Zephyr: Simple, Ready-to-use Software-based Power Evaluation for Background Sensing Smartphone Applications

K. Shankari
Jonathan Fürst
Yawen Wang
Philippe Bonnet
David E. Culler
Randy H. Katz

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-168
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-168.html

December 13, 2018

Acknowledgement

This work is supported by the National Science Foundation, under grant CPS-1239552 (SDB).

Zephyr: Simple, Ready-to-use Software-based Power Evaluation for Background Sensing Smartphone Applications

K. Shankari Jonathan Fürst Yawen Wang Philippe Bonnet
David E. Culler Randy H. Katz

December 13, 2018

Abstract

Innovations in mobile hardware and software need corresponding advances in the accurate assessment of power consumption under realistic conditions. This is especially relevant for smartphone-based background sensing applications. Assessing the power consumption of such applications requires ease of use, deployment *in situ* and well-understood error characteristics.

Existing measurement methods, based on external power meters or power models, are increasingly unable to keep up with these requirements. External power meters require access to device batteries and do not capture context-sensitive power drain. Power models must be rebuilt for each specific device, adapted to each new OS version, and require administrator access to instrument fine-grained system-level APIs. These limitations impede the inclusion of accurate, universal evaluations in the research literature.

We propose a simple and portable alternative, Zephyr, which infers an application's power drain using the relative State of Charge change rate (SoCCR) via the phone's battery sensor. We validate our methodology through experiments that characterize SoCCR on Android and iOS devices and show that they are consistent with hardware readings, across identical phones, for the same phone over time and over both slowly and quickly varying workloads.

The Zephyr implementation is modular, open source, and available for Android and iOS today.

This work is supported by the National Science Foundation, under grant CPS-1239552 (SDB). Author's addresses: K. Shankari, D. Culler, and R. Katz, Computer Science Division, University of California Berkeley; J. Fürst, and P. Bonnet, IT University of Copenhagen; Y. Wang, Cornell University.

1 Introduction

Consider a researcher working in ubiquitous computing today. She has a novel application of background-sensed data from smartphones that she wants to evaluate for publication. She is aware that there are power/accuracy trade-offs inherent in inferring meaning from sensed data, and wants to include them in her evaluation. She looks at the literature to see what prior ubiquitous computing projects have used, and discovers that none of the techniques work any more.

She cannot use an external power meter, because the Nexus 6 used for testing does not have a removable battery. When she does find an older Galaxy Nexus on eBay, it cannot be upgraded to the latest version of Android. She cannot use the PowerTutor modeling app [16] from the Play Store, because it only gives accurate numbers for the G1, G2 and Nexus One phones. She tries it anyway and discovers that on Android 7, it only displays the LCD power (Figure 1). She does find a few papers that use battery run-down tests, which are simple, but she is not sure they will be accepted by the research community as sufficiently accurate. Since the power measurement methodology is not the focus of the paper in which the studies appear, they do not define a clear approach that she can re-use. When she complains to her friend who has just completed an iOS-based application, he responds that his problem is more severe, since he has not been able to find a single evaluation using iOS in the literature! There is a power evaluation problem for smartphone based systems. In this paper, we characterize this problem and propose a solution.

1.1 Power Evaluation Methods

There are three classes of methods for evaluating power consumption on smartphones, today, based on (i) models, (ii) external power meters or (iii) battery rundown. We briefly review them.

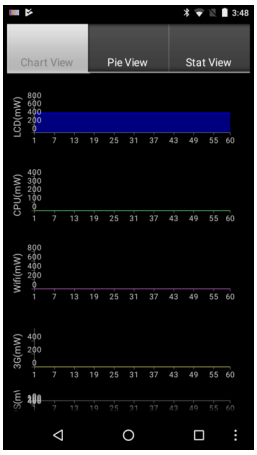


Figure 1: Power tutor on Android 7 with no readings

1.1.1 Model-Based

Typically, a model estimates the power consumption of different components. A trace captures system usage. Based on the trace, it is possible to infer how the components are used and thus derive power consumption.

To build an accurate model, we need, at a minimum, a list of all the components and the set of power states for each component. This list is already challenging to compile for the wide variety of existing phones. Constant maintenance is required to keep track of the power profile of evolving components.

To collect a trace of system usage we need to determine how applications and Operating System (OS) features use phone components. New OS features like sensor batching and sensor fusion [15] make it challenging to infer component usage from application behavior. For example, consider an application running on Android 6+ that senses location every 10 seconds. The fused location API automatically chooses between GPS, WiFi and cell tower sources to determine the location. Therefore, we cannot assume that the GPS is invoked every 10 seconds. Sensor batching ensures that although we may request updates every 10 seconds, the OS may choose to return data faster or slower depending on the context[1].

Systems such as PowerForecaster/PADA[9, 36] overcome this limitation by using `/proc` nodes to trace, rather than infer, component access traces under various scenarios. While this reduces the predictive power of the model—the power consumption can only be estimated for scenarios for which traces are available—emulators can be used to quickly estimate power consumption for the entire set of collected traces. However, increasing concerns about side channel attacks have resulted in Android 7+ restricting access to both `/sys` and `/proc` nodes [2]. This restriction is explicitly intended to remove the ability of regular applications to sense the behavior of other applications (e.g. "`/proc` access can be used to monitor applications launching or enabling phishing attacks"). This change already broke compatibility with most existing model-based solutions (e.g., the popular PowerTutor, see Figure 1). On iOS, similar sandboxing has been in place since iOS 2.0[3] [4], released in 2008 [5]. Even if researchers in the field discover a loophole that again allows fine-grained observation of system behavior, it is likely that it will be closed.

1.1.2 External Power Meter

The external power meter typically intercepts the smartphone battery terminals and measures instantaneous voltage and current. The power consumption of a state (e.g., when an application is running or a component is exercised) is computed by subtracting the baseline power consumption of the system from the power consumption while in the state. This approach is frequently used to build power models for hardware components.

Since this method relies on external instruments, such as an oscilloscope, the accuracy of the observations can be independently established. Also, the external instrument is independently powered and transparent to the phone hardware. As a result, the voltage and current can essentially be observed instantaneously with

[1] `https://developers.google.com/android/reference/com/google/android/gms/location/LocationRequest`, "All location requests are considered hints, and you may receive locations that are more/less accurate, and faster/slower than requested."

[2] https://issuetracker.google.com/issues/37091475

[3] `https://www.theiphonewiki.com/wiki/Sandbox`

[4] `https://stackoverflow.com/questions/16026920/why-sandbox-in-ios-6-1-1-still-exists-for-app-even-after-i-have-jailbreak`

[5] `https://en.wikipedia.org/wiki/IOS_version_history\#iPhone_OS_2`

no observer effect. This allows fine-grained observation of uneven loads such as power spikes corresponding to increased CPU or network activity.

The limitations of external power meters are well-known and have been extensively argued in the model-based monitoring literature (e.g. [57, 41]): They are bulky, so they constrain power measurements to be carried out in laboratory settings, decoupling the components of the accuracy v/s power trade-off. They can only map the power drain of the entire phone, which makes it challenging to evaluate application-specific impact. Finally, they require access to the battery terminals, which is vanishingly rare in modern smartphones.

1.1.3 Battery run-down

This technique effectively treats the battery as a sensor that generates a single sample per experiment. Similar to the external power meter, battery run-down tests also measure the power drain of the entire phone - determining the application-specific impact is not trivial. Not all existing uses of this technique seem to account for this complexity - some simply state that the average battery life of the app is n hours. This is not rigorous because it does not account for the power consumption of system components, or the differences in contexts that the phones are exposed to. The coarse sampling of the battery run-down method also makes it hard to empirically observe differences in power consumption associated with sensing changes.

