Hindawi
Scientific Programming
Volume 2019, Article ID 8235458, 19 pages
https://doi.org/10.1155/2019/8235458

Research Article

An Energy Efficiency Study of Web-Based Communication in Android Phones

Inmaculada Ayala, Mercedes Amor, and Lidia Fuentes

Lenguajes y Ciencias de la Computación, Universidad de Málaga, Andalucía Tech, Málaga 29071, Spain

Correspondence should be addressed to Inmaculada Ayala; ayala@lcc.uma.es

Received 15 November 2018; Revised 24 January 2019; Accepted 25 February 2019; Published 4 April 2019

Academic Editor: Michele Risi

Currently, mobile devices are the most popular pervasive computing devices, and they are becoming the primary way for accessing Internet. Battery is a critical resource in such personal computing gadgets, network communications being one of the primary energy consuming activities in any mobile app. Indeed, as web-based communication is the most used explicitly or implicitly by mobile devices, HTTP-based traffic is the most power demanding one. So, mobile web developers should be aware of how much energy demands the different web-based communication alternatives. The goal of this paper is to measure and compare the energy consumption of three asynchronous HTTP-based methods in mobile devices in different browsers. Our experiments focus on three HTTP-based asynchronous communication models that allow a web server to push data to a client browser through a HTTP/1.1 interaction: Polling, Long Polling, and WebSockets. The resulted measurements are then analysed to get more accurate understanding of the impact of the selected method, and the mobile browser, in the energy consumption of the asynchronous HTTP-based communication. The utility of these experiments is to show developers what are the factors and settings that mostly influence the energy consumption when different web-based asynchronous communication methods are used, helping them to choose the most beneficial solution if possible. With this information, mobile web developers should be able to reduce the power consumption of the front-end of web applications for mobile devices, just selecting and configuring the best asynchronous method or mobile browser, improving the performance of HTTP-based communication in terms of energy demand.

1. Introduction

The number of mobile phone users in the world is expected to pass the five billion mark by 2019. In 2016, an estimated 62.9% of the population worldwide already owned a mobile phone. The mobile phone penetration is forecasted to continue to grow, rounding up to 67% by 2019 [1]. The use of mobile phones varies through the day, smartphone use being overwhelmingly popular for some activities such as social media, messaging, and browsing Internet to catch up with news and updated information. And, it is precisely frequent Internet access which drains the battery. Most of smartphone users (in 2018, Android copes more than 75% of market share) find their Android battery is regularly emptied. Although it is possible to squeeze more life out of the battery by optimizing some phone settings, some battery drain may be due to badly designed apps that are constantly receiving notifications with updated content without the user intervention. These hidden communications are often the culprits of battery draining: apps that frequently get online for updates and notifications in the background. Another issue is that while other phone hardware such as screens and motherboards have been constantly improved to be better and more powerful, battery technology has not seen a similar advancement. So, even brand-new phones with larger charge capacities may not last any longer than their predecessors. And, while it is always recommended downloading software updates, older phones that have been upgraded to the very latest operative system (OS) version may also experience drastic battery drain as a result. Users can find several tips that can improve Android phone battery life depending on the Android version. From the system

perspective, network communication (WiFi, 3G, 4G, more recently 5G, or Bluetooth) is one of the primary energy consuming operations in mobile devices. On average, network communications can consume over 40% or more of the total non-idle state energy of an app or a mobile device [2–4]. So, one of the most popular tips is turning off notifications. However, this option is not very useful in apps that intend to get real-time updates like email or social networks.

Given that smartphones are now becoming the most popular way to browse Internet, mobile browsers can be considered one of the heaviest drains on the battery. Mobile web usage has recently overtaken desktop Internet access for the first time [5]. Even when users are not actively using the browser, it continues consuming in the background. Among all kinds of network activities, those related with the Hypertext Transfer Protocol (HTTP, defined in [6, 7]) are the most energy consuming, representing almost 80% of all network-related energy consumption [2, 8].

Different statistics illustrate what it means in figures: in April 2018, mobile devices excluding tablets accounted for 51.2% of web page views worldwide. In 2021, mobile data traffic worldwide is expected to reach 49 exabytes per month at a compound annual growth rate of 47%. According to Statcounter, most of the Internet traffic of mobile devices is HTTP-based (the data excludes the use of apps such as Facebook and WhatsApp, which account for a significant proportion of mobile Internet usage). Therefore, reducing browser energy consumption due to data transfer can have a significant impact on the overall energy consumption of the device and improve the overall user experience by increasing the underlying device's battery life. In order to contribute to increase battery life when using browsers, it is necessary to identify which are the variable factors that influence the energy consumption of browsers and can be tuned and configured by the Software developer. In addition to data size, another issue that must be considered is data rate (i.e., the speed at which data are updated or the number of times data is sent in a particular period). This factor depends on (i) the type of the data source, which determines when data are generated (e.g., bursty, interactive, real-timed, or at a fixed rate); and (ii) the mechanism or technology used to push the data.

At the beginning, HTTP-based interactions occurred when users retrieve or update web content proactively. Every time a user clicked on a link, new content was retrieved and shown in the browser. Nowadays, however, most of the web content is automatically retrieved without user intervention (e.g., instant messaging, web mail, and real-time content). This means that whenever new content is available, the server is able to push that information out to the browser, allowing users to maintain timely updates from sites inadvertently. Server push technology can be tackled by the application protocol or by the application itself. Recognizing the necessity of server push, HTTP/2, which was released in 2017, supports server push as part of its specification. Prior to that new version, HTTP/1.x is purely a synchronous request/response application protocol. So, in order to achieve server pushing using HTTP/1.x, developers are advocated to use solutions at the application level if they want an asynchronous interaction. However, HTTP/2 is used just by 32.9% of all the websites in 2019. So, the browser always has to initiate a request to get web content. Nowadays, Web developers can adopt different technologies to achieve server pushing in HTTP/1.x. Most of the web applications require the server to send data to the client asynchronously as the state of a dynamic system changes and without the need for the user to interact with the browser interface.

Although HTTP is synchronous, Web software developers have different approaches to achieve asynchronous interaction. The first solution adopted, which is still widely used, consisted in emulating asynchronous communication over a synchronous communication channel using a continuous polling technique. Web Applications that need to listen for server originated data can use a continuous client-originated Polling. A popular variation of Polling is the Long Polling, where the HTTP response is delayed for a specific time or until data are available. More recently, the markup language HTML introduces in its latest version 5, the WebSocket protocol [9] to address such asynchronous interaction, which is not supported by HTTP/1. WebSocket allows a bidirectional, full-duplex, persistent socket connection between the client and server. Based on the bidirectional connection, the server can actively send (i.e., push) data to the browser. A WebSocket connection is built on top of TCP and has only small overhead in comparison to HTTP [10]. A newer solution for server push at the application level is the W3C Push API [11], which lets a web page or app send notifications at the system level even if the app is idle or in the background. All these solutions have in common that their use lean on JavaScript APIs.

Another distinct feature that may affect performance and then energy consumption is the browser selected to show the content. Although all the browsers provide the same functionality, internally, they may differ in the browser engine, which is the core software component of every major web browser. Because the Web platform is an open standard, there are multiple browser engine implementations, which try to differ in their performance and resource management when rendering the content. When accessing the Web from a mobile device, it is especially important to pay attention to the consumption and performance of the used mobile browser. So, mobile web developers also should bear in mind the limited power available on mobile devices and may have to design accordingly their applications. In smartphones, battery power capabilities are not keeping up with the advances in other device resources (e.g., processing and memory), which are considered firstly to improve user experience and especially in view of the growth in usage of mobile browsers to access Internet. The deficiency in battery power and the need for optimized energy consumption provide motivation for web software developer to use energy efficient techniques in order to manage the power consumption in async HTTP-based communications, which may vary from a mobile browser to another.

Our goal is to measure and compare how the use of three different web applications using different asynchronous

HTTP-based communication techniques (Polling, Long Polling, and WebSocket) and two different mobile browsers (Chrome and Firefox) behave regarding energy consumption. The scope of our study lies at the application level and using HTTP/1.X. Although HTTP/2 improves the performance of websites and web applications, this version is still used just by 28.3% of all the websites, so its benefits are not significant yet. In addition, the use of asynchronous solution at the application level is used by the 90% of web developments, because they are well-known and widespread solutions. This work is part of our ongoing study on the energy consumption of asynchronous communication mechanisms. This work extends our previous contribution [12] with a new experimental set and a new kind of experiment (i.e., longer interaction experiments). Additionally, we make the web browser part of our study and analyse whether there is a statistically significant difference on the power consumption of the mechanisms.

