Deconstructing the Energy Consumption of the Mobile Page Load

YI CAO, Stony Brook University
JAVAD NEJATI, Stony Brook University
MUHAMMAD WAJAHAT, Stony Brook University
ARUNA BALASUBRAMANIAN, Stony Brook University
ANSHUL GANDHI, Stony Brook University

Modeling the energy consumption of applications on mobile devices is an important topic that has received much attention in recent years. However, there has been very little research on modeling the energy consumption of the mobile Web. This is primarily due to the short-lived yet complex page load process that makes it infeasible to rely on coarse-grained resource monitoring for accurate power estimation.

We present *RECON*, a modeling approach that accurately estimates the energy consumption of any Web page load and deconstructs it into the energy contributions of individual page load activities. Our key intuition is to leverage low-level application semantics in addition to coarse-grained resource utilizations for modeling the page load energy consumption. By exploiting fine-grained information about the individual activities that make up the page load, *RECON* enables fast and accurate energy estimations without requiring complex models. Experiments across 80 Web pages and under four different optimizations show that *RECON* can estimate the energy consumption for a Web page load with an average error of less than 7%. Importantly, *RECON* helps to analyze and explain the energy effects of an optimization on the individual components of Web page loads.

CCS Concepts: • **Networks → Network performance modeling**; • **Computing methodologies → Modeling methodologies**;

ACM Reference format:
Yi Cao, Javad Nejati, Muhammad Wajahat, Aruna Balasubramanian, and Anshul Gandhi. 2017. Deconstructing the Energy Consumption of the Mobile Page Load. *Proc. ACM Meas. Anal. Comput. Syst.* 1, 1, Article 5 (June 2017), 25 pages.
DOI: http://dx.doi.org/10.1145/3084443

1 INTRODUCTION

Mobile Web page performance is critical to content providers [1, 25], service providers [19], and users [3], as Web browsers are one of the most popular apps on phones [8]. Slow Web pages are known to adversely affect profits [1, 25] and lead to user abandonment [3]. The importance of Web page performance extends even to mobile apps: a recent Evans Data survey showed that three-quarters of mobile app developers polled said they include or plan to include mobile browsers/HTML5 in their apps [2]. Not surprisingly, several optimizations have been proposed to improve mobile Web performance [13, 14, 17, 19, 21].

An important problem that is often overlooked is the *energy consumption* of Web page loads. Mobile devices are severely constrained by energy; in fact browser vendors tout their effect on battery life as a critical selling

The authors would like to thank Pavan Maguluri for his help. This work was supported by the National Science Foundation through the grant CNS-1551909, grant CNS-1566260, grant CNS-1617046, grant CNS-1464151, and a Google Research Grant R2-2015-839.

DOI: http://dx.doi.org/10.1145/3084443

point [5, 16]. The problem is that energy and performance are separate metrics. While Web optimizations or any changes to the Web ecosystem are often studied in terms of performance, their effect on energy is not easy to measure. Not knowing the effect of a Web enhancement on energy consumption can have severe consequences; a recent software update to Chrome resulted in excessive battery drain, leading to severe backlash [6].

Today, content providers and browser vendors use power monitors to measure the energy consumption of the page load process before and after an enhancement. Power monitors that are in-built in the phone are known to be grossly inaccurate [43]. Instead, external power monitors such as the Monsoon power monitor are commonly used to measure power consumption of Web page loads more accurately. These external monitors have higher accuracy, but are not easy to use (§2). Importantly, the *power monitors only provide aggregate power consumption* of the device at any time, without providing any information on how much power was consumed by the *individual page load activities* such as image loading or JavaScript evaluation, or how an enhancement impacted the power consumption of these individual activities. For example, we find that certain ad blockers [4] that remove ads and other malware significantly increase energy consumption. A power monitor can detect this energy increase, but can *not* explain *why* the energy changes.

Our goal in this paper is to provide: (a) quick, accurate, and fine-grained estimations of the power and energy consumption for a page load instantiation, and (b) meaningful analyses of the power profile of the Web page load.

Unfortunately, estimating the energy consumption of a page load is challenging because of:

- **Transience**: The page load process is relatively short-lived, ranging from several milliseconds to a few seconds. Fine-grained resource monitoring on such short timescales to model energy consumption is known to incur substantial *overhead* [23, 33]; our experiments on a Galaxy S4 reveal that resource monitoring at a frequency of 100 Hz can incur 30% CPU overhead.
- **Complexity**: Web pages are complex [39]. A Web enhancement can have widely varying effects on different page load activities.

Thus, studying the energy impact of a Web enhancement on page loads requires understanding its effects on *each* page load activity. Existing approaches to analyzing mobile energy typically focus on profiling and modeling the resource consumption of the device during execution [33, 34] (see §8). Such approaches consider long-running services and apps such as games, audio, and video streaming [23, 43], for which low-overhead, coarse-grained resource monitoring suffices. For page loads, however, coarse-grained resource monitoring is *not* sufficient to analyze the energy consumption of individual, short-lived, page load activities.

We present *RECON* (REsource- and COmpoNent-based modeling), a modeling approach that addresses the above challenges to estimate the energy consumption of any Web page load. The key intuition behind *RECON* is to go beyond resource-level information and *exploit application-level semantics* to capture the individual Web page load activities. Instead of modeling the energy consumption at the full page load level, which is too coarse grained, *RECON* models at a much finer *component* level granularity. Components are individual page load activities such as loading objects, parsing the page, or evaluating JavaScript.

To do this, *RECON* combines coarse-grained resource utilization and component-level Web page load information available from existing tools (§2.1). During the initial training stage, *RECON* uses a power monitor to measure the energy consumption during a set of page load processes and juxtaposes this power consumption with coarse-grained resource and component information. *RECON* uses both simple linear regression and more complex neural networks to build a model of the power consumption as a function of the resources used and the individual page load components.

Using the model, *RECON* can estimate the energy consumption of *any* Web page loaded as-is or upon applying *any* enhancement, *without* the monitor. It is important to note that *RECON*'s model does *not* have to be trained on all Web pages or on any enhancements. Since Web page loads exhibit high variance [40], *RECON* estimates the power consumption for a *given* instantiation of a Web page load.

We experimentally evaluate *RECON* on the Samsung Galaxy S4, S5, and Nexus devices using 80 Web pages. Comparisons with actual power measurements from a fine-grained power meter show that, using the linear regression model, *RECON* can estimate the energy consumption of the entire page load with a mean error of 6.3% and that of individual page load activity segments with a mean error of 16.4%. When trained as a neural network, *RECON*'s mean error for page energy estimation reduces to 5.4% and the mean segment error is 16.5%. We show that *RECON* can accurately estimate the energy consumption of a Web page under different network conditions, such as lower bandwidth or higher RTT, even when the model is trained under a default network condition. *RECON* also accurately estimates the energy consumption of a Web page after applying popular Web enhancements [9] including ad blocking, inlining, compression, and caching.

The *key* application of *RECON* is to analyze how and why Web page enhancements affect energy consumption. To this end, we look at four case studies where a Web enhancement exhibits non-intuitive results. The enhancements (and the non-intuitive behaviors) studied are: *(i)* An Ad blocker [4] that *significantly hurts* energy even though the page load time is not significantly affected, *(ii)* Caching that improves energy *disproportionately* compared to page load time, *(iii)* A more powerful compression optimization providing *worse* performance than a less powerful one, and *(iv)* Inlining optimization that helps performance and energy under one network condition, but *hurts* them under another network condition. In each of these cases, *RECON* breaks down the energy consumption into its constituents, and provides *useful insights* into the energy behavior that are *not* possible using power meters or resource-based power models.

2 BACKGROUND, MOTIVATION, & SCOPE

2.1 Page Load Process

The page load process starts with the user issuing a URL. As a first step, the browser downloads the html file corresponding to the URL. When the first part of the html file is received, html parsing begins; parsing is a computationally intensive process. When the parser encounters a tag for an image, JavaScript (js), or Cascading Style Sheets (css), it downloads the object, which is a networking process. If the object is a js or a css, then these are further evaluated, again a compute process. Progressively, the rendering engine renders the page on the browser. In effect, the page load process is a series of network and compute processes; we call each of these processes as *components*.

In *RECON*, we leverage a tool called WProf-M to obtain the component-level information. WProf-M [31] is an in-browser profiler that, among other things, decomposes the mobile page load process into its constituent components. We use the WProf-M tool to get detailed timing information about the start and the end of each component.

While we use WProf-M, other tools that provide component-level information, such as the Scout tool [32] or a combination of `chrome://tracing` and Chrome developer tools [11], can also be used in *RECON*.

2.2 Energy versus Performance

The components of the page load process together form a dependency graph [39]; many of the components can execute in parallel, while some components are serialized. Figure 1 shows the components for an example page, with certain components occurring in parallel. WProf-M provides enough information to visualize any Web page load similar to the visualization in Figure 1. Naturally, when components occur in parallel, external power monitors can *not* isolate the power consumption of each component.

It is important to note that energy and performance are fundamentally different metrics. Performance, usually measured using the Page Load Time (PLT) metric, is the length of the critical path on the dependency graph [39]; the critical path in Figure 1 is shown using a red dotted line. The power and energy consumption, of course, depend on *all* the components that make up the page load process, not just those on the critical path. In Figure 1,

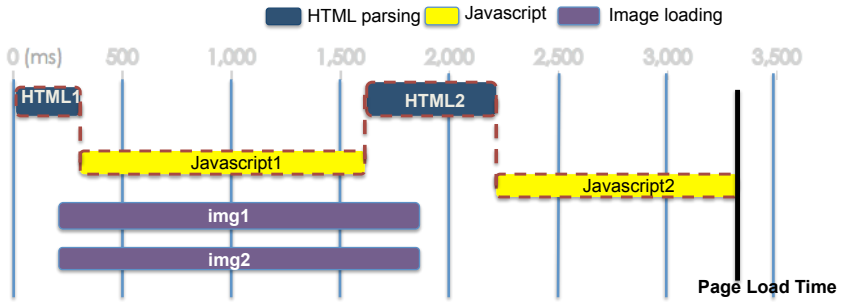


Fig. 1. An example page load process decomposed into various components such as HTML parsing, JavaScript, etc.

the energy consumption includes components on the critical path as well as those off the critical path, such as `img1` and `img2`.