One technique to increase the granularity of this approach is to switch from battery *life* to battery *charge*. Today, smartphones embed a fuel gauge chip that senses voltage and current values. They are converted by the phone OS into a battery percent level, or State of Charge (SoC), and made available to applications through the *battery sensor*. Although early fuel gauges had high *instantaneous error* [11], modern devices which report both voltage and current, are reported to be within $2\% \pm 0.02$ of external power meters [6]. The accuracy of the State of Charge (SoC) provided by the phone OS, as compared to raw voltage and current values measured by an external power meter is an open question.

1.2 Zephyr

All three methods of power evaluation have inherent limitations. What is the most simple and stable method to evaluate the power consumption of smartphone-based systems? How accurate and consistent can such evaluations be? In this paper, we focus on using the state of charge as an internal power meter. We propose a methodology, *Zephyr*, that provides consistent, accurate and low overhead measurements for smartphone-based applications. Zephyr uses SoC values provided by the phone OS over time to compute the SoC Change Rate (SoCCR). It compares the SoCCR slope between runs of the application under controlled conditions, and uses the difference to measure increased or decreased power consumption. To evaluate the power usage of an adaptive sensing application, we use two phones of the same model, same OS version and same baseline applications installed and exposed to the same stimuli. One phone runs the application; the other phone is the baseline. The difference in SoCCR between the two phones represents the power consumption of the application.

This minimalist approach ensures that Zephyr is: (i) *easy to use*, since it does not require access to the battery terminals or training of power models, (ii) *cross-platform*, since it only relies on the phone's built-in `battery` sensor, (iii) *usable outside the lab*, since it allows mobility during measurements, and (iv) *stable*, since the sensor is unlikely to be restricted even for normal apps on smartphones[6].

In order to evaluate the accuracy of this approach, we explore characteristics of SoCCR on modern smartphones. We first show that, under constant load, SoCCR measurements are consistent (0.43 – 1.62 root mean square error (RMSE)), both across identical phones and for the same phone over time. Next, we show that applying varying loads, the SoCCR slope changes consistently across phones (0.2 – 2.6 RMSE). This indicates that the using the *difference* in SoCCR can compensate for the effects of background sensing system apps (if any). Zephyr has a configurable polling period. We show that it is able to observe coarse

[6]https://developer.android.com/training/monitoring-device-state/battery-monitoring.html "When you're altering the frequency of your background updates to reduce the effect of those updates on battery life, checking the current battery level and charging state is a good place to start."

grained sensing changes directly, and is able to observe the cumulative effective of more fine grained changes as well. Researchers can choose the polling granularity that is appropriate for their sensing project and obtain the responsiveness that they need.

Our contributions are the following:

1. We start with a comprehensive survey of mobile system literature from 2011–2017 (MobiSys, UbiComp and SenSys), and derive a set of requirements for the *power evaluation problem* (Section 2).
2. We describe Zephyr, the experimental procedure we propose. It accounts for the effect of standard context-sensitive apps (e.g. Google Now) on the phone and isolate the effect of the app being evaluated (Section 3.1).
3. We describe the architecture and implementation of a cross-platform smartphone app that can be used to measure SoCCR and facilitate reproducible results (Section 4).
4. We characterize the behavior of SoCCR under various scenarios that demonstrate the viability of this approach and estimate the error rate (Section 5).

Collectively, our contributions demonstrate that, with the proper methodology, it is possible to obtain accurate and responsive power evaluation using only the built-in battery sensor on the phone. We posit that the intuitive simplicity of our approach, combined with clearly documented results, will help researchers conduct power evaluations in ubiquituous computing projects, even in the face of future mobile hardware and software restrictions.

2 The State of Power Evaluation on Mobile Systems

We conduct a comprehensive literature review on how mobile systems researchers perform power evaluation in their ubiquitous sensing projects. We then derive requirements for power evaluation in the context of background sensing applications on smartphones.

2.1 Literature review on the extent of power evaluation

We surveyed every full paper published in the UbiComp proceedings for years 2011–2016 (480 papers in total). Figure 2 shows the percentage of papers focused on background sensing applications on smartphones and the percentage of those that did not include any power evaluation. We generated these results by adding the following labels to every paper on the list. Note, that we were conservative in assigning labels—in case of ambiguity, we assumed that the project was not focused on background sensing applications, or that the power was evaluated.

(i) smartphone-based: whether the data was sensed using a smartphone or smartwatch, as opposed to set of sensors connected to a microcontroller;

(ii) context-sensitive: if smartphone-based, whether the sensing occurred in the background depending on some context (e.g., location), or only when users explicitly performed some action; and

(iii) power evaluated: if this was a background sensing project running on COTS smartphones, whether there was a quantitative, empirical power evaluation.

Figure 2 shows that **background sensing applications on smartphones accounted for a consistent 15–25% of papers *every year*. Of these papers, 55–70% of papers had no quantitative power evaluation.**

We hypothesize that the reason for this absence of power evaluation is the lack of a standard, easy to use evaluation methodology. Our survey showed that several of the papers that skipped the evaluation were sensitive to power considerations. For example, [13] attempted to minimize the impact on the device's battery life by using event-driven operation with no processing on the phone, while [22] stated that users reported problems with the power consumption of their applications.

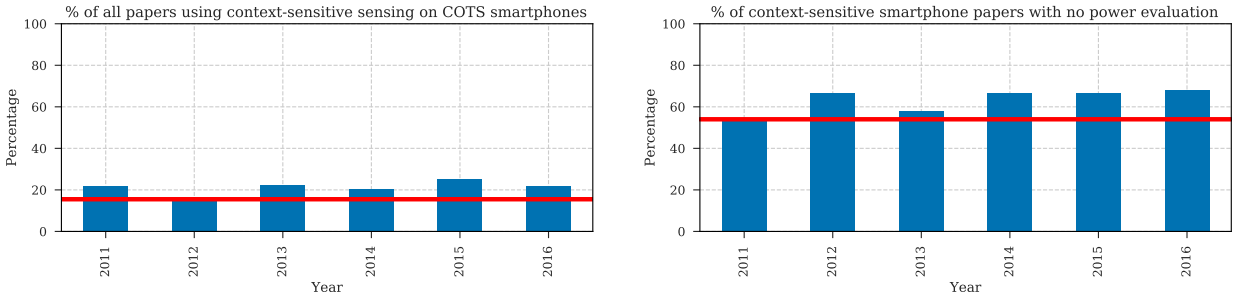


Figure 2: Percentage of papers with smartphone-based background sensing applications, and no power evaluation. Smartphone-based sensing is an important area of ubicomp, accounting for at least 15% of the papers every year, and a majority of those papers (min=55%) did not include a power evaluation.

2.2 Literature review on the techniques for power evaluation

We then looked more carefully at the papers that did include a power evaluation and identified the techniques used. Since there were only a small number of such papers, we expanded our search to papers published in MobiSys and SenSys from 2011–2016. We identified a combined 48 papers that both utilized continuous sensing and reported power analysis of their design. As we can see from **Table 1, we find measuring power with an external hardware power monitor to be the dominant technique for evaluating smartphone power consumption.** Other techniques, in decreasing order of popularity, are battery run-down tests, ad-hoc power models, numbers without provenance, and PowerTutor.

