BattOr: Plug-and-debug energy debugging for applications on smartphones and laptops

Stanford CS Tech Report

Aaron Schulman* Tanuj Thapliyal* Sachin Katti* Neil Spring† Dave Levin† Prabal Dutta‡
*Stanford University †University of Maryland ‡University of Michigan

Abstract

As battery-operated devices like smartphones and laptops have become the norm for users, a critical feature of applications is to ensure low power consumption. Yet, surprisingly, energy bugs remain difficult to find and fix because developers lack the tools to allow them to quickly and authoritatively reason about how much power the various components of their applications consume. To address this need, we present BattOr, a power monitoring system consisting of (1) a hardware component that collects accurate measurements without requiring soldering (developers can plug-and-debug), and (2) a software component that precisely synchronizes these measurements with low-level system events already captured by OSes. A distinguishing feature of BattOr is that it is compatible with existing, continuous integration testing systems, and has been evaluated in production environments. We present case studies, including one in which BattOr was used to find and fix a long-lived energy bug that had resulted in a 38% increase in power consumption of Chrome on OS X as compared to Safari.

1 Introduction

Energy is the defining constraint in mobile computing; users choose apps and devices based on their battery life [15, 17]. Consequently, developers have the responsibility to find and fix any energy waste bugs in their apps. Google's developers, for instance, have responded to pressure from users to improve Chrome's energy consumption [25] by finding and patching energy bugs [23]. ARM even created a new big.LITTLE CPU architecture [1] aimed at improving energy efficiency by having low-power cores for light processor workloads and high-power cores for heavy workloads.

Yet, surprisingly, energy bugs remain notoriously difficult to find and fix [19], often resulting in bugs escaping into the wild and being detected by the user population, rather than in-house.

In this paper, we present *BattOr*, a power monitoring system that collects and synchronizes accurate power measurements with logs of low-level system events, such as what devices and threads are active. BattOr was used by Google's Chrome team to discover and fix a bug that was resulting in 38% higher power consumption for video playback on OS X for Chrome compared to Safari (§6).

BattOr fills a surprisingly open gap in the research and application of power monitoring systems. Despite recent advancements—including inferred power models [20, 21, 28], small and portable external power monitors [2, 26], and even inferring power usage from internally measured voltage [27]—Chrome bug reports [7] reveal that developers still rely on crude battery rundown tests, which take hours to run, and can require months to find a pernicious bug.

Developers rely on rundown tests because they are the only existing power monitoring tool that meets developers' requirements. In particular, developers require three key attributes from a power monitoring tool: First, it must provide *accurate* power measurements, so that developers can confirm the existence of a reported bug, and verify that it is fixed before they push to users. Second, the power monitor must be *portable*, so that developers can reproduce the exact environment (e.g., signal strength) in which the bug appears. Finally, the power monitor must be able to collect accurate, portable measurements *without requiring soldering* wires onto the smartphone or laptop the developers are testing. While soldering may appear to be simply a nuisance, it is in fact a surprisingly limiting factor when equipping software developers with power monitors (§2).

The extensive research in this space has not transitioned widely to practice because none meet the requirements of accurate measurements, portability, and not requiring soldering: Models are not accurate, every model has cases where the error is significant and developers do not know if the bug they are testing is one of those cases.

External power monitors require soldering so they are only usable by a select few software developers who have access to soldering technicians and equipment. Also, some external power monitors are not portable [16]. Finally, internal power monitors do not provide accurate power measurements as the processor being monitored is consuming energy to collect its own measurements.

We describe in this paper the design, implementation, and application of BattOr, and show that it provides all of the properties necessary for developer use. BattOr comprises a portable hardware component a small, external power monitor that is powered off of the device's own battery (§4). Moreover, BattOr provides accurate measurements without requiring any soldering; we achieve this by exploiting the insight that many devices have one of just a few commonly deployed battery connectors to interpose between a device and its power source (battery in the case of a smartphone, and external power supply in the case of a laptop). Though some previous power monitors that are both accurate and portable, they derive these properties through soldering or cutting into batteries. We also show that it is possible to provide portable instrumentation of power consumption while powering the monitor off of the same battery that is under test, contrary to the assertion of Brouwers et al. [2].

In addition to the hardware component that meets developers' requirements, BattOr also comprises a software component that enables an automated and fast testing framework for developers (§5). The key technical challenges in incorporating an external power monitor like BattOr into an automated testing infrastructure are to take samples with high enough time precision to be able to instrument short-lived events (such as the handling of one type of video frame versus another), and to synchronize these fine-grained power measurements with events or state changes on the device itself. We present synchronization techniques that use smartphones' camera flashes to induce predictable power consumption in our measurement traces as well as events in most OSes' logs, allowing us to synchronize the two to the order of microseconds. With its novel hardware and software components, BattOr cuts the time for each energy measurement from 5–10 *hours* for battery drain tests to 1–5 *minutes* with BattOr.

We demonstrate how BattOr's accurate, reproducible power monitoring and precise synchronization are applicable to production environments through actual case studies (§3, §6). BattOr has been evaluated in a production setting to improve the power consumption of an application that is used by over one in four Internet users [18], the Chrome web browser. We present several case studies, including how the Chrome team applied BattOr to fix an energy bug that caused 38% higher power consumption in video playback on OS X Chrome as compared to Safari.

2 Background and Related Work

In this section, we describe the traditional workflow involved in energy debugging, and review related work in light of each of the workflow's steps. We classify related work into three categories: external power monitors [16, 8, 10, 2, 26, 14], internal power monitors [13] and software power models [20, 21, 28]. The standard energy debugging workflow parallels that of more traditional bugs, in that it consists of the four broad steps of discovery, triage, root cause identification, and fixing. But, as we will see, today, energy debugging typically involves developers waiting hours for battery rundown tests, and as a result, it can take months to track down pernicious energy bugs.

1. Discovery The debugging process begins with discovery: identifying a version of the code that resulted in an *energy regression* (an overall increase in consumed power). Energy regressions are typically found only after a release, once a large population of users have had the chance to repeatedly rundown their battery and notice overall increases in power consumption. Conversely, more traditional regression tests—e.g., for performance or correctness—are performed in-house, as part of continuous integration testing.

Efficiently discovering energy bugs benefits from being able to obtain power measurements from the same usage scenarios as what caused them in the first place, and thus would benefit from a power monitoring tool that is easy to deploy widely to users and developers. However, external power monitors—Monsoon [16], iCount+Quanto [8, 10], NEAT [2], and even just direct measurements through an oscilloscope—all require soldering onto or cutting into a mobile device's battery connection. Soldering and cutting requires skilled labor, is time-consuming, difficult, and for some devices even *dangerous* task as the battery itself must be modified; moreover, because these altered devices are not widely available, these monitors are limited to deployments of only a few developers. Software-based power models as well as internal power monitors permit deployment to virtually all users, and are therefore particularly useful in the discovery phase. But as we discuss below, they do not provide the accurate measurements necessary in the subsequent debugging steps.