By knowing the impact in battery consumption of the selected method for asynchronous communication, the mobile browser, and other factors (pushed data size and data pushing rate), mobile web developers could reduce the power consumption of the front-end of web applications on mobile devices selecting the most appropriate Web browser and asynchronous mechanism to send asynchronously pushed data. Although energy is a critical resource for smartphones, developers lack of quantitative and objective information about the behaviour of apps with respect to energy consumption. This paper analyses and compares the energy consumption of an Android device while using three typical asynchronous HTTP-based communication techniques: Polling, Long Polling, and WebSockets; and two Web browsers: Chrome and Firefox, both for Android. The study is conducted according to the goal-question-metrics approach with the goal "Analyse asynchronous HTTP-based communications for Android, from the point of view of web software developers" and two research questions that aim to find out which are the factors influencing energy consumption in the different experimental scenarios, and if the differences found are statistically significant. For developers, the results would show if it is worth learning how to use a technique or use a concrete mechanism in terms of battery consumption. The major beneficiaries of this energy saving would be users who would benefit of being updated while their batteries do not drain off.

This paper is organized as follows: The second section provides an overview of the HTTP protocol and asynchronous communication mechanisms, the third section introduces the energy profile tool used in this study, the fourth section explains the results of this study using the goal-question-metrics approach, the fifth section illustrates the threats to the validity of the presented results, the sixth section overviews work related to our study, and finally, the seventh section concludes the paper and introduces future work.

2. Summary on HTTP Protocol

In this section, we describe the HTTP protocol and the mechanism or technology used by Web application to push data at the application level. We pay special attention to the relationship between the size of the data and the rate of update, JavaScript libraries that support them, and the reasons for that selection.

The Hypertext Transfer Protocol [6] is a request/response protocol. A client (i.e., a browser) establishes TCP connections to a Web server with the purpose of sending HTTP requests. Each HTTP request only allows requesting a resource. Once a HTTP server accepts a request, it sends back the requested resource in the entity-body content of an HTTP response. HTTP communication takes place over TCP/IP connections. The number of TCP connections used depends on the HTTP version. The use of inline images and other associated data often require a client to make multiple requests of the same server in a short amount of time. In HTTP/1.0, most browser implementations use a different TCP connection for each HTTP request/response exchange, thus increasing the load on HTTP servers and causing congestion on the Internet. However, at the same time, it provides a lower load and rendering time since all the resources are downloaded almost simultaneously in connections that run in parallel. In the next version of this protocol, HTTP/1.1, the same persistent TCP connection may be used for one or more request/response exchanges, although connections may be closed for a variety of reasons. Persistent HTTP connections have several advantages [6], which can influence positively to power consumption, by opening and closing fewer TCP connections, CPU time, and memory are saved. On the contrary, the use of TCP persistent connection can be affected by the head-of-line blocking problem, which limits performance and affects end-user experience. In mobile devices, maintaining the connection open also contributes to save energy. It seems that maintaining a connection open requires less energy than a re-establishing or opening a new connection [13]. In addition, latency and power consumption on subsequent requests is reduced since there is no time or energy spent in TCP's connection opening handshake and slow start. Although TCP persistent connections are the default behaviour of any HTTP connection, many browsers use nonpersistent TCP connection for improving end-user experience, despite the overload and the resources required.

The opening and controlling of TCP connections, the instantiation and sending of HTTP requests through TCP sockets, and the reception and processing of HTTP responses is the responsibility of the browser, so the choice of one browser or another can affect battery consumption. The process of sending and receiving HTTP messages can consume a large amount of energy due to the underlying operations that such request entails. HTTP is part of a multilayer network protocol stack, which includes TCP, IP, and various hardware level protocols. These operations, which involve many system operations (such as calculating checksums, copying data, referencing data buffers, and processing protocol data units of different layers), are part of the operating system. In mobile devices, there is extra power consumption, since each HTTP API request has tail energy, which is independent from the size of the request. Tail energy occurs when the system keeps the network radio in

the active state after an HTTP request is finished. This is typically done to attempt to reduce the high energy overhead of starting and shutting down the wireless radio. Although seemingly small, the overhead of an HTTP request can have a significant impact on its energy efficiency.

2.1. Polling. In the standard HTTP model, a server cannot initiate a connection with a client nor send an unrequested HTTP response to the client; thus, the server cannot push asynchronously events or data to browsers. Therefore, in order to receive asynchronously new data as soon as possible, the browser polls the server periodically for new content by sending an HTTP request. With the traditional or "short" polling technique, a client sends regular requests to the server and each request attempts to "pull" any available events or data. If there are no events or data available, the server returns an empty response and the client waits for some time before sending another poll request (i.e., polling interval). The polling frequency depends on the latency that the client can tolerate in retrieving updated information from the server. Polling implementation on the client-side relies on features included by default in browsers, such as JavaScript, rather than on nondefault plugins.

Regarding energy consumption, continual polling can consume significant bandwidth and energy by forcing a request/response round trip when no data are available. It can also be inefficient because it reduces the responsiveness of the application since new data are queued until the server receives the next poll request from the client [14].

This results in high energy consumption for even a single HTTP request. HTTP also poses a significant overhead of extra data and messages to be sent when it makes a request. Therefore, not only the size of the HTTP data sent affects the transmission cost but also the set of headers, which can range from 200 B to 2 KB worth of headers.

Given a compliant server and a compliant browser, the client just has to instantiate an XMLHttpRequest object. All modern browsers support the XMLHttpRequest object, a JavaScript object which allows accessing the DOM. The XMLHttpRequest object can be used to exchange data with a web server without the user intervention (also known as AJAX). This means that it is possible to update parts of a web page, without reloading the whole page. The client instantiates an XMLHttpRequest and sends it to the server. After that, the client reads the HTTP response, which contains updated content if data of the server have new data available from the last HTTP request. Otherwise, the response does not contain any payload in the body. The basic communication cycle of an application using "HTTP polling" is as follows (Figure 1(a)):

(1) The client makes an initial request and then waits for a response.

(2) If there are data available, the server sends an HTTP response with the data. Otherwise, it sends an empty response (no data in the body).

(3) The client typically sends a new poll request after a pause (poll interval) to allow an acceptable latency period.

However, Polling, despite its simplicity, has a priori drawback: the resources consumed by the client (browser processing and network) strongly depend on the frequency data are updated and the polling interval. If the data availability is low (e.g., of the order of seconds), then a high polling frequency can cause an unacceptable burden on the browser, the network, or both.

Figure 2 shows a comparison of the simplified JavaScript code required for Polling (top), Long Polling (middle), and WebSocket (bottom). For simple Polling, the client has a function (*pollServer* in Figure 2) that at every *interval* sends a request (*.get* function in JQuery) and immediately, receives and process the response. The client must establish, usually *hardcoded*, the polling interval, which determines when the server is polled by the front-end of the application (indicated by the clock in the upper code in Figure 2).

2.2. Long Polling. In order to improve Polling shortcomings, there are several server-side programming mechanisms often grouped under the common label "Comet" [15]. One of the most common server push mechanisms is HTTP "Long Polling", in which the server attempts to "hold open" (not immediately reply to) each HTTP request, responding (i.e., sending the HTTP response) only when there are events to deliver. Then, there is always a pending request to which the server can reply sending data as it is available. This solution enables a web server to send data to clients when new data are available, and the browser does not have to be aware of polling periodically, nor adjusting the polling interval. Long Polling can deliver updates to browsers in a timelier manner while avoiding the latency experienced by client applications due to the poll interval or the empty responses, thereby minimizing the latency in message delivery and the use of processing/network resources.