Table 1: Power evaluation techniques used in UbiComp 2011–2016

Mobile sensing applications	**Power evaluation technique**	**n**
Caloric expenditure of bicyclists [58]; Automatic labeling of transit stations semantics [12]; UbiTouch: smartphones as touchpads [54]; Area classification for fingerprint based indoor localization [19]	PowerTutor: model-based power estimation app	4
Indoor/outdoor detection [62]; Bus arrival time estimation [61]; Social fMRI [1]; Loci: semantic location services [28]; Crowd++: speaker count [56]; Monarca:disease insight [14]; CrossNavi: crossroad navigation for the blind [45]; Geofencing 2.0 [43]; MaLoc: Localization [55]; Room-level Locating System [33]; Recognizing Eating Moments [53]; RunBuddy [18]	Battery run-down test	12
Transportation mode detection [20]; Indoor/outdoor detection [42]; Driving routes detection [39]; Energy-efficient trip detection [23]; ARIEL: Room fingerprinting [25]; CAS: Context-aware application scheduling [31][7]; Opportunistic Position Update for LBS [4][8]; PocketParker [38]; Experiences with eNav [21]	Ad-hoc power model: built offline using published specs for the phone used, or measurements from an external power monitor, consumption computed through component usage traces or estimates	9
Device interaction [51]; Transportation mode detection [44]; Hotword detection [59]; Meeting membership detection [52]; Pulse: automatic content rating [5]; NLify: spoken natural language interface [17]; Headio: Heading acquisition [50]; Storage-aware energy savings [40]; AFV: application function virtualization [29]; GreenTouch: energy-efficient cellular radios [3]; Sensing WiFi packets in the air [7]; MobileMiner [48]; SymDetector [49]; EnTrack [32]; Experiences with eNav [21]; Sandra helps you learn [37]	External power monitor: power measurement of the whole design	16
Indoor air quality monitoring [24]; StressSense: through audio [34]; Dynamic home screen based on app usage [47]; Place-centric crowdsourcing [8]; Smartphone addiction detection [46]; Tasker: mobile crowdsourcing [26][9]; data hiding for sensitive smartphone data [35][10]; DeepEar [30]	Unclear; numbers provided without details on their provenance	7

2.3 Shortcomings of power evaluation in the literature

Our review identifies the shortcomings of the power evaluations performed for smartphone-based background sensing applications in the literature. They indicate a general, deeper problem with prevalent evaluation methods.

1. **Accuracy isolated from power.** One common theme across the papers we studied is that system accuracy and power evaluation is done in isolation. Accuracy is evaluated outside the lab by following fixed traces or recruiting volunteers. However, most power measurements are collected in a controlled laboratory setting. This practice persists even for mobility-based designs such as indoor/outdoor detection and transportation mode detection [20, 44, 42, 62]. The segregation between evaluation of accuracy and power hinders a more thorough understanding of the trade-off between these two fundamental design goals.

2. **Lack of cross-platform evaluation.** Most recent power evaluations were only done on Android (89%), with some older evaluations on Nokia and Tizen phones and only on two (4%) on iPhones (despite its current U.S. market share of $> 30\%$ [27]). This can be explained by various reasons—Android is a (partly) open source project, the phones are cheaper, and most older phones have easy physical access to the battery terminals. However, focusing on only one of the two popular smartphone platforms limits the generality of the results. For example, although [10] aims to evaluate the power consumption of *cross-platform frameworks*, their experimental evaluation restricts itself to Android. The authors' stated reason is that iOS devices do not provide access to the battery terminals.

3. **Inability to integrate with power-saving OS features.** Evaluations are performed on older smartphones that lack hardware support for new, battery-saving OS features in order to preserve access to the battery terminals for external power meters. This reduces the *applicability* of the results, since real-world deployments will have access to these features. For example, [44] was unable to evaluate the effect of *sensor batching* because it was only supported on Android 5+, and they were "...unable to tap into the battery of the Nexus 5 for power measurements".

4. **Use of invalid or obsolete models.** The authors in [58] used PowerTutor with "Motorola Droid, Samsung Nexus S, and Samsung Galaxy 2" phones in 2012, although PowerTutor does not support them. On the other hand, in spite of having their work published in 2016, [12] had to resort to using a HTC Nexus One phone, released in 2010, in order to be compatible with PowerTutor.

2.4 Requirements for Power Evaluation in Context-sensitive Sensing Projects

We have shown that power evaluations in the mobile systems literature suffer from various shortcomings. These shortcomings could be overcome by an easy to use *power evaluation* technique for background sensing applications on smartphones with a known error rate, and a standard experimental method, using stock OSes. Any solution to the power evaluation problem should satisfy the following requirements:

1. It should be able to accurately characterize the power consumption of **one single application**, which is the one that is being evaluated,

2. It should be able to exercise a variety of **realistic scenarios**, evaluating the power v/s accuracy tradeoff under a combination of those scenarios.

3. It should be able to use **off-the-shelf software on stock phones** with minimal configuration. Our literature survey (Table 1) shows researchers use existing power models, even when the models are not applicable to their hardware.

4. It should be **well-understood** with an clear experimental procedure, and a known estimate of the measurement error under that procedure.

We can also consider characteristics of the solution that are desirable, but not strictly required.

1. It should be *configurable*, supporting both fine-grained and coarse-grained measurement, as applicable for the needs of the project.
2. It should be *reproducible*, with data stored automatically and made available for download and analysis in the future.
3. It should be *cross-platform*, supporting android and iOS devices at a minimum.

3 The Zephyr Way

Our approach is to substitute the popular, but limited, external power meter (see Section 2) with the internal power meter represented by the battery sensor. Since it is built into the phone, the battery sensor addresses the shortcomings of external power meters - it is portable, and does not require any hardware modifications to access. It also meets several of the requirements in Section 2.4, since it is: (i) *cross-platform*, on both Android and iOS; (ii) *easy to use*, available without unlocking the bootloader and rooting the phone; (iii) *readily available*, through official APIs to allow applications to adapt behavior based on the SoC.

3.1 Methodology

Our technique is simple and builds on relative measurements against a baseline. Concretely, in Zephyr, to compare n different *sensing regimes* to each other, we perform $n + 1$ runs of the experiment on identical phones, one run for each sensing regime (e.g., different granularities for localization) and one run for the baseline.

Runs can executed in parallel, serially or with a hybrid of both. For serial runs, we use one phone and expose it to the same environment over $n + 1$ time periods, while running n sensing regimes. For parallel runs, we use $n + 1$ phones, exposed to the same environment, for one time period, while running n sensing regimes. This flexibility allows researchers to evaluate their projects on configurations ranging from a single phone (taking significantly longer time, but fewer resources) up to a fully fledged parallel evaluation using additional hardware.

Example: Location Tracking. A researcher wants to explore the design space for her new location tracking system in terms of localization accuracy and power consumption. She first identifies a sequence of trips that represent a typical day. Then, she executes the sequence of trips in multiple *runs*, one run per phone. For a serial run, she can perform the sequence with no tracking on day one (for the baseline), with high accuracy tracking on day two and with medium accuracy tracking on day three. For a parallel run, she can use three phones—phone 1 with tracking turned off, phone 2 with high accuracy tracking and phone 3 with medium accuracy tracking—and perform the sequence of trips while carrying all phones simultaneously during a single day. If she now wants to compare her approach against other location tracking projects (e.g. [28, 39]), she can include runs of the alternative approaches, either in parallel or in serial.

3.2 Rigorous Experiment v/s Daily Activities

Precisely defined experiments—the sequence of trips in above example—are important for the serial case to ensure that the phone is exposed to the same environment on every run. In contrast, parallel runs can be less prescriptive—it is possible to skip the definition of the trips since the phones are exposed to the same environment by virtue of being carried around together. So, given sufficient hardware resources, this allows for in-the-wild tests, where volunteers are handed $n + 1$ phones and asked to carry them around during unscripted daily activities.

