2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)

Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps

Ivano Malavolta, Giuseppe Procaccianti, Paul Noorland, Petar Vukmirović
Computer Science Department, Vrije Universiteit Amsterdam, The Netherlands
{i.malavolta | g.procaccianti | p.vukmirovic}@vu.nl, p.d.noorland@student.vu.nl

***Abstract*—*Context.* Mobile web apps represent a large share of the Internet today. However, they still lag behind native apps in terms of user experience. Progressive Web Apps (PWAs) are a new technology introduced by Google that aims at bridging this gap, with a set of APIs known as service workers at its core.**
***Goal.* In this paper, we present an empirical study that evaluates the impact of service workers on the energy efficiency of PWAs, when operating in different network conditions on two different generations of mobile devices.**
***Method.* We designed an empirical experiment with two main factors: the use of service workers and the type of network available (2G or WiFi). We performed the experiment by running a total of 7 PWAs on two devices (an LG G2 and a Nexus 6P) that we evaluated as blocking factor. Our response variable is the energy consumption of the devices.**
***Results.* Our results show that service workers do not have a significant impact over the energy consumption of the two devices, regardless of the network conditions. Also, no interaction was detected between the two factors. However, some patterns in the data show different behaviors among PWAs.**
***Conclusions.* This paper represents a first empirical investigation on PWAs. Our results show that the PWA and service workers technology is promising in terms of energy efficiency.**

I. Introduction

Today the total activity on mobile devices like smartphones and tablets accounts for an incredible 67% of the time spent on digital media in the United States [1]. A considerable share of this amount of time is spent on the mobile web (i.e. accessing mobile-optimized websites via a browser on a mobile device), where a growth of 62% in terms of digital time spent has been observed in the last three years [1].

The mobile web is based on web apps conforming to standard languages like HTML5, CSS3, and JavaScript, which offer (among many) the advantage of full application portability across platforms (e.g. Android, Apple). Even if the browser is becoming more and more a fully-fledged software platform (e.g. the HTML5 standard provides APIs for geolocation, accessing the camera, microphone), as of today the mobile web struggles in providing a satisfactory experience to the user, mainly due to the strong dependence on network conditions, the lack of support for push notifications, and so on.

In this context, *Progressive Web Apps* (PWAs[1]) are a new technology advocated by Google as a way of overcoming the above mentioned limitations. The advantages of PWAs are clear when they are compared to classical web apps, for example: they can be launched from an icon in the home screen of the device (like native apps do), they instantly load regardless of the network availability, they support push notifications. At the core of this new technology is the concept of *service worker*, a set of APIs that allows developers to programmatically cache and preload assets and data, manage push notifications, and others. Technically, a service worker is a JavaScript module running in its own thread and providing generic entry points for event-driven background processing (e.g. reaction to the receiving of a push notification).

If on one side service workers have been advertised as performance boosters, network savers, and providers of better user experience, on the other side they are additional code to be downloaded, parsed, and run by the browser. Under this perspective, it is interesting to investigate the price that web developers and users may have to pay for those features in terms of other software quality aspects, e.g. energy efficiency (battery usage), performance, code complexity.

The **goal** of this paper is to assess the impact of service workers on the *energy efficiency* of PWAs. In this study we focus on the energy efficiency of PWAs because (i) energy is one of the most scarce resources in mobile devices [8] and (ii) there is no evidence about how service workers internally use battery-draining resources like the network. In order to achieve the aforementioned goal, we designed, executed, and reported an empirical study on the energy efficiency of 7 real PWAs running with and without service workers, under different network conditions (2G and WiFi), and on different mobile devices (i.e., low-end and high-end). *To the best of our knowledge, this paper is the first empirical investigation involving service workers and progressive web apps.*

The main **contributions** of this paper are:
• the results of an experiment on the energy efficiency of 7 real progressive web apps under different conditions;
• a discussion of the obtained results and their implications;
• the experiment replication package with all the measured PWAs, the analysis scripts, and raw data.

The **target audience** of this paper is composed of mobile web developers, browser vendors, and researchers. We support mobile web developers in knowing how the service workers technology can impact the energy efficiency of their PWAs; we aim to provide browser vendors with objective evidence about the impact that service workers may have on the energy efficiency of PWAs; finally, we aim at informing researchers on the state of the practice about energy efficiency of PWAs.

[1]http://developers.google.com/web

978-1-5386-2669-6/17 $31.00 © 2017 IEEE
DOI 10.1109/MOBILESoft.2017.7

The remainder of this paper is organized as follows. Section II provides basic concepts about PWAs and service workers, Section III presents the overall planning of our experiment and Section IV describes the setup of our infrastructure for executing the experiment. Section V presents and discusses its results. Threats to validity and related work are reported in Sections VI and VII, whereas in Section VIII we close the paper and discuss future work.

II. Background

A. Progressive Web Apps

A **mobile web app** is developed by using web technologies like HTML5, CSS3, and different flavors of JavaScript. Differently from hybrid mobile apps [7], [6], it is hosted on a remote servers, served via the HTTP standard protocol, and accessed by end users via a unique URL [5]. In other words, mobile web apps are mobile-optimized websites accessed via the browser apps installed on end users' mobile devices (e.g., Chrome, Firefox, Opera).

Introduced in 2015, **Progressive Web Apps** are special kinds of mobile web apps aiming at improving the mobile web experience from the following four perspectives:

• *Conversions*: as suggested by their name, PWAs are based on the progressive enhancement strategy. Specifically, a PWA firstly targets the lowest common denominator with respect to browsers functionality (e.g., static HTML contents only), then, depending on the specific browser it is running in, more advanced functionalities and refinements (e.g., animations, asynchronous network access) are progressively enacted.

• *Reliability*: PWAs can be loaded instantly, even with low or no network connectivity. This is achieved by using service workers as client-side proxies for programmatically caching and preloading assets and data, in principles eliminating the dependence on the network.

• *Performance*: in addition to caching and preloading, service workers can be also used for background processing in order to ensure an instant and reliable experience for users.

• *Engagement*: PWAs can be installed in the device and can be directly accessed from its home screen; also, PWAs support push notifications from the cloud, allowing developers to easily re-engage users; PWAs can provide an immersive experience to users by running in a full-screen mode.

A PWA is served from a remote server and is initially accessed as a standard web app via a browser. After some accesses, the user can decide to install the PWA in the device, thus *promoting* it to a top-level mobile app and gaining from the above described benefits. From a technical perspective, three conditions must hold for considering a mobile web app as a PWA, namely: (i) it is served over HTTPS for security reasons, (ii) it comes with a web app manifest[2] for declaring app metadata like its name, icons, base URL, and (iii) it uses at least a service worker (see next section).