One implication of this difference is that energy consumption may *not* always correlate with PLT. For instance, an optimization may significantly shorten the critical path (and thus the PLT) in Figure 1, but its effect on energy may be much less significant if it increases the loading time of images, especially if image loading turns out to be a power-hungry process. In some rare cases, optimizations that help PLT may even *hurt* energy consumption. In §7, we show examples of real Web pages, where the PLT and energy consumption are poorly correlated when a given optimization or enhancement is applied.

2.3 RECON Usage Model

The energy consumption of a Web page load varies across different runs since Web page loads experience high variance [40]. This means that predicting the energy consumption of a Web page is difficult and prone to errors. Instead, *RECON* estimates the energy consumption for an *instantiation* of a Web page load; that is, the energy consumption for a single run or load of the Web page.

Today, the energy consumption of a Web page is measured using external power monitors. While external power monitors, such as Monsoon [7], are accurate, they are not easy to work with. The monitors are connected to the phone using *battery bypass* technology that requires connecting the phone's battery leads directly to the external monitor and performing experiments in this tethered setting. Figure 3 illustrates such a setting for our experimental setup.

The user of the *RECON* system can accurately estimate the power consumption of several Web page loads without significant human effort. The user first learns the power models by training over a set of Web page loads, that we call the training set, using a power monitor. The only requirement is that the Web pages in the training set should contain at least one instance of all components that make up the target page load. The learnt power model is a function of the Web page components and resource consumption, which can be obtained at run time as the Web page loads. Then, the user simply loads the target Web page and obtains the component and resource consumption information (§4.2). *RECON* then estimates the power and energy consumption of that Web page instantiation using the model *without* requiring the power monitor.

The estimation process is completely automated: the Web page loading and the component and resource information gathering is done programmatically. Similar to prior modeling work [33, 43], *RECON* builds predictive models for the user-specific network types (WiFi vs Cellular) and devices.

2.4 RECON Users

The *RECON* user could be the content provider, the Web designer, or the browser vendor, who wishes to quickly and accurately determine the energy consumption of Web pages. For instance, a Web designer can use *RECON* to design energy-efficient Web pages by continually estimating the energy consumption as she is designing the page. Since the process is automated, measuring the energy consumption of the Web page requires no effort from the designer.

RECON estimates the energy consumption of Web page loads after an optimization or page enhancement and can thus help study pathological cases where energy increases in response to page enhancements. For instance, we find that certain Ad block extensions that remove ads from the Web page significantly increase energy consumption (§7.2). *RECON* can identify these cases and help understand *why* the energy increases. This capability can be invaluable for Web developers who wish to design useful Web optimizations.

In some cases, phone vendors can train the power model and publish the *RECON* coefficients for their specific device. In this case, the *RECON* user does not even have to learn the *RECON* model, and can simply use the published model to estimate energy consumption on the specific device. In the future, we envision *RECON* also helping end-users and others assess the energy impact *in-the-wild*. This in-the-wild usage scenario requires models that work across various devices and network types. We hope to address this challenge as part of future work to extend the benefits of *RECON*.

3 RECON

The key idea in our modeling approach is to exploit *page-specific component-level information* and integrate it with coarse-grained resource logging; hence the term *RECON* (resource- and component-based modeling). Because *RECON* leverages component-level information, it does not require fine-grained resource logging needed to model the power consumption of the *short-lived* page load process. The component-level information also allows imparts explanatory power to *RECON* for deconstructing the impact of a page enhancement on energy (see §7).

3.1 The Design of RECON

At a high-level, *RECON* works by developing a parameterized *page-independent* power model that incorporates the component- and resource-level information to make accurate estimations. We train the model by loading a few Web pages and monitoring the power consumption using a power meter. Once trained, we use the model to estimate the power consumption of any Web page, loaded as-is or under any enhancement, without the power meter.

3.2 RECON Modeling Overview

Building an accurate energy model using components and resource information is challenging. The *monitored power consumption is a result of several simultaneously executing components and their aggregate resource demand.* At any given time slice, the number of components of the page load process can be different. Given a new Web page load with an arbitrary distribution of components and aggregate resource utilizations, how can we estimate its power and/or energy consumption?

Our modeling approach is to break down the page load process into "segments", where a segment is defined as an interval of page load activity during which the components of the Web page do not change. Figure 2 shows the component-level decomposition for the `instagram.com` page, via WProf-M, juxtaposed with its power consumption obtained via the power monitor. Segments of the `instagram.com` page are illustrated in Figure 2 via dashed blue lines. By definition, a segment is composed of at least one component. Further, the entire page load process can be partitioned into discrete (non-overlapping) segments, as shown in Figure 2.

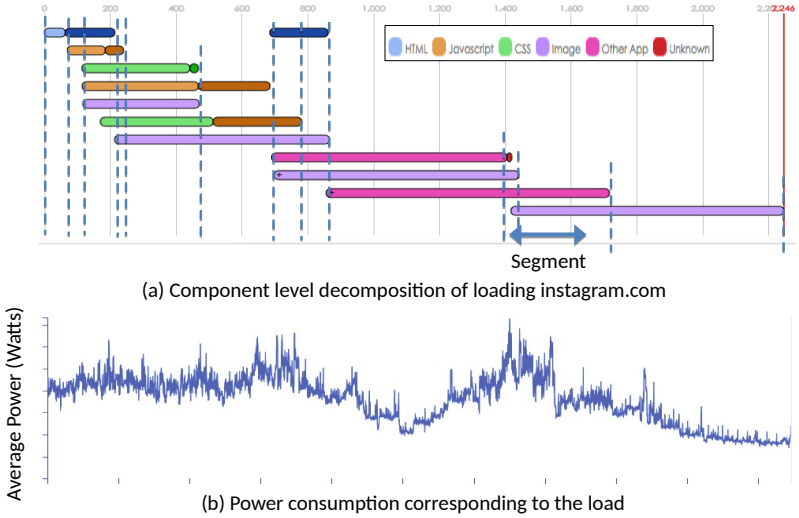


Fig. 2. (a) Using WProf-M to decompose the components when loading the `instagram.com` Web page. (b) The instantaneous aggregate power draw recorded as the Web page loads. Segments are shown in dashed lines.

Component and resource information: For component-level information, we leverage WProf-M [31] to obtain three pieces of information: (i) the set of all components, including their type and how often they appear, that make up the segment, (ii) the start time of each component, and (iii) the end time of each component. The specific component types we leverage for *RECON* are discussed in §4.2.

For resource-level information, we log several resource utilization values for the device; the set of resources we monitor and the frequency of monitoring is discussed in §4.2. We average the resource usage over the segment length.

Modeling goal: Given a segmented page load process, our goal is to *decompose the monitored power consumption of a segment to its constituent components and resource utilizations* during that segment.

To this end, we model the power consumption of a segment, s, as a function:

$$\hat{P_s} = f\left(\underset{i \in Resources}{R_i}, C_s\right), \tag{1}$$

where i represents the various resources we monitor, such as CPU utilization, and R_i represents the average utilization for that resource during s; C_s represents the set of components that make up segment s. If we can determine an appropriate function, f, as in Eq. (1), then we can estimate the power consumption of any segment given the distribution of its components and resource utilizations.

Modeling techniques: We investigate two machine learning techniques to build our power model, f, in Eq. (1): (i) Linear Regression (LR, §3.3), and (ii) Neural Networks (NN, §3.4). These represent two very different learning techniques that model the dependent variable (segment-level power consumption, in our case) in terms of the independent input variables (component- and resource-level information). LR is a statistical technique that models the dependent variable as a simple linear weighted combination of the independent variables. NN is a learning algorithm that learns how to best combine the independent variables, possibly in a non-linear manner, using adaptive weights, to estimate the dependent variable. While LR is easy-to-use and quick to train, it cannot model non-linearities and dependencies between independent variables. NN, on the other hand, can model non-linearities and dependencies, but is more complex and slow to train.

3.3 Linear Regression-Based Power Model

The key idea behind the LR model is to consider the per-component and per-resource power contributions as *invariants*, and model the device power consumption as a linear combination of these variables. That is, we assume that the power consumption of a given component, such as css, is constant, regardless of the Web page it appears in. Likewise, we assume that the power consumption for each unit of a given type of resource, such as CPU utilization, is constant. Lastly, we assume that the power consumption does not change during the length of a segment. In our experiments, we find that 90% of all segments have lengths below 94ms, a relatively short duration. Further, as we discuss in §4.2, our resource sampling interval is 100ms. Thus, we believe that our assumption about power being relatively constant during the segment length does not adversely affect our results since we only get one resource sample per segment in most cases. For longer segments, we average the resource utilization samples collected during the segment length. Of course, we can also impose an upper limit on segment length and split longer segments into multiple, shorter segments, each with its own constant power draw.