BattOr is an external power monitor that cleanly fits into existing devices, is small and portable and requires no soldering. Because BattOr is an external device, it is not feasible to deploy to *all* users, however, BattOr is feasible to deploy to all members of a development team. Therefore, with BattOr, it is possible to incorporate power measurements into a development team's automated performance regression testing system (e.g., Google Chrome's Telemetry[1]).

2. Triage The next step in the debugging process is triage: verifying that the discovered bug is indeed a bug, and not, e.g., noise in the measurement or specific software build. Triage is particularly important with user-reported discoveries, as they can be anecdotal and extremely noisy, and if not triaged, can waste significant developer time. The goal of triage is thus to replicate the reported bug report with authoritative measurements of the energy consumption under those conditions, and to verify if it has increased as reported for that app. Ideally, measurements during triage can be performed automatically and quickly to improve developer productivity.

External power monitors, including BattOr, and internal power monitors do provide the authoritative measurements needed for triage but software based power models do not. A software model's accuracy is dependent on many factors, such as the number of devices modeled, the complexity of the devices, and the time precision modeled. The most advanced power model available for smartphones, EProf [20] reports that only the 80th percentile of 50 msec intervals have less than 10% error; when applied to the YouTube application, however, EProf had 45% error for 90% of the 50 msec intervals.[2]

3. Root cause identification Once a developer has verified the existence of an energy bug, the next step is root cause identification: attributing the energy bug to a specific task within a specific application. With traditional battery rundown tests, this often requires selectively turning off modules, doing a rundown, and iterating until the culprit module can be identified. Not only is this extremely time-consuming, but it can easily lead developers down the wrong path, e.g., if turning off one module results in increased power consumption in another. We show with several case studies that BattOr is able to identify energy bugs in minutes.

Root cause analysis requires the ability to synchronize power measurements to activities on the device being measured, such as CPU activity, which threads were active at any point in time, and so on so that one can pinpoint the cause for the energy bug. This idea was first introduced in PowerScope [9]. However, PowerScope achieved precise clock synchronization by modifying the kernel to create synchronization pulses on an external GPIO pin (via the PC's parallel port). BattOr does not require kernel modifications, and its clock synchronization method works with today's smartphones and laptops, which often do not have external GPIO pins.

The necessary level of synchronization should be approximately one order of magnitude faster than the rate at which state changes on a device—for frame rendering and tracking threads, synchronization on the order of 100 µsec suffices. Software-based modeling permits perfect synchronization, in the sense that its "measurements" are a direct function of the activity on the phone. But software models, as we have discussed above, are not accurate. Internal power monitors report accurate measurement but do not provide the time resolution of measurement that's needed for synchronization. Specifically, they usually provide power measurements at 10ms intervals, whereas root cause identification needs an order of magnitude higher resolution.

External power monitors such as Monsoon are accurate and have sufficient time resolution, but do not provide any way to synchronize the power measurement with the device activity log. Specifically, since the external monitor is running on a different clock than the device being measured, timestamps are unreliable. BattOr is an external power monitor but it also provides a mechanism to precisely synchronize the power measurement trace with the device's logs. Also, the Monsoon is not portable, it must be plugged into mains power.

4. Fixing After attributing the bug to a root cause, the final step in the debugging process is to determine if it needs a fix and, if so, to implement that fix. Traditionally, implementing a fix would involve making iterative changes and performing multiple battery rundown tests to validate the fix. Efficiently fixing an energy bug benefits from having a power measurement tool that is accurate and can quickly collect sufficient data evaluate if the fix helped. The requirements for this step are the same as root cause identification, and hence the same arguments about related work apply.

Automation To integrate all of the above steps into a single, cohesive workflow, a power measurement tool must also permit a high degree of automation. Continuous integration systems have become an essential tool for software developers for speeding up the development process and improving software quality. These systems automatically find regressions by continuously building and testing new commits to the tree. Examples of continuous integration systems include *Buildbot* [3] which is used by Google Chrome and Firefox as well as Facebook's *Sandcastle*.

Combining the excellent coverage of continuous integration systems with the authoritative power measurements from hardware power monitors would signifi-

[1]https://www.chromium.org/developers/telemetry

[2]EProf used Monsoon power monitor measurements as ground truth.

cantly shorten the time between when an energy bug is introduced and when it is discovered and fixed. With continuous integration, commits to the tree could even be rejected immediately if they do not pass energy regression tests.

Performing power regression testing with a continuous integration system requires collecting power measurements without physical intervention. The problem is continuous integration systems must be able to control the device, and for smartphones, the control and charging channels are coupled in USB. Even if the device was controlled over another medium such as wireless, the device must be charged occasionally or powered externally so the battery does not drain.

This feature has not been addressed in any of the prior work. BattOr is the first system to demonstrate that it is feasible to automate hardware power measurement of smartphones, and we go even further to demonstrate that it is also possible to have automated hardware power measurements with laptops as well.

3 An Overview of the BattOr Workflow

BattOr provides developers with immediate, accurate power measurements, and synchronizes them with low-level logs. In this section, we provide an overview of how these features enable a far faster, more insightful workflow. We root our discussion with a real case study, in which we identified a previously uncaught energy regression in the Chrome web browser.

Setup BattOr comprises a small, noninvasive hardware device that interposes between a device (e.g., a smartphone or a laptop) and its battery, collecting accurate power measurements at very high rates (on the order of tens of microseconds). To use BattOr, a developer first physically connects the BattOr monitor hardware to the device they wish to test (e.g., a smartphone or a laptop). For smartphones, this involves opening the back case of the device to expose the battery, and snapping BattOr's battery interceptor onto one of several standard battery connectors in the phone, then connecting the BattOr power monitor hardware to the interceptor. We note that this requires no soldering or dangerous modifications to the battery; at worst, some phone cases can be a little tricky for a novice to open [12].

In our case study, we connected BattOr to a Nexus 5 smartphone, and chose to debug the Chromium browser. The Chromium project releases incremental builds of the browser after every few commits to the tree[3]. There were 39,850 versions of the browser available for download at the time of writing dating back to April 2013. We emphasize that collecting BattOr measurements requires no modifications to the software on the smartphone itself: the developer can then run their unmodified code through regression tests.

1. Discovery BattOr is compatible with continuous integration testing because it runs without requiring modifications to the software being tested; one simply runs the standard tests and BattOr complements them with fine-grained, accurate power measurement data. In our case study, we ran the WebKit 3D CSS animation benchmark called *Poster Circle*[4] on two Chromium builds: one at the time of this writing (#349421) and one from seven months earlier (#315477).

