Google Play. On average the top 5% most popular apps ran on 1009 devices for 23.7 minutes per day each, while the bottom 80% least popular apps ran on 20 devices for 56.0 minutes per day each.

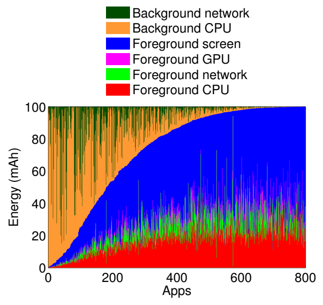
4.1 Energy Drain, Screen-on vs. Screen-off

Figure 11(a) shows the total daily energy drain of the 800 apps in the sorted order. We see that the energy drain follows an exponential distribution: the top 5% of apps drain an average of 133.6 mAh per day, while the bottom 80% drain an average of 6.8 mAh per day.

A common misconception is that apps only drain energy when running in the foreground during screen-on periods. Figure 11(a) further breaks down the total daily energy drain of each app into foreground and background energy drain. We see that the background energy drain fluctuates and there is no strong correlation between background energy and total energy drain of the apps. For a better view, Figure 11(b) shows the relative energy drain of different components by the apps when running in foreground and in background, sorted according to the increasing percentage of foreground energy percentages. We see that (1) background energy can be significant for many apps – the fraction of background energy is more than 50% for 22.5% of the apps, and the average fraction of background across the 1520 apps is 27.1%. (2) Within the foreground app energy, screen energy is the largest portion, at 62.3% on average out of its total energy; (3) Within either foreground or background energy, CPU and GPU energy dominates networking energy – the average ratios are 2.7x and 2.8x for foreground and background energy, respectively, across the 800 apps.



(a) Absolute breakdown



(b) Relative breakdown

Figure 11: Distribution of daily energy breakdown across apps.

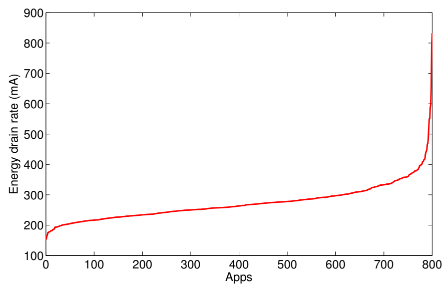


Figure 12: App energy drain rate.

4.2 App Energy Drain Rate

Since an app's total energy drain is the accumulation of its power draw over its runtime, the longer an app is played, the higher its energy. Thus the highest energy drain app may not be the most power-hungry app. To compare the power draw of the apps, we plot in Figure 12 the foreground energy drain rate (EDR), defined as the total foreground energy drain of an app divided by the total foreground time. We observe that over 92.6% of the apps have an average power draw between 200-400 mA. However, the last 4 apps, Speedtest.net, Deezer Music, BBC iPlayer, Kill Shot (0.5%) have average power draw between 600 mA and 832mA. The first three apps (one WiFi test app and two music apps) consume 66.8%, 46.5% and 49.7% of total energy on network respectively. On the other hand, the last app (a game) consumes 39.8% energy on CPU.

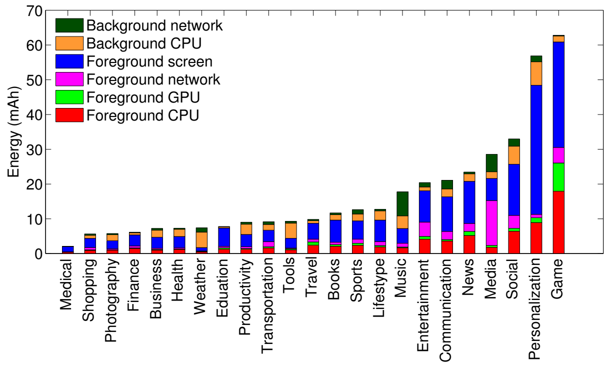


Figure 13: Average daily app energy drain by Google Play category.

4.3 App Categories

Next, we study the app energy drain by different app categories, where the categories follow from Google Play's classification.