[2]http://www.w3.org/TR/appmanifest/

B. Service Workers

Service workers are a set of APIs for allowing developers to programmatically cache and preload assets and data, managing push notifications, and others. Service workers are standardized by W3C[3] since 2009. According to the standard, a service worker is a special case of web worker[4], it is implemented in a dedicated JavaScript file and runs in a separate thread with respect to the main JavaScript thread. Intuitively, a service worker can be considered as a piece of JavaScript code running in parallel to the main page, providing persistent background processing, and interacting with the rest of the PWA in an event-driven fashion. A service worker can listen to events dispatched from the main page (e.g., the `fetch` event is raised when the main page makes network requests) or other sources (e.g., the `push` event is raised when a push notification is received from a remote server). Since providing all the details about service workers is out of the scope of this paper, in the following we describe the basics of service workers by reusing the source code of a PWA belonging to our dataset (see Listing 1).

```
 1 var filesToCache = [ /*...*/ ];
 2 // ...
 3 self.addEventListener('install', function(e) {
 4   e.waitUntil(
 5     caches.open(cacheName).then(function(cache) {
 6       return cache.addAll(filesToCache);
 7     })
 8   );
 9 });
10 // ...
11 self.addEventListener('fetch', function(e) {
12   var dataUrl, analyticsUrl;
13   // ...
14   if(e.request.url.indexOf(analyticsUrl) === 0) {
15     fetch(e.request);
16   } else {
17     e.respondWith(
18       caches.match(e.request).then(function(response) {
19         // ...
20         return response || fetch(e.request).then(function(
                 resp) {
21           // ...
22           caches.open(cacheName).then(function(cache) {
23             cache.put(e.request.url, response.clone());
24           });
25           return resp;
26         };
27     })));
28   }
29 });
```

Listing 1. Example of service worker (adapted from the Billings Gazette PWA)

The meaning of the listing is the following: when the service worker is installed a set of static resources are added to a local cache (lines 1-11) and when the main page makes any network request (e.g., an Ajax call), then the service worker intercepts it (line 13), leading to two cases: **a)** the requested resource is related to Google Analytics, so it is always directly performed (lines 14-16); **b)** otherwise (line 16), a cache of previous requests is queried (line 18) and the cached object is responded (if any, line 20), otherwise the network request is actually performed (line 20 again), its result is put into

[3]http://www.w3.org/TR/service-workers
[4]http://www.w3.org/TR/workers

the cache for future use (lines 22 and 23) and returned as a response to the main page (lines 20 and 25).

The conjecture of this paper is that the execution of a service worker may impact the energy consumption of the web page either positively (a service worker is additional software that is executed, thus it consumes additional resources) or negatively (caching may imply less access to the network).

III. Experiment Plan

The goal of our experiment is to assess the impact of service workers on the energy efficiency of PWAs. In order to achieve the above mentioned goal in an objective and replicable manner we planned our experiment by following well-known guidelines on empirical software engineering [15], [10]. In the following we report how we planned our experiment.

A. Research Questions

The research research questions of our study are:
RQ1 - *How does the use of service workers impact the energy efficiency of progressive web apps?* To answer this question the app will be served from a proxy with the service worker turned on and off. Statistical analysis will be used to determine whether there is a significant difference.
RQ2 - *How does the use of service workers impact the energy efficiency of progressive web apps under different network conditions?* Since PWAs can leverage service workers to be able to work offline in bad network conditions (or when no network is available) and cache app shells, it is expected that they will make less network requests, thus using less energy. Network conditions can be simulated using a proxy server, providing a method to test if this premise is true.

B. Subjects

As for the choice of subjects, that is PWAs to experiment with, first alexa.com's list of most popular websites was tried to be mined to identify the PWAs and choose a reasonable number of apps mined from the list. However, it turned out that very small percentage of the top 500 apps are actually PWAs and down sampling this small set for the ones that are network-intensive (and thus more informative for the experiment) would be an nonviable option. Instead, we selected 7 apps from a repository of PWAs (https://pwa.rocks). The selection has been performed in a pseudo-random manner because we wanted to avoid toy examples or PWAs with very specific characteristics (e.g., games), which could have been not representative of the population of data-driven mobile PWAs, the most recurrent application scenario for PWAs.

Table I presents the PWAs we selected for our experiment, they belong to different categories (e.g., news, shopping), are quite variegated in terms of both overall size and the size of their service workers (loc_{sw}, the last column of the table). The overall size of a PWA has been computed by automatically loading each PWA in the browser, locally downloading the whole front-end (i.e., all JavaScript scripts, images, HTML and CSS files, and other resources), and then calculating the size of the downloaded resources. The value for loc_{sw} has been computed by (i) identifying the JavaScript file of the used service workers via a suitably crafted JavaScript instruction injected into the browser during the loading of the PWA, (ii) beautifying the source code of the service worker (in some cases it was minified/obfuscated), and (iii) reporting the total number of lines of code after the beautification. Finally, it is interesting to note that in 5 cases over 7 the considered PWAs are real PWAs (e.g., AliExpress, the Washington Post), meaning that they have a real large user base, a complex business logic, and they have not been developed for simple experimentation or testing purposes; the other 2 PWAs (i.e., The Billings Gazette and Wiki offline) are experimental or unofficial apps.

ID	**Name**	**Category**	**Size**	loc_{sw}
aliexpress	AliExpress	Shopping	2.1Mb	69
babe	Babe News	News	1.2Mb	156
flipkart	Flipkart	Shopping	3.8Mb	907
gazette	The Billings Gazette	News	2.1Mb	60
googleevents	Google I/O 2016	Events	4.2Mb	358
washingtonpost	The Washington Post	News	4.0Mb	85
wikipedia	Wiki offline	Knowledge	800Kb	1009

TABLE I
Selected progressive web apps

C. Variables and Hypotheses

Three independent variables are used in the experiment: type of Android device, the type of the network connection and the service worker status (is it enabled or disabled).

The type of device has two treatments: low-end and high-end. A low-end device is considered to be a smartphone that is reasonably priced (around 200 euros) but of modest/legacy performance. A high-end device is described as the flagship by its manufacturer and has high performance. The low-end and high-end devices used for this experiment are respectively a LG G2 and a Huawei Nexus 6P. The specs for each device are reported in Table II.