Each of these tests took exactly *two minutes* to run, and we collected BattOr's measurements during one minute of the animation running. The latest build consumed 100.16 J, and the earlier build consumed 92.64 J, revealing there has been an 8% regression. When we ran these tests, Google's Chrome team was not aware of this regression that BattOr had discovered.

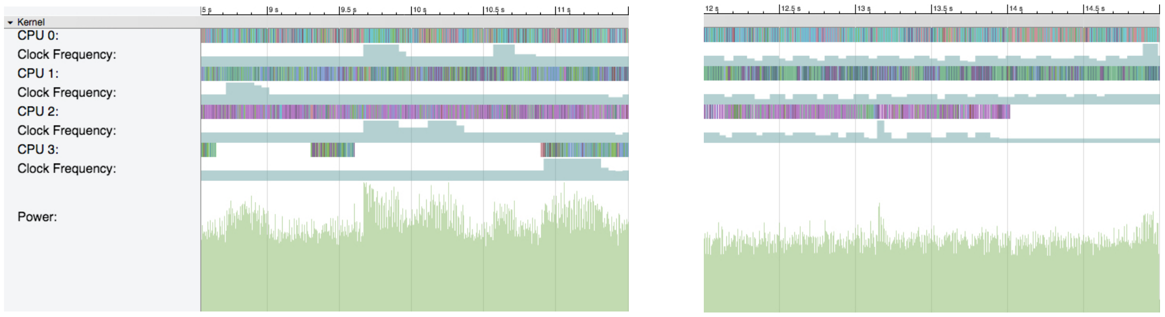
2. Triage We ran the 60 sec measurement on several other builds of the Chrome browser from before and after the regression. Altogether, these tests took under ten minutes; in practice, we envision developers taking these measurements with each commit, amortizing triage time. We found that the regression shows up in several of the builds before #349421, and in none of the builds before #315477, indicating that it is not an ephemeral issue with a single build, but rather that we have discovered a persistent energy bug.

3. Root cause identification BattOr replaces the traditional time-consuming, iterative process by *synchronizing* its fine-grained measurements with logs maintained by the kernel on, e.g., CPU activity and what threads are running. Google's *Systrace* [11] tool and the Linux kernel's ftrace collects these data, which are often used for Android system debugging.

Figure 1 shows three seconds of output of Systrace run on the pre-regression (right) and latest (left) versions of Chrome. We chose these three-second windows arbitrarily; this particular test is highly repetitive, and so both are representative of any other similarly sized window. Included in this trace are power logs from BattOr, sampled at 10 KHz, and synchronized with 100 µsec precision with Linux kernel ftrace log of the scheduler activity on each of the four of the Nexus 5's CPUs, as well as the CPUs' frequency over time.

[3]https://commondatastorage.googleapis.com/chromium-browser-snapshots/index.html?prefix=Android/

[4]https://www.webkit.org/blog-files/3d-transforms/poster-circle.html



Latest Chrome commit (#349421) Pre-regression Chrome commit (#315477)

Figure 1: Root causing the regression by comparing precise power traces from the BattOr with logs from the kernel.

At first glance, it is readily evident that there is an energy regression: the latest commit's power measurements often peak above those of the pre-regression commit. Additionally, we draw two important conclusions from this trace simply by visual inspection. First, the clock frequency of the CPUs trend higher than those in the pre-regression commit, and with each spike in clock frequency, there is a corresponding spike in the power level. Second, the latest commit often offloads processing onto the third core, while the pre-regression commit does not use it at all. These two observations—increased frequency and extra CPU offloading—indicate that there has been an increase in the amount of CPU processing required to render the frames in the 3D CSS animation test. The fact that the increase in CPU frequency is sporadic indicates that the increased load arises from some particular subset of frames.

To find the specific commit that introduced this bug, we "bisected" the 39,850 commits between the two we had originally tested. Bisection involves effectively performing a binary search over the commits, running the same Poster Circle test on each and collecting BattOr measurements to pinpoint when the bug was introduced. This bisection ran for 15 rounds, taking a total of 15 minutes. Figure 2 shows the energy consumption of the commits before and after the culprit commit found by bisection. From these results, the potential culprit becomes clear: all commits prior to #320096 consumed lower energy, and there has been a consistent energy regression as of no later than #320097.

Finally, we consulted the commit logs to determine which code modification ultimately resulted in the regression. The description of #320096's commit notes that the developers added sorting of 3D layers into a Binary Space Partitions (BSP) tree to make it easier to render 3D layers the correct sorted order. We therefore concluded that the additional computing of the BSP tree is the cause for the increase in CPU load that results in the increased power consumption. We reported this to the Chromium team, and they confirmed that this was indeed the cause of the energy regression.

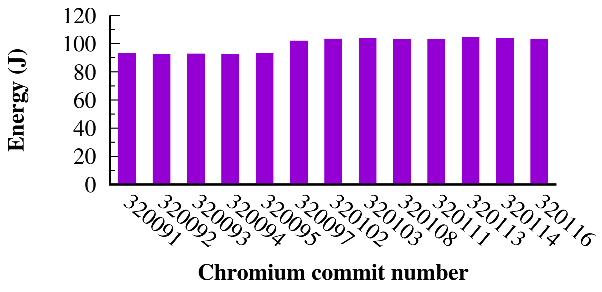


Figure 2: A Chromium 3D rendering energy regression appears between commits #320095 and #320097.

We note that we were able to identify the root cause of this energy bug *in minutes*, without having a deep familiarity with the relevant rendering code.

4. Fixing BattOr's fine-grained, synchronized measurements permit small, iterative changes that are more conducive to quickly understanding and fixing a bug. To minimize the time it would take a developer to fix an energy bug, each developer would have a BattOr hardware power monitor incorporated into their testing rig. It is therefore important that a power monitor be deployable to a large group of developers.

To conclude our case study, we reported the bug to the Chromium team. Their developers reported that the Poster Circle benchmark that we tested with is a pathological case for BSP trees, and so there is no need for a fix. This however provides a good example of how much less time is wasted to reach the conclusion that a fix does not need to be made: following traditional steps, the discovery, triage, and root cause attribution phases would have spent days to ultimately conclude that no further action was needed, diverting resources and developer time from bugs that did require fixes.

Summary BattOr replaces battery rundown tests with high-fidelity power measurements, resulting in a significant speedup of each stage in the debugging process. Moreover, BattOr can capture measurements in tandem with other regression tests, allowing it to be naturally integrated with today's typical software development processes—if BattOr could be deployed to each developer, then energy regressions could become a standard check for all code commits, at last putting energy on equal footing as more traditional performance metrics. In this section's case study, a developer using BattOr could have identified the regression, attributed the root cause, and provided justification in her code commit, all within minutes (as we did), as opposed to spending days or more with battery rundown tests. In the remainder of this paper, we detail how BattOr achieves these properties.