Figure 13 shows the average foreground and background energy drain and the 6-way splits as before of the apps in each of the 23 app categories. We make the following observations. (1) **Total energy:** On average, apps in two app categories, Games and Personalization (*e.g.,* Candy Crush and Cover Lock Screen), drain far more energy each day than apps in the rest categories, *e.g.,* about 2.1x high than those in Social, Media and News categories, and about 9.7–8.2x high than those in Finance, Business, and Health categories. (2) **Background energy:** The fraction of background energy varies significantly across app categories and have little correlation with the total energy drain. For example, the Health, Weather, Education, Productivity categories have comparable EDR, but their average fractions of background energy, 32.5%, 76.9%, 5.3%, and 39.0%, are quite different. (3) **Screen energy:** The screen energy remains the dominating chunk of the total energy across all categories, ranging from 73.6% for Medical and 68.3% for Education to 13.0% for Weather and 22.3% for Media. (4) **GPU energy:** Game apps draw on average 12.9% of total energy on GPU, much higher than all other app categories, (5) **CPU energy:** The highest CPU energy draining categories are Game, Travel, and Finance, at 28.6%, 24.8%, and 24.1%, respectively. (6) **Network energy:** The top network energy draining app categories are Media and Music, where on average the network accounts for 62.8% and 45.1% of the total app energy drain.

5. EVOLUTION STUDY

In this section, we analyze the energy drain of the 1520 devices along several evolution dimensions, including hardware evolution, Android evolution, cellular evolution, and app evolution, and draw implications to the various key players of the Android phone ecosystem, including phone vendors (*e.g.,* Samsung), Android developers, app developers, and ultimately phone users.

5.1 Device Evolution: S3 vs. S4

Since both S3 and S4 devices could be running Android JellyBean and KitKat, to decouple of the impact of Android versions from device versions, we separately plot the daily CPU time breakdown and energy breakdown for four groups of devices: S3 devices running Jellybean (S3/JB), S3 devices running KitKat (S3/KK), S4 devices running Jellybean (S4/JB), and S4 devices running KitKat (S4/KK). The results are shown in Figure 14.

We first compare S3 devices with S4 devices. Figure 14(a) shows the average CPU time of S4/JB devices is 8.1% longer than S3/JB devices. The increase mainly comes from increased screen-on time (14.7%), within which CPU idle time increases by 72.3% and CPU busy time by foreground apps, background apps and background services shrink by 62.7%, 48.8% and 71.3% respectively.

The increase of screen-on time suggests users with S4 are more active. The decrease of CPU busy time during screen-on is mainly due to the faster CPU of S4 with more cores and higher maximum frequency and the CPU tends to turn on all the cores when there are tasks during screen-on period which we observe in the traces. On the other hand, during screen-off periods the CPU usually keeps one core on to process background tasks to save battery life, which explains the little change of background CPU busy time. We observed similar trend of changes between S3/KK and S4/KK devices.

Figure 14(b) shows that the 8.1% increase in total CPU time of S4/JB over S3/JB devices translates into 11.3% energy increase. In particular, during screen-on, the 72.3% CPU idle time increase translates to 15.1% energy increase, because the CPU power draw is minimal during CPU idle, but the 62.4% CPU busy time decrease translates to only 1.6% energy reduction, due to the faster CPU of S4 than S3; S3 has a dualcore 1.5GHz Krait CPU, and S4 has a quadcore 1.9GHz Krait 300 CPU. For the same amount of computation, more cores and higher frequency translate into less CPU time but also higher instantaneous CPU power, and the total CPU energy given by their product can be either higher or lower which determines the CPU energy efficiency.

To better understand the CPU efficiency of S3 and S4, we further break down foreground app energy into CPU, GPU and network in Table 5. If we assume users using different devices have similar behavior, the amount of computation for foreground apps can be measure by screen-on time. Figure 14(a) shows S4/JB devices have 14.7% more screen-on time than S3/JB, while the foreground app energy of S4/JB devices is 16.3% higher than S3/JB. This indicates the CPU of S3 is slightly more energy efficient than S4.

Finally, the SOC, WiFi scanning, WiFi beacon and cellular paging energy of S4 increases by 39.1% since S4 has a higher SOC suspended power.