Device Model	*Huawei Nexus 6P*	*LG G2*
CPU	Qualcomm Snapdragon 810	Qualcomm Snapdragon 800
Processor speed	2 GHz	2.3 GHz
Total cores	8	4
RAM	3 GB	2GB
Resolution	2560x1440	1920x1080
Screen diagonal	5.7"	5.2"
OS	Android 6.0	Android 4.4

TABLE II
Specifications of used Android devices

The network conditions variable has two treatments: 2G and WiFi. To simulate those network conditions the proxy serves the pages with reference latencies and bandwidths.These were determined based on a well-accepted network performance source [2] and by adding the latency of the experiment network connection (5 ms). The service worker status variable has two treatments as well: on and off. To turn the service worker off the proxy rewrites every response that is an HTML page with

JavaScript code that removes the `serviceWorker` object from `window.navigator`.

The dependent variable is the energy consumption of the device. Its values are collected by means of the Trepn Power Profiler[5], a software-based tool that estimates the power consumption of Android devices. This tool has been reported as sufficiently accurate with respect to hardware power measurement (e.g. the Monsoon power monitor), with an error margin of 99% [3]. For the dependent variable, we hypothesize an additive linear model of the form:

$$y_i = \mu + \tau_{i,j} + \nu_{i,j} + (\tau\nu)_{i,j} + \beta_i \qquad (1)$$

Where y_i represents our response variable (energy consumption), μ is the overall mean of the response, $\tau_{i,j}$ and $\nu_{i,j}$ are the effects of treatments τ and ν for observations i (respectively, service workers and network conditions), $(\tau\nu)_{i,j}$ is the effect of the interaction of the treatments and β_i is the blocking factor (the device model). Hence, the following two-tailed hypotheses are formulated:

H1: If the effect on energy consumption of the sites browsed with service worker off is labeled with τ_1 and the effect of the sites browsed with service worker on with τ_2, then the null and alternate hypothesis are defined as:
$H1_0 : \tau_1 = \tau_2$ $\qquad H1_1 : \tau_1 \neq \tau_2$

H2: If the effect on energy consumption of the sites browsed with network conditions 2G and WiFi respectively is labeled with ν_1 and ν_2 the null and alternate hypothesis can be defined as:
$H2_0 : \nu_1 = \nu_2$ $\qquad H2_1 : \nu_1 \neq \nu_2$

H3: The null hypothesis is that the service worker status and the network condition do not interact in affecting the energy consumption of the website. The interaction of the effect of the service worker status and the effect of the network condition on the energy consumption is labeled as $(\tau\nu)$. The hypothesis are:
$H3_0 : (\tau\nu)_{i,j} = 0 \ \forall \ i, j$
$H3_1 : \exists \ i, j \ \textit{such that} \ (\tau\nu)_{i,j} \neq 0$

D. Experiment Design

To get the data that will represent the observed situation accurately, a full 2x2x2 factorial design has been performed. In other words, all possible combinations of treatments have been assigned to each of our factors. During the experiment, we automatically performed a series of tap and/or gesture paths for each app in order to simulate the average user behavior on the subject and represent the app test scenario. Each series consists of roughly 10-15 commands. For each possible combination of treatments 8 measurements are conducted. The measurements executions are randomized.

E. Data Analysis

The data gathered by the experiment will be analyzed quantitatively. Beginning this analysis the first test indicates whether the gathered data can be assumed to be approximately normally distributed. For a first indication the mean and the median of the data are used. For a visual indication box plots are used and for a quantitative measurement the Shapiro-Wilk test is used.

It is expected that the data will be analyzed using a three-way ANOVA. However, this assumption will be tested in the result section before continuing with the three-way ANOVA. To verify this assumption the normality of the distribution of the residuals is tested with a Shapiro-Wilk test. The homoscedasticity is verified by a Levene's test.

Provided that the verification of the assumptions shows that an ANOVA test can be performed a three-way ANOVA test will be performed to analyse the data. This test has a full factorial design, thus measuring all possible combinations of variables. This will eventually be used to determine whether the null hypotheses can be rejected.

All the above mentioned test will be performed using a *95%* confidence interval, thus *p*-value ¡ 0.05.

F. Replicability of the Experiment

To allow easy replication and verification of our experiment, a complete replication package[6] is publicly available to interested researchers. Our replication package includes: a 38-pages report presenting the details of the experiment, the source code of the measurement scripts, raw data for each phase of the experiment, and the R scripts for exploring, summarizing, and analyzing measurement data.

IV. Experiment Execution

To perform our experiments, as discussed earlier, we leveraged several tools that help automation of task execution on the Android platform. In the following we will describe the tool stack we used to achieve the goal of deterministic scenario execution accompanied with energy consumption information gathering. The order of operations that are performed in one experiment execution is shown in Figure 1

To "freeze" the PWA contents and behavior we first record all of the traffic needed to perform one execution of the scenario for a particular PWA on a local desktop computer that acts as a proxy between the mobile device and the Internet (as seen in steps 1 and 2 of Figure 1). When all the contents served by the PWA have been recorded, local PC is ready to inject SW disabling script and/or set latency and bandwidth constraints to simulate different experiment factors. In other words, PC will act as a server from which mobile device will request the PWA. Content recording and response rewriting features described above are courtesy of the Fiddler Proxy software[7].

Then the Python script running inside Monkeyrunner[8] tool that orchestrates the scenario execution can be started, as depicted in step 3. This orchestration script will get as input the website to test, specific values for factors of the experiment

[5]https://developer.qualcomm.com/software/trepn-power-profiler

[6]http://s2group.cs.vu.nl/PWA2017ReplicationPackage

[7]http://www.telerik.com/fiddler

[8]https://developer.android.com/studio/test/monkeyrunner/index.html

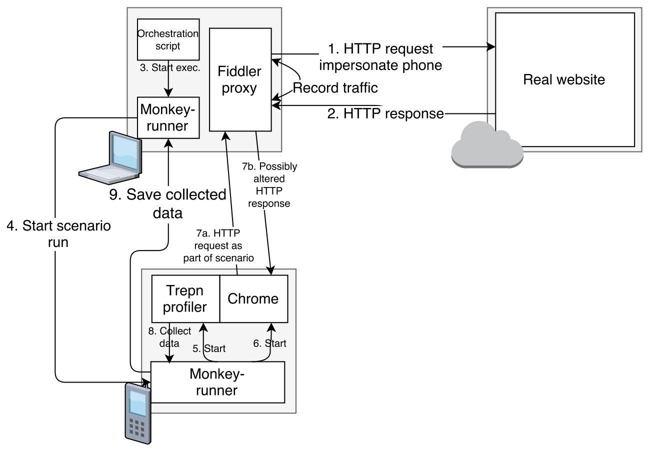


Fig. 1. Scenario execution. First record traffic, then replay that traffic on mobile device request, possibly altering the responses and then finally record energy consumption data.

and the text file, referred to as *scenario file*), that represents the series of screen taps and/or gestures that have to be performed as part of this experiment run and then wake up the device to start the execution of scenario file (step 4).