4 BattOr: Hardware Design

BattOr can measure the power consumption of a device without soldering wires to it. BattOr also is portable—its circuit board (shown in Figure 3) measures 1.24 in by 1.99 in—and it is powered directly off the battery on the device under test without affecting the power measurements. In the following section we describe how BattOr achieves each of these properties.

4.1 How does the BattOr connect & measure without soldering

BattOr physically measures power by intercepting the wires between the power supply (e.g., battery) and the load (e.g., smartphone) *without cutting wires or soldering*. We present two different solutions to this problem, one for smartphones and one for laptops.

Intercepting smartphone batteries A naive approach to power measurement would be to intercept the USB charging connector and remove the phone's battery during the experiments. However, because USB does not supply enough current to power the entire phone. So the battery must be connected, which in turn will mean that the device will be powered by the battery, hence the only option is to intercept the battery connectors between the battery and the device.

Our key observation here is that in spite of the vast array of phone models, there are in fact only a few (3–4) different battery connectors that are used in these models. The reason is consolidation in the battery connector industry. Smartphones are becoming smaller and thinner which has put an interesting strain on the physical design of smartphone batteries: their connectors to the

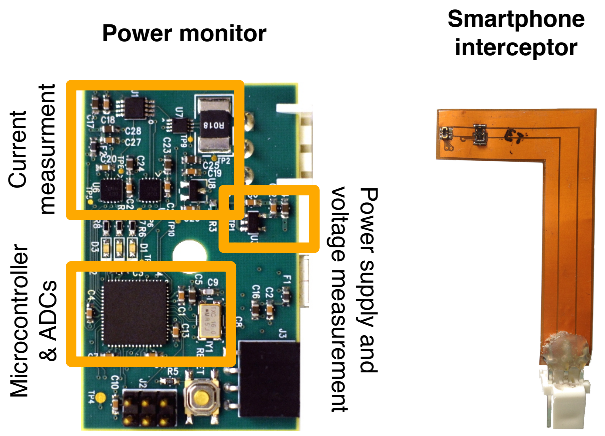


Figure 3: BattOr's power monitoring hardware consists of three main parts, the power monitor circuit (left–actual size), the battery interceptors for smartphones (right), and the laptop power adapter interceptors (not shown).

phone must also become thinner and smaller. It difficult to design a battery connector that can handle several amps of current draw from today's complex smartphones, but still be very small and thin (with limited surface area to have a low resistance connection with the phone). Consequently, only a few commercial electronic connector manufacturers have mastered the technology of designing and manufacturing such high current low profile battery connectors. As a result, there are only a few standard models of battery connectors that are used in most of today's smartphones.

We provide support for this observation by tearing down 12 popular phones and finding out what battery connectors they have within them. Figure 4 shows the results of this teardown study. We found that there are two connectors that are used by at least four phones each, even across different manufacturers. Two of the connectors were only used by one manufacturer, but for several versions of their devices. We also found that these connectors are built by familiar names in the connector industry such as Hirose and SMK, they are not one offs from smaller suppliers. The Hirose and Panasonic parts are even popular enough that they are available for immediate order in low quantities on Digikey, an online electrical part retailer.

Given this trend, we can thus build a few specialized battery interceptors for the primary types of battery connectors. These interceptors can be *snapped* directly onto the device's mainboard and battery connection points. The remaining question is, how do we get the power wires outside of the phone's enclosure so it can be attached to the BattOr external power monitor? We discovered that Flexible PCBs could be the answer. Flexible

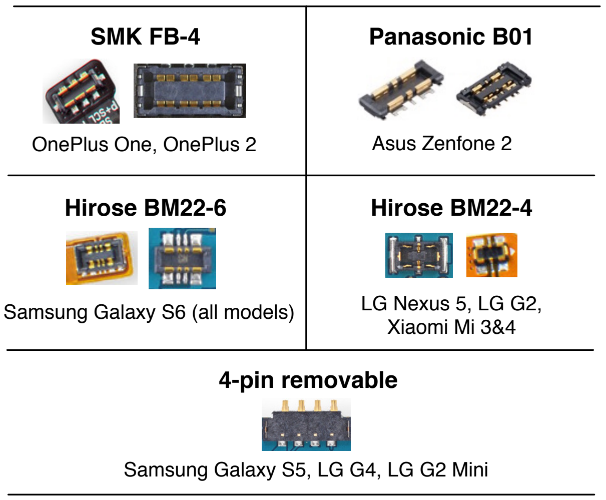

SMK FB-4
OnePlus One, OnePlus 2
Panasonic B01
Asus Zenfone 2
Hirose BM22-6
Samsung Galaxy S6 (all models)
Hirose BM22-4
LG Nexus 5, LG G2, Xiaomi Mi 3&4
4-pin removable
Samsung Galaxy S5, LG G4, LG G2 Mini

Figure 4: There only a few commodity battery connectors that are used in many of today's Android smartphones.

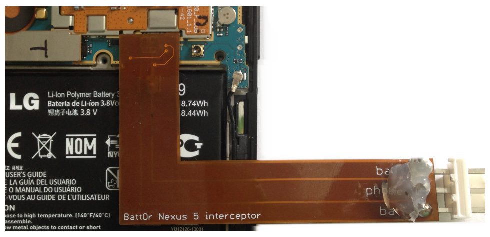


Figure 5: BattOr's custom Flex-PCB battery interceptor snaps onto the Nexus 5's Hirose BM22-4 battery connector. The back case can then be closed around the connector (not shown).

PCB is a very thin circuit board that can be snaked within the phone's enclosure and snapped onto the battery connectors while only adding very tiny bulk, and is flexible enough that it can be bent around the phone to hold the BattOr and phone in one hand. Flex PCB technology is becoming very inexpensive. We built 70 Nexus 5 interceptors in two weeks for $42 each; the price would drop at larger quantities and with longer lead times.

Intercepting laptop power adapters We also designed the BattOr for use with laptops, as they are battery dependent devices that have significant reported power issues [22, 25], but have mostly been overlooked as a platform that needs power monitoring. Our laptop interceptors do not require soldering, but unfortunately the laptop interceptors are not portable, the laptop's power adapter must be plugged into mains power. This is an acceptable trade off to make because the energy consumption of laptops is not significantly affected by the environment (they often lack cellular modems and arrays of

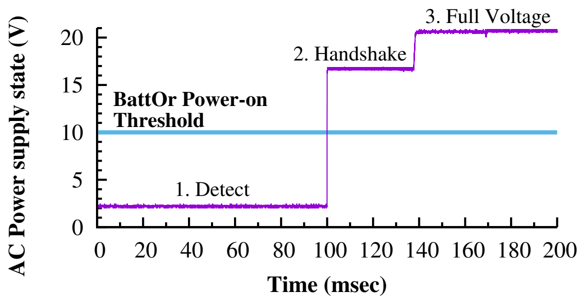


Figure 6: Apple MagSafe 2 AC adapters use resistance measured at the detection phase to set the Handshaking voltage (connection to 15" MacBook Pro shown)

.

sensors).

