Analyzing the Power Consumption of the Mobile Page Load

ABSTRACT

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 predicts the energy consumption of any Web page load. Our key intuition is to leverage low-level application semantics in addition to coarse-grained resource utilizations while 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 predictions without requiring complex models or analyses. Experiments across 80 Web pages and under four different optimizations show that *RECON* can predict the energy consumption for a Web page load with an average error of less than 7%. Further, *RECON* helps to analyze and explain the energy effects of an optimization on Web page loads.

1. INTRODUCTION

Mobile Web page performance is extremely critical to content providers [6, 28], service providers [14], and users [7]. Slow Web pages are known to adversely affect profits [6, 28] and lead to user abandonment [7]. Not surprisingly, several optimizations have been proposed to improve mobile Web performance [4, 12, 14, 17, 24].

However, 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 point [3, 11]. The problem is that energy and performance are separate metrics, and Web optimizations that improve Web performance may not have the same effect on energy consumption. Not knowing how a Web optimization affects energy consumption can have severe consequences; a recent software update on Chrome resulted in excessive battery drain, leading to severe backlash [5].

As a result, content providers and browser vendors often meticulously examine the impact of any enhancement to their products on device energy consumption; enhancements include optimizations, addition of new features, or updates. Today, the energy consumption of a Web page load is examined using power monitors such as the Monsoon power monitor [9]. To assess the energy effects of Web enhancements, one needs to repeat the power measurements on each enhancement or combinations thereof using the monitor. Since Web page loads exhibit high variance [33], the power measurements have to be repeated for several runs for every update to the page. Worse, power meters can only report 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.

Our goal is to provide quick, accurate, and fine-grained estimations of the energy consumption for a page load instantiation. 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* [19, 27]; 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 [32]. A Web enhancement can have widely varying effects on different page load activities. Thus, studying the energy impact of a Web enhancement on page load requires understanding its effects on *each* page load activity. Further, Web page loads exhibit high variance [33].

Existing approaches to analyzing mobile energy typically focus on profiling and modeling the resource consumption of the device during execution [27, 29]. Such approaches consider long-running services and apps such as games, audio, and video streaming [19, 36], 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 predict 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 page load activities. In particular, 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; *RECON* lever-

ages component-level information such as activity type, start time, and end time. Such information can be obtained from several sources including Scout [26] or a combination of `chrome://tracing` and Chrome developer tools [2]; for *RECON*, we leverage WProf-M [25].

Modeling at the component level allows predicting the energy consumption of sub-second Web page loads, and lets us study the power consumption of each page load activity. We use multiple linear regression to model the Web page power consumption. While we do initially require the power monitor to train our regression model, we can then predict the energy consumption of the Web page when loaded again as-is or upon applying any enhancement, *without* the power monitor.

We experimentally evaluate *RECON* on the Samsung Galaxy S4, S5, and Nexus devices using 80 Web pages. We validate our modeling approach by predicting the energy consumption of Web page loads and comparing with actual power measurements from a fine-grained power meter; results show that *RECON* can predict the energy consumption of the entire page load with a mean error of 6.9% and that of individual page load activity segments with a mean error of 13.1%. By contrast, modeling resources alone increases the activity segments error by 14%. We further show that *RECON* can accurately predict the energy consumption of a Web page under different network conditions, even when the model is trained under a default network condition.

One of the key applications of *RECON* is to study how Web page enhancements affect energy consumption. To this end, we study four optimizations: *Compression*, *Minification*, *Inlining*, and *Ad-block*. *RECON* is able to predict the energy consumption of a given optimization with an average prediction error of 5.45%. *RECON* trains its models on *unoptimized* runs of the Web page, and predicts the energy consumption when the optimized version of the page is loaded.

We leverage *RECON*'s component-level analysis to: *(i)* show that *compression* optimization can increase the power consumption of Javascript analysis and HTML parsing, even though it reduces the power consumption of network loads, *(ii)* study why certain optimizations such as *inlining* and *compression* can increase the energy consumption of specific Web page loads, and *(iii)* help choose the best design parameters for a given optimization by quickly predicting the energy consumption for a diverse set of optimization parameters.

To summarize, we make the following contributions:

1. We present *RECON*, a modeling approach that uses component and resource information to predict Web page energy consumption with a 6.9% mean error.
2. *RECON* accurately predicts the impact of four Web optimizations on page load energy. *RECON* is able to analyze how and why an optimization has varying effects on the energy consumption of different pages.
3. *RECON* further allows us to quickly evaluate a large number of optimization design choices with respect to tradeoffs in energy and performance.

2. BACKGROUND

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 from the network. 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*, or *Cascading Style Sheets (CSS)*, it downloads the object. If the object is a Javascript or a CSS, then these are further *evaluated*, and if needed, the embedded objects are fetched. The HTML parser and the script evaluator together build the Document Object Model (DOM) progressively. The rendering engine reads the DOM and renders the page on the browser. The time taken for the entire page to load is called the Page Load Time (PLT).

WProf/WProf-M: Many of the components of the page load process can be executed in parallel; for example, the HTML parsing and an image download can happen in parallel. However, some components depend on each other and will have to be executed in order. The WProf tool [32] is an in-browser profiler that extracts both the browser dependencies and low-level timing information of page loads.

WProf-M [25] is an extension of WProf for mobile browsers. *RECON* uses WProf-M for decomposing the mobile page load process into its various components, and for providing fine-grained timing information for each component. While WProf-M also identifies the critical path and dependencies, *RECON* does not use these features. Figure 2 (a) shows the component level decomposition of the page load process, as provided by WProf-M.

While we use WProf-M, other tools that provide component-level information such as the Scout tool [26] or Google developer tools can also be used in *RECON*.

3. EXPERIMENTAL SETUP

RECON logs the power, Web page components, and coarse-grained resources during the page load for modeling. We describe each in turn, including our setup.

3.1 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

Figure 1: Our hardware setup showing the Samsung S4 under test connected to the Monsoon power monitor.

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 in Pittsburgh.