Further, as part of steps 5 and 6, the part of Monkeyrunner tool running on Android device will start the Trepn Profiler energy monitor and *clean* instance of Chrome browser (i.e. all persistent website data such as cookies and caches will be deleted), instrumenting it to load the desired website and start performing screen taps and gestures.

Those taps and gestures will incur network requests that fetch the resources needed for the PWA being executed. However, since the phone is connected to our preset proxy, PWAs will not be fetched directly, but will be served from our local machine. With proxy up and running, we are able to intercept the request and alter it, by for example sending the reply with specific bandwidth and/or latency or injecting JavaScript code to remove the `navigator.serviceWorker` object from `window` top-level object (essentially removing the entry point into the ServiceWorker API).

What was previously described is a series of HTTP requests and replies that is shown as step 7(a for request and b for reply), which will be stopped when the end of scenario text file is reached. When this event happens, Chrome will be closed and Trepn Profiler will save the data, which can later on be pushed to the local PC, which is represented by steps 8 and 9 in Figure 1 that are the last steps performed in experiment execution.

V. Results

This section presents the descriptive statistics for our dataset (Section V-A) and the results of our hypotheses testing (Section V-B). Moreover, in Sections V-C and V-D we answer the research questions of our experiment by elaborating and interpreting the results of our statistical analyses. Finally, Section V-E discusses how each specific PWA performed in terms of energy consumption across the whole experiment.

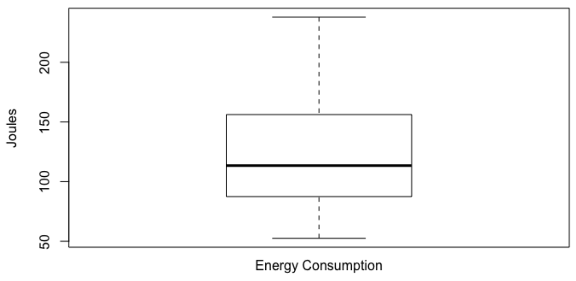


Fig. 2. Measured energy consumption values

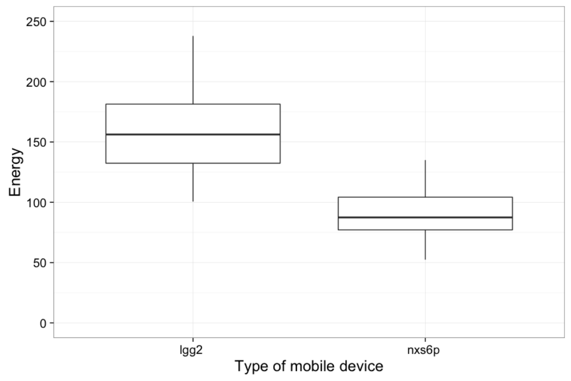


Fig. 3. Measured energy consumption values per mobile device (in Joules)

A. Descriptive Statistics

The energy consumption of all PWAs (two versions for each PWA, with service workers on and off) of our dataset is summarized in Table III.

	Energy Consumption					
Phone	Min.	Max.	Median	Mean	SD	CV
Both	52.44	237.90	113.41	124.33	45.32	0.36
LG G2	100.61	237.90	156.21	157.38	37.78	0.24
Nexus	52.44	134.97	87.44	91.28	22.14	0.24

TABLE III
Descriptive statistics for the energy consumption values (in Joules) – SD= standard deviation, CV = coefficient of variation)

As shown in the boxplot in Figure 2, the dataset appears quite positively skewed, with the mean higher than the median. Indeed, the data scores negatively for normality (*W = 0.95141, p-value = 0.02459*). This is due to the high difference in the energy consumption values between the two mobile devices, as can be also observed in the boxplot in Figure 3. For this reason, for the rest of our analysis we will use the type of mobile device as a blocking factor.

B. Hypothesis Testing

As anticipated in Section III-E, we performed a 3-way ANOVA to test our hypotheses. We verified the assumptions of ANOVA as follows:

- *errors are statistically independent*: this assumption is satisfied by the randomization of our measurement runs.
- *residuals are normally distributed*: we verified this by performing a Shapiro-Wilk normality test on the residuals (*W = 0.97009, p-value = 0.1775*).
- *Homoscedasticity*: we verified homoscedasticity by means of Levene's test (*F=0.0966, p-value=0.9616*).

Then, we proceeded with testing our hypothesized model. The raw analysis data from the three-way ANOVA test is shown in table IV. None of the *p*-values is below our significance threshold, except for the blocking factor (i.e. the type of mobile device, see below). This means that we are unable to detect a significant difference between the energy consumption of a PWA using service workers and that of the same PWA not using service workers. Furthermore, no significant interaction between treatments is detected by the ANOVA. Thus, **we cannot reject the null hypotheses** $H1_0, H2_0$ **and** $H3_0$ and we cannot claim that service workers influence the energy consumption of a PWA running on a mobile device.

	Df	Sum Sq	Mean Sq	F value	p-val
SW	1	98.15	98.15	0.11	0.75
Network	1	3164.28	3164.28	3.41	0.07
Phone	1	67974.31	67974.31	73.33	¡0.05
SW:Network	1	66.93	66.93	0.07	0.79
Residuals	52	48201.45	926.95		

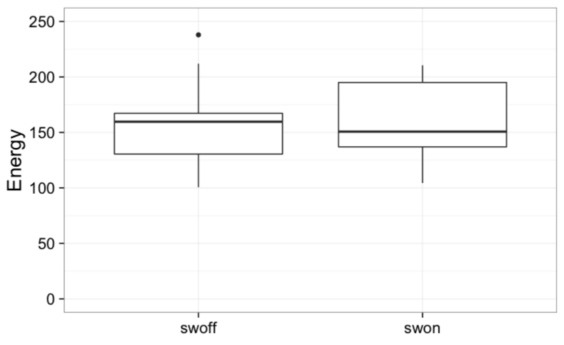
TABLE IV
THREE-WAY ANOVA RAW RESULTS

The ANOVA did show also that the type of mobile device has a very strong impact on energy consumption. This further validates our design choice of considering the type of mobile device as a blocking factor for our experiment. We expected such outcome, as different screen sizes, chip sets and other hardware specifics of mobile devices will very likely lead to different energy consumption values. Additionally, newer mobile devices tend to be more energy efficient and the Huawei Nexus 6P is several generations newer than LG G2. This observation was more than expected, nevertheless it is still interesting because it confirms that the experiment has been conducted in a valid manner.