Given a compliant server (supporting async servlets of Servlet 3.0) and a compliant browser, to send the request (code shadowed in blue in the middle of Figure 2), the client has just to instantiate an XMLHttpRequest object and send it to the server (methods *open* and *send*). After that, the client waits until the Http response is ready. Once the server has new data available, it completes the HTTP interaction sending the corresponding Http response. The basic communication cycle of an application using "HTTP Long Polling" is as follows (Figure 2(b)):

(1) The client makes an initial request and then waits for a response

(2) The server defers its response until an update is available or until a particular status or timeout has occurred (if this occurs, the HTTP response has no data)

(3) When an update is available, the server sends a complete response to the client

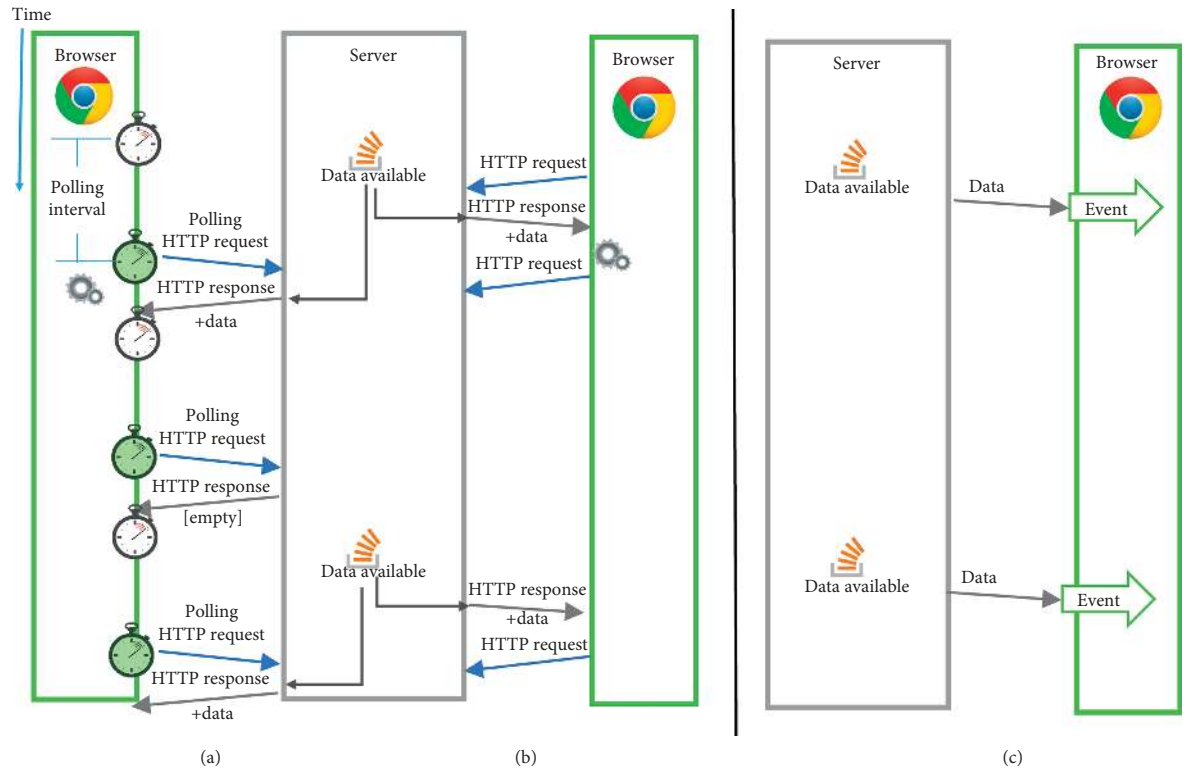

Figure 1: Communication scheme in asynchronous HTTP communication using Polling (a), Long Polling (b), and WebSocket (c).

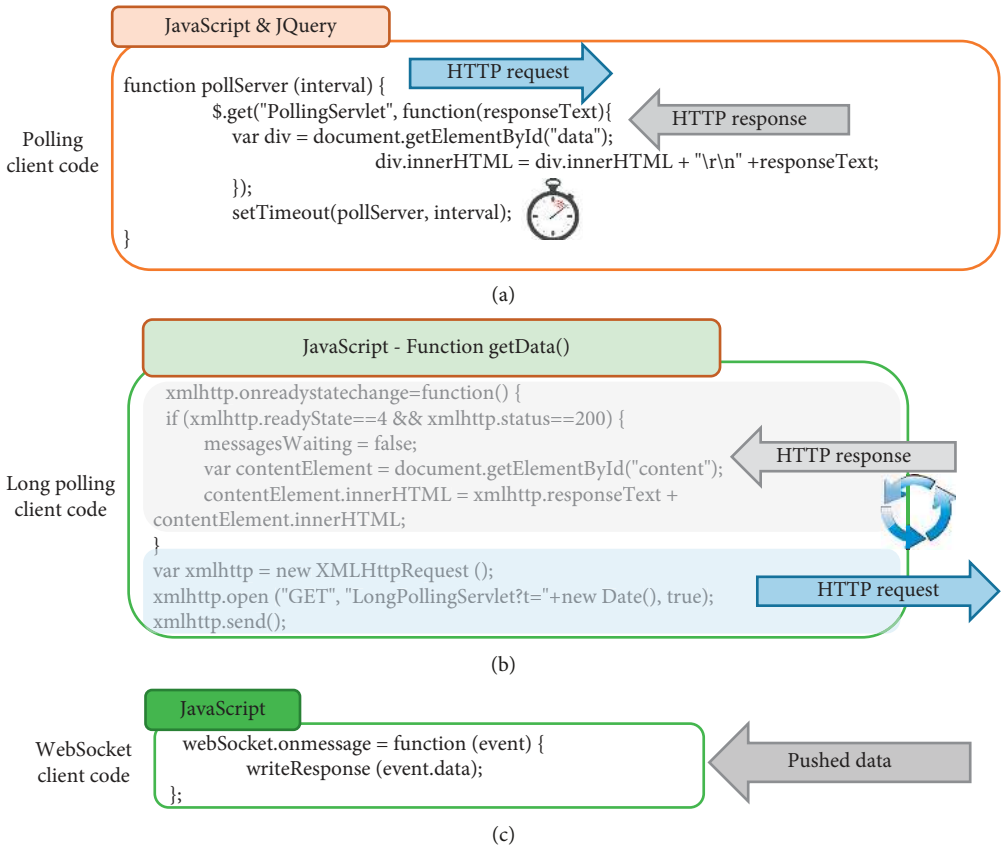

Figure 2: JavaScript client code for Polling (a), Long Polling (b), and WebSocket (c).

(4) The client typically sends a new Long Polling request, either immediately upon receiving a response or after a pause to allow an acceptable latency period

The HTTP Long Polling mechanism can be applied to either persistent or nonpersistent HTTP connections. However, the use of persistent HTTP connections will avoid the additional overhead of establishing a new TCP/IP connection [16] for every Long Polling request.

2.3. WebSocket Protocol. WebSocket is a protocol defined in [9], which allows to use the TCP connection established between a browser and a Web server as a full-duplex and persistent socket-like channel for exchanging non-HTTP messages with only a small overhead in comparison to HTTP [10]. Using this connection, the back-end in the Web server is able to actively and asynchronously push data to the client whenever it is available. Prior to data/message exchange through the connection, the WebSocket protocol requires an initial handshake to establish the WebSocket connection. The message exchange is executed in form of frames, which contain either text or binary data [9]. Different subprotocols for WebSocket are defined like SOAP Over WebSocket Protocol Binding (MSSWSB), WebSocket Application Messaging Protocol (WAMP), or Simple Text Oriented Messaging Protocol (STOMP).

Given a WebSocket compliant server and a compliant browser, the client, using a JavaScript API, just has to instantiate a WebSocket object and starts listening to server-pushed data events (method *onmessage*) (see example code in the bottom of Figure 2). The basic communication cycle of a client application using WebSockets is as follows (Figure 1(c)):

(1) The client makes an initial handshake and then waits for a response

(2) When an update is available, the server sends the data to the client through the WebSocket

(3) Incoming data are available to the browser through an asynchronous event (method *onmessage*)

3. Energy Profiling Tool

To measure energy consumption of software applications running on mobile phones, there are multiple tools based on both hardware and software [17–24]. Although hardware measurement offers higher precision, we cannot make use of it, since it estimates the energy consumed by the whole machine, and our study investigates the consumption at the application level. Additionally, selecting and configuring a hardware equipment may represent a complex task [25], which can introduce additional bias. Some solutions require special equipment and are difficult to apply with the available documentation [18, 19]. Other solutions offer applications that can be installed in devices easily [20, 21] but they are restricted to specific devices and architectures. PETrA, which is an acronym for a Power Estimation Tool for Android apps, is able to measure power consumption at a method-level granularity, but it requires some app preprocessing before measuring energy consumption. Energy measurements provided by PETRA have been compared against the actual energy consumption computed by a hardware toolkit, obtaining similar results [22]. In our case, taking into account the available devices for testing, we find two software solutions that fitted our requirements: the GreenOracle [26, 27] and the Trepn Profiler [28]. GreenOracle is an accurate software energy model that is now improved by GreenScaler [24]. The latter improves training process by automating the generation process of the software tests. A summary and a brief comparison of the existing Android testing tools for model building test generation can be found in [24]. An overview of the empirical studies on energy consumption in mobile devices is presented in the Related Work section.

In a previous contribution [12], we have used both GreenOracle and Trepn Profiler tools to profile the power consumption of different Android devices. As the results of the experiments were remarkably similar for all the devices and profiling tools, in this contribution, we have opted to use just one of the tools. Specifically, we have opted for Green Oracle, widely used in other experimental researches [29, 30], which supports to experiment with the device connected to a power source, allowing to speed up the experimentation process.