3.2 Power, Component, and Resource logging

Power Logging: To measure the device power consumption, *RECON* uses an external power monitor, Monsoon [9], which performs fine-grained power measurement at a 5KHz frequency. The power monitor is only used to build the models and is not used for predictions.

Figure 1 shows the Samsung Galaxy S4 device loading a mobile Web page, *fico.com*, while recording the instantaneous current draw through the Monsoon power monitor. We maintain a constant voltage level throughout our experiments.

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 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)).

Resource logging: *RECON* combines component-level modeling with coarse-grained resource monitoring. To monitor resources, we use a simple android service that records resource consumption values. For CPU, we collect per-core CPU utilization from `/proc/stat` and per-core CPU frequency from `/sys/devices`. For network, we collect number of bytes transmitted and received during an interval from `/proc/net/dev`. To maintain low monitoring overhead, we log resources every 0.1 seconds. When we increase the resource monitoring frequency to once every 0.01 seconds, the CPU utilization increases by 30%, which is clearly infeasible.

Our resource monitoring only focusses on CPU and network. Although the screen power consumption is critical to Web page loads, Chen et al. [19] 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 prediction accuracy of *RECON*; we thus omit it from our monitoring.

Both WProf-M and resource monitoring use logcat, Android's logging software, to record raw data that is then analyzed offline. 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* [1] 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 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 [25]. 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.

3.3 Web pages

We experiment with 80 Web pages. 60 Web pages are randomly selected from top 100 Alexa Web Pages [15] from 10 different countries. The remaining 20 Web pages are randomly selected from pages ranked between 100 to 10,000 on the Alexa site. 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. The local HTML only ensures that the same set of objects are embedded for each run of the Web page.

All Web page loads are cold loads and the cache is cleared after each load.

4. 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).

In this section, we first discuss the challenges involved in modeling Web page power consumption in §4.1. We

then present, in §4.2, the design of *RECON*, followed by a description of our modeling approach in §4.3.

4.1 Challenges in Web page power modeling

The primary challenge in power modeling is the *short-lived* nature of the page load process. Employing resource utilization-based approaches for modeling Web pages will require very fine-grained (high frequency) resource monitoring, which is known to incur significant overhead and leads to poor modeling accuracy [19, 27].

The next challenge is the *complexity* of the page load process. The page load consists of various components, and a given page enhancement can affect each component differently. Without component-level visibility, it is difficult to investigate the impact of enhancements.

A related challenge is the *variance* in the page load process [33]. Our own experiments show that Web page loads exhibit high variance, both in terms of PLT and energy. Variance is caused by changes in the network, mobile OS, web server, etc. Because of variance, measuring (via power monitors) the power consumption of one instantiation of the Web page load is not enough. Since variance is inherent in the page load, the challenge is to model energy consumption despite the variance.

4.2 The design of RECON

RECON uses component-level Web page information that provides significant insight into the page load process and augments this with coarse-grained resource monitoring. Together, this information provides the *visibility* needed to understand the complexities and dependencies within the short-lived page load process without incurring significant overhead. We show in §5.4 why *both* component-level and resource-level information are necessary for achieving high accuracy.

At a high-level, *RECON* works by developing a parameterized power model that incorporates the component- and resource-level information to make accurate predictions using multiple linear regression. Linear regression works by minimizing the variance of the prediction error over the model, and is thus well suited for the highly variable page load process. We also minimize the effect of noise in the page load process by training our model over several instantiations of the Web page.

4.3 RECON modeling

WProf-M gives us the decomposition of the Web page into its components, as illustrated in Figure 2 for the *instagram.com* page, juxtaposed with its power consumption obtained via the power monitor. The problem is that the *monitored power consumption is a result of several simultaneously executing components and their aggregate resource demand.* Given a new Web page load with an arbitrary distribution of components and resource utilizations, how can we predict its energy?

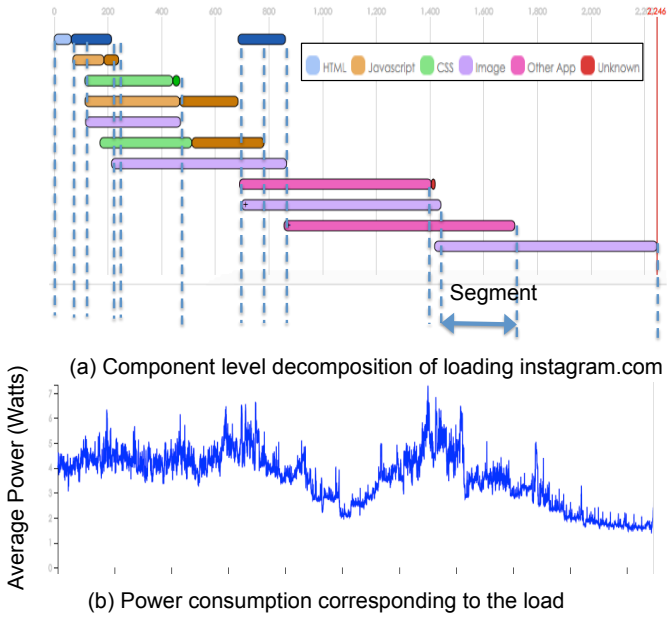


Figure 2: (a) Using WProf-M to decompose the components when loading the *instagram.com* Web page. (b) The instantaneous power draw recorded as the Web page loads. Segments are shown in dotted lines.

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. Segments of the *instagram.com* page are illustrated in Figure 2. 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 indicated by the dotted blue lines in Figure 2.

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. We model the power consumption of a segment, s, at any time as:

$$\hat{P}_s = w_0 + \sum_{i \in Resources} w_i R_i + \sum_{j \in s} w_j, \qquad (1)$$

where i represents the various resources we monitor, such as CPU utilization, and R_i represents the average utilization for that resource during s; j represents a component, and thus $j \in s$ is the set of all components that make up segment s; finally, w_i and w_j are coefficients (independent of s) representing the power contribution of the resources and components, and are variables that need to be determined. w_0 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 weights, $\vec{w}$.