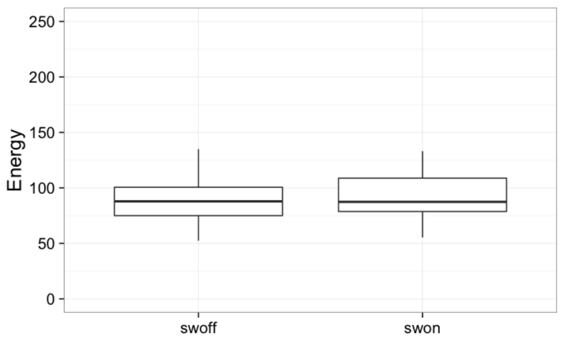
Factor	Diff.	Lower Bound	Upper Bound	p-val. (adj.)
SW on vs. SW off	2.56	-12.88	18.00	0.74
WIFI vs. 2G	-14.52	-29.97	-0.919	0.065
Nexus vs. LG	-67.47	-82.94	-51.99	¡0.05

TABLE V
RESULTS OF TUKEY'S HSD TEST WITH 95% CONFIDENCE INTERVALS.

In order to further validate our analysis, we performed post-hoc tests (namely, Tukey's Honest Significance Difference test [13]) on our ANOVA. Results of the test are reported in Table V. For brevity reasons, we omit the results related to the interactions between factors, also considering they do not achieve statistical significance.



(a) Service workers vs. energy on LG G2



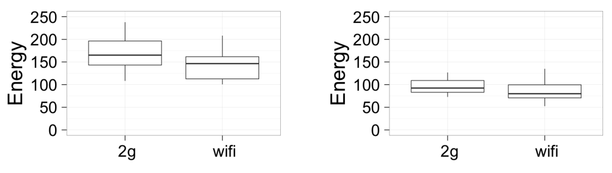
(b) Service workers vs. energy on Nexus 6P

Fig. 4. Energy consumption of the PWAs with service workers on and off

C. Does the use of service workers impact the energy efficiency of PWAs? (RQ1)

In the previous section we already assessed that, in overall, the use of service workers does not have a statistically significant impact on the energy efficiency of PWAs. However, as shown in Figure 4, we can observe that the median of the consumed energy of PWAs with service workers on is always lower than the median of the consumed energy of PWAs with service workers off.

Interestingly, if we consider how PWAs consume energy across mobile devices (see Table VI), this difference is quite evident in low-end mobile devices, i.e. the LG G2, where it accounts to 159.6 - 150.7 = 8.9 Joules), whereas it gets extremely minimal when considering high-end mobile devices, i.e. the Nexus 6P, where it accounts to 87.86 - 87.44 = 0.42 Joules only (well within the measurement error margin). This result is quite encouraging since it may be an indication of the improvements that hardware manufacturers and platform vendors (i.e., Google for Android in this case) are making with respect to the energy efficiency of the software running on their products.



(a) Network on LG G2 (b) Network on Nexus 6P

Fig. 5. Energy consumption of the PWAs under different network conditions

D. Does the use of service workers impact the energy efficiency of PWAs under different network conditions? (RQ2)

As shown in Figure 5 and 6, our PWAs performed in a consistent manner with respect to the networking conditions, independently of whether service workers were active or not. As detailed in Table VII, both low- and high-end devices consumed less energy when running PWAs under a WiFi network, with energy consumption with a median of 146.5 and 79.97 Joules for the LG G2 and Nexus 6P mobile devices, respectively. Also, since WiFi radios provide higher bandwidth, this means that in principles the browser running a PWA downloads the needed assets and data (e.g., the initial `index.html` file, images, CSS stylesheets, JavaScript files) in less time, thus consuming less energy.

We also investigate on how the presence of service workers may impact the energy consumption of the PWAs according to all the possible combinations of (i) type of mobile device and (ii) network conditions. As shown in Figure 6, the use of service workers has a marginal impact on the energy efficiency of PWAs when considering the same mobile device and network conditions.

This means that if we fix the mobile device and network conditions, the gains provided by service workers in terms

SW	Min.	Max.	Median	Mean	SD	CV
LG G2						
sw off	100.6	237.9	159.6	156.1	39.72	0.25
sw on	104.4	210.5	150.7	158.7	37.18	0.23
Nexus 6P						
sw off	52.44	135.0	87.86	89.82	21.83	0.25
sw on	55.30	133.10	87.44	108.8	23.17	0.23

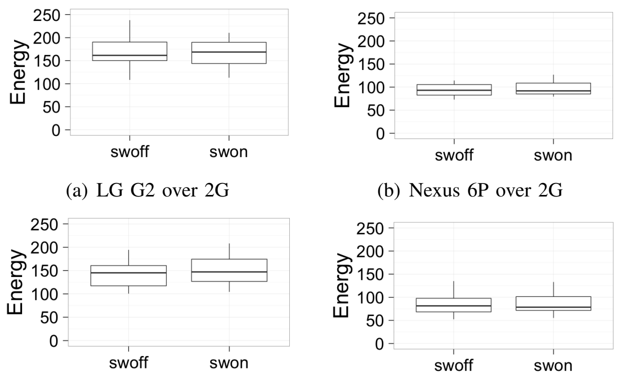
Energy Consumption

TABLE VI
STATISTICS ON THE ENERGY CONSUMPTION OF PWAS WITH SERVICES WORKERS ON OR OFF ACROSS DEVICES

SW	Min.	Max.	Median	Mean	SD	CV
LG G2						
2G	108.3	237.9	165.1	167.7	37.98	0.22
WiFi	100.6	208.2	146.5	147.0	35.92	0.24
Nexus 6P						
2G	72.89	126.8	92.42	95.85	17.00	0.17
WiFi	52.44	135.0	79.97	86.7	26.15	0.30