Zephyr also supports hybrid solutions in which we have m phones, for $m < n + 1$ and we run the experiment in multiple serial batches of size m. If we are able to define a rigorous experiment, we end up with $\lceil \frac{n+1}{m} \rceil$ serial batches, since we only need to run the baseline once. If we want to use a more natural

experiment, we need $\lceil \frac{n}{m-1} \rceil$ batches, since we need to include a baseline phone as part of each batch. In the second case, regimes in different batches cannot be directly compared against each other. Instead, each must be compared to the baseline in its batch and only then can the results be compared against each other.

3.3 Granularity and the observer effect

In software monitoring, because we read the battery sensor from the same phone whose battery drain we measure, we could be subject to the *observer effect*, in which the act of measuring the value can change it. In general, since we compare values with the baseline, the observer effect will cancel out and we can measure fine grained values.

However, if we want to include the impact of OS-level optimizations such as Android's Doze mode, we need to ensure that the Zephyr is passive for long enough to activate the OS optimizations. In that case, we read the SoC with a large interval (t_m) and Zephyr cannot be used for fine grained measurement. However, although Zephyr cannot measure fine-grained transitions when used in this mode, it can still measure the **cumulative effect** of those transitions. If we keep reducing the granularity, we will eventually end up with a single reading per experiment, which reduces to the *battery run-down* methodology.

Concretely, consider a situation in which the sensing regime switches between states $\{S_1, S_2, S_3 \ldots\}$ with an interval t_a. If $t_a < t_m$, we may not get any readings in some of the states, which means that we may not be able to model the power consumption in each state. Even so, the impact of power consumption changes in each state accumulates, and when we compare against the baseline, Zephyr is able to assess power consumption over the duration of the experiment, or over a representative set of daily activities.

3.4 Identical phones

Since we rely on relative measurements to cancel out the effect of the baseline power drain, we need to ensure that the compared phones are identical. Concretely, this means that we use the same phone model, OS version, carrier settings, OS settings (data-gathering settings like Android location API, automatic updates), WiFi and Bluetooth state (e.g., connected to same network, same pairing state) and set of applications. In addition, automatic updates of all applications, including system components, should be turned off. Otherwise, an update could change the energy profile, and since it can be applied at different times on different phones, the power consumption across phones may no longer be consistent.

4 Implementation

Zephyr periodically reads the battery level (SoC) through framework APIs, stores it locally in a database and periodically posts the data to a remote server using HTTP requests to a REST API. The information read includes SoC and charging state. These two sensor values represent a common denominator between Android and iOS.

Zephyr consists of two periodic tasks, *read sensor* and *server sync*. Each of these tasks can be triggered by multiple possible events. A summary of these events, along with the one that we chose for our current implementation on Android and iOS, is shown in Table 2.

Table 2: Periodic invocations and triggers across platforms

	read sensor	**server sync**
silent remote push	iOS	iOS
local periodic scheduling	Android	Android
other sensor callback	-	-

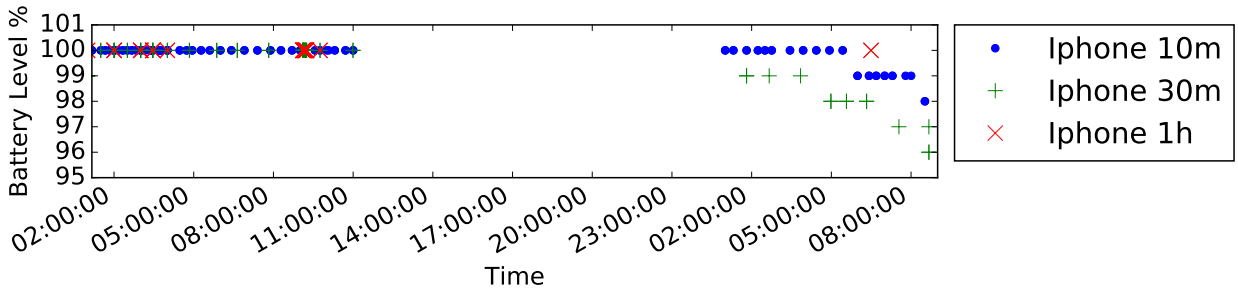


Figure 3: iOS background fetch: We use local, periodic scheduling on different time intervals. On iOS, events are not scheduled reliably because of OS power-saving features that make scheduling context-dependent.

We align read and sync tasks by performing them sequentially in time for both platforms (i.e., a sync task is triggered, reads sensor values and performs the HTTP request to publish to the REST API). However, the trigger event is distinct for both platforms: On Android, we start a sync task based on local periodic scheduling, on iOS we use silent push.[11] We use silent push instead of local scheduling on iOS, because the locally scheduled *background fetch* mechanism is context-sensitive, without time guarantees, and thus not suitable for interval-based measurements. Figure 3 shows the results of using local, periodic scheduling on iOS for different time intervals. It empirically shows that scheduling features on iOS are unreliable. Similarly, on Android 6.0+, local scheduling is affected by *doze mode*, but despite that, our experiments have shown that it is a reliable scheduling method and occurs at the same time across phone instances. We did not tie the read sensor task or the server sync to callbacks with other sensor values (third row from Table 2). That is because if the base sensor (i.e., the sensor that a power-measured application uses, e.g., location) was context sensitive, it would add another variable to the frequency of power measurement, making the following results harder to interpret. Reducing the overhead of Zephyr (see Section 5.3) by piggy-backing power measurements on existing callbacks is a topic for future work.

On the server, we receive time series data from the phones, convert the Android and iOS data structures into a common interface, and store time series data for each phone instance. The data in the repository can then be pulled to a local MongoDB database for further analysis, using standard statistical tools such as Python, Matlab and R.

We run a public server instance that also stores similar time series data from regular user phones[12]. Users of Zephyr can choose to use this infrastructure (and make their experimental data public), or configure their own server using our implementation. The data underlying all graphs and tables in this paper is publicly available, so others can reproduce our results.

5 Evaluation

We evaluate our proposed technique for *consistency*, *accuracy*, and *overhead*. We evaluate the consistency by comparing the battery drain of identical phones exposed to the same environment. We study how Zephyr captures power consumption for workloads that vary in time. We evaluate the accuracy by comparing the power drain reported by Zephyr with the results from hardware power monitoring. Finally, we show that a 30 minute measurement interval incurs negligible overhead.

These results collectively demonstrate that Zephyr is an alternative to traditional methods for determining the relative power drain of continuous sensing applications on modern phones in the wild.

Our evaluation devices consist of four iPhone6 (iOS 9.3, 2014), four Nexus 6 (Android 6.01, 2014), two Nexus 7 (Android 6.01, 2013) and two Moto E (Android 6.0, 2015). The software version and settings are

[11] Silent push describes a notification that is sent to an application on the user's phone, waking it up for processing, but without displaying notification to the user [2].

[12] URL removed for double blind reviewing.

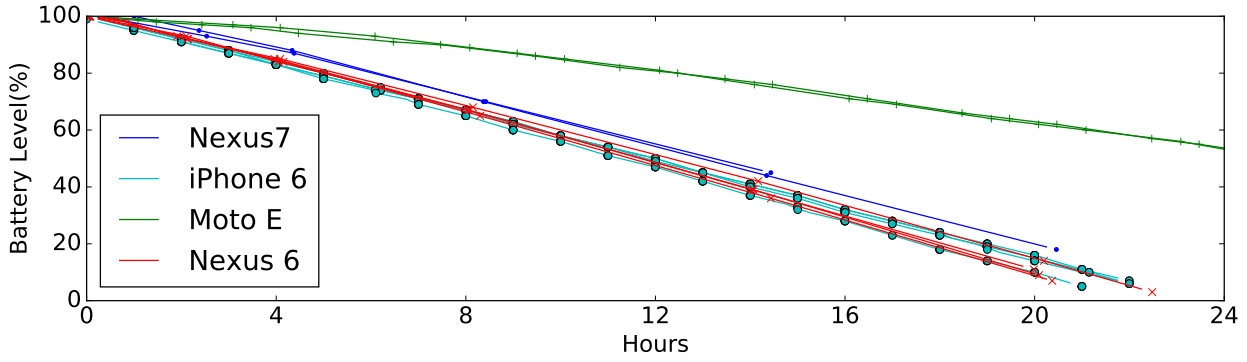


Figure 4: Continuous location tracking on different devices. We use multiple instances of each phone model. Phones continuously track their location on high accuracy, while being placed in the same location.

consistent across devices of the same model. Besides the Zephyr application, all devices only have the set of applications that are pre-installed.