Multiple Linear Regression: 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 in Eq. (1) using indicator variables. For resources, we use 14 variables: the time-averaged cpu utilization, frequency, and the product of utilization and frequency for all 4 logical cores, and throughput of bytes sent and received. The power consumed by the device CPU is known to depend on the (utilization, frequency) pair [19,37]; we use the product of utilization and frequency to account for this dependence. Further, network throughput, CPU frequency, and CPU utilization, are known to linearly affect power consumption [21–23], thus motivating our linear model in Eq. (1). Note that resource utilizations are averaged over the length of the segment.

While more powerful (yet complicated) models, such as neural networks, can be used for modeling the power consumption, $\hat{P}_s$, our focus here is on exploiting application-level semantics to obtain a rich feature set. We find that using simple multiple linear regression with these features suffices to achieve high accuracy without incurring substantial overhead or complexity.

Training: We train our regression model on nine instantiations (or runs) of a given Web page and test on the tenth instantiation. Our error results are based on a 10-fold cross validation. For each run, *RECON* records the power, page load components, and resources. We use the instantaneous power measurements to calculate the average power consumption, $\hat{P}$, for each segment. We then use regression over the several segments collected during the nine runs to derive weights, $\vec{w}$, for each observed component and resource variable.

Testing: The weights, derived via training, for our power model in Eq. (1) can now be used to predict the power and energy consumption of any new instantiation of the Web page without requiring the power monitor. The inputs are the set of components involved in the page load process and the corresponding coarse-grained resource consumptions for the new instantiation. We apply our trained model on the test data by substituting the learned weights, $\vec{w}$, in Eq. (1) along with the above-mentioned inputs. This gives us the predicted power consumption for every segment; we then predict the energy consumption by multiplying the predicted 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 predicted energy consumption of the new Web page load.

Note that the energy consumption of the new Web page load is likely to vary significantly from the page load instantiations used for model training because of variance that affects the page components. To address this, *RECON* models per-component *power* consumption (and not energy). This allows us to predict the energy consumption of a new page load, despite the variance, by obtaining the component lengths and multiplying them with the component power estimates (weights).

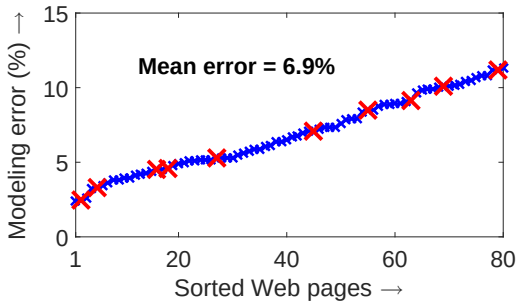


Figure 3: Modeling error (sorted) for predicted Web page energy consumption for all 80 pages. *Red* crosses indicate specific pages that we analyze in Figure 4.

An advantage of *RECON* is that we can *train our model online in a few minutes* without having to build detailed subsystem-level models as in prior work. For example, recent work [19] 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. Instead, *RECON* runs a handful of Web page loads for training, that together take a few minutes to complete. Of course, our focus here is only on modeling the energy consumption for the *browser* which allows for much faster model training. We discuss the usage scenarios for *RECON* in §8.

5. MODEL VALIDATION AND RESULTS

We now validate our model and present results from our *RECON* study. The validation results are presented in §5.1. We then present results under different network conditions in §5.2 and for different devices in §5.3. Finally, we compare *RECON*, quantitatively and qualitatively, with resource-only modeling in §5.4.

5.1 Model validation

Web page energy consumption: We employ *RECON* to predict the energy consumption of 80 Web pages (as discussed in §3.3) and compare the model-predicted values with actual measurements from the Monsoon power meter. The average prediction error across these Web pages is 6.9% (using 10-fold cross validation for each). Figure 3 shows the sorted model prediction error for the 80 Web pages; note that each prediction error value is obtained by averaging over ten instantiations for each Web page as described in §4.3.

Figure 4 takes a closer look at the error for ten well-known Web pages indicated by the red crosses in Figure 3. Figure 4 shows the actual and predicted mean energy for these pages along with the 95th percentile confidence intervals. The confidence intervals are obtained

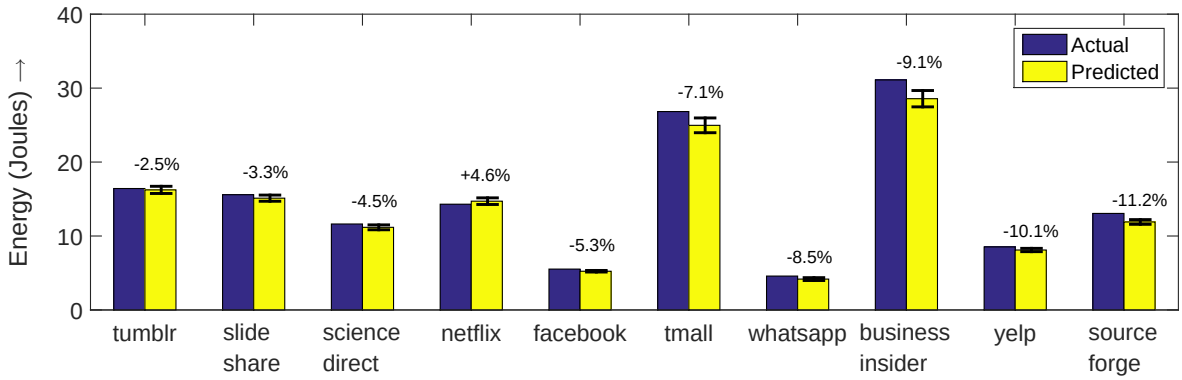


Figure 4: Actual versus predicted energy consumption for ten selected Web pages from Figure 3. The percentages above the bars indicate modeling error. Confidence intervals around the predicted mean energy are also shown.