Unlike smartphones, laptop battery connectors are unfortunately not yet standardized, and they can be very difficult to access. Further, due to the high high voltage (e.g., 14 V) and high current (e.g., 3 A) of laptop batteries, it can be dangerous to intercept them when they have any charge. We wondered then if maybe we could instrument power through the external power adapter instead.

Fortunately, laptop battery charging design is different from that of smartphones. Laptop power adapters are designed to be able to both fully power the laptop as well as charge the battery at the same time. This is why laptops can be turned immediately even if the battery is empty, whereas smartphones need time to charge the battery before they can be powered on.

When a laptop battery is fully charged, the charging circuit disconnects the battery entirely from the laptop. This is unlike the smartphone where the battery remains connected because the USB can not supply enough instantaneous current to run the phone. This presents an opportunity to instrument just the power adapter of the laptop in order to measure the laptop's power consumption.

Our laptop power adapter interceptor design is simply a circuit that connects between the laptop's power adapter and the plug. Intercepting this connection does not require soldering by the end user. Users simply buy a universal laptop power adapter that has detachable DC cables, then the user simply plugs the interceptor in between the cable and the power adapter.

However, intercepting Apple's MagSafe power adapters with a BattOr requires a special circuit to keep the BattOr disconnected until the power adapter handshakes with the laptop. Apple's MagSafe power adapters detect the voltage that they should operate based on the resistance of the laptop when the adapter is connected. The BattOr adds resistance to the connection, so it will prevent the adapter from detecting the laptop. The MagSafe's voltage detection process is shown in

Figure 6. The BattOr MagSafe interceptor circuit keeps the BattOr disconnected until the handshake completes by adding p-Channel FET transistor with a threshold at 10 V between the BattOr and the power adapter. An additional problem with MagSafe power cables is that the DC wire is not user detachable. However, the need for these modified power cables creates a market for selling intercepted MagSafe power supplies.

4.2 How does the BattOr measure power?

A diagram of the BattOr power measurement circuit is shown in Figure 7, and a picture of the circuit is shown in Figure 3. The current BattOr design is based on an Atmel XMega 192A3U System-on-Chip. Voltage and current measurements are captured with the XMega's built-in 11-bit ADCs, their sample rate is 10 kHz (100 µsec sampling interval). We use a shunt resistor with 1% tolerance to measure the current. In low-current gain mode BattOr can achieve a precision of 1 mA with an accuracy of ±4 mA and in high-current gain mode a precision of 325 µA with accuracy of ± 1.3 mA. We used a Keithley 2420 precision current source for this test.

BattOr overcomes two power measurement challenges presented by smartphones: (1) smartphones operate at very low current (<10 mA) when they are suspended and very high current (>1 A) when they are active, also (2) powering the BattOr from the smartphone's battery for portable operation may affect the power measurements.

Auto gain Power measurement of smartphones is complicated by the large range of power signals. When the device is in idle mode it draws less than 10 mA of current, and when it is in active mode it consumes more than 1 A. Due to the limited dynamic range of any ADC that is measuring current, the suspend current could have limited precision when sampled with the same amplification as the high current. This presents a problem for observing bugs that affect the energy consumption of the device when it is suspended. The BattOr solves this resolution problem by having a software controlled amplifier on the current measurement gain circuit. When the current drops below a certain threshold for a set period of time, the firmware switches the amplifier to high gain, and when the current rises above a certain threshold, it switches the amplifier to low gain.

Powering the BattOr from the device's battery Unlike existing power monitors, the BattOr does not require any power from either a mains or external battery. The BattOr is powered from the battery on the smartphone or the power adapter of the laptop.

One might think that drawing from the battery would affect the power measurement itself. Brouwers et al. estimated that this aspect of the BattOr's design causes significant error [2] to justify using an external battery. Although any load on a battery will affect its voltage due to the internal resistance of the battery, because of a confluence of interesting design points, this distortion does not reduce the accuracy of a power measurement.

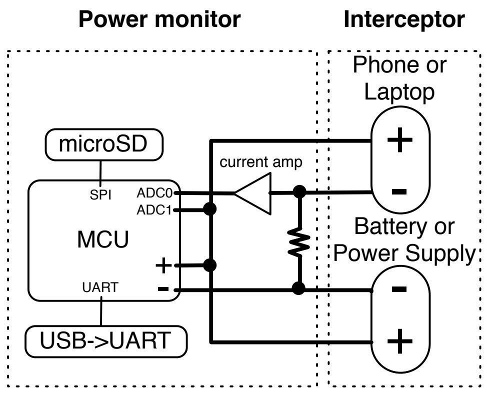


Figure 7: BattOr power measurement overview.

The reason is smartphones and laptops have voltage regulation in them to convert the higher DC input voltage (generally 4 V for smartphones, 19 V for laptops)—that also decays as the battery drains—to the lower voltage of the components in the device (generally 1.8V). These voltage regulators are often switching regulators, and by and large switching regulators are known to be constant power. This means that within reasonable bounds of the input to the voltage regulator, if the voltage drops down, then the current consumption of the regulator will proportionally increase to maintain constant power. Consequently there is no significant effect on the total power consumption by the BattOr decreasing the voltage periodically. We evaluate this phenomenon empirically in Section 6.2.

The BattOr does not have a significant effect on smartphone battery life. It consumes at most 250 mW when writing to the SD card. For an 8 Wh battery of a Nexus 5, for example, this would result in 32 hours of runtime when the smartphone is suspended. When the phone is active, the 250 mW consumed by the BattOr is a small fraction of the active power consumption, which can be 2.8 W when the GPU is actively rendering frames.

5 BattOr: Software Design

In this section, we describe how the BattOr software enables accurately syncing power measurements with logs without soldering wires, and automation for continuous integration testing. The BattOr software (1) modulates a sync signal from the phone to the BattOr using the measurements as a communication channel, (2) recovers and subtracting that synchronization signal from the power measurement, and (3) controls the battery charging circuit so the phone may be connected to USB data channel for automated continuous testing, but while USB charging is disabled—since it would disrupt the measurement.

5.1 Synchronization without soldering

While BattOr can sample power at 10 kHz, the operating system and applications on the device under test must also log power-relevant events. Synchronization is the process of aligning power logs with event logs despite the lack of a shared clock and the potential for relative clock drift. With combined synchronized logs, developers can precisely determine how much energy each portion of the application is consuming.