5.1 Consistency

We evaluate the consistency of our approach for (1) parallel and (2) serial application of our methodology.

5.1.1 Validity of the parallel technique

We show that the drain across multiple phones in the same environment and with the same regime is consistent, for constant and variable workloads.

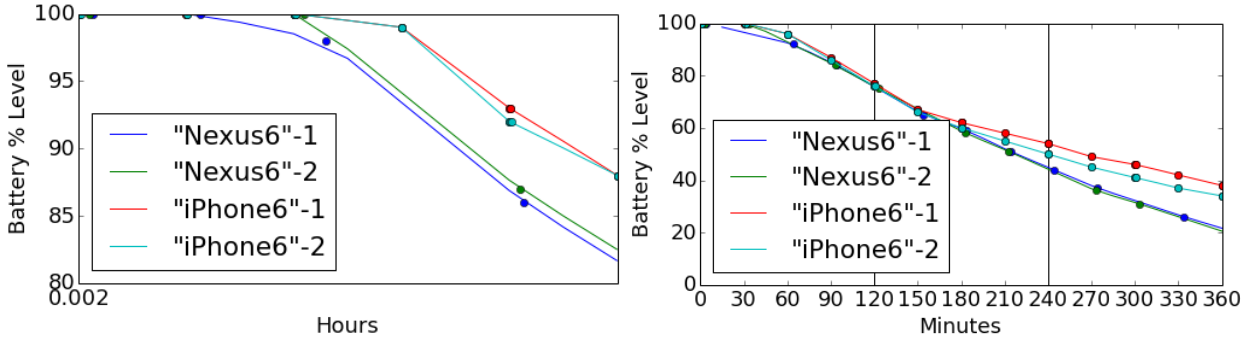
Constant Workload Our constant workload benchmark consists of running high-accuracy location tracking on all phone instances while phones are stationary. I.e., on Android, we use `android.location.Criteria.ACCURACY_HIGH` and on iOS we use `kCLLocationAccuracyBest`. We then continuously track the phones' location until they are fully discharged. Zephyr is configured with a measurement interval of 1 hour.

Fig. 4 depicts the result for this experiment. It shows that phone instances of the same model follow the same battery drain pattern. Nexus 7, iPhones and Nexus 6 phones all show similar battery drain rates. The Moto E phones, however, are separated from the other models. Concretely, the average drain rate between phones has low standard deviation, and the root mean square error (RMSE) is low (Table 3). This shows high consistency between phones and a good fit to a linear model.

We conclude that battery drain rate depends on the phone make and model, and battery drain rate can differ substantially on different phone types for the same benchmark.

Table 3: Battery drain across phones for a constant location tracking workload: Smartphone instances of the same model show similar battery-drain behavior.

Model	Linear fit RMSE	Quadr. fit RMSE	Avg drain rate (% drop/hr)
iPhone6	0.438 ± 0.091	0.343 ± 0.059	-4.34 ± 0.126
Nexus6	1.193 ± 0.141	0.372 ± 0.120	-4.52 ± 0.136
Nexus7	1.239 ± 0.198	0.765 ± 0.345	-4.13 ± 0.141
MotoE	1.692 ± 0.040	1.484 ± 0.319	-2.18 ± 0.056



(a) Fast variable workload

(b) Slow variable workload

Figure 5: Video playback at different brightness on device pairs. Vertical lines indicate state changes, slope changes appear to lag.

Variable Workload Our variable workload consists of a white screen video playback on pairs of iPhone 6 and Android 6 phones. In order to induce controlled variability, we change the screen brightness, cycling between 100%, 50% and 0%.[13] We perform two permutations of this workload: (1) a fast adapting workload and (2) a slow adapting workload. For the fast adapting workload, the brightness is "adapted" every 15 min while the battery value is measured every 30 minutes. In the slow adapting workload, the brightness is changed every 2 hours while the measurement interval remains 30 min.

Fig. 5 depicts the results for pairs of phone instances for both fast adapting (a) and slow adapting (b) workloads. As expected, in the fast adapting workload, we are unable to detect the 15 min interval state changes with a 30 min measurement interval. However, the overall drain is consistent across instances. In the slow adapting workload, we are able to detect small changes in the slope when the state changes. Note the divergence in the iPhone slopes when auto-brightness was inadvertently turned on, and the return to parallel once it was fixed. Compared to the constant workload, battery drain rate across phone instances is less consistent (0.2% - 2.6% RMSE). However, the results are consistent enough to allow the comparison between different variable workloads.

5.1.2 Validity of the Serial Technique

Researchers might have only a limited number of phones available. We now evaluate Zephyr's serial technique, in which experiments are performed on the same model and environment, but at different times. Concretely, we show that the battery drain on the same phone, but across multiple identical days, is consistent. We use the constant workload described in Section 5.1.1.

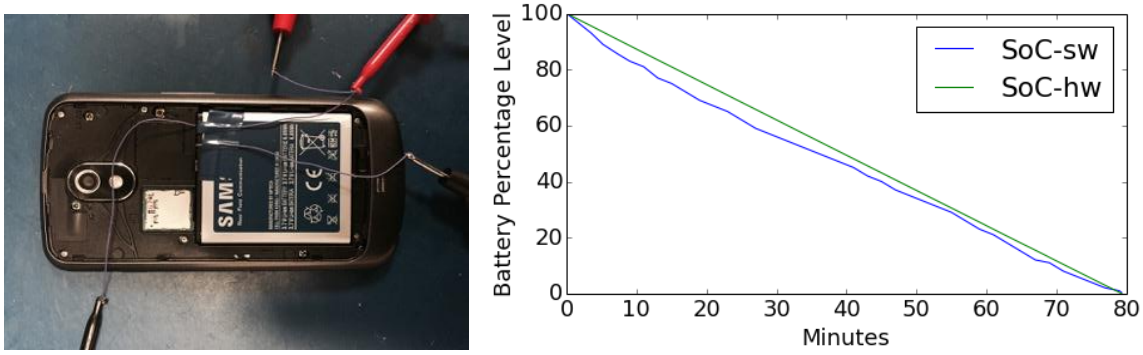
Our results are summarized in Table 4. They show that the variation in the average drain rate is low (max. < 0.233 and for most instances < 0.1).

In summary, all our experiments show highly consistent results across smartphone instances of the same model, for constant and variable workloads and for Zephyr's parallel and serial technique. Next, we look at accuracy in relation to hardware based power monitoring.

5.2 Accuracy

In order to show that Zephyr accurately evaluates actual power drain, we compare its results against a baseline obtained through hardware measurements. This required the use of a phone with removable battery, for which we picked a Galaxy Nexus Phone (Android OS 4.4.2) from 2011. This phone model has also been used in several of the papers discussed in Section 2.

[13] Screen brightness is one of the main impact factors on a smartphone energy consumption [60].