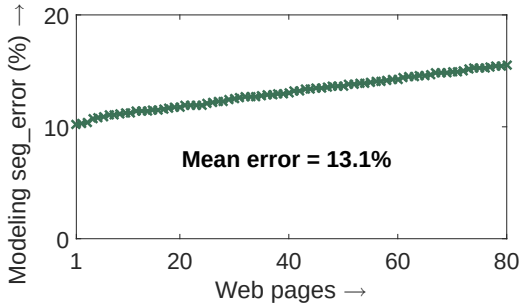


Figure 5: Average modeling seg_error (sorted) for predicted segment energy consumption for all 80 pages.

from the standard deviation of the ten per-instantiation energy predictions for each page. The tight intervals suggest that our energy predictions have low variability.

Segment energy consumption: *RECON* can also be used to predict fine-grained segment-level energy consumption directly using Eq. (1). 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 §6. The average segment-level modeling error, referred to as seg_error (to distinguish from full page prediction error), across all 80 Web pages is 13.1%. Note that the full Web page prediction 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 5 shows the sorted model energy prediction seg_error for the 80 Web pages; the order of the Web pages here is slightly different from that in Figure 3. As before, each prediction error value is obtained by averaging the per-instantiation seg_error over ten instantiations for each Web page. Figure 6 shows the actual and predicted segment-level power consumption for specific instantiations of three Web pages. We see that the predicted values closely track the measured values.

Analysis of model weights: One immediate application of *RECON* is to study the relative energy consumption of different page load activities (or components) such as HTML parsing and image loads. Recall (§4.3) that *RECON* derives weights, w, for each component of the page load to estimate total power. These weights represent the relative contribution of each component and resource to total power consumption.

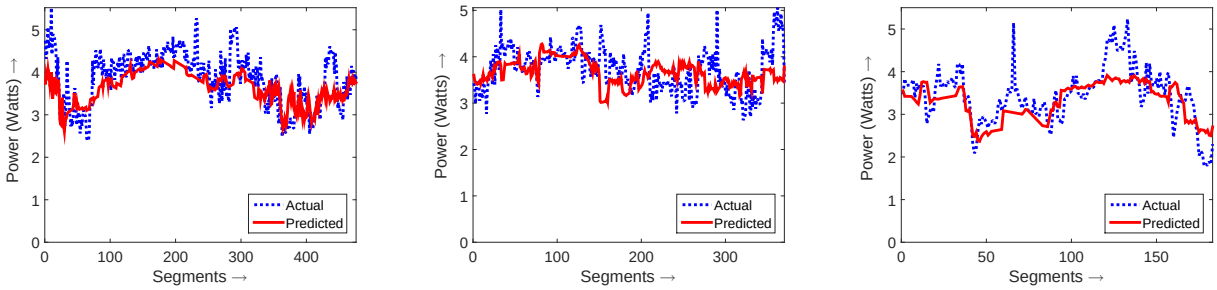
Figure 7 shows the relative component power contributions for *alexa.com* and *fico.com*. For ease of presentation, we adjust the power numbers such that the smallest contribution is 0 while holding the relative differences constant. Here, js refers to Javascript. Note that we often encounter multiple types of components, such as different types of Javascript, CSS, etc. We classify them as different components in our modeling. In our experiments, we find that the baseline weight, w_0, typically accounts for 14-61% of the total power.

In general, we find that the highest contributors to power consumption are objects that require external or internal application support, such as application/octet-stream and Javascript. However, as is evident from the figures, the relative ordering of the components is different. For example, text and Javascript are the most power consuming components for *alexa.com*, while octet-stream (typically binary files or executables) is the most power consuming component for *fico.com*.

We can apply the same weight analysis to determine the power contribution of resources. For example, a similar analysis over the lifetime of *alexa.com* reveals that network-related activities consume 57.7% less power than CPU-related activities, indicating that *alexa.com*'s page load process is more computationally intensive in terms of power consumption (not shown in figure).

5.2 Modeling results under different networks

Thus far our experimental results employed the default WiFi network described in §3.1. It is interesting to ask whether the power model trained on this default network can be used to accurately predict the power consumption on a different network. Specifically, *can the weights (in Eq.* (1)*) derived via training on one net-*



(a) Segment-level modeling for *www.mirror.co.uk*. Error=9.6%.

(b) Segment-level modeling for *s.kakaku.com*. Error=8.8%.

(c) Segment-level modeling for *www.alexa.com*. Error=9.7%.

Figure 6: Actual versus predicted segment-level power consumptions for specific Web page instantiations.

work provide accurate estimates for power and energy consumption on another network? We use the above-derived model (trained on the default network) to predict 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 prediction error across ten different Web pages is an impressive 4.8% and the seg_error is 14.9%.

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 prediction error across ten different Web pages is only 5.7% and the seg_error is 18%.

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; this increases the average PLT by about 27% over the default WiFi network. Our average energy consumption prediction error across ten different Web pages is a disappointing 21.2% and the seg_error is 22.3%.

The above results reveal that cross-network model predictions can be accurate, as long as the same network interface is used for training and testing. Intuitively, this makes sense as the behavior and dynamics of the network are interface-specific, and can thus affect PLT and energy consumption [36]. In fact, when we train on the cellular network and then test our model on ten different Web pages also on the cellular network, our modeling error decreases to a satisfactory 4.8% and the seg_error also reduces to 14.2%.

5.3 Modeling results 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 predict the energy consumption of ten Web pages on the Galaxy S5 and Galaxy Nexus devices (see §3). We obtain a low full page mean modeling error of 7.1% and seg_error of 15.6% for the S5, and modeling error of 9.1% and seg_error of 15.7% 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 [27, 36].

5.4 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 [19,31,34, 36], to construct similar models. However, such models do not perform as well as *RECON*.

In particular, when we model power consumption using only resource-level information, the seg_error is 14% higher than *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 same linear regression model. The seg_error is much higher in specific cases; in particular, the seg_error is 30% higher than *RECON* for segment sizes smaller than 0.1s. This is because such resource-only models are limited by the resource monitoring frequency which needs to be low (10/second, in our case) to ensure low CPU and power overhead (see §3.2).

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 decompressing it locally. These different phases are treated equally under WProf-M, resulting in poor accuracy for such components. Figure 8 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.