Power monitoring approaches based on invasive soldering connect a trigger wire to the phone to provide clock synchronization. For example, NEAT [2] by Brouwers et al. requires de-soldering the phone's vibrating alert motor, removing it, and soldering in its place a line to trigger their power meter.

BattOr uses the current draw itself as a means of introducing a recognizable synchronization signal into the power measurement logs. This is similar to Nemo [29], a power monitoring device for sensor motes. Nemo modulates the current load to send commands to the power monitor, we modulate the current to synchronize the power monitor's clock with the device's clock.

Smartphones include a LED flash for the camera that is used only when taking a picture or operating as a flashlight: it is unlikely to be used as part of an application whose power consumption is being measured, making it available for use in synchronization. The flash also can turn on and off at least one order of magnitude faster than the shortest thread execution times ($\sim$1 msec). We modulate the flash a pseudo random pattern to aid in aligning the modulated signal to the log of device timestamps when the LED was cycled on and off.

Figure 8 shows the bits of the pseudo random sequence as well as the power measurement of the device while the sync signal is modulated with the flash. he LED flash consumes 2 W on the Nexus 5 smartphone. This is a significant amount of power that is not drowned out by the varying, substantial power consumption on the device during this test. The background power draw in this test was an active screen while scrolling on the home screen.

5.2 Programmatically disabling USB charging for continuous testing

We designed BattOr to support continuous integration testing for energy consumption, where each commit into a source tree is potentially run through a series of energy

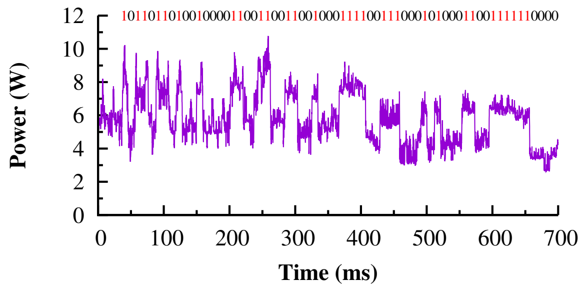


Figure 8: Pseudo random sequence for clock synchronization modulated into the power measurement by turning on and off the camera flash LED for 10 msec every bit.

tests to immediately notice a regression. For continuous testing smartphones would need to be capable of running energy tests unattended. On many smartphones, automated software installation and testing can be performed over USB; however, USB also charges the phone. Allowing USB to charge the phone would make the power measurements that BattOr collects from the battery connection inaccurate. Cutting USB power entirely is not desirable either, since we do need the battery to charge between tests, and it also violates the USB specification which requires the USB power connection to detect the presence of an attached USB device.

To make automated energy testing possible requires the ability to disable USB power supply to the phone. Simply disabling charging of the battery is not enough, because the USB connection could still partially supply power to the smartphone.

We discovered that smartphones often have the ability to control the connection of the USB power supply *in software*. All smartphones feature charge controllers either in the Power Management Integrated Circuits (PMICs), or in a separate battery controller chip such as the Texas Instruments bq24192 in the Nexus 5.

These charge controllers often feature software control over a p-Channel FET transistor that can cut completely the USB supply current. There are three reasons to cut the USB supply current. First, overvoltage protection guards against low-quality USB power supplies that might incorrectly pass too high a voltage. Although overvoltage protection could be implemented entirely in hardware, it is often a software-controlled feature. Second, temperature control can require software control to modulate the temperature of the battery by enabling or disabling charging. There have been many instances of overheating causing batteries to swell, and the algorithms for controlling temperature are still being refined in software. Finally, software control over input power can be used in assembly line testing. Taken together, we expect that software control over the input power will remain.

Device	Charge disable command
Nexus 4	echo 1 > /sys/module/pm8921_charger/parameters/disabled
Nexus 5	echo 0xCA > /sys/kernel/debug/bq24192/INPUT_SRC_CONT
Galaxy S5	echo 1 > /sys/class/power_supply/battery/test_mode && echo 0 > /sys/class/power_supply/sec-charger/current_now
Galaxy S6	echo 1 > /sys/class/power_supply/battery/test_mode && echo 0 > /sys/class/power_supply/max77843-charger/current_now
Nexus 9	echo C > /sys/bus/i2c/drivers/bq2419x/0-006b/input_cable_state

Table 1: Popular Android phones can disable USB charging in software.

We performed a study where we inspected the kernel drivers for many popular Android smartphones to see if the PMIC exposed the hardware control that we need to disable charging. Specifically, we searched for for enable_charging and disable_charging functions as well as the string "BATFET", and considered specifically how the CURRENT_NOW and CURRENT_MAX properties are set. These properties are exposed in a driver-specific way to userland through sysfs, and often they require root permissions to be able to modify them. We verified that the USB charging can in fact be disabled via sysfs by using an intercepted USB cable connected to a BattOr power monitor.

Table 1 shows how the USB power supply can be disabled in software for a few Android devices from this study. The table shows that there is no consistent way of disabling USB power, it is on a driver-by-driver basis quite different. However since for Android phones these drivers are all open source, it is not difficult to find the userland interface to disabling charging.

6 Evaluation

6.1 Accuracy of clock synchronization

The modulated clock synchronization signal may not be precisely as planned: we rely on the kernel scheduler to awaken the synchronization process after a nanosleep(). If other processes are running, or the timer granularity is insufficient, individual sleep calls may be longer than intended. Because the signal may thus be distorted by contention for the processor, we must recover the actual timing of when the LED flash was cycled on or off for precise synchronization.

The simplest approach, invoking gettimeofday() at user level near the call to toggle the flash state has significant potential for error based on processor scheduling. Since Android phones are based on the Linux kernel, however, we have access to a better timestamp that more closely represents the time the LED changes state. There is an API in the Linux kernel for controlling the voltage regulators that set CPU frequency through its voltage, turn off peripherals such as the LCD backlight when they are not in use, and, fortuitously, control power to the LED flash.

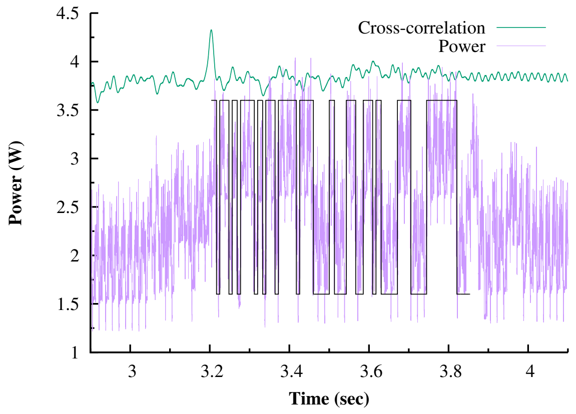


Figure 9: Cross-correlation (top) of injected pseudo random signal (square wave, aligned) as observed in the power trace (main series with noise). The correlation spike is clear.