Mathematically, we model the power consumption of a segment, s, at any time as:

$$\hat{P_s} \quad = \quad \alpha + \sum_{i \in Resources} \beta_i R_i + \sum_{j \in C_s} \gamma_j F_j, \tag{2}$$

where R_i represents the average utilization for resource i during s; j represents a component type, and thus $j \in C_s$ is the set of all component types that make up segment s; F_j is the frequency of component type j, meaning the number of type-j components in the segment; finally, β_i and γ_j are coefficients (*independent* of s) representing the power contribution of the resources and components, respectively, and are variables that need to be determined. α represents the baseline power draw of the phone, and accounts for background activities, screen brightness, etc. Given this model, our goal now is to estimate the coefficients, $\vec{w} = (\alpha, \vec{\beta}, \vec{\gamma})$.

Note that a component's power consumption profile will not always be the same. While we model the power contribution of a component of type j using a constant, γ_j, we also capture the possibly changing resource usage during the component's execution, which can account for its inconsistent power profile. In other words, we capture the variation between different instances of the same component by also considering their possibly different resource utilizations. The resource-level information thus complements the component-level information under *RECON*.

We use multiple linear regression to derive the weights, $\vec{w}$, that are indicative of the power contribution of each component and resource. We obtain the components of a segment via Wprof-M and represent their contribution to power using indicator variables; summing up all contributions/occurences of a component type in a segment gives us its frequency, which we use in Eq. (2). Our use of a linear power model is motivated by the following observations: (i) for resources, prior work has shown that resource usage, such as CPU and network utilization, affect power consumption linearly to some extent [26, 41–43], and (ii) each Web component has its own modeled

power draw (represented by γ_j in Eq. (2)), so we linearly sum all component power contributions. Note that Eq. (2) implicitly assumes that the individual power contributions of resources and components are independent, and thus can be added together. While this is not necessarily true, we find that, in practice, the model represented by Eq. (2) accurately tracks total power consumption.

3.4 Neural Network-Based Power Model

We employ NN, more specifically, a multi-layer feed-forward network with a sigmoid activation function in the hidden layer, to model segment power consumption. NN offers a number of advantages, including the ability to implicitly detect complex non-linear relationships between dependent and independent variables [38]. A disadvantage is that NN is prone to over-fitting [24]. For more details, we refer the readers to Haykin [27].

NN takes the component- and resource-level information detailed in §3.2 as input nodes, and then learns how to best combine them, using adaptive weights, to estimate the segment power consumption that is close to the observed power. Note that these are the exact same inputs as we use for LR in Eq. (2), namely, utilization R_i for each resource i, and frequency F_j for each component type j. We use a single hidden layer in our network since the Universal Approximation Theorem is well known for feed-forward networks with sigmoid functions [27].

Mathematically, the NN model for power consumption of a segment, s, is:

$$\hat{P_s} = y_0 + \sum_{k=1}^{m} y_k \left(1 + \exp(-(x_k + \sum_{i \in Res} \theta_{k,i} R_i + \sum_{j \in C_s} \phi_{k,j} F_j)) \right)^{-1}, \tag{3}$$

where i, R_i, j and F_j are the same as in Eq. (2). m is the number of nodes in the hidden layer of the NN; we set m to be the average of the number of input and output nodes, as is commonly suggested for NN [28]. The $\vec{x}$ and $\vec{y}$ vectors, and the $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ matrices, are weights (independent of s) that need to be learned. Note that the number of weights to be learned for NN is at least a factor (of m) higher than that for LR. Also note that while the exponent for NN in Eq. (3) appears to be similar to the expression in the LR model, the weights can be different, and further, there are m different exponents in NN. We use the Truncated Newton algorithm [30] to derive the weight vectors and matrices that minimize the estimation error for the NN model in Eq. (3).

3.5 Model Training and Testing

We now discuss our methodology for model training and testing, which is common for both the LR and NN models.

Training: We train our model on one set of Web pages and test on a different set of Web pages. In the training period, we randomly choose a subset of Web pages and train on multiple instantiations (or runs) of those selected Web pages. We then leverage the trained model to estimate the energy consumption of the remaining Web pages, once again for several instantiations.

For each run, *RECON* records the power consumption values, page load component information, and resource utilizations. We use the instantaneous power measurements to calculate the average power, $\hat{P}$, for each segment. We then use regression for LL and the Newton algorithm for NN over the segments collected during the training runs to derive coefficients/weights for the models.

Note that our models can be *trained online without having to build detailed subsystem-level models* as in prior work. For example, recent work [23] developed a CPU specific power model by running microbenchmarks at *each* possible frequency for *each* combination of CPU cores to train their model. Similar training experiments were carried out for other subsystems. For our LR and NN models, we can obtain all component- and resource-level coefficients and weights *simultaneously* by simply training over a set of Web page loads; we do not have to perform separate, controlled training experiments for each component or resource. Of course, our focus here is

only on modeling the energy consumption for the *browser*, which allows for faster training.

Testing: The coefficients/weights, derived via training, for our power models in Eqs. (2) and (3) can now be used to estimate the power and energy consumption of any new instantiation of any Web page without requiring the power monitor. The inputs are the set of components involved in the page load process and the resource consumptions for the new instantiation, all of which can be obtained *at run time* (see §4.2). We apply our trained model on the test data by substituting the learned weights, $\vec{w}$ in Eq. (2) for LR, and vectors $\vec{x}$ and $\vec{y}$ and matrices $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ in Eq. (3) for NN, along with the above-mentioned inputs. This gives us the estimated power consumption for every segment; we then estimate the energy consumption by multiplying the estimated power with the observed segment length (obtained via Wprof-M). Summing up the energy consumption across all segments of the instantiated page gives us the estimated energy consumption of the new Web page load.

4 EXPERIMENTAL SETUP

We now discuss our experimental setup which we use for training, testing, and evaluation of *RECON*. Results for RECON validation and evaluation are presented in the subsequent sections (§5 and §6).

4.1 Devices and Network

Our experiments are conducted using three phones: (i) Samsung Galaxy S4 (Android 4.3, Jelly Bean), (ii) Samsung Galaxy S5 (Android 5.1.1, Lollipop), and (iii) Galaxy Nexus (Android 5.1.1, Lollipop). Unless otherwise specified, we present results from the Galaxy S4.

We experiment under several network conditions, including WiFi with different traffic conditions, and a cellular (4G) network. Unless specified otherwise, we use the default WiFi network with 30 Mbps download and 20 Mbps upload bandwidth, and a 50ms RTT to a reference server hosted by pair Networks.

4.2 Power, Component, and Resource Logging

RECON logs the power, Web page components, and coarse-grained resources during the page load for modeling. We now describe each of these in turn.

Logging power consumption: To measure the device power consumption, *RECON* uses an external power monitor, Monsoon [7], which performs fine-grained power measurement at a 5KHz frequency. *RECON* uses the power monitor only to train the models and not for estimations.

Figure 3 shows our experimental setup with the Samsung Galaxy S4 device loading a sample mobile Web page, `fico.com`, while recording the instantaneous current draw and maintaining a constant voltage through the Monsoon power monitor.

Logging Web page components: We leverage WProf-M (described in §2) to log the components of the Web page load. WProf-M instruments the Android Chromium browser, Version 31.0.1626.0. We run all our experiments on the instrumented mobile browser. The instrumentation logs provide fine-grained timing information to decompose the page load process into various components (an example decomposition is shown in Figure 2(a)).

Since our focus is on Web page energy consumption, we classify components according to their expected energy behavior. Specifically, we group all components into: (i) downloads (all types, including text and images), (ii) js evaluation, (iii) css evaluation, and (iv) html evaluation; we group all downloads together since the energy consumption for a download should only depend on the object size, and not the object type. *RECON* can be easily extended to also consider the type of download, or even the type of image being downloaded, since the modeling is *independent* of the choice of component classification.

Fig. 3. Our hardware setup showing the Samsung S4 under test connected to the Monsoon power monitor.

Logging resource usage: *RECON* combines component-level modeling with coarse-grained resource monitoring. To monitor resources, we use a simple android service that records resource consumption values. Based on our prior work [31], we find that CPU and network are important power contributors for mobile browsers. For CPU, we collect per-core CPU utilization from `/proc/stat` and per-core CPU frequency from `/sys/devices`. The device CPU power consumption is known to depend on the {utilization, frequency} pair [23, 44]; we use the *product* of utilization and frequency to account for this non-linear dependence. For network, we collect number of bytes transmitted and received during an interval from `/proc/net/dev`. Although the screen power consumption is critical to Web page loads, Chen et al. [23] show that screen power remains fairly constant when displaying at a fixed brightness. Accordingly, we set the screen brightness to a constant and model the baseline power consumption of the device instead; in the future, we will study the impact of newer OLED displays, whose energy consumption changes with the content on the screen. We find that monitoring memory usage does not improve the estimation accuracy of *RECON*; we thus omit it from our logging. For our setup, we disable other functionalities such as GPS and audio, and do not consider them in our power model. The p-value for the 14 resource variables (12 for CPU and 2 for network) we consider are small, thus validating our selection (see §5.4).

Logging methodology:

Both WProf-M and resource monitoring use logcat, Android's logging software, to record raw data that is then analyzed offline. To maintain low monitoring overhead, we log resources every 0.1 seconds, resulting in a less than 2% increase in CPU utilization. When we increase the resource monitoring frequency to once every 0.01 seconds, the CPU utilization increases by more than 30%, which is clearly infeasible. WProf-M and resource logging together add only about 0.2W to the total power consumption. Compared to the average device power consumption when loading a Web page without WProf-M or resource logging, this 0.2W accounts for less than 5% of the total power consumption.