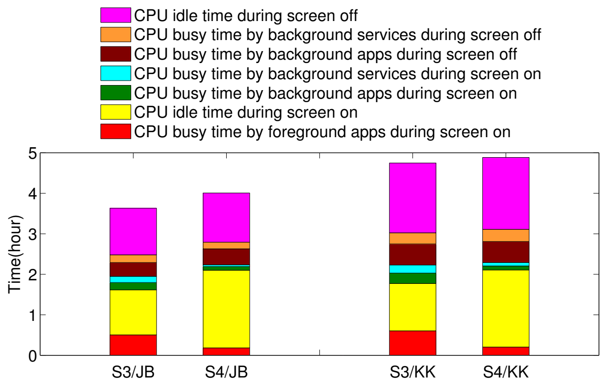
Table 5: Average daily CPU, GPU and network energy of foreground apps of the four groups of devices.

Devices	CPU (mAh)	GPU (mAh)	Network (mAh)
S3/JB	109.8	14.4	122.2
S4/JB	127.3	21.1	71.2
S3/KK	121.5	71.2	61.0
S4/KK	133.4	27.7	57.1

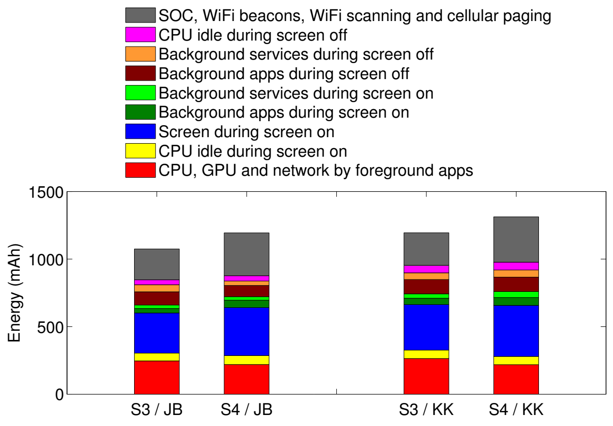
5.2 Android Evolution: Jellybean vs. KitKat

We next measure the impact of Android versions on the usage pattern and energy drain of the devices. Figure 14(a) shows the average CPU time of S3/KK and S4/KK devices are 33.2% and 22.5% higher than the corresponding S3/JB and S4/JB. The increase appears to be mainly coming from increased background CPU time during screen-off, 37.6% and 38.4%. To understand the reason for the increase, we calculated the average numbers of apps/services on the S3/JB, S4/JB, S3/KK and S4/KK devices which come out to be 67.2, 79.3, 79.0 and 86.4, respectively. Such higher numbers of apps/services on KitKat devices explain their higher background CPU busy time since more apps/services are likely to introduce more background tasks and traffic. It also indicates users with newer devices (*e.g.,* S4 compared to S3) tend to install more apps likely because of the perception of their faster CPU and larger storage.

The 33.2% and 22.5% increase in total CPU time of S3/KK and S4/KK over S3/JB and S4/KK devices translates into only 9.7%



(a) Total CPU time breakdown



(b) Total energy breakdown

Figure 14: Comparison of CPU time and energy breakdown among Galaxy S3, Galaxy S4, JellyBean and KitKat.

and 8.0% energy increase. This is because the main contribution of CPU time increase, background apps and services, (about 40%) only result in energy increase of 34.6% and 26.8% during screen-on and 3.4% and 37.8% during screen-off, respectively.

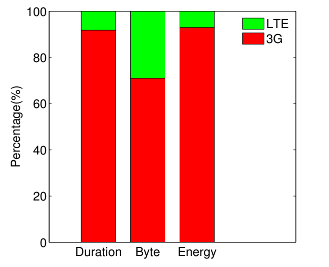
KitKat offer a new API that allows apps to specify networking (or background activities) to happen within a window so the scheduler may co-locate them and thus aggregate CPU wakeups as well as networking activities which in turn translates into few network tails, saving network energy. Since in screen-off periods, our CPU logging is infrequent and network logging is every 1 second, we use networking activity per interval to estimate whether the CPU woke up in the interval. We calculated the number of wakeups during screen-off periods per day and found on average KitKat devices have 68.2% fewer wakeups than Jellybean devices. We further observed the number of 3G/LTE tails during screen-off per day to be 64.7% lower on KitKat devices. Finally, the aggregated network activities leads to the lower average normalized network energy of KitKat devices of 0.71mAh/MB compared to 2.07mAh/MB of JellyBean devices.

Finally, KitKat brings significant performance improvements on GPU acceleration and make it easily accessible: any apps that use RenderScript on a supported device will benefit from GPU acceleration, without any code updates or recompiling, and RenderScript functionality can be accessed directly from native code. Table 5 shows the average GPU energy drain of S3/kk and S4/kk are 5x and 1.3x faster than those of of S3/JB and S4/JB, respectively.