Use of this API can be timestamped by the Linux *ftrace* system, which can also be used to log other power relevant events. To get in-kernel timestamps of LED flash events requires tracing the voltage regulator API calls, regulator_enable() and regulator_disable(). The timestamps of these events can be copied from the phone after the experiment with any other logs. To align the timestamps from the phone with power measurements from BattOr requires straightforward signal-processing techniques, finding the peak of the cross-correlation between the pseudo random sequence as it was sent to the LED driver and the observed power draw.

We next show that the synchronization is precise by showing how narrow the peak correlation turns out to be. If the peak were not so pronounced, there would be uncertainty as to when the synchronization signal was sent. An example of this correlation is shown in Fig 9. The cross-correlation is normalized by dividing by the average power over the pseudo random window. Normalizing prevents relatively poor correlation at high power from appearing as a better match than good correlation at low power. Note that there is one alignment that is far better correlated than any other.

6.2 Reproducible power measurements

Power regression testing requires measurements that are reproducible—a developer must be able to trust that a possible regression is not due to noise in the power measurement. Power measurements taken at the battery may not be reproducible, because battery voltage decreases as the battery discharges. As the battery voltage decreases toward the operating voltage of the circuits in the phone

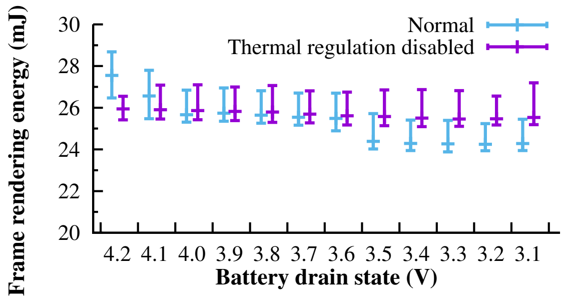


Figure 10: Without thermal regulation, the Nexus 5 does not become significantly less efficient as the battery drains (5pct, median, 95pct). Note the y-axis starts at 20 mJ.

(∼1.8 V), the voltage regulators in the smartphone operate with increasing efficiency.

To test the effect of the decreasing battery voltage on the reproducibility of power measurements, we ran the Poster Circle animation on a Nexus 5 while recording the energy consumed to render each frame. The "Normal" bars in Figure 10 show the distribution of per-frame energy consumption in each 100 mV bin of the battery discharge cycle. It appears that as the battery drains, the energy consumed to render the frames decreases significantly: the median drops by 10%. This seems to confirm our hypothesis that battery voltage has a significant effect on the power consumption of the device.

However there could be another, more insidious cause for the drop in power consumption: *thermal regulation*. Smartphones are built with powerful processors, but they are enclosed in a plastic case with no active cooling. Smartphones therefore rely on aggressive thermal regulation to progressively throttle the frequency of the CPUs, GPUs, and even radios as the phone gets hotter [24].

To test the effect of thermal regulation on the power consumption of the device, we disabled it manually by renaming the thermal regulation configuration file */etc/thermal-engine.conf*, and reran our experiment. We plot both results in Figure 10. Surprisingly, the power consumption stays mostly stable except for edge cases when the battery charge is full and almost empty. This indicates that in the prior experiment it was increasing thermal throttling, and not decreasing battery drain that produced the inconsistent per-frame energy.

We conclude from these results that thermal regulation has a significant effect on the reproducibility of power measurements. Thermal regulation should be monitored carefully while collecting power measurements, and disabled when adequate cooling is possible.

6.3 Case study: Comparing Video Playback on Safari and Chrome for Mac

We conclude the evaluation with a detailed case study describing how BattOr was used to reduce energy consumption in Google Chrome on Mac OSX by 38% by the Chrome development team. In this case study the Chrome developers used the MagSafe intercepted the power adapter. The study shows how the precision and accuracy of BattOr enabled a significant energy saving in an application used on two out of every five laptops.

1. Discovery and Triage Both Chrome users and the popular press had widely reported on Chrome's battery inefficiency compared to Safari on Mac OSX [25]. To put the gap in context, users reported that their laptops lasted over three hours less if they played YouTube videos on Chrome compared to Safari.

BattOr made it quick and easy to reproduce this bug. We collected power traces of a 360p quality YouTube video playing on Safari and comparing it to Chrome for Mac for the version of Chrome that was used around the time of the discovery of the problem (#300000). The energy consumption of Chrome was found to be 37% higher than Safari when rendering the same video.

2. Root cause Given that the energy consumption regression happened when videos were played in Chrome but not in Safari, the team's hypothesis was that the frame rendering in Chrome compared to Safari was consuming too much energy. To check this, the first step was visually inspecting the power measurements in terms of per-frame rendering (16 msec) energy as produced by BattOr while a video is being played. Delving into this problem at that granularity is feasible because of BattOr's 10 kHz sample rate. Figure 11 shows per-frame power traces of Safari (left) compared with Chrome near the time of the discovery of the problem (middle). Each of the colored lines represents the power consumption of one frame. Note that these are collected at 10 kHz on the BattOr, so there are 160 power samples for the rendering of each frame (16 msec at 60 Hz). One can clearly see that for Safari, the power consumption for rendering frames is very stable, before each frame appears on the screen it is rendered with a bit of energy, then the GPU goes to sleep until the rendering of the next frame. Chrome's power signal, however, has significant variability.

For this bug, just looking at the plot of power consumption of Chrome for these rendering tasks was enough to identify the root cause. In this case the bug is a comparison bug, where the regression is apparent when seen in comparison with software running on the same platform doing the same activity but with less power (i.e., Safari playing the same video).

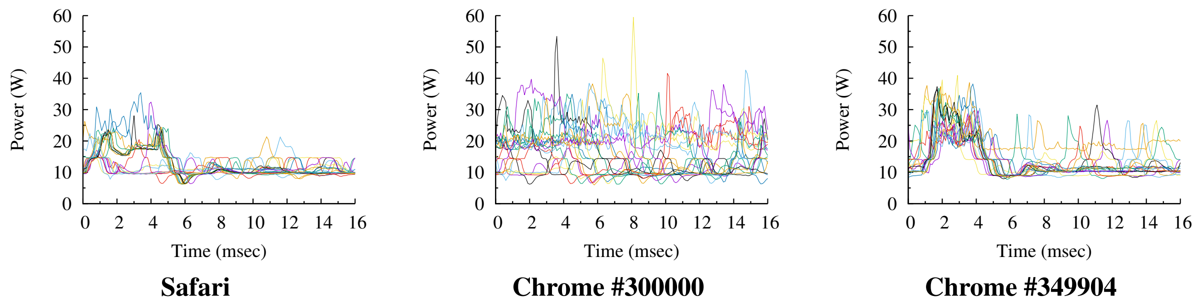


Figure 11: Per-frame power consumption on a 15" MacBook Pro of Safari and Chrome while playing a YouTube video. Chrome consumed 38% more energy than Safari in rendering frames for at least a year; Chrome at commit #300000 (and earlier) shows this. Based on the comparison of the per-frame power plots the root cause was identified and it was recently fixed with now only a 13% gap remaining.