An important step in our methodology is to synchronize the power monitor measurements with the Web page load times. Our testing framework is completely *automated* using the calabash [10] scripting language that starts the power monitor and then loads the Web page programmatically. We let the power monitor run for a few seconds to stabilize, and then load the Web page. The start of the Web page load creates a spike in current, that can be identified in the power monitor's logs. We mark this time as the start of the Web page loading process. We note that this synchronization may not be accurate at a microsecond scale, but given that the page load

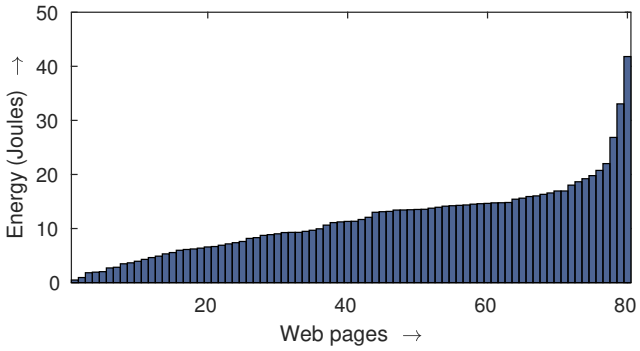


Fig. 4. Average energy consumption (measured) across all runs for all 80 Web pages, sorted in ascending order.

times are on the order of several hundreds of milliseconds, this level of synchronization suffices for our purposes. We determine the end of the page load using WProf-M logs; the page load ends when the DOMLoad event is fired [31]. These two events together allow us to identify portions of the power logs that correspond to the start and end time of the Web page load.

4.3 Web Pages

We experiment with 80 Web pages. 60 Web pages are randomly selected from top 100 Alexa Web Pages [18] across 10 different countries. The remaining 20 Web pages are randomly selected from pages ranked between 100 to 10,000 on the Alexa site for diversity in page selection. We load each Web page 10 times in our experiments.

The Web pages vary significantly in terms of their PLT and power/energy consumption. Figure 4 shows the measured energy consumption for each of the 80 Web pages, averaged across their 10 runs, sorted in ascending order.

Since Web pages change over time, we download the main html page locally on our server, and load the page from this local copy. Note that all the objects embedded in the page are still fetched from the original remote server over the network. Downloading the main html locally ensures that roughly the same set of objects are loaded in each run, though the object loads may vary because of dynamic JavaScript. Despite loading the Web pages locally, our experiments still show significant variance across runs, as we show for example Web pages in Figure 9.

Unless specified otherwise, all Web page loads are cold loads and the cache is cleared after each load. We show in §6.4 that the accuracy of our modeling remains high irrespective or whether we employ caching or not.

5 RECON MODEL VALIDATION

We now thoroughly validate our LR and NN models and contrast the results. We start by presenting the Web page-level error in §5.1 and §5.2, followed by the segment-level error in §5.3. We then contrast LR and NN in §5.4 and finalize our power model for *RECON*.

5.1 Modeling Error for Web Page Energy

We employ *RECON* to estimate the energy consumption of 80 Web pages (as discussed in §4.3) using LR and NN models, and compare the model-estimated values with actual measurements from the Monsoon power meter. The 4-fold cross-validation error averaged across all instantiations of these Web pages is 6.29% under LR and 5.40% under NN. For cross-validation, we split the list of 80 Web pages into 4 sets of 20 pages each, and then train

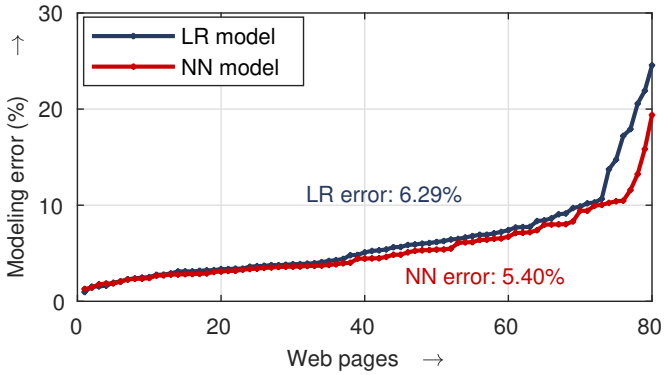


Fig. 5. ***RECON* validation:** Percentage error in energy estimation for LR and NN models across the 80 Web pages, sorted by the error. The mean error is obtained by averaging across 10 instantiations of each Web page under 4 runs of cross-validation. While this figure shows the average, Figure 6 shows the complete CDF of the errors.

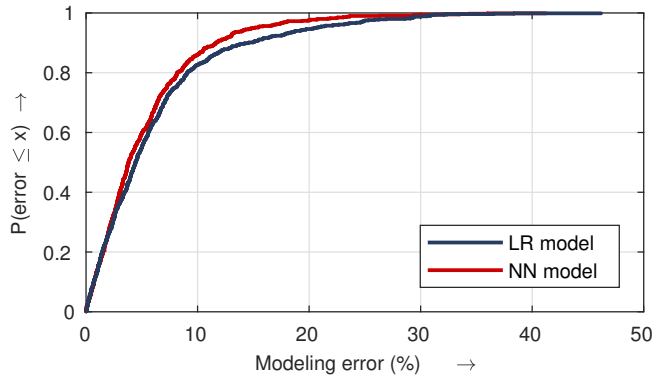


Fig. 6. ***RECON* error CDF:** The CDF shows the LR and NN energy estimation errors across all runs of all Web pages (800 data points). 83% of the LR errors and 86% of the NN errors are below 10%.

the models on 3 of these sets. Note that each Web page is loaded 10 times, so the training set consists of 600 page loads and the test set consists of 200 page loads. We then estimate the energy consumption of the 200 page loads in the test set, and compute the *test error* for each of the 20 Web pages by averaging over its 10 instantiations. We repeat this training/testing over all 4 combinations of the sets and report the average test error across all 4 test sets spanning all 80 Web pages.

Figure 5 shows the (sorted) estimation error for all Web pages under LR and NN models. The low estimation error is obtained by computing the *page-independent* model coefficients and weights for Eq. (2) and Eq. (3). If we instead train a separate model for each Web page by training over several runs of the same Web page to derive the weights, the error further reduces by about 1%. However, this severely limits the applicability of such page-dependent models in practice. Thus, *RECON* employs practical page-independent modeling which provides sufficiently high accuracy.

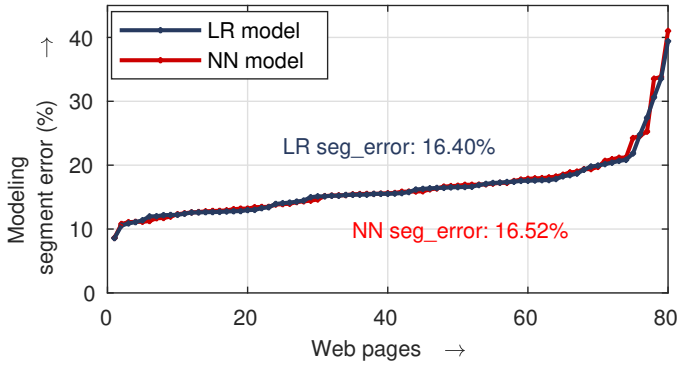


Fig. 7. Average modeling segment error (sorted) for LR and NN estimated segment energy consumption for all pages.

5.2 CDF of Web Page Energy Modeling Error

Figure 6 shows the energy estimation error for all 10 runs of all 80 Web pages under LR and NN. We see that more than 80% of the errors for both LR and NN are below 10%, and more than 90% of the errors are below 20%. This shows that the error for almost all Web page loads (not just the average) is low.

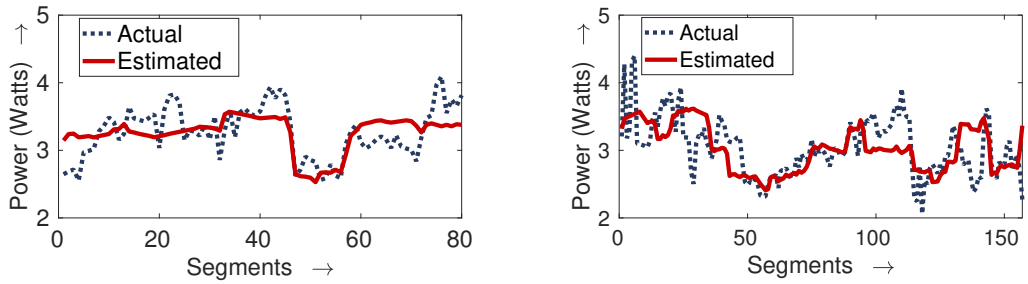
5.3 Modeling Error for Segment Energy

RECON can also be used to estimate fine-grained segment-level energy consumption directly using Eq. (2) for LR and Eq. (3) for NN. This is a valuable feature that allows us to analyze the impact of Web optimizations on the energy consumption of individual components of a page; we highlight this advantage later in §7. The average segment-level modeling error, referred to as *seg_error* (to distinguish from full page estimation error), across all 80 Web pages is 16.40% under LR and 16.52% under NN. The full Web page estimation error is lower than seg_error as the over-estimation of energy for some segments is countered by the under-estimation of energy for other segments when computing full page energy.

Figure 7 shows the sorted energy estimation seg_error for the 80 Web pages under LR and NN; the order of the Web pages here is different from that in Figure 5.

Figure 8 shows the actual and estimated segment-level power for specific instantiations of two Web pages using the LR model. We see that the estimated values closely track the measured values; results are similar for the NN model.

To investigate the modeling error further, we consider page loads that have a high modeling error (such as those in the tail of Figure 6), and then focus on all segments in these page loads with seg_error > 50%. We find that downloads and html evaluation are among the most frequently occurring objects in these segments. Furthermore, downloads and html evaluation components have long (cumulative) lengths in most of these page loads, hence contributing significantly to the page load energy. Based on the above observations, we conclude that much of the modeling error can be attributed to the object downloads and html evaluation components. Note that it is not possible to evaluate the per-component modeling error since we cannot directly measure the actual power draw of each component; we can only measure the *aggregate* power consumption of the device.