Energy Consumption

TABLE VII
STATISTICS ON THE ENERGY CONSUMPTION OF PWAS UNDER DIFFERENT NETWORK CONDITIONS ACROSS DEVICES



(a) LG G2 over 2G (b) Nexus 6P over 2G

(c) LG G2 over WiFi (d) Nexus 6P over WiFi

Fig. 6. Energy consumption of the PWAs (in Joule) on different mobile devices and under different network conditions

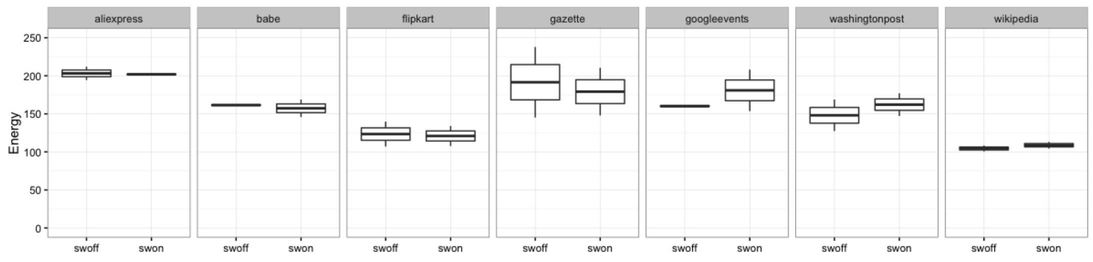
of user experience seem to have a very low impact on the energy consumption of the mobile device running them. This result can be seen as an encouraging indication that service workers is a viable technology, both for mobile web developers and users, providing features that are strongly needed by users (e.g., support of push notifications, better performance of the web) without relevant drawbacks in terms of energy consumption, which is one of the main concerns the mobile development arena.

E. PWA-specific Results

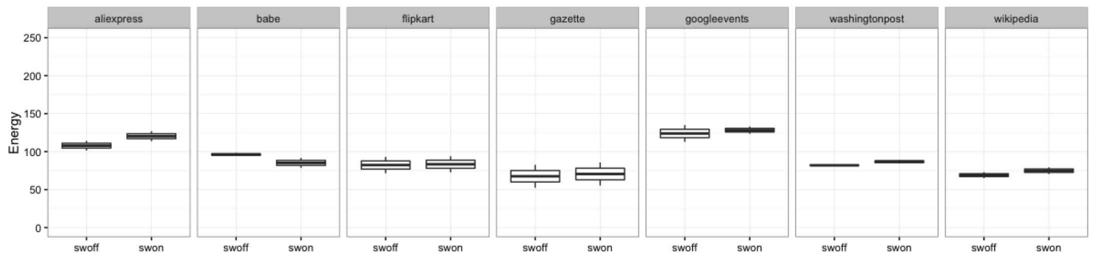
In this section we zoom into the specificities of each single PWA of our experiment as a way to better understand and provide additional insights about the previously discussed results. More specifically we (i) analysed the energy consumption of each single PWA across the various factor-treatment combinations of our experiment and (ii) reviewed the source code of the service worker of each PWA in order to explain its specific performance during the experiment.

Energy consumption of each PWA. Figure 7 presents the energy consumption values of each PWA across the considered mobile devices and network conditions. We can observe that different PWAs have different energy consumption values, even under the same contextual conditions (i.e., when running on the same mobile device and with the same available network); for example, the Ali Express PWA consumes far more energy than Flipkart when running on the LG G2 device (first and third facets of Figure 7(a)). This result is expected since each PWA differs from the others in terms of size, business logic, interaction with browser resources, interaction with the network.

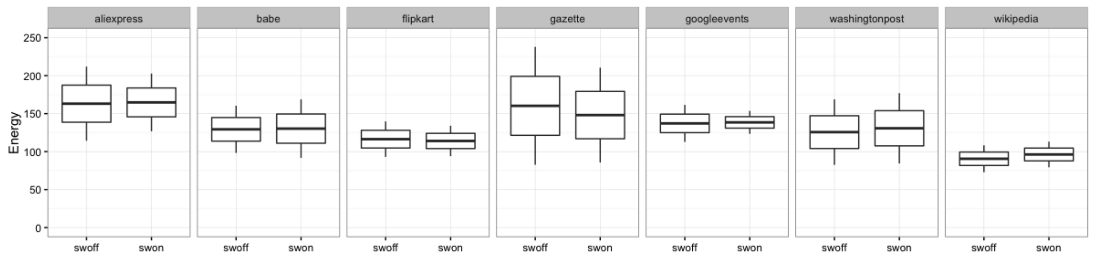
It is also interesting to note that there seem to be no evident pattern about the energy efficiency of a PWA under different contextual conditions; for example, The Billings Gazette with service worker consumes more when running on the Nexus 6P device or over a WiFi network, wheres it consumes less when running on the LG G2 device or over a 2G network. This result may depend on the specific implementations of service workers, they are discussed later in this section.



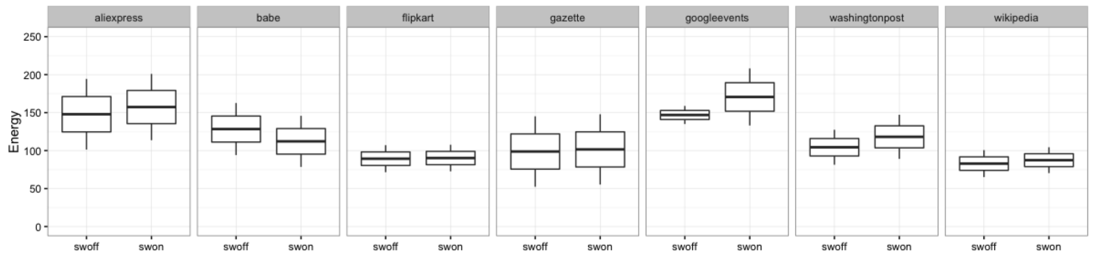
(a) Energy consumption values with service workers on and off on the LG G2



(b) Energy consumption values with service workers on and off on the Nexus 6P



(c) Energy consumption values with service workers on and off under a 2G network



(d) Energy consumption values with service workers on and off under a WiFi network

Fig. 7. Energy consumption values of each PWA under different contextual conditions

Trends analysis. In order to better understand how service workers may impact the energy consumption of the considered PWAs, we categorize each PWA according to whether it consumes more in combination with (i) the presence of service workers in the PWA and (ii) the context in which it is running, i.e., mobile device and network conditions. The results of such categorization is a set of 28 combinations (7 PWAs * 2 devices * 2 network conditions) and are shown in Table VIII.