Web page	Component only	Resource only	*RECON*
naver.jp	18.0%	6.3%	6.2%
xiami	17.8%	5.9%	5.4%
sohu	14.9%	8.5%	4.4%
live	14.7%	6.7%	4.2%
baidu	10.9%	1.9%	1.1%
mirror	6.9%	17.4%	6.3%
acfun.tv	9.6%	11.8%	2.5%
github	3.2%	8.5%	1.8%
paypal	2.9%	6.9%	1.3%
booking	5.4%	6.2%	1.8%

Table 1: Modeling seg_error for specific Web page instantiations. The first five rows depict pages with high component-only seg_error and the last five depict pages with high resource-only seg_error. In all cases, *RECON* (combined model) has the lowest seg_error.

Table 1 highlights the modeling seg_error for resource-only and component-only approaches and compares them with *RECON* for certain Web pages. In some cases resource-only models perform well but component-only models perform poorly, whereas in other cases we see the opposite behavior. Regardless, in all these cases, *RECON* performs well by leveraging information from both sources. Note that the seg_error reported in Table 1 is for specific instantiations of Web pages, and is different from the mean per-Web page seg_errors shown in Figure 5 that are obtained by averaging across several instantiations of the same Web page.

6. APPLICATIONS: OPTIMIZATION

One of the key applications of *RECON* is to examine the energy effects of Web page enhancements. To this end, we study four optimizations. Three of these, *Compression*, *Minification*, and *Inlining*, are commonly used best practices to improve Web performance [10]. In fact, Google's Web optimization tool, PageSpeed [4], uses all of the three optimizations. The fourth optimization we consider is *Ad-block*; removing ads is a common technique to reduce resource consumption.

RECON is able to predict the energy consumption of a given optimization with an average prediction error of 5.45%. By leveraging this low prediction error, we use *RECON* to enable the following applications:

- Analyze the effect of the optimization on Web page components. For example, to compare the energy effects of an optimization on HTML parsing vs. Javascript evaluation vs. network loading.
- Examine not only how an optimization affects the energy consumption of page load, but also study why the energy consumption changes.
- Quickly evaluate a large number of optimization design choices with respect to tradeoffs in energy and performance.

Optimization	ΔPLT	ΔEnergy	Error
Compression	-21.1%	-24.4%	4.3%
Inlining	+15.9%	+6.7%	5.3%
Minification	-5.5%	-1.7%	7.9%
Ad-block	-29.3%	-28.5%	4.1%

Table 2: Modeling results for various optimizations.

Note that the above three applications cannot be enabled without *RECON*, even if one employs power meters to measure the Web page power consumption.

6.1 Optimization methodology

To analyze the effects of an optimization on Web page energy consumption, we first train the *RECON* model on ten runs of the *unoptimized* version of that Web page, and then predict the energy consumption of the optimized Web page (without having trained on it). We analyze optimizations for 10 randomly selected Web pages from the original 80 described in §3.3.

As before, *RECON* predicts the power consumption of a given instantiation of a Web page load. An optimized version of the Web page is loaded, and the resource- and component-level information is collected. *RECON* then uses its trained model to predict the power consumption of the optimized Web page. We run the Web pages with and without optimization for 10 runs for each optimization.

Unlike the methodology in §3, for the optimization experiments, we download and store the HTML *and* all the associated objects on our own local Web server. We do this because we need to apply the optimization on the Web page without server support, and this is only possible if the entire Web page is available locally. Below we describe the four optimizations and our methodology in applying these optimizations.

1. **Compression** reduces the size of embedded Web objects by applying various compression algorithms on the server side. This reduces the network time and energy to download the objects at the client. However, the objects have to be decompressed, adding a computational overhead. Typically, the decompression overhead is higher for larger objects, because of the increased CPU cost in decompressing them [8]. To apply the compression optimization on a Web page, we enable *mod_deflate* [8], a compression module, on our local Web server.
2. **Inlining** embeds all the external Javascript and CSS in the root HTML file. The inlined HTML file is bigger than the original HTML, resulting in increased network latency and energy to download it. However, once downloaded, there is no need to fetch the external Javascript and CSS, thus avoiding several

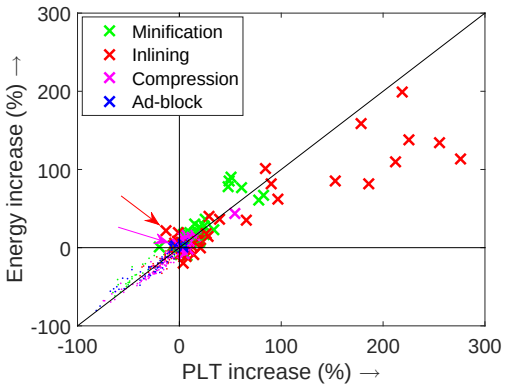


Figure 9: Change in PLT and energy consumption under various optimizations.

small downloads. Importantly, inlining reduces network dependencies, since the browser need not wait for external scripts to be downloaded to act on them. To apply the inlining optimization, we parse the Javascript and CSS embedded in the HTML file and add them to the HTML. We only inline the first-level scripts, and do not inline recursively embedded scripts.

3. **Minification** removes comments and spaces in the HTML file to reduce the size of the file, potentially reducing the network latency and energy. Minification has no effect on computation.
   To apply the minification optimization, we use Yahoo's YUI Compressor [13] that minifies scripts.
4. **Ad-block** removes all ads from the Web page. We emulate actual ad-blockers by manually analyzing unmodified HTML and removing all ad content.

Of course, several other optimizations have been designed to improve Web performance [12, 14]; we plan to study these optimizations as part of future work.

6.2 Energy effects of an optimization

RECON is able to predict the energy consumption of a given optimization with a low mean prediction error of 5.45%, averaged across multiple runs of 10 different Web pages.