(a) HW Measurement Setup

(b) SW vs HW SOC

Figure 6: Hardware validation of continuous workload

Figure 6a shows the experimental setup, with the phone's battery terminals used for connecting to the hardware power source. We measure voltage and current simultaneously using two digital multimeters. Then we compute the power consumption by multiplying the current and voltage measurements. To be able to compare these hardware measurements with our software based SoC measurements, we obtain a comparable, hardware-based SoC value as follows: After a full discharge, we integrate measured power values over time to uncover the (initial) amount of energy stored in the phone battery in Joules. This allows us to compute the "hardware SoC" by dividing the amount of energy remaining by the initial capacity in subsequent experiments.

5.2.1 Constant Workload

Our constant workload uses the white screen video technique from Section 5.1.1, except that we fix the screen brightness at 100% to have a constant power draw. We then measure a full discharge cycle using Zephyr and both multimeters. We compute the "hardware SoC" as explained above.

Figure 6b shows the results, comparing between hardware and software SoC for a constant workload. We achieve a corresponding error of 4.27% in the battery drain rate. Note that the Galaxy Nexus phone is a five year old model. The aging battery in Galaxy Nexus and improved fuel gauge technology inside newer Nexus phones explain why the change in SoC of the Galaxy Nexus (Figure 6b) is not as linear as a comparable result for the Nexus 6 (Figure 4).

5.2.2 Fast Variable Workload

As a fast varying workload, we use the fast adapting brightness workload from Section 5.1.1 with 15 min intervals of 100%, 50% and 0% brightness levels, repeated twice, and measured every 30 mins. Again, we use the same technique as in Section 5.2.1 to compute a hardware based-SoC as our baseline.

Table 4: Avg. drain rate (mean ± std_dev) across two days

Model	phone1	phone2	phone3	phone4
iPhone 6	4.27 ±0.021	4.18 ±0.085	4.24 ±0.057	4.47 ±0.13
Nexus 6	4.54 ±0.071	4.32 ±0.028	4.53 ±0.028	4.49 ±0.233

Our results are depicted in Figure 7. They show that Zephyr yields a ratio of 11.443 for the cumulative power drain for the fast variable workload, compared to the baseline. The hardware measurement yields 13.368 for the same ratio, a difference of 14.40% between Zephyr and hardware-based monitoring. Fig 7 also shows a strong correlation between the slopes representing both the experiment and the baseline. Further, as expected, because the workload varies faster than we can measure it, the software slope does not vary with the workload state. The hardware slope does show small variance at the middle of each cycle, but the lines converge again at the end of each cycle, indicating that Zephyr is able to accurately measure the cumulative effect of quickly varying workloads when we extend the runtime of experiments.

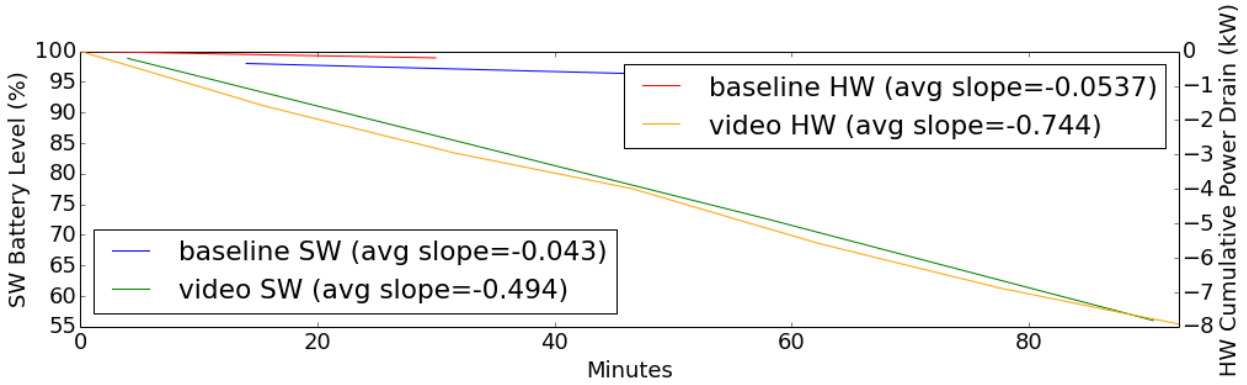


Figure 7: Hardware validation for a quickly varying workload where $f_a(1/15m) > f_m(1/30m)$

5.2.3 Slow Variable Workload

We now investigate if we can also detect the state of varying workloads with Zephyr. Note that this is not necessary for the benchmarking in most of the context-sensitive sensing papers discussed in Section 2, where the daily battery impact of an application is paramount.

In our experiment, we increase the time in each state of the varying workload (see Section 5.1.1) to be greater than Zephyr's measurement interval. We expect to be able to see a change in power drain, as reflected by a change in slope, for both Zephyr and hardware-based measurements. We switch between 100% brightness and 0% every 20 min, while measuring every minute.

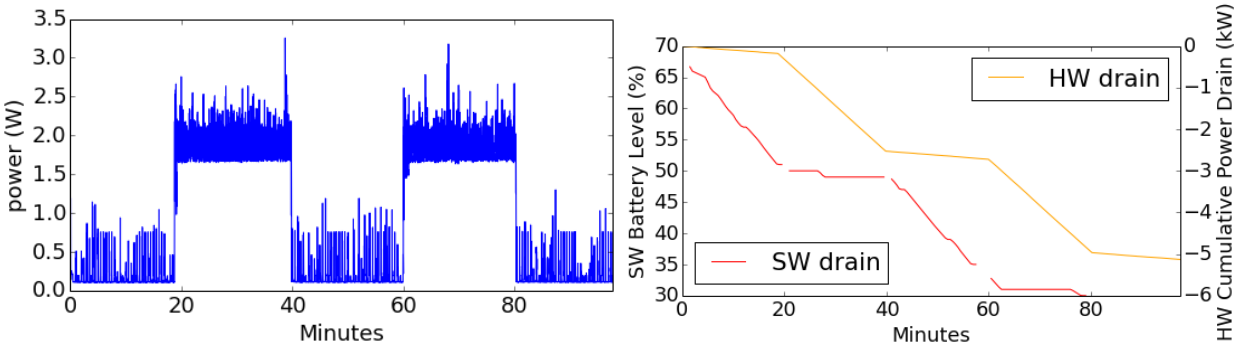
Figure 8a shows the raw hardware measurements, while Figure 8b shows hardware and software based SoC. Note that, the hardware and software plots are offset because only one starts with the baseline. However, the plots show a clear parallel pattern. Further, the mean power drain ratio reported by Zephyr is 13.224, while that reported by the hardware is 11.616, a difference of only 13.845%.

In conclusion, Zephyr shows only small error rates ($\approx$ 10%) when compared to hardware based measurement. Detecting the state of variable workloads with Zephyr is possible to some extent, limited by the granularity of the phone's fuel gauge, the ratio between Zephyr's measurement interval and the interval of the variable workload, and OS-level restrictions on background execution. However, detecting such variable workloads is not required for most context-sensitive sensing papers that we surveyed in Section 2.

5.3 Overhead

Lastly, we investigate the overhead of Zephyr itself by investigating its observer effect on power consumption. In our experiments, we run Zephyr with different measurement intervals to measure unloaded power consumption. We then use our previously described external power meter setup to directly measure the power consumption of the baseline, and of Zephyr at multiple granularities.