(a) Segment-level modeling for `yelp.com`. Error=7.8%.

(b) Segment-level modeling for `sfr.fr`. Error=9.7%.

Fig. 8. Actual versus LR-estimated segment-level power consumptions for specific Web page instantiations.

5.4 Why We Pick LR over NN for RECON

The LR model is motivated by its ease-of-use and the fact that LR is fast to train. In fact, we *train our LR model online in a few seconds*; each of our four folds of training takes about 2 seconds. The possible disadvantage of LR is that the model is simplistic, and may result in poor accuracy.

The NN model is motivated by the fact that it can model non-linearities in the input variables (component- and resource-level information) and any dependencies that might exist between them, thus providing higher accuracy. The disadvantage of NN is that the model training is time consuming, about 20 mins for a single fold of our NN training, and requires expert knowledge to determine the model parameters such as the number of intermediate nodes and layers.

The 4-fold cross-validation error for estimating the Web page energy consumption is 6.29% for LR and 5.40% for NN, as shown in Figure 5. Clearly, both models have high accuracy, and LR has only marginally higher (by 0.89%) test error. Given the simplicity and fast training time for LR, we are inclined to employ LR in our modeling.

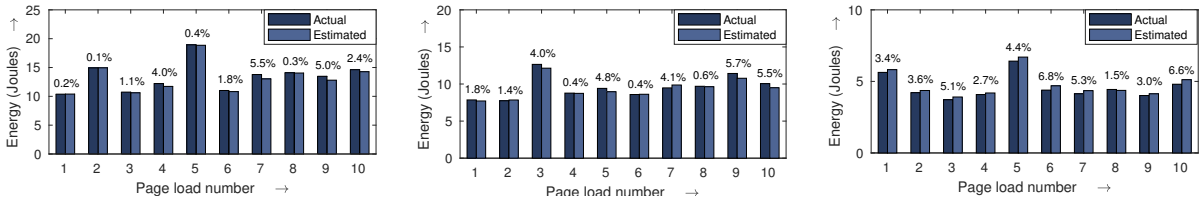
But most importantly, LR allows us to easily *deconstruct* the individual power contributions of each component and resource, which is the main motivation for the design of *RECON*. Based on Eq. (2) for LR, the power contribution of component j is γ_j, and that of resource i is β_i per unit of utilization. Using R [35], we find that the p-values of all components and resources in the LR model are less than 10^{-9}. The low p-values indicate that the specific component and resource variables we choose in the LR model (see §4.2) are statistically significant.

While NN modeling also provides weights for all nodes, it is not obvious what these values represent since they could be weights for non-linear combinations of components and resources, making it hard to deconstruct the individual power contributions of each component or resource. For example, collecting all weights for any R_i or F_j in Eq. (3) is clearly non-trivial given the several exponent functions that appear in a summation of non-linear terms.

For these reasons, *we choose the LR model for RECON* in the rest of this paper.

6 RECON EVALUATION

In this section we provide further evaluation of *RECON* using our LR model. In particular, we evaluate *RECON* for different devices (§6.1), under different network conditions (§6.2), and in the presence of page load variance (§6.3) and Web enhancements (§6.4). Finally, we compare *RECON*, quantitatively and qualitatively, with resource-only and component-only modeling in §6.5.



(a) Actual and estimated energy for `newegg.com` across 10 different runs.

(b) Actual and estimated energy for `stackoverflow.com` across 10 different runs.

(c) Actual and estimated energy for `whatsapp.com` across 10 different runs.

Fig. 9. **Error across runs of the same page.** Despite the variance in energy, *RECON* accurately estimates the energy consumption for each run for the above example Web pages. Numbers above the bars denote the error for each run.

6.1 Evaluating RECON for Different Devices

RECON's online modeling approach is not specific to the S4 device we use and can be extended to other devices as well. We use our modeling approach to train and estimate the energy consumption of ten Web pages on the Galaxy S5 and Galaxy Nexus devices (see §4.1). We obtain a low full page mean modeling error of 6.9% and seg_error of 19.7% for the S5, and modeling error of 9.6% and seg_error of 16.9% for the Nexus. Note that the model has to be retrained for each device because of the significant differences in the architecture and features between them; the need for device-specific models was also emphasized in prior work [33, 43]. However, we emphasize that, within the scope of our evaluation, our model is page-independent, as illustrated by the results in §5.1 and §5.2.

6.2 Evaluating RECON for Different Networks

Thus far our experimental results employed the default WiFi network described in §4.1. It is interesting to ask whether the power model trained on this default network can be used to accurately estimate the energy consumption on a different network. Specifically, *can the weights derived in Eq.* (2) *via training on one network provide accurate estimates for energy consumption on another network?* We use the above-derived model, trained on the default network, to estimate energy consumption of ten different Web pages under three different networks.
Lower bandwidth: We use Linux's traffic control (`tc`) to lower the upload and download bandwidth to 5 Mbps; this increases the average PLT by about 10%. Our average energy estimation error across ten different Web pages is 7.3% and the seg_error is 13.3%.
Higher RTT: We use `tc` to increase the RTT (150ms to the reference server) while keeping the default network bandwidth; this increases average PLT by about 50%. Our average estimation error across ten different Web pages is 8.8% and the seg_error is 16.4%.
Cellular network: We also experiment with a 4G LTE cellular network (AT&T) instead of WiFi. The cellular network has 10 Mbps download and 2 Mbps upload bandwidth, and 70ms RTT to the reference server. We train our model on the cellular network and then test on ten different Web pages also on the cellular network. The modeling error is 6.1% and the seg_error is 14.4%. Since the behavior and dynamics of the network are interface-specific, even for the same device, *RECON* results in high modeling error when trained on, for example, the WiFi environment and tested on the cellular environment (21.2% modeling error). As a result, the model has to be retrained for each network type.

Enhancement	Caching	Compression	Inlining	Ad block
Error	8.9%	8.4%	5.4%	8.5%

Table 1. RECON modeling error for different enhancements.

6.3 RECON under Page Load Variance

There is significant variance between loads of the same Web page. It is thus interesting to ask whether *RECON* can accurately estimate the energy consumption for *each* instantiation of a Web page. While Figure 6 shows the CDF across all instantiations of all Web pages, Figure 9 shows the actual and estimated energy consumption for all instantiations of 3 specific Web pages, `newegg.com`, `stackoverflow.com` and `whatsapp.com`. We see that the error for *every* instantiation is less than 7%. The energy consumption across different runs varies by as much as 44%. *RECON*'s estimated energy consumption is in agreement with the actual energy consumption across all instantiations, despite the variance.

6.4 RECON under Page Enhancements

RECON can also be used to estimate Web page energy consumption under Web enhancements, including compression, inlining, ad blockers, and caching (that is, without cold loads). Table 1 shows the *RECON* modeling error for different enhancements; we explain these enhancements in detail, including their setup and implementation, in §7.

6.5 Comparison with Resource-only Modeling

RECON leverages both resource-level information and component-level information when modeling Web page and segment-level energy consumption. Instead, one could leverage only resources, as in prior work (e.g., PowerTutor [43], V-edge [41], and WattsOn [29]), to construct similar models. However, such models do not perform as well as *RECON*.

When we model power consumption using *only resource-level information*, the modeling accuracy is limited by the resource monitoring frequency which needs to be low (10/second, in our case) to ensure low CPU and power overhead (see §4.2). For example, using resource-only modeling increases the error by about 34% for `bing.com` and about 41% for `craigslist.org` when compared to *RECON*. This result was obtained using the exact same 14 resource metrics used for *RECON*, collected at the same frequency (once every 0.1s), and using the LR model. Although related works suggest logging resources every few seconds [23], we find that lowering the resource monitoring frequency to 2 - 5 seconds increases average modeling error across all page loads by 38%. In the context of this work, resource-only models can *not* provide visibility into the power consumption of individual page load activities. *RECON*, on the other hand, can provide such visibility, thus enabling the analyses of Web optimizations as presented in §7.

However, resource-level information is important and cannot be completely dismissed. In particular, models that *only rely on component-level information* perform poorly as they cannot distinguish between the resource utilization levels for various phases of a component load. For example, an image load might involve fetching the image from the server and possibly rendering it locally. These different phases are treated equally under WProf-M, and can result in poor accuracy for such components when not leveraging resource information. Figure 10 illustrates such an example for a long image/gif component encountered as the sole component of a segment during the loading of `fico.com`. As shown, the power and resource usage vary considerably during the loading of this segment; component-only models cannot capture this information and often have poor accuracy.

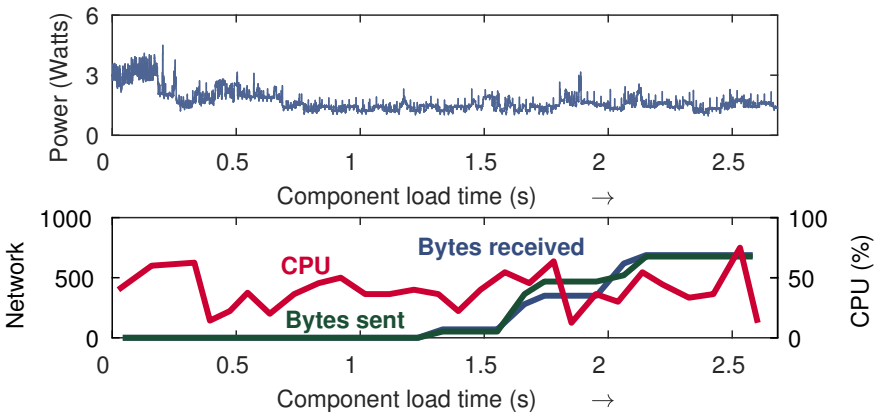


Fig. 10. The figure depicts the variations in power consumption (top) and resource usage (bottom) for a segment of `fico.com` where only *one* long component (image/gif download) was present.