Regarding the equipment, tests have been executed on a Samsung Galaxy Nexus (https://en.wikipedia.org/wiki/Galaxy_Nexus), which is the device used to develop the GreenMiner. The back-end of the web sites is deployed in a Glassfish web server in an Ubuntu 14. The PC is an Intel Core i7 3.50 GHz with 16 gigabytes of RAM memory. The web server has a 1 gigabit Ethernet Internet connection. On the other hand, the smartphone uses a WiFi connection in a different local network to the web server.

3.1. GreenOracle. GreenOracle is an accurate energy model generated using a big-data approach and hundreds of energy measurements obtained by the GreenMiner [19]. According to authors, GreenOracle has an upper error-bound close to 10% which is similar to other methods like the ones discussed in [18, 31]. Additionally, it is relatively easy to apply as it is based on information that can be extracted from the operative system of the device (i.e., Android), and authors have well documented the workflow to apply the energy model. Finally, GreenMiner, the source of energy measurement data, has been applied in several works [29, 30, 32]. This supports the accuracy of the data used to generate the energy model.

The framework used to apply the GreenOracle model is composed of different applications and tools (Figure 3). We have developed several scripts for Android ADB [33] that gather the information of the operative system and interact in an automated way with the mobile phone screen avoiding introducing additional bias. This information is processed by a Java application, which finally applies the model. The energy model requires information of the CPU consumption, interruptions, major faults, and context switches that are extracted from the operative system files *"/proc/stat"* and

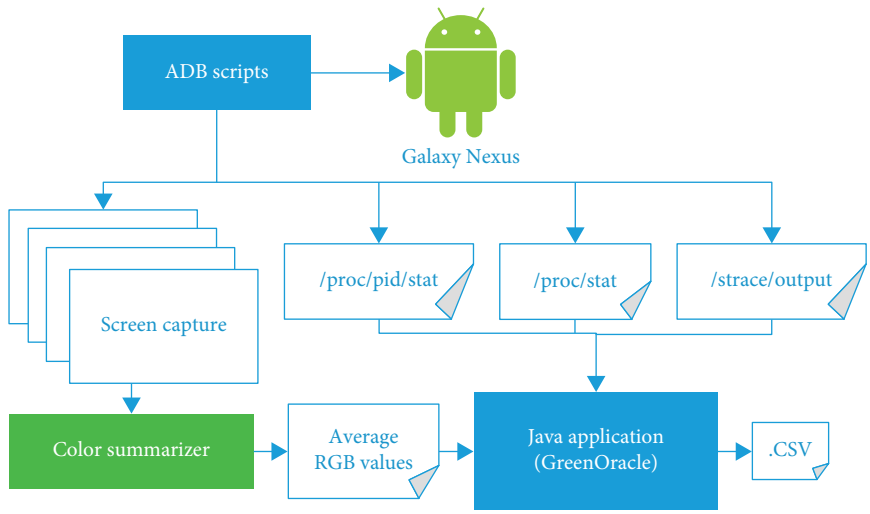

Figure 3: Framework to apply the GreenOracle Energy Model.

"/proc/pid/stat". Other necessary information is the system calls performed as a result of the execution of the application, which is extracted using the "strace" command, and average RGB values of colour in the screen during the execution. This information is generated capturing periodically the screen during the work of the application (by means of an ADB script) and processing captures using the utility Image Color Summarizer [34]. All this information is processed by the Java Application, which contains the GreenOracle model, and generates csv files that can be processed by Microsoft Excel or other similar tools. The components of this framework are available at https://goo.gl/sZsMZ3.

4. Empirical Study

In this section, we present the experimental planning and the energy profile results obtained by the experiments. The measurement unit used is joule (J). One joule is the equivalent of one watt of power radiated or dissipated per second.

4.1. Objectives and Research Questions. The methodology of this study is defined according to the goal-question-metrics approach [35] as follows: "Analyse asynchronous HTTP-based communications for Android, from the point of view of web software developers". To achieve this goal, we set the following research questions (RQs):

RQ1. What is the factor (polling/pushing interval time or data size) that most influences energy consumption in asynchronous communication? This question explores the influence of data size, data latency; and polling/pushing time interval, and their relationship in the power consumption of the three methods considered. Also, the data availability frequency in relation with polling interval is also considered in the simple polling method, because in polling mechanism, if client polling interval is minor than data availability frequency, it can cause the reception of empty HTTP responses. Other factors, such as the use of two mobile web browsers with different browser engines, will also be considered.

RQ2. Which asynchronous communication method is the most efficient in terms of energy consumption? Are statistically significant the differences in terms of energy consumption of the three communication mechanisms? RQ2 compares and overviews whether the power consumption of each asynchronous method is significantly different, or there exists different scenarios or browsers where a method is better than others. The answer to this question is the key to make the best choice in terms of energy consumption and application requirements. Additionally, we analyse whether the quantitative saving in power consumption of one communication mechanism of a mobile browser is significant with regard to others. This is especially interesting for developers that consider to change the implementation of a Web application to get a mobile front-end more respectful with device energy.

4.2. Data Collection. In order to measure and compare the consumption of the three asynchronous communication mechanisms in two different mobile browsers, we have developed three simple web sites compatible with mobile browsers (http://caosd.lcc.uma.es:9000/). Each of them allows testing one of the three communication mechanisms considered. The user interface of the front-end allows configuring the parameters for each experiment: the polling/push frequencies and size of the data received from the server (Figure 4). All the web sites allow to configure the server to generate data randomly (both size and time). The server generates events randomly according two normal (or Gaussian) distributions.

The three web sites present the same simple user interface. Our tests focus on energy consumption at data reception, so we have measured the energy consumption of the mobile browser when it is receiving data using the three communication mechanisms in different scenarios with different durations, data sizes, and frequencies. We considered experiments of two durations: 1 minute and 5 minutes. We have selected 1 minute because, according to

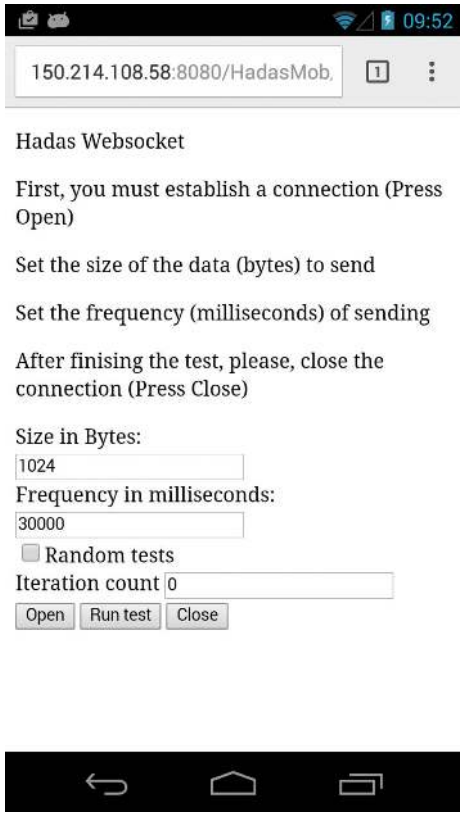

Figure 4: Minimal UI to test WebSocket reception in Google.

different works [36, 37], the interaction of people with mobile phones is for short periods of time of around 1 minute. Additionally, we consider longer experiments of five minutes. Interactions with tablets or specific Web apps, such as gambling, news consult, or access to social networks, are longer than 1 minute and usually of this duration [38]. Most notifications and updates occur at intervals of more than 30 seconds. With regard to the data size, we consider five different message sizes (140 bytes, 280 bytes, 560 bytes, 1120 bytes, and 2240 bytes) that comes from a usual twitter message (i.e., 140 bytes) to messages that requires fragmentation at the IP level (i.e., 2240 bytes). The data update periods selected for the tests are (in milliseconds) 1000, 5000, 10000, 30000, and 50000. These update periods represent the polling frequency for simple polling and interval time between data availability at the server for Long Polling and WebSockets. Each test has been repeated 20 times.

Figures 5–7 show the results of these experiments. The results are conclusive enough with a standard deviation lower than 15 joules for experiments of 1 minute and lower than 25 joules for experiments of 5 minutes. According to our experiments, there are different patterns of energy consumption depending on the web browser and the duration of the experiments. In general, Chrome has a lower energy consumption than Firefox. However, it seems more sensible to lower polling periods. As expected, WebSocket has the lowest energy consumption for all experiments. Other interesting issue is that it seems that data size has little impact on energy consumption.