Figure 9a and Table 9b show the results for multiple runs, using different Zephyr measurement intervals from 1 min to 30 min as well as Zephyr not running in the background (no sync). As can be seen in Table 9b,



(a) Raw power readings

(b) HW v/s SW drain

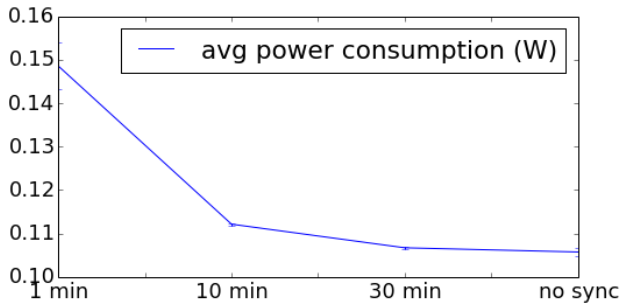
Figure 8: Hardware validation for a slowly varying workload where $f_a(1/20m) < f_m(1/1m)$. The lines are offset because HW monitoring started with standby while software monitoring started with the experiment.

periods longer than 30 min result in negligible power overhead (< 0.001 W).

In conclusion, Zephyr's overhead is small compared to the energy impact of the application under measurement when using sensible measurement intervals (30 min). Besides that, its overhead is neutralized by Zephyr's methodology that compares application measurement against baseline measurements.

Figure 9: Observer effect at various measurement intervals. The graph does have error bars but they are almost invisible since the standard deviation is very low.

(a) Knee at 30 mins



(b) Low std dev

	mean (W)	overhead (W)	std
1 min	0.148	0.042	0.005387
10 min	0.112	0.006	0.000306
30 min	0.107	0.001	0.000283
no sync	0.106	-	0.000950

6 Conclusion

With Zephyr we propose a software-based smartphone power monitoring technique that overcomes shortcomings of traditional power monitoring tools. Zephyr enables simple power consumption monitoring across platforms and in the wild by relying solely on software reported State of Charge (SoC) data. Our evaluation has shown that this is a valid and accurate way to evaluate power drain of continuous mobile sensing designs. In the presence of the limitations of existing power monitoring techniques and the shift towards non-removable batteries in the mobile industry, we hope our proposed technique can help simplify the power monitoring process, enable more testing under real-world conditions and lead to a better understanding of the power/accuracy trade-off of design decisions. Looking forward, we would like to initiate a discussion among mobile researchers that leads towards the establishment of a simple and reproducible power evaluation

methodology without relying on native power measurements. All our code is freely available[14].

References

[1] N. Aharony, W. Pan, C. Ip, I. Khayal, and A. Pentland. The social fmri: Measuring, understanding, and designing social mechanisms in the real world. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11, pages 445–454, New York, NY, USA, 2011. ACM.

[2] Apple. Apple Push Notification service (APNs). `https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html\#//apple_ref/doc/uid/TP40008194-CH8-SW1`, 11 2017.

[3] S. Aras and C. Gniady. Greentouch: Transparent energy management for cellular data radios. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '16, pages 970–980, New York, NY, USA, 2016. ACM.

[4] P. Baier, F. Dürr, and K. Rothermel. Opportunistic position update protocols for mobile devices. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 787–796, New York, NY, USA, 2013. ACM.

[5] X. Bao, S. Fan, A. Varshavsky, K. Li, and R. Roy Choudhury. Your reactions suggest you liked the movie: Automatic content rating via reaction sensing. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 197–206, New York, NY, USA, 2013. ACM.

[6] J. Bornholt, T. Mytkowicz, and K. S. McKinley. The model is not enough: Understanding energy consumption in mobile devices. *Power (watts)*, 1(2):3, 2012.

[7] Y. Chon, S. Kim, S. Lee, D. Kim, Y. Kim, and H. Cha. Sensing wifi packets in the air: practicality and implications in urban mobility monitoring. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 189–200. ACM, 2014.

[8] Y. Chon, N. D. Lane, Y. Kim, F. Zhao, and H. Cha. Understanding the coverage and scalability of place-centric crowdsensing. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 3–12, New York, NY, USA, 2013. ACM.

[9] Chulhong Min, Youngki Lee, Chungkuk Yoo, Seungwoo Kang, Sangwon Choi, Pillsoon Park, Inseok Hwang, Younghyun Ju, Seungpyo Choi, and Junehwa Song. PowerForecaster: Predicting Smartphone Power Impact of Continuous Sensing Applications at Pre-installation Time. Seoul, Nov. 2015.

[10] M. Ciman and O. Gaggi. Evaluating impact of cross-platform frameworks in energy consumption of mobile applications. In *WEBIST*, 2014.

[11] M. Dong and 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*, pages 335–348. ACM, 2011.

[12] M. Elhamshary, M. Youssef, A. Uchiyama, H. Yamaguchi, and T. Higashino. Transitlabel: A crowd-sensing system for automatic labeling of transit stations semantics. In *MobiSys'16*. ACM, 2016.

[13] D. Ferreira, E. Ferreira, J. Goncalves, V. Kostakos, and A. K. Dey. Revisiting human-battery interaction with an interactive battery interface. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 563–572, New York, NY, USA, 2013. ACM.

[14] URL removed for double blind reviewing.

[14] M. Frost, A. Doryab, M. Faurholt-Jepsen, L. V. Kessing, and J. E. Bardram. Supporting disease insight through data analysis: Refinements of the monarca self-assessment system. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 133–142, New York, NY, USA, 2013. ACM.

[15] Google. Android Sensor Stack. `https://source.android.com/devices/sensors/sensor-stack`, 2017.

[16] M. Gordon, L. Zhang, B. Tiwana, R. Dick, Z. M. Mao, and L. Yang. Powertutor. `http://ziyang.eecs.umich.edu/projects/powertutor`, 2009.

[17] S. Han, M. Philipose, and Y.-C. Ju. Nlify: Lightweight spoken natural language interfaces via exhaustive paraphrasing. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 429–438, New York, NY, USA, 2013. ACM.

[18] T. Hao, G. Xing, and G. Zhou. Runbuddy: A smartphone system for running rhythm monitoring. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 133–144. ACM, 2015.

[19] S. He, J. Tan, and S.-H. G. Chan. Towards area classification for large-scale fingerprint-based system. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '16, pages 232–243, New York, NY, USA, 2016. ACM.

[20] S. Hemminki, P. Nurmi, and S. Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *SenSys'13*. ACM, 2013.

[21] S. Hu, L. Su, S. Li, S. Wang, C. Pan, S. Gu, M. T. Al Amin, H. Liu, S. Nath, R. R. Choudhury, et al. Experiences with enav: A low-power vehicular navigation system. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 433–444. ACM, 2015.

[22] S. Intille, C. Haynes, D. Maniar, A. Ponnada, and J. Manjourides. μema: Microinteraction-based ecological momentary assessment (ema) using a smartwatch. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '16, pages 1124–1128, New York, NY, USA, 2016. ACM.

[23] Y. Jiang, D. Li, G. Yang, Q. Lv, and Z. Liu. Deliberation for intuition: A framework for energy-efficient trip detection on cellular phones. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11, pages 315–324, New York, NY, USA, 2011. ACM.

[24] Y. Jiang, K. Li, L. Tian, R. Piedrahita, X. Yun, O. Mansata, Q. Lv, R. P. Dick, M. Hannigan, and L. Shang. Maqs: A personalized mobile sensing system for indoor air quality monitoring. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11, pages 271–280, New York, NY, USA, 2011. ACM.

[25] Y. Jiang, X. Pan, K. Li, Q. Lv, R. P. Dick, M. Hannigan, and L. Shang. Ariel: Automatic wi-fi based room fingerprinting for indoor localization. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 441–450, New York, NY, USA, 2012. ACM.