3. Fixing and Verification After isolating the root cause around frame rendering, the developers realized that there were two main problems, both of which were around rendering API choices in OSX. At the time the API choices were made, the expectation was that they wont have any impact on power or in fact might help improve it. But that expectation could not be verified because of the lack of tools like BattOr, so design choices were made in a blind fashion with regard to energy.

The first issue that the Chrome team found was that the OSX API that was used to push frames to the screen, CAOpenGLLayer, while it looked promising because it avoided an extra copy, was found to in fact introduce a significant energy regression when tested with BattOr. This was unknown to the developers as this "feature" was not documented [4]. In contrast, a different API in OSX, the IOSurfaces API [6] was found to not have the same issue when tested with BattOr.

The second issue, which was known beforehand, was that Chrome's rendering on Mac was not as efficient as it could be, because the entire screen was refreshed on each rendering operation. After seeing the benefits of the first fix, the developers realized that implementing a partial screen swap that only updates the parts of the screen that had changed would likely bring down frame rendering energy too [5]. After this fix was implemented and tested, BattOr helped verify that the energy consumption of Chrome for frame rendering was now on par with Safari. Measurements on the latest build of Chrome at the time of writing (#349904) showed there was only a 13% gap remaining.

7 Conclusion

To find and fix energy bugs, teams of developers require tools that allow them to understand how their code consumes power. In this paper, we have identified the unique set of challenges facing such a tool—accurate measurements, portability, and no-soldering—and have introduced BattOr, the first power monitoring system that meets these challenges. BattOr does so with a set of novel techniques that could be applied more broadly, including: hardware for interposing between a battery and a device without requiring soldering, automated charge disabling, and synchronization without soldering. Power monitoring tools are best evaluated with a simple question—*can they actually help developers find and fix bugs?*—and we have shown with several case studies that BattOr was responsible for identifying a costly bug in Chrome on OS X and a potential regression in Chrome for Android.

References

[1] ARM. big.little technology. http://www.arm.com/products/processors/technologies/biglittleprocessing.php.

[2] N. Brouwers, M. Zuniga, and K. Langendoen. NEAT: A novel energy analysis toolkit for free-roaming smartphones. In *Proc. ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2014.

[3] Buildbot. Buildbot: The continuous integration framework. http://buildbot.net.

[4] ccameron@chromium.org. Mac chrome compositor power. https://docs.google.com/document/d/19YX5WYnpJj-h9nMM4r5WwlGOT99OQ7aEAm2DhWl-Ltg/edit?usp=sharing, June 2015.

[5] ccameron@chromium.org. Mac partial swaps. https://docs.google.com/document/d/1iWb1Y0ubpVZ0CKr1duA2EvQQfZd5mh1EE2Ha8rsJYnA/edit#heading=h.k8s4a4o2fcy5, Apr. 2015.

[6] ccameron@chromium.org. Mac: Use IOSurfaces as CALayer content. https://code.google.com/p/chromium/issues/detail?id=510087, July 2015.

[7] ccameron@chromium.org. Power consumption elevated when drawing relative to competition. https://code.google.com/p/chromium/issues/detail?id=484304, May 2015.

[8] P. Dutta, M. Feldmeier, J. Taneja, J. Paradiso, and D. Culler. Energy metering for free: Augmenting switching regulators for real-time monitoring. In *Proc. ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2008.

[9] J. Flinn and M. Satyanarayanan. PowerScope: A tool for profiling the energy usage of mobile applications. In *Proc. IEEE Workshop on Mobile Computer Systems and Applications*, 1999.

[10] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *Proc. Symposium on Operating Systems Design and Implementation (OSDI)*, 2008.

[11] Google. Systrace. http://developer.android.com/tools/help/systrace.html.

[12] iFixIt: The Free Repair Manual. https://www.ifixit.com/.

[13] Intel. Intel power gadget. https://software.intel.com/en-us/articles/intel-power-gadget-20, Jan. 2014.

[14] S. Keranidis, G. Kazdaridis, V. Passas, G. Igoumenos, T. Korakis, I. Koutsopoulos, and L. Tassiulas. NITOS mobile monitoring solution: realistic energy consumption profiling of mobile devices. In *Proc. Conference on Future Energy Systems (e-Energy)*, 2014.

[15] C. Mims. Our one wish? longer battery life. http://www.wsj.com/articles/our-one-wish-longer-battery-life-1424650700, Jan. 2015.

[16] Monsoon. PowerMonitor. http://www.msoon.com/LabEquipment/PowerMonitor, 2011.

[17] A. Murray. What do consumers want? better batteries, not wearables. http://fortune.com/2015/01/07/what-do-consumers-want-better-batteries-not-wearables, Feb. 2015.

[18] NetMarketShare. Market share statistics for internet technologies. https://netmarketshare.com/, Aug. 2015.

[19] A. Pathak, Y. C. Hu, and M. Zhang. Bootstrapping energy debugging on smartphones: A first look at energy bugs in mobile devices. In *Proc. Workshop on Hot Topics in Networks (HotNets)*, 2011.

[20] 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. European Conference on Computer Systems (EuroSys)*, 2012.

[21] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proc. European Conference on Computer Systems (EuroSys)*, 2011.

[22] PCWorld. Intel, microsoft working to squash windows 10 battery life bug. http://www.pcworld.com/article/2952393/windows/intel-microsoft-working-to-squash-windows-10-battery-life-bug.html, July 2015.

[23] J. Rossignol. Google improving Chrome for OS X performance to better rival Safari, 2015.

[24] O. Sahin and A. K. Coskun. On the impacts of greedy thermal management in mobile devices. *IEEE Embedded Systems Letters*, 2015.

[25] V. Savov. Chrome is still a threat to your MacBook's battery. http://www.theverge.com/2015/4/10/8381447/chrome-macbook-battery-life, 2015.

[26] A. Schulman, T. Schmid, P. Dutta, and N. Spring. **Demo:** Phone power monitoring with BattOr. In *Proc. ACM Conference on Mobile Computing and Networking (MobiCom)*, 2011.

[27] F. Xu, Y. Liu, Q. Li, and Y. Zhang. V-edge: Fast self-constructive power modeling of smartphones based on battery voltage dynamics. In *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

[28] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. Conference on Hardware/Software Codesign and System Synthesis*, 2010.

[29] R. Zhou and G. Xing. Nemo: A high-fidelity non-invasive power meter system for wireless sensor networks. In *Proc. ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2013.