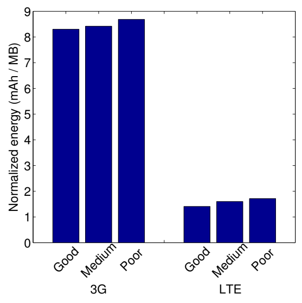
5.3 Cellular Evolution: 3G vs. LTE

We next study the impact of mobile data technology evolution on the energy drain of devices. The power model comparison of 3G and LTE in Appendix suggests that the tail time of 3G and LTE are similar while the tail power of LTE is much lower than that of 3G. Since a same device can be using 3G and LTE during different time of the day, we first compare the average daily energy drain (excluding paging), bytes transmitted, and duration during connectivity with 3G and LTE across the 1520 devices. Figure 15(a) shows that although LTE accounts for 29.0% of total bytes are transmitted, it only accounts for 8.7% of the time connected to 3G/LTE and only consumes 6.9% of the total 3G/LTE network energy.

Since the two technologies offer different data rates, we also compare the energy drain normalized by the traffic volume under the two technologies. Figure 15(b) shows on average 3G drains 5.9x, 5.3x, and 5.1x more energy per MB transmitted under good,



(a) Percentage energy, bytes, and duration



(b) Normalized energy by bytes

Figure 15: Traffic volume, duration and energy drain in 3G and LTE.

medium and poor signal strength, respectively, which are defined as -85dBm, -95dBm and -105dBm for 3G and -90dBm, -100dBm, and -110dBm for LTE.

5.4 App Evolution: App Updates

Finally, we study how the energy drain rate of apps change with app versions, *e.g.,* from different app updates. We pick 4 popular apps, Facebook, Dropbox, Gmail, and Chrome, and calculate the average app energy drain rate across their foreground runs on all the devices, as shown in Figure 16. We omit the background energy drain analysis due to page limit. We make the following observations. (1) Across the different versions of the 4 popular apps, the average power draw fluctuates by up to 150.8%, 70.9%, 33.1%, 99.6%, respectively. (2) Facebook has more than 120 versions used by 1520 users within the two-mouth trace period, with the foreground power ranging from 184.4mA to 462.6mA with an average of 307.3mA. The fluctuating and high foreground power happens mainly due to its frequently updated new features. (3) Dropbox and Gmail have less foreground power variation across their few versions (10 and 10, respectively) and a low average foreground power, 250.6mA and 254.9mA, respectively. This is mainly because these two apps usually synchronize with servers in background, minimizing foreground network energy. On the other hand, they both have simple UI which helps to lower the foreground CPU and GPU energy. (4) The Chrome foreground power has some repeating fluctuation among its earlier 9 versions but appears to sta-

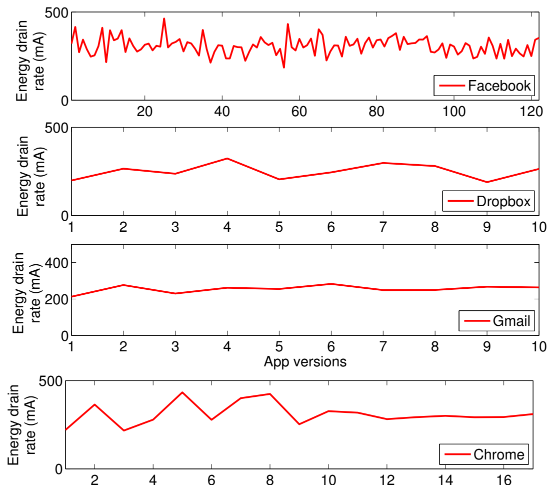


Figure 16: Comparison of average foreground energy drain rate of 4 popular apps over different versions.

bilize in the more recent 8 versions. Since it is difficult for browsers to prefetch network traffic, the foreground power of Chrome, 311.0mA on average, is higher than that of Dropbox and Gmail.

6. RELATED WORK

Power modeling of smartphones. We already discussed various previous work on power modeling of smartphones in §2.1.