Using the same examples as for resource-only modeling above, component-only modeling increases the modeling error by about 12% for `bing.com` and about 59% for `craigslist.org`, when compared to *RECON*.

6.6 Comparison with PLT-only Modeling

We now compare *RECON* with a simple power model that only relies on PLT. In particular, we consider a linear regression model which uses PLT as the only explanatory variable:

$$\hat{E}_{page} \quad = \quad c_0 + c_1 \cdot PLT, \tag{4}$$

where $\hat{E}_{page}$ is the estimated energy consumption of the page load and c_0 and c_1 are coefficients to be deterimined (via regression). We test this model on the full set of page loads, and find that the PLT-only model given by Eq. (4) has a 66% higher 4-fold cross-validation error than *RECON*. This simple result highlights the need to incorporate information about components and resources into the model to account for variations in power throughout the page load.

7 CASE STUDIES ENABLED BY RECON

A key application of *RECON* is in providing visibility into both *how* and *why* Web page enhancements affect energy consumption. We show four case studies that exemplify *RECON*'s explanatory power. In each of the four cases, we find that an enhanced Web page results in non-intuitive energy behavior, and use *RECON*'s constituent (component- and resource-level) analysis to analyze this behavior. The enhancements (and the non-intuitive behavior) studied are: *(i)* An Ad blocker [4] that significantly hurts energy even though the PLT is not significantly affected, *(ii)* Caching that improves energy disproportionately compared to PLT, *(iii)* A more powerful compression optimization providing worse performance than a less powerful one, and *(iv)* Inlining optimization that helps PLT and energy under one network condition, but hurts PLT and energy under another network condition.

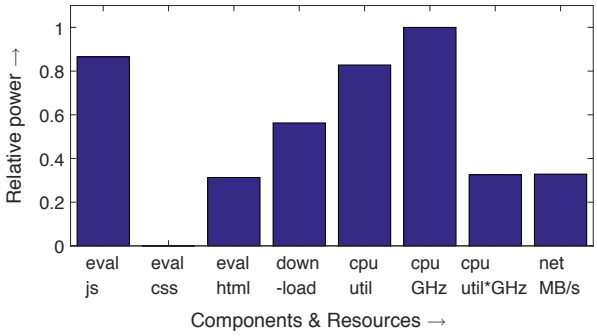


Fig. 11. Normalized power contribution estimations for components and resources based on our LR modeling.

Experimental setup: For each case study, we load 10 Web pages with and without the enhancement, and measure the PLT and energy consumption over multiple runs. The Web pages were chosen from Alexa's top million list to reflect a broad range of page sizes; for Ad blocker, we also consider sites that are known to contain ads so as to trigger the ad blocking. We note that, extensively studying each of these Web enhancements is outside the scope of this work. Our goal is to use *RECON* to explain certain non-trivial behaviors observed in our case study, thus highlighting the advantages of our main contribution, *RECON*. As shown in §6.4, our mean estimation error for the four enhancements using this setup is less than 10%.

To enable the enhancements, we serve these Web pages from our local Web server by downloading all contents from the remote servers; note that many of the Web enhancements are applied at the Web server requiring that we have control over the Web server. To maintain the links, we launch a separate virtual DNS and Web server for all domains involved.

7.1 Breaking Down Energy Consumption into its Constituents Using RECON

We use *RECON* to analyze any non-intuitive behavior that we observe in our case studies. Specifically, *RECON* breaks down the estimated Web page energy consumption into the energy consumed by Web components and resources, both before and after the enhancement is applied. As discussed in §4.2, we classify components as object downloads, and evaluation of js, css, and html; for resources, we focus on CPU and network related metrics.

Recall, from §3.3, that the coefficients ($\vec{w}$) in the LR model (Eq. (2)) correspond to the relative contribution of each component and resource to total power consumption. Multiplying the coefficients with the component and resource lengths gives us their energy contribution — the energy spent in downloading objects, the energy spent is evaluating js, css, and html, and the energy consumed by the device due to CPU and network activities unrelated to the components. This provides the needed visibility into the energy effects of the page enhancement; such visibility is *not* possible by relying solely on power meters that only report aggregate power consumption at any time, or resource-based power models that can *not* deconstruct the power into constituent Web component contributions.

To illustrate this visibility, consider Figure 11, which shows our model-estimated power contributions for components and resources. For ease of presentation, we normalize the contributions such that they lie in the $[0, 1]$ range, with the smallest contribution, eval css, set to 0. The components from left to right are evaluation of js, css, and html, and downloads; these are followed by resource variables which represent the power consumed by the CPU per unit of utilization, per unit of GHz, and per unit of (utilization · GHz), and the power consumed

by the network transfer activities. Of course, these are only power contributions. We must also consider the component and resource lengths for each segment or page to obtain their energy contributions.

7.2 Case Study #1: Ad Blocker

One popular technique to block unwanted advertisements and malware is to block them at the name resolution phase. This technique, also called the hosts file ad blocking, maintains a blacklist of malware and ad domains. Before the browser loads an object, it checks this blacklist. If the object domain is blacklisted, it will be resolved to an IP address (e.g., 127.0.0.1) with no service. We use a popular hosts file ad blocker called `BSDgeek_Jake` in our case study [36].

For two Web pages that were known to have malware, the ad blocker significantly reduced both PLT and energy. This is not surprising, since the ad blocker saves time and energy by blocking blacklisted objects that will then not be loaded.

However, for the remaining 8 Web pages (that did not have any blacklisted objects embedded in them), loading the Web page with the ad blocker *increased energy by 50%*. Surprisingly, the PLT did not increase correspondingly, and on an average, the PLT increase was only 13%. This *non-correlation* between PLT and energy is problematic for a Web developer or even the user. The `BSDgeek_Jake` is popular since the blocker either improves PLT or leaves PLT unaffected; but its negative effect on energy needs to be understood to make informed decisions about its use.

Performing the energy break down of all 10 Web pages, we find that when using ad blocker, the *CPU resource energy increases by an average of 200%*. Recall (§7.1) that the CPU resource energy is the energy consumed by the CPU, not by the components. In fact, using *RECON*, we find that the other constituents that make up the total energy, including the Web components and network usage, see little change with and without the ad blocker for the 8 Web pages that have no blacklisted objects.

This indicates that it is not the Web page components, but other external compute activities, that are causing this energy increase. This external computation activity likely involves scanning the entire blacklist file before loading each object. The `BSDgeek_Jake` has more than 150K entries for ads/malware domain, and matching each domain against this list is known to be power hungry [36]. We note that while a power monitor can detect this increase in energy, it cannot explain *why* the energy increases. Similarly, resource-based power models can identify that the CPU energy is high, but cannot identify if the energy is high due to Web page load activities or other external factors.

7.3 Case Study #2: Effect of Caching

Caching objects locally at the client saves a round trip time during the Web page load process, improving performance. For our experiments, we load the page once and cache all the first-level embedded objects: css, js, and images. Other dynamic objects such as ads are still fetched from the server. The effect of caching is studied by reloading the Web page immediately after the first load.

In most case, as expected, both PLT and energy reduces significantly when using caching, and the reduction is well correlated. But in 3 of the Web pages, *the reduction in energy is much more pronounced compared to PLT*. In some cases, the original Web page load consumes *double the energy* compared to when objects are cached, but the PLT reduction is only 33%. Again, this observation has implications for Web developers. A developer may conduct performance tests, conclude that caching is not useful, and may disable caching, even though it can provide energy benefits.

Figure 12 shows the visualization of `stackoverflow.com` when the Web page is loaded with and without caching. Caching only eliminates one object on the critical path (shown in red). The other activities on the critical path are compute activities that are not affected by caching. As a result caching does not greatly help performance.

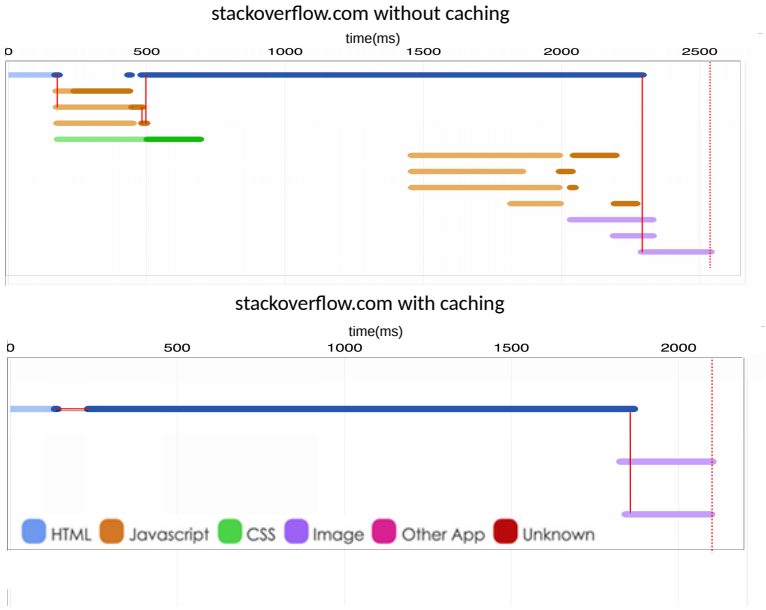


Fig. 12. Visualizing the page load process when loading `stackoverflow.com` with and without caching. The critical path is shown in red.

However, the energy consumption depends on all objects, not only the critical objects. From Figure 12, it is unclear if the additional Web object loads saved by caching indeed contribute to the energy reduction. In other words, we need to determine the energy contribution of the object loads (in the non caching case) to conclude that it is indeed the caching that reduces the energy. *RECON* provides just that. Our constituent analysis shows that the energy contribution of the download components reduces by 81%, significantly reducing the energy consumption under caching.