We can observe that in 20 cases including a service worker in a PWA had a negative impact on the energy efficiency of the app (i.e., the PWA consumes more when a service worker is present). In the 8 remaining cases, the energy consumption is lower in PWAs without service workers with respect to PWAs with service workers. This result suggests that mobile web developers should be careful when including a service worker in their PWA for enhancing the user experience of their PWAs. In many cases the effect on energy consumption is minimal, so it may be worth trading off energy consumption for a better user experience. Nevertheless it is important for mobile web developers to know that in some cases when adopting service workers their PWA may potentially consume more energy. Under this perspective it will be useful for mobile web developers to have a recommendation tool for providing suggestions about this trade off based on the actual energy consumption of the service workers of their PWAs; this is part of our future work.

Manual reviews of service workers code. The lack of evident trends across all PWAs let us conjecture that a potential confounding factor would have been the specific implementation of the service worker of each PWA. In order to give an answer to this conjecture we manually analyzed the source code of the service worker of each PWA with the goal of finding recurrent technical solutions or patterns that may have an impact on its energy consumption.

Table IX shows the development practices we considered when analyzing the source code of the service worker of each PWA. Firstly, we considered the *events* listened by the service worker (e.g., push notification, data fetching). As expected, the majority of service workers are managing the installation and activation of the service worker itself and the fetch event; this is because those are the standard events to be managed for managing the *caching* of remote data and static assets, which is done in almost all the cases. We also categorized service workers according to whether their code has been *obfuscated or minified* because in principles obfuscated/minified JavaScript code can be more complex and its parsing and execution may be more resources demanding; we observed a certain balance between obfuscated/minified and clean code across PWAs. Finally, we report the cyclomatic complexity and cyclomatic complexity density of the source code of the service worker, as identified by the complexity-report open-source tool[9].

In Table IX we highlight in bold the PWAs in which the presence of a service worker consistently impacts their energy efficiency in a negative manner (i.e., they consume more energy with service workers on all devices and all network conditions). Those PWAs are Google Events, The Washington Post, and Wikipedia and they are good candidates for identifying the development practices that may have an impact on the energy consumption of service workers. For all development practices, the differences in the values of those PWAs are not significant and do not follow any evident trend. Nevertheless, we can see a mild prevalence of both (i) the service workers listening to a higher number of events and (ii) those that are obfuscated/minified; those two practices may be seen as two potential causes of higher energy consumption. However, this is only a conjecture as we cannot rigorously conclude that those two practices are linked to higher energy consumption because of the low statistical significance we could achieve with only 7 PWAs (we plan to better investigate on these initial results in our future extension of this work).

[9]http://www.npmjs.com/package/complexity-report

	LG G2		Nexus 6P		2G		WiFi	
PWA	swoff	swon	swoff	swon	swoff	swon	swoff	swon
aliexpress	✓	-	-	✓⃝	-	✓	-	✓
babe	✓	-	✓⃝	-	-	✓	✓⃝	-
flipkart	✓	-	-	✓	✓	-	-	✓
gazette	✓⃝	-	-	✓	✓⃝	-	-	✓
g.events	-	✓⃝	-	✓	-	✓	-	✓⃝
w. post	-	✓⃝	-	✓⃝	-	✓⃝	-	✓⃝
wikipedia	-	✓	-	✓	-	✓⃝	-	✓

TABLE VIII
CASES IN WHICH ENERGY CONSUMPTION IS HIGHER FOR EACH PWA (CIRCLED CHECKMARKS ARE USED WHEN THE DIFFERENCE IS ABOVE 10 JOULES)

PWA	**Listened events**	**Caching**	**Obf./ minif.**	**Compl-exity**	**Compl. density**
aliexpress	P,N	✓	✓	12	21.4
babe	I,A,F,P,N	✓		9	10.5
flipkart	I,A,F	✓	✓	131	19.0
gazette	I,A,F	✓		5	16.1
g. events	I,A,F,M	✓		16	15.8
w. post	I,A,P,N		✓	7	10.6
wikipedia	I,A,F,S,M,N	✓	✓	194	24.7

TABLE IX
DEVELOPMENT PRACTICES WITHIN SERVICE WORKERS. THE VALUES OF LISTENED EVENTS ARE: I = "INSTALL", A = "ACTIVATE", F = "FETCH", P = "PUSH", N = "NOTIFICATIONCLICK", S = "SYNC", M = "MESSAGE".

VI. THREATS TO VALIDITY

In what follows, we are going to categorize threats to validity based on categorization given in [15], and describe how each of those is applicable to our experiment is manifested.

External validity. Since PWAs have recently been introduced as a novel approach to developing mobile web apps, there are still platforms that do not support them. The most notable example is Apple iOS, a major mobile platform that we could not include in our research, which prevented us from generalizing the results of our experiment to Apple devices. Moreover, due to resources constraints, we had to restrict the set of mobile devices used in the experiment to two specific devices (i.e., LG G2 and the Nexus 6P), potentially impacting the generalization of our results to the whole set of Android devices. Nevertheless, we mitigated this potential bias by choosing two mobile devices that cover both low-end and high-end categories of hardware specs and different versions of Android.

Moreover, we executed realistic usage scenarios of the PWAs, which means manual work (e.g., for recording the scenarios) for each considered PWA. As a consequence, we had to select a subset of all existing PWAs for performing the experiment in reasonable time. We mitigated this potential threat to validity by selecting PWAs in a pseudo-random manner, across different categories and with different size; also, the considered PWAs are designed and developed by independent development teams.

Internal validity. Since it is arguably infeasible to reliably perform real-life tests on all of the PWAs,we had to make a selection. Given the fact that the scope of our experiment is mobile web development, we decided to focus only on categories of network intensive PWAs such as shopping or news, and disregard the ones that use specific phone hardware (e.g. QR code scanners, voice recorders) since arguably it would be difficult to implement those using traditional web technologies and service workers play a relatively secondary role in them. This could disable us from generalizing our conclusions to the full population of existing PWAs, but we decided to make such a decision since it gives more insights to comparison between traditional web apps and PWAs. Related to this point is the fact that PWAs have only recently been introduced and that programming techniques used in development in this first batch of PWAs might not utilize the potential of PWAs to the fullest.

Conclusion validity. The setup of this experiment may be subject to random events that may have disturbed our measurements (e.g., background tasks, inner Android OS scheduling). To mitigate this validity threat we performed the experiment with the same fixed set of treatments for the 8 runs of each PWA. To lower the chances of random events, we kill all of the background applications, disable all applications that are crash-prone and make sure that even those applications that are left running since they cannot be closed (e.g. core Android functionalities) are unable to use the network by blocking their requests on Fiddler.