Measurement study. There have been a number of work on measurement studies of smartphone apps and traffics. In [10], Falaki *et al.* characterize the smartphone traffic based on traffic trace collected from 43 users. In [14], Huang *et al.* study the 3G network performance using 3GTest data. In [23], Xu *et al.* study the smartphone usage pattern via network measurement from cellular network provider. AppInsight [20] monitors the performance of mobile apps in the wild by instrumenting app binary. In [22], Sommers and Barford study the WiFi and cellular performance using the Speedtest.net data. None of the work above however study the energy drain of mobile apps in the wild.

A few work study the energy consumption of mobile apps. In [13], Huang *et al.* collect traffic from 20 users and study screen-off radio energy consumption. In [21], Shye *et al.* derive a regression-based power model for HTC G1 phone which only has 2G EDGE and study component energy breakdown from a 20-user trace which does not include any app energy analysis. Different from these work, we collected trace from a much broader user base, developed a power model that captures both utilization-based and FSM-based components (for WiFi, 3G and LTE), and performed detailed activity and energy analysis across devices, components, apps, and technology and app evolutions.

7. CONCLUSIONS

In this paper, we undertook one of the first efforts in understanding where and how energy drains happens in smartphones running in the wild. We developed a hybrid utilization-based and FSM-based model that accurately estimates energy breakdown among activities and phone components without changing the Android framework or rooting the phone. Our analysis of traces collected on 1520 Galaxy S3 and S4 devices in the wild covering 800 apps gained much insight on energy drain across devices (users), device components, apps, and multiple technology and app evolutions. These insights in turn allow us to draw implications to the phone vendors, SOC vendors, cellular carriers, and app developers on better system, network, and app design to extend battery life.

8. REFERENCES

[1] AppBrain, top android phones. www.appbrain.com/stats/top-android-phones.
[2] eMarketer, smartphone users worldwide will total 1.75 billion in 2014. www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-.
[3] Gpu benchmark 3d. play.google.com/store/apps/details?id=com.kortenoeverdev.GPUbench.
[4] GPU benchmark app. play.google.com/store/apps/details?id=name.duzenko.farfaraway.
[5] Monsoon power monitor. www.msoon.com/LabEquipment/PowerMonitor/.
[6] TechCrunch, mobile app usage increases in 2014, as mobile web surfing declines.
[7] theGuardian, your smartphone's best app? battery life, say 89% of britons. www.theguardian.com/technology/2014/may/21/your-smartphones-best-app-battery-life-say-89.
[8] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proc of IMC*, 2009.
[9] N. Ding, *et al.* Characterizing and modeling the impact of wireless signal strength on smartphone battery drain. In *SIGMETRICS*, 2013.
[10] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A first look at traffic on smartphones. In *Proc. of IMC*, 2010.
[11] A. Garcia-Saavedra, P. Serrano, A. Banchs, and G. Bianchi. Energy consumption anatomy of 802.11 devices and its implication on modeling and design. In *CoNEXT*, 2012.
[12] J. Huang, *et al.* A close examination of performance and power characteristics of 4g lte networks. In *Proc. of Mobisys*, 2012.
[13] J. Huang, F. Qian, Z .M. Mao, S. Sen, and O. Spatscheck. Screen-off traffic characterization and optimization in 3g/4g networks. In *IMC*, 2012.
[14] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing application performance difference on smartphones. In *Proc. of Mobisys*, 2010.
[15] C.-Y. Li, *et al.* Energy-based rate adaptation for 802.11n. In *Proc. of ACM MobiCom*, 2012.
[16] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *Proc. of ACM MobiCom*, 2012.
[17] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app? fine grained energy accounting on smartphones with eprof. In *Proc. of EuroSys*, 2012.
[18] F. Qian, *et al.* Profiling resource usage for mobile applications: a cross-layer approach. In *Proc. of Mobisys*, 2011.
[19] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3g networks. In *Proc. of IMC*, 2010.
[20] L. Ravindranath, *et al.* S. Shayandeh. Appinsight: Mobile app performance monitoring in the wild. In *OSDI*, 2012.
[21] A. Shye, B. Scholbrock, and G. Memik. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In *MICRO*, 2009.
[22] J. Sommers and P. Barford. Cell vs. wifi: On the performance of metro area mobile connections. In *IMC*, 2012.
[23] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman. Identifying diverse usage behaviors of smartphone apps. In *Proc. of IMC*, 2011.
[24] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. Mao, and L. Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *Proc. of CODES+ISSS*, 2010.
[25] Y. Zhang, X. Wang, X. Liu, Y. Liu, L. Zhuang, and F. Zhao. Towards better cpu power management on multicore smartphones. In *Proc. of HotPower*, 2013.