In practice, it is difficult to predict the contents of the cache since it is related to the user's browsing history, shared embedded objects among different Web pages, etc. Fortunately, *RECON* does not rely on such predictions; instead, *RECON* leverages the component-level information obtained via WProf-M *after* the Web page load to estimate the page load energy consumption. In particular, if an object is in the cache during the page load, then the (smaller) object load time obtained via WProf will reflect the impact of this caching; *RECON* then leverages this component length information to estimate the page load energy consumption.

7.4 Case Study #3: Compression Levels

The compression optimization compresses textual content so that objects can be downloaded more quickly over the network. Typically, compression only works for the html file, js, and css, since images are already in compressed format. Once compressed, the object has to be decompressed at the client. To apply the compression optimization on a Web page, we enable *mod_gzip* [15], a compression module, on our local Web server and load the Web pages from this Web server. We perform our experiments in an emulated slow network condition. The slow network condition is emulated using linux traffic controller and the bandwidth is set to 1 Mbps and the round trip delay is set to 50ms, which are typical 3G speeds [31].

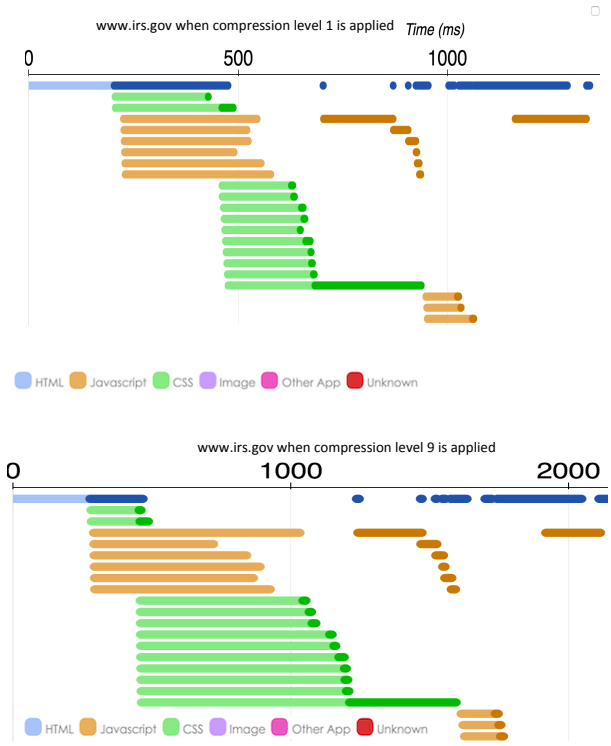


Fig. 13. Illustrating the increased delay in decompressing css (green bars) and js (brown bars) under compression level 9 (bottom) when compared to level 1 (top) for `irs.gov`.

We conduct our experiments under two different compression levels, level 1 and level 9, where the higher compression level is more powerful. For most Web pages, compression provided little benefit in terms of PLT (under 15%).

However, the `irs.gov` Web site exhibits non-intuitive behavior upon compression. The more effective compression level (level 9) reduces PLT by 47%, however, the *less effective compression level reduces PLT by 78%*. In terms of energy, compression level 1 reduces energy by 75% compared to only a 39% reduction in energy under level 9.

Using *RECON*'s energy breakdown, we observe that the energy consumed by the downloads (which includes the decompression energy) reduces by 71% under level 1 when compared to level 9. Figure 13 shows the visualization of `irs.gov` under different compression levels. The visualization of an object download includes the time taken to download the object and the time for decompression. The length of the green css and brown js components under level 9 is longer than level 1 (note the difference in time scales), possibly because of the effect of decompression.

To validate this finding, we find, using *RECON*, that the energy consumed by the CPU component is 37% higher for level 9 compared to level 1. Similar to the ad blocker case study, this increase in CPU component may indicate external factors, such as increased decompression overhead under level 9. Therefore, even if the compression is more effective, the time and energy needed for decompression reduces the effectiveness. Compression level 1, although less effective, also takes less decompression time and energy, as suggested by the lower energy consumed by the CPU.

Finally, we analyze why other Web pages do not see improvements when using compression even under the slow network. We find that, for most of the Web pages in our experiment, the root html file is small, and compressing this file further does not provide enough benefits. Further, for these Web pages, either the js/css objects are small or not on the critical path. Therefore, compressing these objects provides only modest improvements to energy and PLT.

7.5 Case Study #4: Inlining Optimization

The inlining optimization embeds external js and css in the root html file; as in the case of caching, only the first-level objects are inlined. Inlining makes the initial html file larger, increasing the network latency and energy to download. However, once downloaded, there is no need to fetch the external js and css objects since the html file already has them inlined, thus avoiding several small downloads. Further, inlining reduces network dependencies.

We load the 10 Web pages with and without inlining under two different network conditions: A fast network condition which is our typical experimental set up and a slower network condition, as described in the compression case study.

In the fast network condition, 4 of the 10 Web pages benefit from inlining, with an average PLT reduction of 64% and average energy reduction of 48%. The remaining Web pages are small (load within 3 seconds) and do not benefit significantly from inlining. On the slower network, we see a surprising reversal of trends. The smaller Web pages are not affected by inlining, as before. However, for one of the Web pages, `collegehumor.com`, inlining *hurts* PLT by a median 28%. For this same Web page, inlining *helped* PLT by 16% in the fast network. Worse, under the slow network, energy *increases* by 38% upon inlining while the energy *decreases* under the faster network. This behavior has implications for Web developers, who need to consider the *different effects of their optimization under different network conditions*.

Our constituent analysis shows that, under slow network conditions, the energy consumption of the download component increases upon inlining. In other words, the energy consumption for loading one large object (inlining) is higher than the energy consumption of smaller objects (default) under poor network conditions. In the fast network, there is not much difference in the energy of the download components.

But our observation for the download component indicates that all Web pages should see poor performance under inlining under slow networks; however, we see this behavior for only one of the Web pages. Here we find that `collegehumor.com` has a *large number of embedded images* which are dependent on the html download. When the html download gets delayed, the subsequent image downloads also get delayed, leading to increased PLT and increased energy. Other Web pages did not have this dependency.

Note that, in all the above examples, *RECON* can help identify which resources/components are causing the energy drain. Explaining the root cause of why these constituents are causing the energy drain is beyond the scope of *RECON*.

8 RELATED WORK

Given the importance of smartphone energy consumption, there has been considerable interest in modeling device power. Below, we categorize the related work in terms of the techniques used for modeling.

Utilization-based power models: One of the most common modeling techniques for smartphone power is resource utilization-based models. These models leverage the correlation between resource utilization and the energy consumption. The typical modeling approach in to first establish a power model for individual hardware components on the phone including the CPU, GPU, Screen, and the Network. Data for the model training is typically collected using an external power monitor. PowerTutor [43] uses the Monsoon power monitor [7] to measure the energy consumption under various CPU frequencies, WiFi data transfer rates, and screen brightness

settings. PowerTutor estimates the power consumption on phones based on the battery discharge patterns and the utilization-based models.

In many cases, utilization does not directly correlate with power consumption; for example, in cellular networks, the power consumption continues even after all data transfer finishes, because of tail effects [20]. To address this, researchers use advanced models, such as finite state machines (FSM), to represent the power consumption of resources that do not correlate well with utilization alone [34]. PowerTutor itself uses an FSM to build a model for cellular/WiFi power consumption. Chen et al. [23] use a hybrid model which uses a utilization-based model for CPU and GPU, an FSM-based power model for wireless interfaces, and the average power usage of activities such as WiFi beacon, cellular paging, and SOC suspension. Rather than using a commodity external power monitor to model the power consumption of resources, Carroll et al. [22] use special hardware to measure the power consumption. In most of the above works, the resource monitoring frequency is on the order of once per second [23, 33], which is insufficient for modeling Web pages (see §6.5).

ARO [34] is a complementary approach that models the device's radio power consumption by analyzing cellular packet traces to infer the RRC (Radio Resource Control) states. Since ARO focuses on radio, it does not take into account the energy consumed by CPU and Web components. As illustrated by our case studies in §7, the energy consumed by these constituents is non-trivial and can be critical in reasoning about Web page energy consumption. Nonetheless, ARO's bottom-up approach to track the radio energy can be invaluable to analyze high throughput apps, and is complementary to *RECON*.

Power models using battery dynamics and system monitoring: The utilization-based approaches require external power monitors (or custom hardware) and exhaustive training. Instead, other research works [26, 41, 42] propose to build energy models without an external power monitor. Dong et al. [26] leverage the smart battery interface on phones to get accurate battery consumption, and use this to build power models. V-edge [41] models smartphone power by leveraging the instant battery voltage dynamics. Voltage levels change as the battery drains, and V-edge learns this correlation. Appscope [42] models power consumption by monitoring the changes at the kernel. AppScope monitors fine-grained utilization at the Android Binder level and at the system call level. Pathak et al. [33] perform fine-grained system call tracing to model power consumption. They combine the system call tracing with FSM power models to handle non-utilization-based power behaviors, such as the tail power [20].

Building power models for in-the-wild studies: Shye et al. [37] employ the utilization-based modeling approach to study the power behavior in the wild. This 2009 study finds that CPU and screen are the two biggest power consumers. Recently, Chen et al. [23] perform a more sophisticated utilization-based power modeling. The paper describes a large-scale user study that examines the power consumption patterns of 1520 devices.

Power consumption of mobile browsers: There have been relatively few studies on power consumption of specific applications, such as the mobile browser. While Qian et al. [34] study the resource and power consumption of mobile browsers, they focus on the power consumed by the networking component of the browsers alone. Our results (§7.1) show that browsers perform both networking and computing activities, and both consume considerable power. We thus study the power consumption of mobile browsers as a whole.