We have performed experiments to measure energy consumption of Long Polling and WebSocket in more realistic scenarios. Specifically, we have measured energy of both communication mechanisms in scenarios of random data size available at random time. In order to generate randomly data in the back-end, we have used the java class *java.util.Random* and its method *nextGaussian()*. This method is used to get the pseudorandom numbers that are normally distributed (i.e., they follow a Gaussian distribution) with mean 0 and standard deviation 1. We have used equation (1) with the parameters *dev* and *mean* to generate more realistic data patterns. The Gaussian distribution to model the event generation time has a *mean* of 5000 and standard deviation (*dev*) of 1000 (this means that 70 percent of values will fall between 5000 ± 1000, in other words between 4000 and 6000 milliseconds; the 95 percent of values will fall between 3000 and 7000 milliseconds). The standard deviation of the Gaussian distribution used to generate data size has a *mean* of 500 and standard deviation (*dev*) of 100 (the 70 percent of values will fall between 500 ± 100, in other words between 400 and 600 bytes).

$$r.nextGaussian() * dev + mean. \quad (1)$$

Results of random experiments are depicted in Table 1. We can see a great similarity between Long Polling and WebSocket but with slightly lower consumption for WebSocket. Additionally, it is interesting to note that, for random experiments, the energy consumption of Firefox is higher than the consumption of Chrome.

4.3. Answers to Research Questions. RQ1. What is the factor (polling/pushing interval time or data size) that has more influence to energy consumption in asynchronous communication? In order to answer this question, we apply correlation coefficients [39]. A correlation coefficient is a numerical measure of some type of correlation between two variables. Depending on the type of the relationship between the variables (i.e., linear and monotonic), we can use different correlation coefficients like Pearson or Spearman. Therefore, before applying the corresponding coefficient, we will study the relationship between the variables. In order to accomplish this task, we will create scatterplots matrices of the data. In addition, scatterplots can detect whether there is a relationship between the variables.

The scatterplots of our experiments (Figures 8–13) show three important results. Firstly, there is not a relationship between the variables bytes and energy consumption in any of the scenarios studied. Secondly, there is a relationship between frequency and energy consumption and this relationship is monotonic. This is true for all scenarios except the one of Polling using Firefox in 1-minute experiments (Figure 8). In this case, the relationship between the variables is linear. Finally, these scatterplots confirm the independence of our input variables (i.e., bytes and period).

In order to assess the strength of the relationship between frequency and energy consumption, we run Spearman's rank correlation coefficient. This coefficient is a statistical measure of the strength of a monotonic

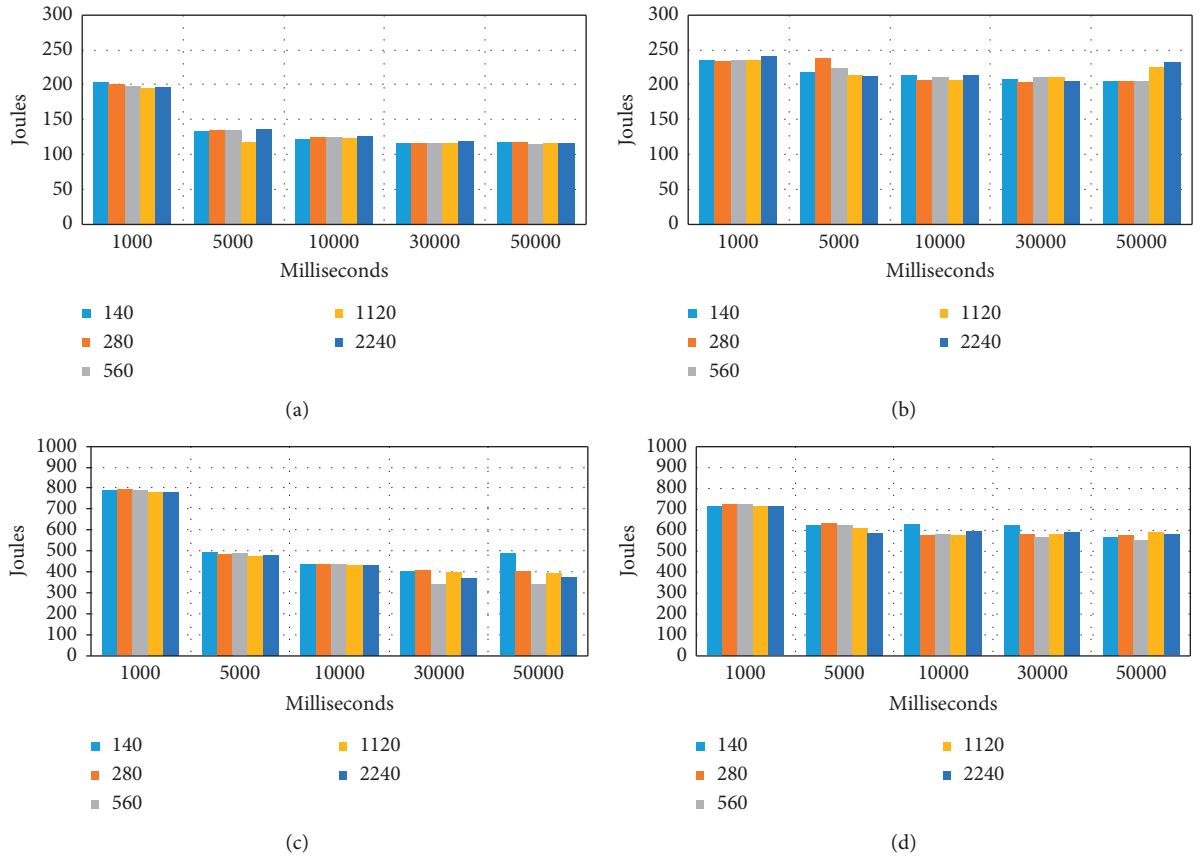

Figure 5: Energy consumption of Polling for 1-minute experiments in Chrome (a) and Firefox (b) and 5-minute experiments in Chrome (c) and Firefox (d).

relationship between paired data. Therefore, it is adequate for variables that are monotonically related. As all the linear relationships are monotonic, we can apply this coefficient to the Polling Firefox 1-minute scenario (Figure 8). Spearman's rank correlation coefficient is denoted by r_s and is constrained by −1 and 1. The closer r_s is to ± 1, the stronger the monotonic relationship is. We have performed this analysis using the statistical tool IBM SPSS.

Results of the application of the Spearman correlation (Table 2) confirm what was evident from the scatterplots; there is a strong negative relationship between energy consumption and period in all scenarios. Thus, short polling periods are associated with high energy consumptions. We have tested the significance of the results using the p value, which is lower than 0.001 in all scenarios, so we can confirm the results obtained.

In conclusion, the polling/pushed interval is the factor that has more influence on energy consumption. We can confirm this fact in bar charts of Figures 5–7. In Chrome, there is a considerable difference between the energy consumed for periods of 1 second and the rest of polling periods regardless of the duration of the experiments. On the one hand, for experiments of 1 minute (Figures 5(a), 5(b)–7(a), 7(b)), the energy consumption for polling periods of 1 second is around 150–200 joules, while for the rest of the polling periods is around 100 joules, with no significance of the data transmitted. The situation for experiments of 5 minutes is similar (Figures 5(c), 5(d)–7(c), 7(d)), for polling periods of 1 second the energy consumption is around 500–800 joules, and for the rest of the polling periods analysed is around 400 joules. These differences are slighter for WebSocket. On the other hand, Firefox seems less sensible to variations in the polling period.

Taking into account the information provided by the bar charts and Spearman's rank coefficient, we state that updating frequency is the factor that has more impact on energy consumption. So, mechanisms that just provoke some interaction when it is necessary will consume less. However, this is not the case of the polling mechanism that will consume energy, regardless of the availability of the data. In view of these data, the answer to question RQ1 is that frequently updating data affects negatively energy consumption, while the size of the data pushed barely influences energy consumption. In order to reduce battery consumption in the mobile front-end, and if data pushing/polling can be set to a fixed rate without jeopardizing the quality of the service and the user experience, the developer should configure the back-end of the web application to push data with a period above 5000 milliseconds. If new data are generated within this interval, it can be sent together in the same delivery without penalizing energy consumption.