Appendix A: Power Model Details

A.1 Power Models for Galaxy S3 and S4

We present the detailed power model of various phone components, derived by following the procedures described in §2.4.

CPU Power Model.. In training the power model for Galaxy S3's dual-core CPU, we first ran CPU microbenchmarks while using the power meter to measure the power draw of the CPU under different frequencies with only core-0 turned on. We then repeated the process with both cores turned on [3]. Table 6 shows the CPU power draw at 100% CPU utilization for Galaxy S3 under a range of frequencies. Single-core results are shown with core-1 turned off.

Table 6: Dual-core CPU power model for Galaxy S3, shown for 6 sample frequencies per core. The unit of CPU power is mW.

Core 0 (MHz)	Core 1 (MHz)						
	0	384	594	810	1026	1242	1512
384	296	744	766	818	873	977	1047
594	359	766	814	866	921	1036	1103
810	411	818	866	918	973	1080	1154
1026	455	873	921	977	1029	1136	1217
1242	555	981	1029	1084	1140	1199	1277
1512	633	1062	1106	1158	1221	1273	1351

In modeling the quad-core CPU on Galaxy S4, we follow the procedure in [25]. The power draw of the quad-core CPU is modeled as $P_{CPU} = P_{B,N_c} + \sum_i^{N_c} u_i \cdot P_\Delta(f_i)$, where P_{B,N_c} is the baseline CPU power with N_c enabled cores, $P_\Delta(f_i)$ is the power increment of core i at frequency f_i, and u_i is its utilization.

We first model the core-0 using the same method as with Galaxy S3. We then vary the number of cores online, but fix all online cores to the same frequency and 100% utilization. The increased power when turning on core-i is considered as the busy power for core-i at this frequency. Then we vary the frequency and repeat the process to obtain the busy power for each core at each frequency. For idle power, the procedure is the same except we keep the online cores idle instead of 100% busy. Table 7 shows the CPU power draw at 100% CPU utilization for Galaxy S4 under a range of frequencies with varying number of online cores.

Table 7: Galaxy S4 CPU power model for 3 sample frequencies with varying number of online cores. The power unit is mW.

	384MHz		1026 MHz		1890 MHz	
	P_{B,N_c}	$P_\Delta(f_i)$	P_{B,N_c}	$P_\Delta(f_i)$	P_{B,N_c}	$P_\Delta(f_i)$
1	86	207	86	438	86	1358
2	269	70	363	228	647	811
3	351	72	464	239	917	891
4	472	75	577	243	1205	962

To use the CPU model, we logged the frequencies of the each core as well as the utilization of each app active during the logging interval. In post-processing, we estimated the energy of each app over each interval based on the logged CPU frequency and the app's utilization, *i.e.,* as the power draw at that frequency under 100% utilization weighted by the app's actual utilization. Finally, we summed up the CPU energy consumed by that app in each interval to arrive at the total CPU energy consumption for the app.

Screen Power Model. Galaxy S3 and S4 phones have AMOLED screens, and thus in principle the screen power model should have two triggers: the brightness, and the content displayed on the screen. However, logging the content will impose unacceptable performance overhead. Further, we compared screen power of 10 popular apps and games under typical brightness settings and found the screen power differ by less than 18.5% for different displayed contents. This is much lower than the 45.5% to 77.0% power draw difference between lowest and highest brightness levels (fixing the displayed content). For these two reasons, we strike a balance between model accuracy and logging overhead by deriving a screen model solely based on the brightness using the following method: we use a set of wallpapers with various color tones, ranging from the darkest (pure black) to the brightest (pure white), and for each wallpaper we measure the screen power draw under each brightness level. Finally, for each brightness level, we use the average power draw across all wallpapers as the screen power under this brightness value.

Table 8 shows the screen power draw for Galaxy S3 and S4 for 6 sample brightness levels. S4 draws more screen power than S3 due to its larger screen size and high resolution.

Table 8: Galaxy S3 and S4 screen power for 6 sample brightness levels.