An optimization may not affect PLT and energy similarly, because these two metrics are measured over different parts of the page load process; PLT only depends on components on the critical path, whereas energy depends on all components. Using our optimization experiments, we study the effect of each optimization on PLT versus energy. Our findings are summarized in Table 2. For instance, compression reduces PLT by 21.1% and reduces energy consumption by 24.4%

Next we look at the effect of the optimization per-page. Figure 9 shows the change in PLT and energy consumption of the Web page loads across all our experiments under all four optimizations. We divide the figure into four quadrants, and show the y=x line for reference. The quadrant view allows us to understand

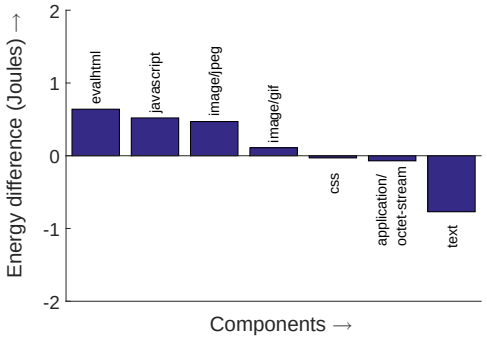


Figure 10: Change in predicted per-component energy usage under compression for *www.fico.com*.

the joint effect of an optimization on both PLT and energy. Note that negative numbers indicate a decrease in PLT or energy consumption after optimization.

Figure 9 shows that optimizing performance improves both PLT and energy in *only* 70% of the instances; these are depicted as the (purposely shrunk) dots in the lower-left quadrant of the figure. Of the remaining 30% (shown as crosses), (i) 20% of the cases fall in the upper-right quadrant where the optimization increases both PLT and energy, (ii) roughly 5% lie in the upper-left quadrant exhibiting a decrease in PLT but *increase* in energy, and (iii) roughly 5% lie in the bottom-right quadrant exhibiting an increase in PLT but *decrease* in energy. Note that the impact of these counterintuitive 30% cases is not negligible. For example, inlining (red crosses), minification (green crosses), and compression (magenta crosses) can increase PLT and energy consumption by about 200%, 90%, and 45%, respectively. Further, inlining and minification have several instances of increase in PLT and energy (upper-right quadrant). We will study some of these instances in detail in §6.4.

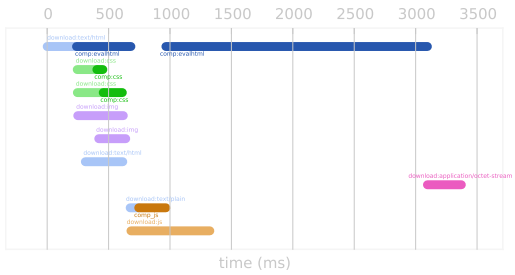
6.3 Component analysis for optimizations

Another related application based on *RECON* is component analysis of a Web page before and after optimization. Specifically, given sufficient training data for unoptimized and optimized Web pages, we can construct regression models for both versions and compare the difference in energy consumption per component.

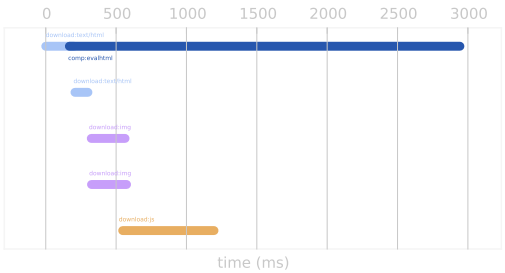
Figure 10 shows our results comparing the component-level energy difference after and before compression for the Web page *fico.com*. We see that, after compression, the energy consumption of evalhtml and Javascript increases, indicating that compression is not useful for these components. On the other hand, the energy consumption for text and CSS decreases, suggesting that compression is useful for these components.

6.4 Analyzing why energy consumption changes

Beyond studying *how* energy consumption changes due to an optimization, *RECON*'s component-level modeling also allows us to explore *why* the energy consump-



(a) *netflix.com* before inlining



(b) *netflix.com* after inlining

Figure 11: Inlining: The figure shows the various components of *netflix.com* (a) before and (b) after inlining.

tion changes. To explore this further, we choose two examples of Web page loads and optimizations: (i) An inlined *netflix.com* Web page which has a 12.72% decrease in PLT but a 22.2% increase in energy (red arrow pointing towards it in Figure 9), and (ii) A compressed *mirror.co.uk* Web page which has a 2.69% decrease in PLT but a 4.76% increase in energy[1] (magenta arrow pointing towards it in Figure 9).

Inlined *netflix.com*: Figure 11 shows the components before and after inlining *netflix.com*. After inlining, the HTML parsing component (shown in *dark blue*) takes about 2800ms compared to about 2500ms (in total) before inlining. This is because inlining increases the size of the HTML by including all the external Javascript and CSS, thereby increasing the parsing time. *RECON*'s component-level prediction reveals that the HTML components after inlining consume 9% more energy compared to the original version; this contributes to the overall increase in energy of the inlined *netflix.com*.

However, inlining helps performance by avoiding the loading of small objects. Figures 11(a) and (b) show that indeed, the number of objects downloaded after inlining is smaller compared to before the optimization (e.g., no (green) CSS downloads). But these objects are downloaded in parallel to the parsing, and there are only a few of these objects; as a result, they do not contribute significantly to the total energy consumption.

[1]While these numbers for mirror.co.uk are within the error range of our modeling, it is interesting to analyze the effect of compression on the individual page load components.

Combining the increase in parsing energy due to inlining with the negligible decrease in energy consumption as a result of not downloading small objects leads to a net increase in the total energy after inlining.

Compressed *mirror.co.uk*: The unoptimized *mirror.co.uk* contains over 70 Javascript objects that can potentially be compressed. On analyzing the components after compression, we find that the network latency reduces by 13% (due to compression); this contributes to the reduction in PLT for *mirror.co.uk*. However, our (predicted) energy for loading the compressed objects increases by 8%. This is because the browser now needs to decompress a large number of objects resulting in higher computational effort and energy. This effort contributes to the increase in energy consumption for *mirror.co.uk*, despite the decrease in PLT.

6.5 Scalably evaluating optimization choices

Since *RECON* enables component-level analysis to tease out the impact of an optimization, we can also use *RECON* to choose the right optimization that provides the best trade-off between decrease in PLT and possible increase in energy. In particular, we can quickly evaluate, without requiring a power monitor, various design options for a given optimization.