RQ2. Which asynchronous communication method is the most efficient in terms of energy consumption? Are the differences significant in terms of energy consumption of the three communication mechanisms? In order to answer this

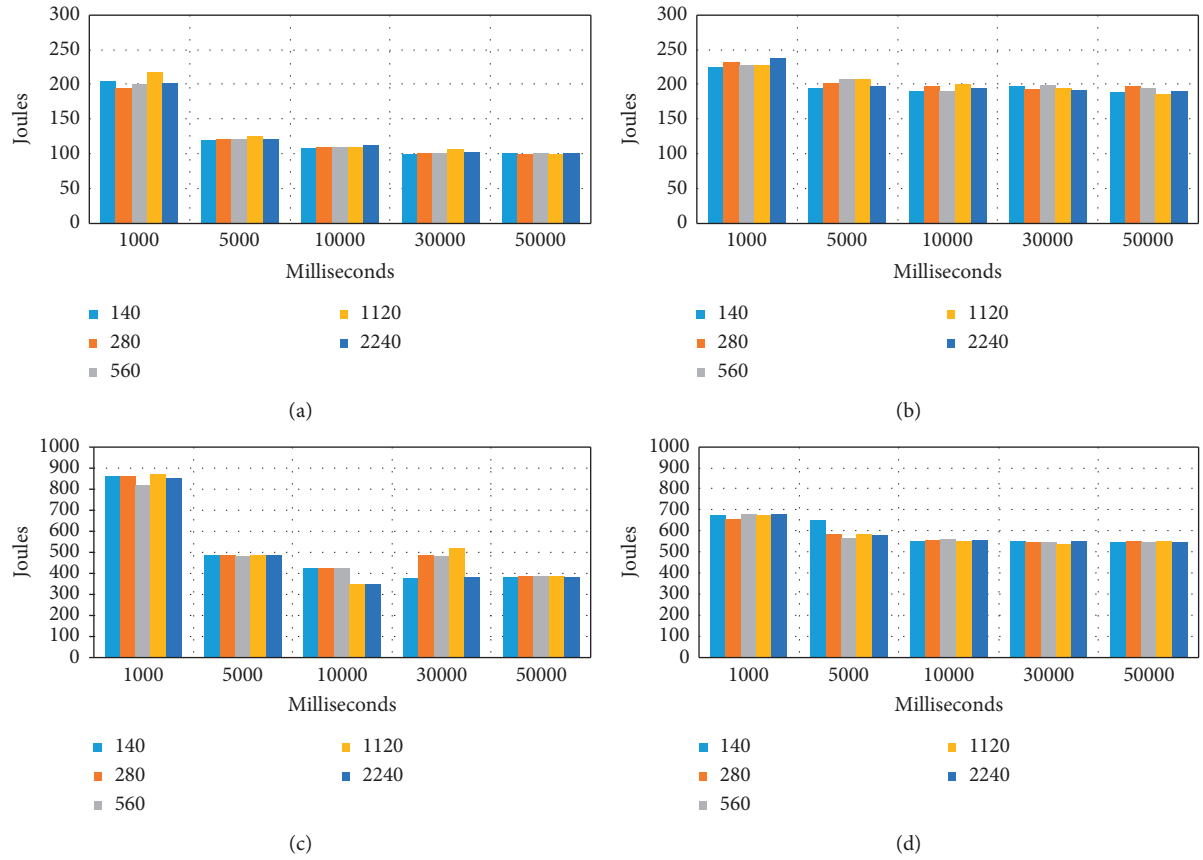

Figure 6: Energy consumption of Long Polling for 1-minute experiments in Chrome (a) and Firefox (b) and 5-minute experiments in Chrome (c) and Firefox (d).

question, we firstly make a quantitative study of the energy consumption values obtained for the different communication mechanisms and web browsers. Table 3 includes the descriptive statistics for these values. Polling is clearly the communication mechanism with the highest energy consumption while there is a strong similarity between Long Polling and WebSockets. In general terms, the use of WebSocket to push data shows the lowest energy consumption. However, for polling/pushing periods longer than 1 second and using Firefox, its consumption is very similar to the other communication mechanisms. Simple Polling is the asynchronous communication mechanism with the highest energy consumption but for longer polling periods its consumption is very similar to Long Polling. The answer is that, according to our results (Figures 5–7), there is no single asynchronous mechanism clearly better in all the scenarios.

This result is coherent with the active processing that the browser has to perform for small periods. For instance, if 80 bytes of data is available every 2000 milliseconds, the Polling client, with a polling interval of 1500 milliseconds, sends 40 HTTP requests of 462 bytes (a data overload of 7760 bytes plus the size of header in bytes of the HTTP response). For the same scenario, the client using Long Polling sends 30 HTTP requests of 462 bytes (plus the size of header in bytes of the HTTP response). Since the WebSocket client does not need to send any request nor data to the server to receive 80 bytes of data every 2000 milliseconds, the overhead is 0. The number of messages sent explains the higher energy consumption. In terms of usefulness, the TCP connection transports more useful application data than Polling and Long Polling.

Tests with random generation of data (see Table 1), which allows to measure energy consumption when data is generated by a bursty or interactive source, have confirmed a *similar energy consumption between Long Polling and WebSockets*. Both in the 1-minute and 5-minute experiments, energy consumption of the *WebSocket client is slightly lower than the energy consumption of Long Polling client*. These random results confirm the advantage of Long Polling and WebSocket over Polling when the source does not generate data at a fixed rate. When the most adequate polling period cannot be set in advance, the client side polls for data more frequently, provoking some empty responses and increasing energy consumption: the average energy consumption of Polling in Chrome for 1-minute experiments is 137 joules and for 5-minute experiments is 498 joules. Using Firefox, the consumption is higher: for 1-minute experiments the consumption is 225 joules, and for 5 minutes experiments is 616 joules. In all the tests (fixed rate/data and random generation), *Firefox energy consumption is higher than the consumption of Chrome*.

Taking into account the results, our *answer to the first part of RQ2* is that the energy consumption of Long Polling and WebSocket is very similar for the experiments performed. Simple polling is the most inefficient in terms of

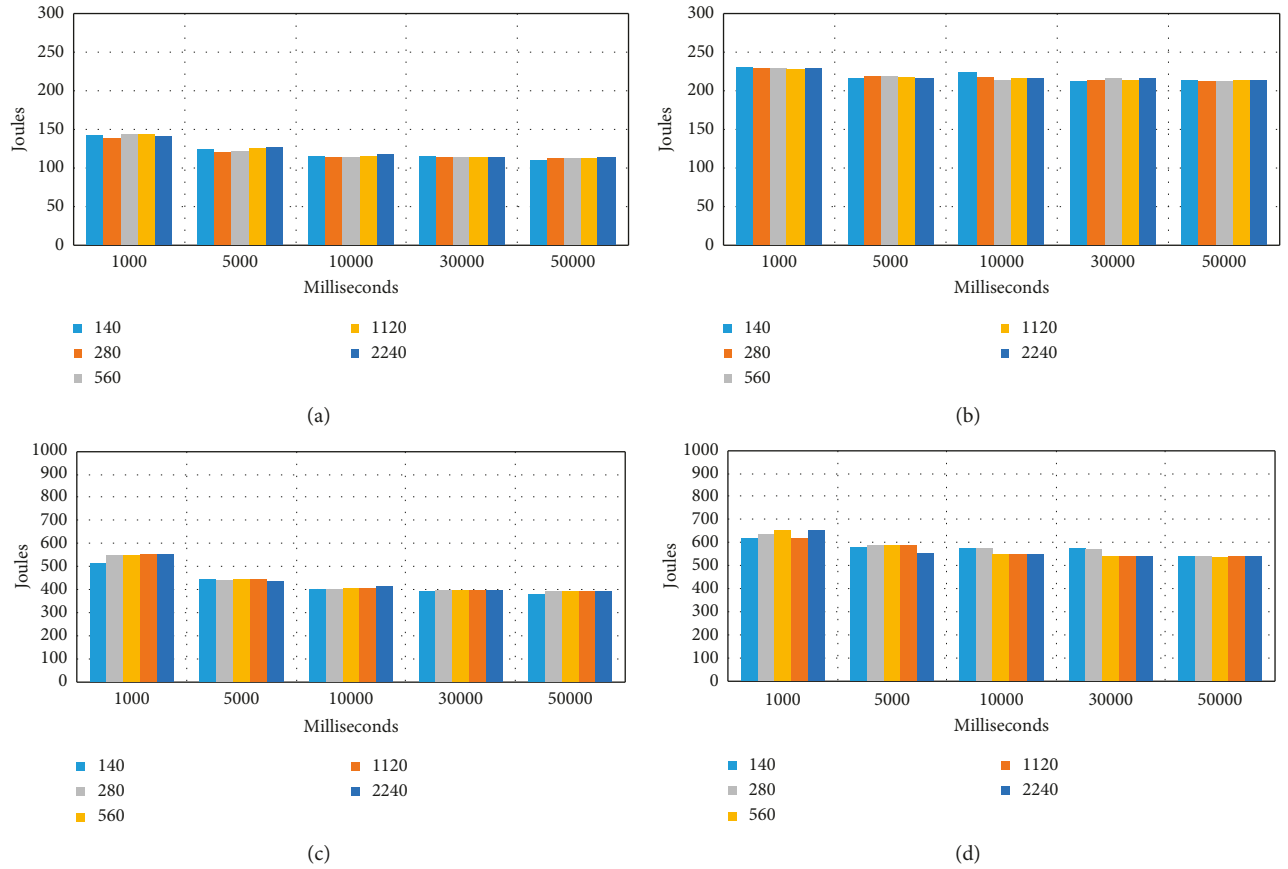

Figure 7: Energy consumption of WebSockets for 1-minute experiments in Chrome (a) and Firefox (b) and 5-minute experiments in Chrome (c) and Firefox (d).