The works described above, with the exception of Qian et al. [34], are macro-level studies: they study the power consumption of the entire smartphone or long-running apps. Instead, the goal of *RECON* is to study the power consumption at the micro-level for a specific application, namely mobile browsers. Leveraging utilization models [23, 43] for such small time scales incurs high resource overhead, and consequently results in poor modeling accuracy. System call or kernel tracing techniques [26, 41, 42] are operating system specific, and not application specific. Instead, *RECON* combines *application-specific* component analysis with coarse-grained resource modeling.

Note that there are other works, such as Zhu et al. [45, 46], that aim to improve browser power consumption, but do not focus on modeling.

9 CONCLUSION

Accurate energy modeling of the page load process is challenging because Web pages are *complex and short-lived*. Deconstructing the energy consumption into constituent component- and resource-level contributions is even more challenging because power monitors only report *aggregate* power consumption values. We present *RECON*, a modeling approach that combines low-level page load information with coarse-grained resource monitoring. We show that *RECON* can predict the energy consumption of 80 Web page loads with a mean error of less than 7%. We employ *RECON* to accurately predict the impact of four different Web page optimizations. Importantly, *RECON*'s component- and resource-level energy deconstruction provides visibility into how and why an optimization affects energy consumption; this information can be invaluable to Web developers and content providers who wish to design efficient Web pages. The *RECON* model is currently implemented in MATLAB. Code and relevant scripts for *RECON* have been made available online [12].

REFERENCES

[1] 2011. How loading time affects bottomline. https://blog.kissmetrics.com/loading-time. (2011).
[2] 2012. 75% of Developers Using HTML5: Survey. http://www.eweek.com/c/a/Application-Development/75-of-Developers-Using-HTML5-Survey-508096. (2012).
[3] 2012. How one second could cost Amazon 1.6 billion in sales. http://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales. (2012).
[4] 2014. BSDGeek Jake Ad Blocker. http://androidforums.com/threads/guide-best-ad-blocking-no-app-needed-for-rooted-phone.853586/. (2014).
[5] 2015. Google chrome promises longer battery life and fast performance, eyes Safari. http://www.techtimes.com/articles/60277/20150614/googles-mac-friendly-chrome-promises-faster-performance-longer-battery-life.htm. (2015).
[6] 2015. Google promises update as users suffer. http://www.technobuffalo.com/2015/06/12/google-promises-chrome-updates-as-users-suffer/. (2015).
[7] 2015. Monsoon Power Monitor. http://msoon.github.io/powermonitor/. (2015).
[8] 2015. No, Apps Aren't Winning. The Mobile Browser Is. http://marketingland.com/morgan-stanley-no-apps-arent-winning-the-mobile-browser-is-144303. (2015).
[9] 2015. Page Speed Insight Rules. https://developers.google.com/speed/docs/insights/rules?hl=en. (2015).
[10] 2017. Calabash. http://calaba.sh. (2017).
[11] 2017. Chrome Developer Tools. https://developers.google.com/web/tools/chrome-devtools/?hl=en. (2017).
[12] 2017. Code and relevant scripts for *RECON*. https://github.com/davycao/Deconstructing-Mobipower. (2017).
[13] 2017. Google Pagespeed Insights. https://developers.google.com/speed/pagespeed/insights. (2017).
[14] 2017. mod_pagespeed. http://www.modpagespeed.com/. (2017).
[15] 2017. Module ngx_http_gzip_module. http://nginx.org/en/docs/http/ngx_http_gzip_module.html. (2017).
[16] 2017. Safari: Longer battery life and faster performance. http://www.apple.com/safari/. (2017).
[17] 2017. SPDY. https://www.chromium.org/spdy/spdy-whitepaper. (2017).
[18] 2017. The top 500 sites on the web. http://www.alexa.com/topsites/. (2017).
[19] Victor Agababov, Michael Buettner, Victor Chudnovsky, Mark Cogan, Ben Greenstein, Shane McDaniel, Michael Piatek, Colin Scott, Matt Welsh, and Bolian Yin. 2015. Flywheel: Google's Data Compression Proxy for the Mobile Web. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI '15)*. Oakland, CA, USA, 367–380. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/agababov
[20] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. 2009. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference (IMC '09)*. Chicago, Illinois, USA, 280–293.
[21] Michael Butkiewicz, Daimeng Wang, Zhe Wu, Harsha V. Madhyastha, and Vyas Sekar. 2015. Klotski: Reprioritizing Web Content to Improve User Experience on Mobile Devices. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI '15)*. Oakland, CA, USA, 439–453.
[22] Aaron Carroll and Gernot Heiser. 2010. An Analysis of Power Consumption in a Smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference (USENIX ATC '10)*. Boston, MA, USA, 21–21. http://dl.acm.org/citation.cfm?id=1855840.1855861
[23] Xiaomeng Chen, Ning Ding, Abhilash Jindal, Y Charlie Hu, Maruti Gupta, and Rath Vannithamby. 2015. Smartphone energy drain in the wild: Analysis and implications. *ACM SIGMETRICS Performance Evaluation Review* 43, 1 (2015), 151–164.

[24] Padraig Cunningham, John Carney, and Saji Jacob. 2000. Stability problems with artificial neural networks and the ensemble solution. *Artificial Intelligence in Medicine* 20, 3 (2000), 217–225.

[25] Philip Dixon. 2009. Shopzilla's Site Redo - You Get What You Measure. In *2009 Web Performance and Operations Conference (Velocity)*. San Jose, CA, USA.

[26] Mian Dong and Lin Zhong. 2011. 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 (MobiSys '11)*. Bethesda, Maryland, USA, 335–348.

[27] Simon Haykin. 2004. *Neural Networks: A Comprehensive Foundation* (3rd ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.

[28] Jeff Heaton. 2008. *Introduction to Neural Networks for Java* (2nd ed.). Heaton Research, Inc., St.Louis, MO, USA.

[29] Radhika Mittal, Aman Kansal, and Ranveer Chandra. 2012. Empowering developers to estimate app energy consumption. In *Proceedings of the 18th annual international conference on Mobile computing and networking (MobiCom '12)*. Istanbul, Turkey.

[30] Stephen G. Nash and Jorge Nocedal. 1991. A Numerical Study of the Limited Memory BFGS Method and the Truncated-Newton Method for Large Scale Optimization. *SIAM Journal on Optimization* 1, 3 (1991), 358–372.

[31] Javad Nejati and Aruna Balasubramanian. 2016. An In-depth Study of Mobile Browser Performance. In *Proceedings of the 25th International Conference on World Wide Web (WWW '16)*. Montreal, Quebec, Canada, 1305–1315.

[32] Ravi Netravali, James Mickens, and Hari Balakrishnan. 2016. Polaris: Faster Page Loads Using Fine-grained Dependency Tracking. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16)*. Santa Clara, CA, USA, 123–136.

[33] Abhinav Pathak, Y Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. 2011. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems (EuroSys '11)*. Salzburg, Austria, 153–168.

[34] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. 2011. Profiling Resource Usage for Mobile Applications: A Cross-layer Approach. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys '11)*. Bethesda, Maryland, USA, 321–334. DOI: https://doi.org/10.1145/1999995.2000026

[35] R Development Core Team. 2008. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org

[36] Kent Rasmussen, Alex Wilson, and Abram Hindle. 2014. Green Mining: Energy Consumption of Advertisement Blocking Methods. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software (GREENS '14)*. Hyderabad, India, 38–45. DOI: https://doi.org/10.1145/2593743.2593749

[37] Alex Shye, Benjamin Scholbrock, and Gokhan Memik. 2009. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42)*. New York, NY, USA, 168–178.

[38] Donald Specht. 1991. A general regression neural network. *IEEE Transactions on Neural Networks* 2, 6 (1991), 568–576.

[39] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2013. Demystifying Page Load Performance with WProf.. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation (NSDI '13)*. Lombard, IL, USA, 473–485.

[40] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2014. How Speedy is SPDY?. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI '14)*. Seattle, WA, USA, 387–399. http://dl.acm.org/citation.cfm?id=2616448.2616484

[41] Fengyuan Xu, Yunxin Liu, Qun Li, and Yongguang Zhang. 2013. V-edge: Fast Self-constructive Power Modeling of Smartphones Based on Battery Voltage Dynamics.. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation (NSDI '13)*, Vol. 13. Lombard, IL, USA, 43–56.

[42] Chanmin Yoon, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, and Hojung Cha. 2012. AppScope: Application Energy Metering Framework for Android Smartphones Using Kernel Activity Monitoring. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference (USENIX ATC '12)*. Boston, MA, USA, 36–36. http://dl.acm.org/citation.cfm?id=2342821.2342857

[43] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P Dick, Zhuoqing Morley Mao, and Lei Yang. 2010. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis (CODES/ISSS '10)*. Scottsdale, Arizona, USA, 105–114.

[44] Yifan Zhang, Xudong Wang, Xuanzhe Liu, Yunxin Liu, Li Zhuang, and Feng Zhao. 2013. Towards Better CPU Power Management on Multicore Smartphones. In *Proceedings of the Workshop on Power-Aware Computing and Systems (HotPower '13)*. Farmington, PA, USA.

[45] Yuhao Zhu and Vijay Janapa Reddi. 2013. High-performance and energy-efficient mobile web browsing on big/little systems. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA '13)*. Shenzhen, China, 13–24.

[46] Yuhao Zhu and Vijay Janapa Reddi. 2014. WebCore: Architectural Support for Mobile Web Browsing. In *Proceeding of the 41st annual international symposium on Computer architecuture (ISCA '14)*. Minneapolis, MN, USA, 541–552.