Consider compression as a concrete example. There are several control knobs for compression such as type of object to compress, compression factor, compression algorithm, etc. Each of these can have an impact on the PLT and energy consumption of the optimized Web page. We focus on five different compression type options: (i) default compression (compresses all supported objects), (ii) only compress text, (iii) only compress CSS, (iv) only compress Javascript, and (v) compress CSS and Javascript. We analyze the energy consumption of four different Web pages under these compression options using *RECON*.

Figure 12 shows the predicted results, averaged across all experiments, of various compression options across all Web pages. The values here are relative to the uncompressed Web page. We see that different options provide different tradeoffs between PLT decrease and energy decrease. The cyan line represents y=x and is provided for reference.

Text compression lies on this y=x line, indicating that it has equal impact on PLT and energy. Further, the graph suggests that compressing text is valuable; this is in agreement with our observation for *fico.com* in §6.3. Except for Javascript compression, all other options lie above the y=x line. This indicates that these compression options decrease energy more than PLT. Interestingly, the decrease in energy and PLT when compressing Javascript alone is quite low, even lower than that for Javascript+CSS compression. This indicates that compressing Javascript (alone) is not very useful; in fact,

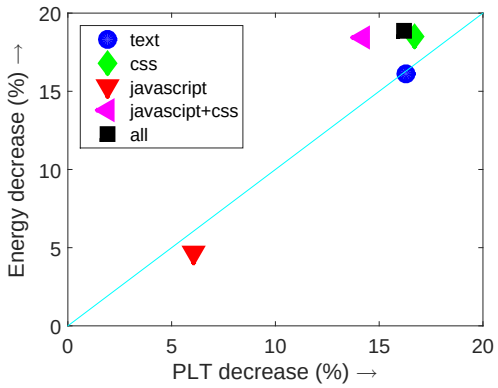


Figure 12: Trade-off between energy decrease % and PLT decrease % for various compression options.

we find that Javascript compression can sometimes *increase* PLT and energy consumption. This observation is again in agreement with our analysis in §6.3.

A similar analysis can be conducted for a given Web page to choose the best *page-specific compression option*. For example, our analysis for *alexa.com* suggests that, surprisingly, CSS-only compression is better, in terms of energy and PLT, than the default compression option that tries to compress all supported objects. These simple examples illustrate how we can use our modeling approach to quickly evaluate the impact of several optimization parameters. One can choose the right optimization in terms of both PLT and energy by using *RECON* to analyze several design options. The high accuracy of *RECON* makes it feasible to use this approach in practice; our modeling error for energy consumption across all compression options and all Web pages is only 4.3%.

7. 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 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 [36] uses the Monsoon power monitor [9] to measure the energy consumption under various CPU frequencies, WiFi data transfer rates, and screen brightness settings. PowerTutor predicts 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 [16]. 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 [29]. PowerTutor itself uses an FSM to build a model for cellular/WiFi power consumption. Chen et al. [19] 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. [18] 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 [19, 27], which is insufficient for modeling Web pages (see §5.4).

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 [20, 34, 35] propose to build energy models without an external power monitor. Dong et al. [20] leverage the smart battery interface on phones to get accurate battery consumption, and use this to build power models. V-edge [34] models smartphone power by leveraging the instant battery voltage dynamics. Voltage levels change as the battery drains, and V-edge learns this correlation. Appscope [35] 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. [27] 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 [16].

Building power models for in-the-wild studies: Shye et al. [31] 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. [19] 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. [30] 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 (§5.1) 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. [30], 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 [19, 36] for such small time scales incurs high resource overhead, and consequently results in poor modeling accuracy. System call or kernel tracing techniques [20, 34, 35] 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. [38, 39], that aim to improve browser power consumption, but do not focus on modeling.

8. USAGE SCENARIOS

Our work addresses the needs of content providers, browser vendors, and in some cases even Web designers, who wish to quickly and accurately evaluate the energy consumption of Web pages, either as-is or under different enhancements. Today, the energy consumption of a Web page load needs to be measured using power monitors, and the measurements have to be performed over several runs to account for variance. With *RECON*, one could quickly and efficiently estimate the energy consumption of Web page load instances without requiring a power monitor. Importantly, *RECON*'s component-level analysis enables several applications that cannot be enabled using power monitor measurements alone. *RECON* can be used to study which components most adversely affect the energy consumption. *RECON* also provides insights into how and why a given optimization affects the energy consumption a certain way.

In the future, we envision *RECON* also helping end-users and others assess the energy impact *in-the-wild*. However, this *in-the-wild* usage scenario requires decoupling the model training (in the lab, with power meters) with component- and resource-monitoring (on the end-user side), thus necessitating models that work across all devices under all possible network conditions. Prior work [27, 36] has shown that it is difficult to build such device- and network-independent energy models. We hope to address this challenge as part of future work to extend the benefits of *RECON*.

9. CONCLUSION

Accurate power modeling of the page load process is challenging because Web pages are complex and short-lived. 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% error. We employ *RECON* to accurately predict the impact of four different Web page optimizations. *RECON*'s component-level information provides visibility into how and why an optimization affects energy consumption. *RECON* can also be used to quickly analyze various optimization choices in terms of both energy and performance.

10. REFERENCES

[1] Calabash. `calaba.sh`.

[2] Chrome Developer Tools. `https://developers.google.com/web/tools/chrome-devtools/?hl=en`.

[3] 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`.

[4] Google Pagespeed Insights. `https://developers.google.com/speed/pagespeed/insights`.

[5] Google promises update as users suffer. `http://www.technobuffalo.com/2015/06/12/google-promises-chrome-updates-as-users-suffer/`.

[6] How loading time affects bottomline. `https://blog.kissmetrics.com/loading-time/`.

[7] 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`.

[8] Measuring the performance effects of mod_deflate. `http://www.webperformance.com/library/reports/moddeflate/`.

[9] Monsoon power monitor. `http://msoon.github.io/powermonitor/`.