Brightness	0	51	102	153	204	255
Power on S3 (mW)	417	452	484	511	542	573
Power on S4 (mW)	507	562	616	671	725	780

GPU Power Model. The GPUs on both Galaxy S3 and S4 has three power states: Active, Nap and Idle, and can be in four different frequencies. Thus the GPU power draw differs in different power state and frequency combination. In GPU power modeling, we run GPU microbenchmarks to generate workload and in the meanwhile measure the power draw using the power meter. The measured power consists of three parts: CPU power, GPU power and screen power. Hence we log the frequency and utilization of CPUs, the frequency and state of GPU, as well as the brightness of the screen. In post-processing, we first isolate the power draw of GPU by subtracting the CPU and screen power (calculated by the CPU and screen power models) from the total power, and then calculate the average GPU power draw under each frequency and state combination to obtain the GPU power model. Table 9 shows the GPU power draw for Galaxy S3 and S4 under each frequency and state. The power of Idle state is always 0 hence not shown.

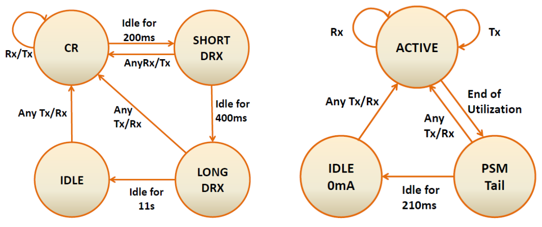
To use the GPU power model, we log the duration of each GPU frequency and state combination every 1 second, and predict the GPU power draw of each interval based on the GPU model.

Table 9: Galaxy S3 and S4 GPU power model.

Galaxy S3				
Frequency (MHz)	128	200	300	400
Active power (mA)	729	975	1217	1482
Nap power (mA)	78	0	0	78
Galaxy S4				
Frequency (MHz)	128	200	320	450
Active power (mW)	293	398	562	1034
Nap power (mW)	0	0	0	164

LTE, 3G, and WiFi Power Models. The LTE interface on smartphones has four power states. The power states and their transitions are shown in Figure 17(a): (1) *IDLE:* The interface is in idle states when the User Equipment (UE) does not send or receive any data. The interface consumes little power under the IDLE state, and periodically wakes up to check whether there are incoming data buffered at the network. (2) *CR:* When the UE sends or receives any data, the interface enters the Continuous Reception (CR) state and consumes high power. (3) *Short DRX:* After the UE finishes data transfer and becomes idle for 200ms, the interface will enter the Short DRX state, during which the interface consumes little power but wakes up frequently to check for incoming traffic. (3) *Long DRX:* The interface enters the Long DRX state after staying in Short DRX for 400ms without receiving any data. Long DRX

[3]In running the benchmarks, we kept the screen display (a wallpaper) static with a fixed brightness level and subtracted the constant screen power which is easy to measure separately.



(a) LTE (b) WiFi

Figure 17: WiFi and LTE state machines for Galaxy S3 and S4.

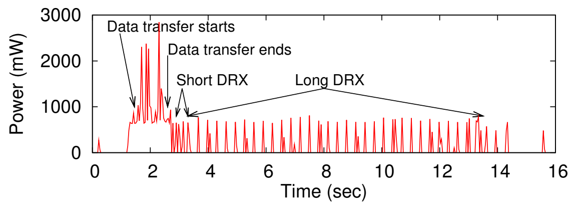


Figure 18: LTE Power states on Galaxy S3

is similar to Short DRX except that the wakeup interval becomes longer. Note in the power model in Table 10 we refer the periodical spikes during Short DRX and Long DRX state as Short DRX and Long DRX, respectively, and refer the low base periods between spikes as LTE tail base, as shown in Figure 18. Finally, if the UE stays in Long DRX for 11s without receiving any data, the interface will return to the IDLE state; otherwise, any data sending or receiving in Short DRX or Long DRX states will trigger the interface to enter the CR state.

The 3G interface has three RRC states: IDLE, FACH and DCH, as well as transition states between RRC states. We refer readers to [19] and [18] for a comprehensive discussion about the 3G states and transition rules.

The WiFi interface also has four power states: Tx, Rx, Tail, and Idle, as shown in Figure 17(b). The interface is in the Idle state when there is no traffic, and will enter the Tx (Rx) state when it starts sending (receiving) data. After data transfer, the interface will stay in the Tail state for 210ms before it returns to the Idle state. The interface consumes very little power in the Idle state, moderate power in the Tail state, and high power in the Tx and Rx states.