Construct validity. Our experiment has been performed in a tightly controlled lab environment, which is a necessary precaution we had to take to minimize randomness, but this lab environment has the consequence of applying the treatments in an artificial way which might not accurately represent the in-the-wild treatment effects. Firstly, we applied the 2G network conditions treatment by accurately simulating the latency and bandwidth of average 2G connection. However, on mobile devices different chips (different radios) take care of WiFi and cellular connections. This might have had a small effect on the energy consumption of a device.Another treatment that may have an effect on PWAs is disabling the service worker. In this case we injected some JavaScript code in the main page of each PWA to remove the service worker capability from the browser. Any injected JavaScript code can have influence on the web site behaviour, but we mitigate this by close inspection and testing of the code that we injected and we kept it as minimal as possible. Lastly, to minimize randomness in terms of latency and bandwidth that is inherent to the Internet, we served the pages from a local proxy in which we could tightly control latency and bandwidth.

VII. Related Work

As introduced in Section I, to the best of our knowledge this is the first empirical study investigating service workers and progressive web apps.

The state-of-the-Art on the energy efficiency of mobile web apps is also scarce, but it provides some interesting insights. For example, Thiagarajan et al. [11] analyze the energy consumption of 26 web apps. Measurements were performed using a hardware multimeter and a patched version of the Android browser. The differences between this study and ours are many; among them, we can highlight that (i) the two studies focus on different subjects (standard web apps VS PWAs), and (ii) during our experiment we measured the energy consumption of PWAs across their main functionalities over time using execution scenarios, rather than focussing only on the initial load of the web app.

Other researchers focused instead on eliciting *best-practices* for energy efficiency. For example, Linares-Vásquez et al. [4] aim at identifying whether some API calls consume more energy than others, and if sequences of API calls (patterns) repeat themselves frequently, causing anomalies in energy consumption. The study analyzed the execution traces of 55 Android applications, looking for the most energy-greedy Android API calls. Results show that APIs related to *GUI & Image Manipulation* and *Database* are the most energy-consuming. Tonini et al. [12] evaluate the energy impact on two best-practices proposed by Google to improve performance in Android applications. The practices can be summarized as: "use appropriate *for* syntax" and "avoid getters/setters". The two best practices were applied in different implementations of the same mobile application and tested upon three different smartphones.

An interesting line of research has been pursued on empirical studies targeting the *performance* of mobile web apps. For example, Vesuna et al. [14] designed and executed a study on how caching may impact the mobile browser performance over a dataset of 400 web pages. Each run the web page was only loaded in the browser, no execution of complex usage scenarios was involved. Differently, Nejati and Balasubramanian [9] measured performance bottlenecks in mobile and desktop browsers. The focus of this work is on the lower levels of the browser engine and it is based on a dedicated in-browser performance profiler for Android Chromium. Both studies have some similar aspects for what concerns the experiment design, but they all differ in the goal of the experiment (i.e., mobile web performance VS energy efficiency assessment).

VIII. Conclusions and Future Work

Up to today there was no evidence about how service workers internally use battery-draining resources when running on a mobile device. In this paper we presented the design, execution, and the results of an empirical study on the energy efficiency of service workers in the context of progressive web apps. Our results show that service workers do not have a significant impact over the energy consumption of the two devices, regardless of the network conditions. Nevertheless, we observed different energy consumption among PWAs; we inspected the source code of the service worker of each PWA to investigate the potential causes of this phenomenon.

As future work we are planning to (i) push further with the level of automation of our measurement infrastructure so that we will be able to perform a large scale study involving hundreds of PWAs; (ii) deepen the analysis of the source code of service workers in order to find additional development practices which may impact the energy consumption of the PWAs hosting them; (iii) investigate on other interesting dimensions in addition to energy efficiency, including the load times of PWAs when leveraging the caching capabilities of service workers.

References

[1] Adam Lella, Andrew Lipsman. The 2016 U.S. Mobile App Report, 2016. comsCore white paper.

[2] I. Grigorik. *High Performance Browser Networking*. O'Reilly Media, 2013.

[3] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao, and S. Tarkoma. Modeling, Profiling, and Debugging the Energy Consumption of Mobile Devices. *ACM Comput. Surv.*, 48(3):39:1–39:40, Dec. 2015.

[4] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk. Mining energy-greedy api usage patterns in android apps: An empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 2–11, New York, NY, USA, 2014. ACM.

[5] I. Malavolta. Beyond Native Apps: Web Technologies to the Rescue! In *Proceedings of the 1st International Workshop on Mobile Development*, Mobile! 2016, pages 1–2, New York, NY, USA, 2016. ACM.

[6] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni. Hybrid mobile apps in the google play store: An exploratory investigation. In *Mobile Software Engineering and Systems (MOBILESoft), 2015 2nd ACM International Conference on*, pages 56–59, May 2015.

[7] I. Malavolta, S. Ruberto, V. Terragni, and T. Soru. End Users Perception of Hybrid Mobile Apps in the Google Play Store. In *Mobile Services (MS), 2015 IEEE International Conference on*, pages 25–32. Institute of Electrical and Electronics Engineers (IEEE), June 2015.

[8] M. Nagappan and E. Shihab. Future trends in software engineering research for mobile apps. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 5, pages 21–32. IEEE, 2016.

[9] J. Nejati and A. Balasubramanian. An in-depth study of mobile browser performance. In *Proceedings of the 25th International Conference on World Wide Web*, pages 1305–1315. International World Wide Web Conferences Steering Committee, 2016.

[10] F. Shull, J. Singer, and D. I. Sjøberg. *Guide to advanced empirical software engineering*, volume 93. Springer, 2008.

[11] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh. Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, pages 41–50. ACM, 2012.

[12] A. R. Tonini, L. M. Fischer, J. C. B. de Mattos, and L. B. de Brisolara. Analysis and evaluation of the android best practices impact on the efficiency of mobile applications. In *2013 III Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 157–158. IEEE Computer Society, 1 Nov. 2013.

[13] J. W. Tukey. Comparing individual means in the analysis of variance. *Biometrics*, 5(2):99–114, June 1949.

[14] J. Vesuna, C. Scott, M. Buettner, M. Piatek, A. Krishnamurthy, and S. Shenker. Caching Doesn't Improve Mobile Web Performance (Much). In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, 2016.

[15] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2012.