[10] Page Speed Insight Rules. `https://developers.google.com/speed/docs/insights/rules?hl=en`.

[11] Safari: Longer battery life and faster performance. `http://www.apple.com/safari/`.

[12] SPDY. `https://www.chromium.org/spdy/spdy-whitepaper`.

[13] Yahoo's YUI Compressor. `http://yui.github.io/yuicompressor/`.

[14] V. Agababov, M. Buettner, V. Chudnovsky, M. Cogan, B. Greenstein, S. McDaniel, M. Piatek, C. Scott, M. Welsh, and B. Yin. Flywheel: Google's data compression proxy for the mobile web. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 367–380, Oakland, CA, May 2015. USENIX Association.

[15] Alexa Internet, Inc. The top 500 sites on the web.

http://www.alexa.com/topsites.

[16] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *Proc. ACM IMC*, November 2009.

[17] M. Butkiewicz, D. Wang, Z. Wu, H. V. Madhyastha, and V. Sekar. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, Oakland, CA, 2015. USENIX Association.

[18] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[19] X. Chen, N. Ding, A. Jindal, Y. C. Hu, M. Gupta, and R. Vannithamby. Smartphone energy drain in the wild: Analysis and implications. 2015.

[20] M. Dong and L. Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 335–348. ACM, 2011.

[21] D. Economou, R. S., K. C., and P. Ranganathan. Full-system power analysis and modeling for server environments. In *Proceedings of the Workshop on Modeling Benchmarking and Simulation*, MOBS '06, 2006.

[22] A. Gandhi, Y. Chen, D. Gmach, M. Arlitt, and M. Marwah. Minimizing Data Center SLA Violations and Power Consumption via Hybrid Resource Provisioning. In *Proceedings of the 2011 International Green Computing Conference*, IGCC '11, pages 49–56, Orlando, FL, USA, 2011.

[23] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal power allocation in server farms. In *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 157–168, Seattle, WA, USA, 2009.

[24] mod_pagespeed. http://www.modpagespeed.com/.

[25] J. Nejati and A. Balasubramanian. An in-depth study of mobile browser performance. In *Proceedings of the 25th International Conference on World Wide Web*, 2016.

[26] R. Netravali, J. Mickens, and H. Balakrishnan. 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, 2016.

[27] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*, pages 153–168. ACM, 2011.

[28] Shopzilla: faster page load time = 12% revenue increase. http://www.strangeloopnetworks.com/resources/infographics/web-performance-and-ecommerce/shopzilla-faster-pages-12-revenue-increase/.

[29] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Profiling resource usage for mobile applications: a cross-layer approach. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 321–334. ACM, 2011.

[30] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. 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, pages 321–334, New York, NY, USA, 2011. ACM.

[31] A. Shye, B. Scholbrock, and G. Memik. 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*, pages 168–178. ACM, 2009.

[32] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. Demystifying page load performance with wprof. In *NSDI*, pages 473–485, 2013.

[33] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. How speedy is spdy? In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 387–399, Berkeley, CA, USA, 2014. USENIX Association.

[34] F. Xu, Y. Liu, Q. Li, and Y. Zhang. V-edge: Fast self-constructive power modeling of smartphones based on battery voltage dynamics. In *NSDI*, volume 13, pages 43–56, 2013.

[35] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. 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, pages 36–36, Berkeley, CA, USA, 2012. USENIX Association.

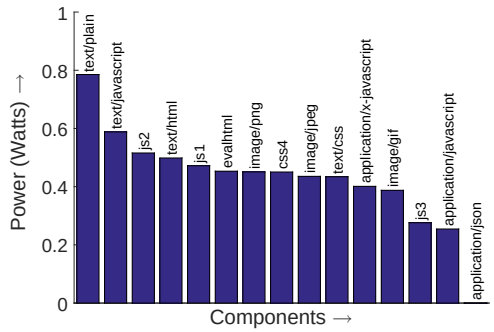
[36] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior

based power model generation for smartphones. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis*, pages 105–114. ACM, 2010.

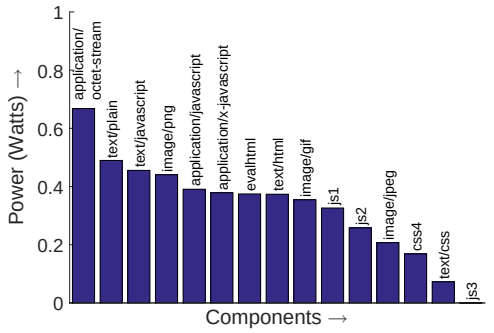
[37] Y. Zhang, X. Wang, X. Liu, Y. Liu, L. Zhuang, and F. Zhao. Towards Better CPU Power Management on Multicore Smartphones. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, HotPower '13, Farmington, PA, USA, 2013.

[38] Y. Zhu and V. J. Reddi. High-performance and energy-efficient mobile web browsing on big/little systems. *Proceedings of the 19th High Performance Computer Architecture*, 2013.

[39] Y. Zhu and V. J. Reddi. Webcore: Architectural support for mobile web browsing. In *International Symposium on Computer Architecture*, 2014.



(a) *www.alexa.com*



(b) *www.fico.com*

Figure 7: Normalized per-component power predictions based on our modeling.

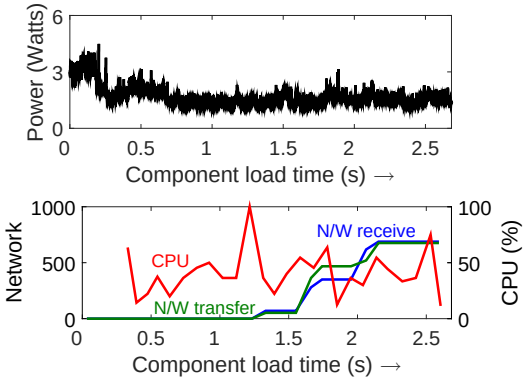


Figure 8: The figure depicts the variations in power consumption (top) and resource usage (bottom) for a segment of *fico.com* where only one long component (image/gif) was present.