Table 1: Energy consumption in joules of random experiments.

	Chrome		Firefox	
	1 min	5 min	1 min	5 min
Long Polling	117.583	420.295	203.734	589.927
WebSocket	104.533	423.350	215.115	491.951

energy consumption. However, when the new data are frequently available, Long Polling introduces an overload derived from the processing and sending of HTTP requests, which consumes more energy (using WebSockets, the client does not have to send request to server). Surprisingly, this is not maintained for longer periods. The reason is the WebSocket protocol sends periodically signalling data to keep the connection opened if it is not used and the energy consumption of this interaction is similar to the energy consumption caused by sending HTTP requests of Long Polling. So, surely many developers may think that WebSockets, being a more recent technology, will consume less than an older one, but we have found that the beliefs of these web developers do not correspond to the reality, showing the great utility of this kind of experiments.

Answering the second part of RQ2 implies to analyse whether these differences on energy consumption are statistically significant or not. We have visually analysed the energy consumption of the communication mechanisms using a box plot (Figure 14). These graphs evidence the different patterns of energy consumption depending on the web browser and the duration of the experiments. Differences in energy consumption are remarkable for 1-minute experiments (see Figures 14(a) and 14(b)). It is interesting to note that the lowest and the highest values of energy consumption are for Long Polling in all experiments except for Firefox in 5-minute experiments. We can see outliers in all experiments that can means statistically significant difference between the three communication mechanisms.

Hence, we also performed a Wilcoxon rank sum test (a.k.a. Mann–Whitney U-test) for each pair of groups to test observed differences in their averages. The Wilcoxon rank sum test is a nonparametric test appropriate for small samples as in our case. This test is used to test whether two samples are likely to derive from the same population. Therefore, the null hypothesis of this test is "there is not a statistically significant difference between two data sets." If the p value of this test is lower than 0.05, the null hypothesis is rejected. The p values for these tests are summarized in Table 4, the null hypothesis is rejected for Chrome in 1-minute experiments, but surprisingly is accepted for Firefox in 1 minute but between Polling and Long Polling.

So, in general, there are statistically significant differences depending on the communication mechanism. In 1-minute experiments, this is true with the exception of Firefox for which the differences are not significant. We have

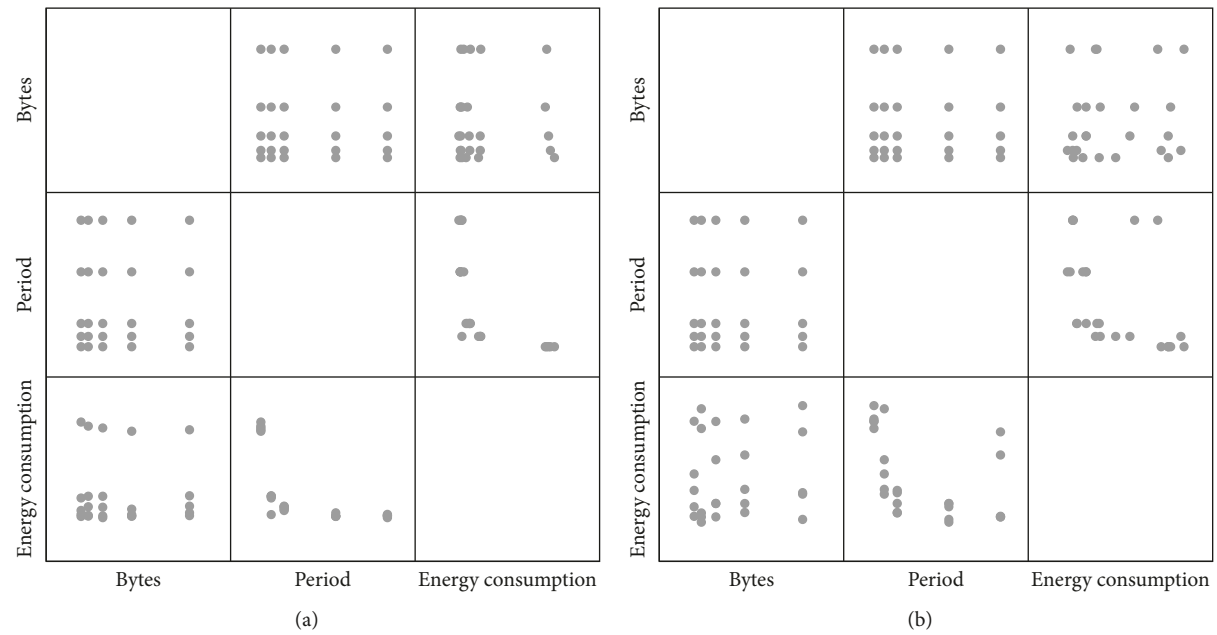

Figure 8: Scatterplots for Polling in Chrome (a) and Firefox (b) in 1-minute experiments.

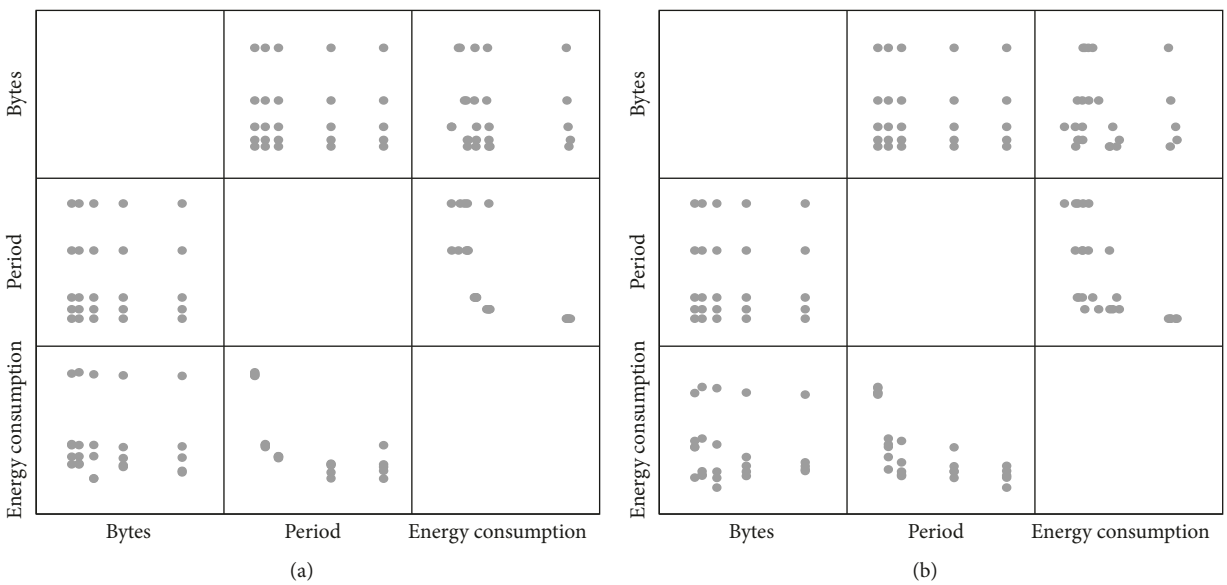

Figure 9: Scatterplots for Polling in Chrome (a) and Firefox (b) in 5-minute experiments.

different results for 5 minutes experiments and the *differences are statistically significant between Polling and Long Polling, and Long Polling and WebSockets in Chrome. In Firefox, there is a statistically significant difference between Polling and Long Polling, and Polling and WebSocket*.

The Wilcoxon sum rank test has demonstrated there is a statistically significant difference between some scenarios. However, we did not assess the relevance of these differences. In order to accomplish this task, we run an effect size test Cohen d for the cases where Wilcoxon has discarded the null hypothesis (Table 4, where black cells mean there is no statistically significant difference for Wilcoxon, so the Cohen d does not apply for that cases). According to the rule of thumb, values of Cohen d higher than 0.8 shows a large effect or, in other words, the differences between the two groups are relevant. In our experiments, this is true for Polling-Long Polling using Firefox, and Polling-WebSocket for Firefox in 5-minute experiment. On the other hand, values of Cohen d higher than 0.5 show a medium effect, which is the case of Polling-WebSocket for Chrome in 1-minute experiments and Long Polling-WebSocket for Chrome in the 5-minute scenario.