For both LTE and WiFi interfaces, the power draw and duration at each state and state transitions are affected by the wireless signal strength [9]. We followed the procedure described in §2.4 to derive all the parameters in the power state machines for the WiFi and LTE interfaces for Galaxy S3 and S4 phones. The parameters are shown in Table 10. We note the Tx/Rx power generally increases with weaker signal strength, but can decrease when the signal strength is extremely weak, at which point the throughput is significantly lower. Finally, Figure 18 plots the LTE power states on Galaxy S3 in a 100KB download under good signal strength (-90dBm).

We note the power models for these two phones differ from that in [12] conducted about three years ago. For example, while the LTE base has a similar duration of 11s, the tail base power is down from about 1000mW on the HTC phone used by [12] to zero in our measurement. An author of [12] has confirmed with us that they now also observe close to zero tail base power in their LTE measurements. Thus this difference from [12] is likely due to the newer LTE deployment.

A.2 Comparing S3 and S4 Power Models

We make the following observations in comparing the power models for S3 and S4:

Table 10: Parameters of signal-strength-aware power models for WiFi and LTE on Galaxy S3 and S4. The power unit is mW.

WiFi						
RSSI (dBm)	Galaxy S3			Galaxy S4		
	Tx	Rx	Tail	Tx	Rx	Tail
-50	564	396	242	654	451	289
-60	596	422	242	723	528	289
-70	641	431	242	1019	592	289
-80	704	400	242	1113	633	289
-85	702	382	242	892	514	289
The duration of WiFi tail for both phones is 210ms.						

Galaxy S3 3G			
	promotion	DCH tail	FACH tail
-85	836mW, 1.6s	783mW, 3.3s	486mW, 6.7s
-95	836mW, 1.6s	1034mW, 3.3s	486mW, 6.7s
-105	836mW, 1.6s	1224mW, 3.3s	486mW, 6.7s

Galaxy S4 3G			
	promotion	DCH tail	FACH tail
-85	647mW, 2.1s	577mW, 3.3s	332mW, 1.7s
-95	663mW, 2.1s	679mW, 3.3s	390mW, 1.7s
-105	807mW, 2.2s	722mW, 3.3s	390mW, 1.7s

	Galaxy S3 3G		Galaxy S4 3G	
RSSI (dBm)	Tx (mW)	Rx (mW)	Tx (mW)	Rx (mW)
-85	1414	1300	667	843
-95	1737	1718	835	1043
-105	2280	2060	1772	1545

Galaxy S3 LTE			
	Power(mW)	Duration(ms)	Periodicity(ms)
LTE promotion	1200	200	N/A
Short DRX	788	41	100
Long DRX	788	45	320
LTE tail base	61	11000	N/A
DRX in IDLE	570	32	1280

Galaxy S4 LTE			
	Power(mW)	Duration(ms)	Periodicity(ms)
LTE promotion	1326	200	N/A
Short DRX	N/A	N/A	N/A
Long DRX	585	30	320
LTE tail base	69	11000	N/A
DRX in IDLE	452	24	1280

	S3 LTE		S4 LTE	
RSRP (dBm)	Tx (mW)	Rx (mW)	Tx (mW)	Rx (mW)
-85	1218	1085	1177	938
-95	1683	1264	1849	1110
-105	1840	1271	1699	1140

(1) **Screen:** Due to the larger size and higher resolution, the screen of Galaxy S4 consumes high power compared to S3. The difference monotonically increases from 21.6% under the lowest brightness to 36.1% under the highest brightness.

(2) **SOC:** Galaxy S4 SOC suspension power is 34.2% higher than S3 in airplane mode, and 79.4% higher with LTE enabled. This directly translates into higher power consumption when the phone is in standby mode.

(3) **CPU:** Galaxy S4 has a 1.9GHz quad-core Krait 300 CPU, while S3 has a 1.5GHz dual-core Krait CPU. More cores and higher frequency on S4 brings better performance, but also higher power consumption. For example, when both phones have two cores working at the highest frequency, the CPU power of S4 is 68.0% higher than that of S3.

(4) **GPU:** The Adreno 320 GPU used on S4 is much more power efficient compared to Adreno 225 on S3: it consumes 59.8%, 30.2% less power under lowest and highest frequency, respectively.