[26] T. Kandappu, N. Jaiman, R. Tandriansyah, A. Misra, S.-F. Cheng, C. Chen, H. C. Lau, D. Chander, and K. Dasgupta. Tasker: Behavioral insights via campus-based experimental mobile crowd-sourcing. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '16, pages 392–402, New York, NY, USA, 2016. ACM.

[27] Kantar. Smartphone os sales market share. `https://www.kantarworldpanel.com/smartphone-os-market-share/`, 2017.

[28] D. H. Kim, K. Han, and D. Estrin. Employing user feedback for semantic location services. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11, pages 217–226, New York, NY, USA, 2011. ACM.

[29] H. Kolamunna, Y. Hu, D. Perino, K. Thilakarathna, D. Makaroff, X. Guan, and A. Seneviratne. Afv: Enabling application function virtualization and scheduling in wearable networks. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '16, pages 981–991, New York, NY, USA, 2016. ACM.

[30] N. D. Lane, P. Georgiev, and L. Qendro. Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 283–294. ACM, 2015.

[31] J. Lee, K. Lee, E. Jeong, J. Jo, and N. B. Shroff. Context-aware application scheduling in mobile systems: What will users do and not do next? In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '16, pages 1235–1246, New York, NY, USA, 2016. ACM.

[32] S. Lee, W. Jung, Y. Chon, and H. Cha. Entrack: a system facility for analyzing energy consumption of android system services. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 191–202. ACM, 2015.

[33] S. Lee, Y. Kim, D. Ahn, R. Ha, K. Lee, and H. Cha. Non-obstructive room-level locating system in home environments using activity fingerprints from smartwatch. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 939–950. ACM, 2015.

[34] H. Lu, D. Frauendorfer, M. Rabbi, M. S. Mast, G. T. Chittaranjan, A. T. Campbell, D. Gatica-Perez, and T. Choudhury. Stresssense: Detecting stress in unconstrained acoustic environments using smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 351–360, New York, NY, USA, 2012. ACM.

[35] C. Luo, A. Fylakis, J. Partala, S. Klakegg, J. Goncalves, K. Liang, T. Seppänen, and V. Kostakos. A data hiding approach for sensitive smartphone data. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '16, pages 557–568, New York, NY, USA, 2016. ACM.

[36] C. Min, S. Lee, C. Lee, Y. Lee, S. Kang, S. Choi, W. Kim, and J. Song. Pada: Power-aware development assistant for mobile sensing applications. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '16, pages 946–957, New York, NY, USA, 2016. ACM.

[37] C. Min, C. Yoo, I. Hwang, S. Kang, Y. Lee, S. Lee, P. Park, C. Lee, S. Choi, and J. Song. Sandra helps you learn: the more you walk, the more battery your phone drains. In *Proceedings of the 2015 ACM international joint conference on Pervasive and ubiquitous computing*, pages 421–432. ACM, 2015.

[38] A. Nandugudi, T. Ki, C. Nuessle, and G. Challen. Pocketparker: Pocketsourcing parking lot availability. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 963–973. ACM, 2014.

[39] S. Nawaz and C. Mascolo. Mining users' significant driving routes with low-power sensors. In *SenSys'14*. ACM, 2014.

[40] D. T. Nguyen, G. Zhou, X. Qi, G. Peng, J. Zhao, T. Nguyen, and D. Le. Storage-aware smartphone energy savings. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 677–686, New York, NY, USA, 2013. ACM.

[41] A. Nika, Y. Zhu, N. Ding, A. Jindal, Y. C. Hu, X. Zhou, B. Y. Zhao, and H. Zheng. Energy and Performance of Smartphone Radio Bundling in Outdoor Environments. pages 809–819. ACM Press, 2015.

[42] V. Radu, P. Katsikouli, R. Sarkar, and M. K. Marina. A semi-supervised learning approach for robust indoor-outdoor detection with smartphones. In *SenSys'14*. ACM, 2014.

[43] S. Rodriguez Garzon and B. Deva. Geofencing 2.0: taking location-based notifications to the next level. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 921–932. ACM, 2014.

[44] K. Sankaran, M. Zhu, X. F. Guo, A. L. Ananda, M. C. Chan, and L.-S. Peh. Using mobile phone barometer for low-power transportation context detection. In *SenSys'14*. ACM, 2014.

[45] L. Shangguan, Z. Yang, Z. Zhou, X. Zheng, C. Wu, and Y. Liu. Crossnavi: enabling real-time crossroad navigation for the blind with commodity phones. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 787–798. ACM, 2014.

[46] C. Shin and A. K. Dey. Automatically detecting problematic use of smartphones. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 335–344, New York, NY, USA, 2013. ACM.

[47] C. Shin, J.-H. Hong, and A. K. Dey. Understanding and prediction of mobile application usage for smart phones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 173–182, New York, NY, USA, 2012. ACM.

[48] V. Srinivasan, S. Moghaddam, A. Mukherji, K. K. Rachuri, C. Xu, and E. M. Tapia. Mobileminer: Mining your frequent patterns on your phone. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 389–400. ACM, 2014.

[49] X. Sun, Z. Lu, W. Hu, and G. Cao. Symdetector: detecting sound-related respiratory symptoms using smartphones. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 97–108. ACM, 2015.

[50] Z. Sun, S. Pan, Y.-C. Su, and P. Zhang. Headio: Zero-configured heading acquisition for indoor mobile devices through multimodal context sensing. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 33–42, New York, NY, USA, 2013. ACM.

[51] Z. Sun, A. Purohit, R. Bose, and P. Zhang. Spartacus: spatially-aware interaction for mobile devices through energy-efficient audio sensing. In *MobiSys'13*. ACM, 2013.

[52] W.-T. Tan, M. Baker, B. Lee, and R. Samadani. The sound of silence. In *SenSys'13*. ACM, 2013.

[53] E. Thomaz, I. Essa, and G. D. Abowd. A practical approach for recognizing eating moments with wrist-mounted inertial sensing. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 1029–1040. ACM, 2015.

[54] E. Wen, W. Seah, B. Ng, X. Liu, and J. Cao. Ubitouch: Ubiquitous smartphone touchpads using built-in proximity and ambient light sensors. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '16, pages 286–297, New York, NY, USA, 2016. ACM.

[55] H. Xie, T. Gu, X. Tao, H. Ye, and J. Lv. Maloc: A practical magnetic fingerprinting approach to indoor localization using smartphones. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 243–253. ACM, 2014.

[56] C. Xu, S. Li, G. Liu, Y. Zhang, E. Miluzzo, Y.-F. Chen, J. Li, and B. Firner. Crowd++: Unsupervised speaker count with smartphones. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 43–52, New York, NY, USA, 2013. ACM.

[57] f. xu, y. liu, q. li, and y. zhang. v-edge: fast self-constructive power modeling of smartphones based on battery voltage dynamics. In *nsdi*, volume 13, pages 43–56, 2013.

[58] A. Zhan, M. Chang, Y. Chen, and A. Terzis. Accurate caloric expenditure of bicyclists using cellphones. In *SenSys'12*. ACM, 2012.

[59] L. Zhang, P. H. Pathak, M. Wu, Y. Zhao, and P. Mohapatra. Accelword: Energy efficient hotword detection through accelerometer. In *MobiSys'15*. ACM, 2015.

[60] L. Zhang, B. Tiwana, R. P. Dick, Z. Qian, Z. M. Mao, Z. Wang, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2010 IEEE/ACM/IFIP International Conference on*, pages 105–114. IEEE, 2010.

[61] P. Zhou, Y. Zheng, and M. Li. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *MobiSys'12*. ACM, 2012.

[62] P. Zhou, Y. Zheng, Z. Li, M. Li, and G. Shen. Iodetector: a generic service for indoor outdoor detection. In *SenSys'12*. ACM, 2012.