The answer to the second part of RQ2 is that, taking into account the results of the analysis, we conclude that *the communication mechanism with the lowest energy consumption is WebSocket. However, the difference on energy*

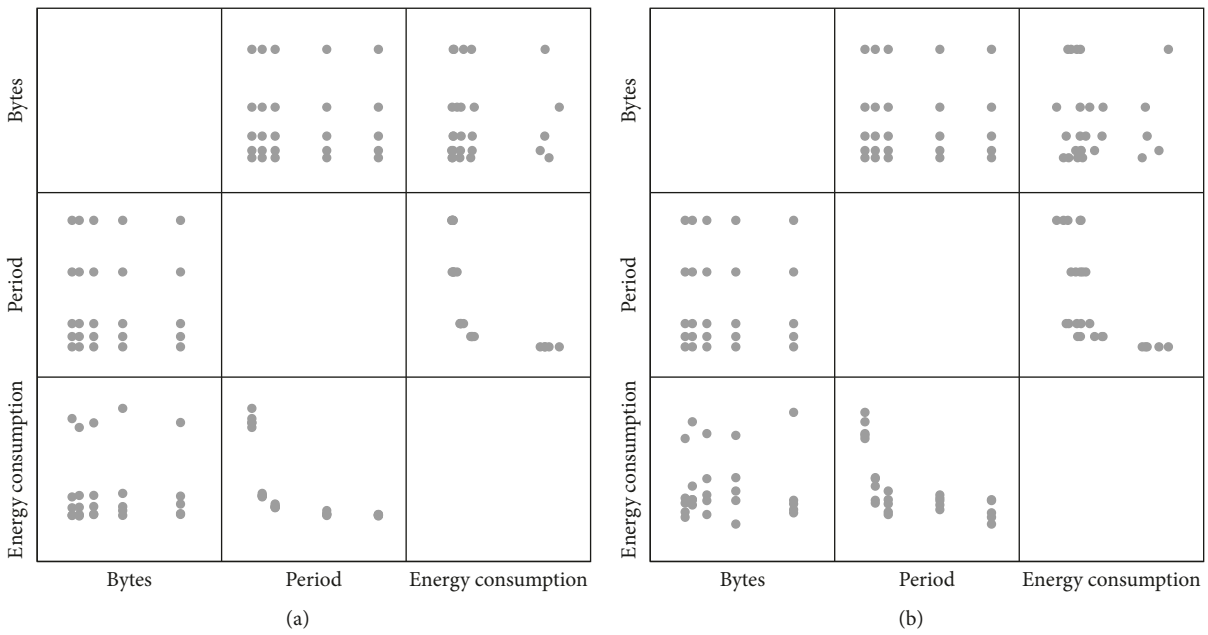

Figure 10: Scatterplots for Long Polling in Chrome (a) and Firefox (b) in 1-minute experiments.

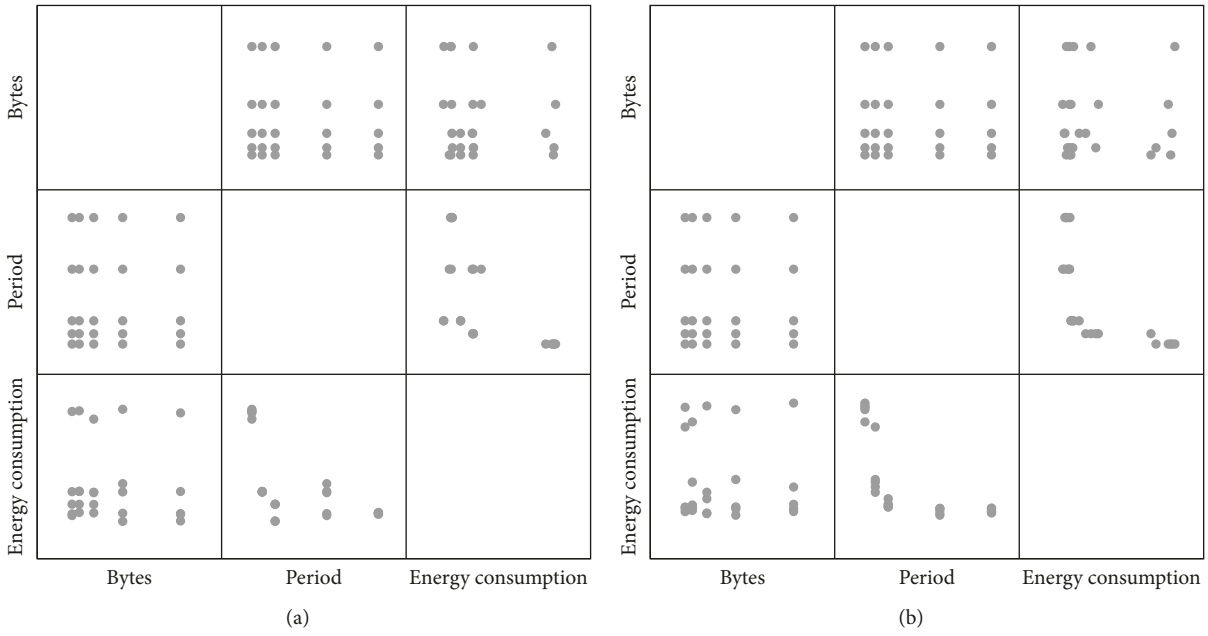

Figure 11: Scatterplots for Long Polling in Chrome (a) and Firefox (b) in 5-minute experiments.

consumption of this mechanism is only clearly relevant compared with Polling in Chrome for 1-minute experiments and Firefox for 5-minute experiments. According to our analysis, the most important difference on power consumption is between Polling and Long Polling when the web browser is Firefox.

5. Threats to Validity

This section briefly discusses the internal validity, construct validity, and external validity of our study. The internal validity intends to explore whether the results obtained are influenced or not by other factors. The threat to construct validity is concerned with how good is the relation between theory and observation. However, the external validity analyses whether the results obtained in the experiments can be generalized or not.

With regard to the internal validity, we should analyse the accuracy of the results provided by the energy-measuring tool. Although hardware solutions usually offer more precision in energy measurements, we have opted for using a software measuring tools, GreenOracle. As previously stated

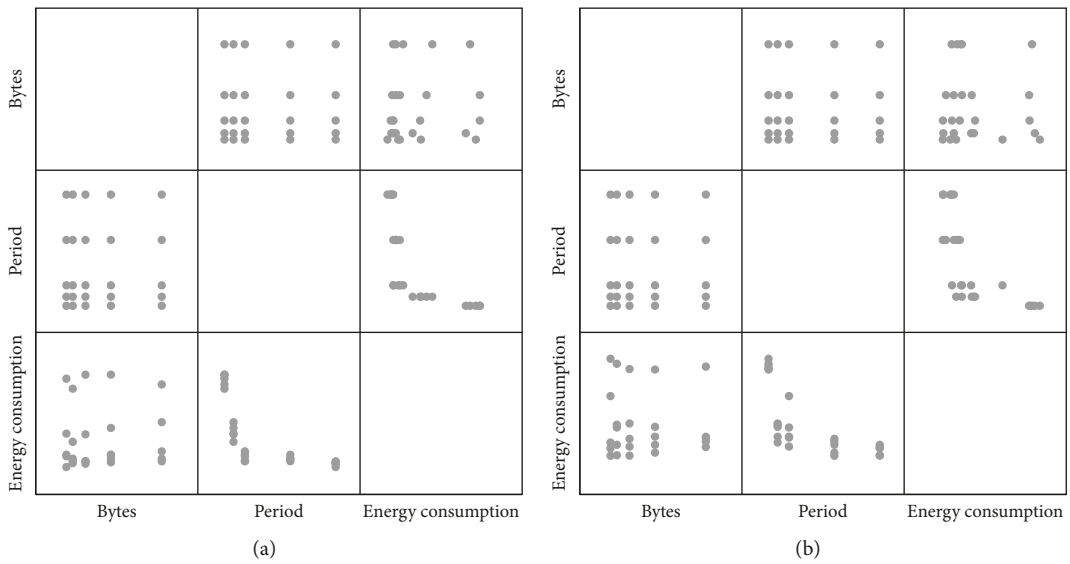

Figure 12: Scatterplots for WebSocket in Chrome (a) and Firefox (b) in 1-minute experiments.

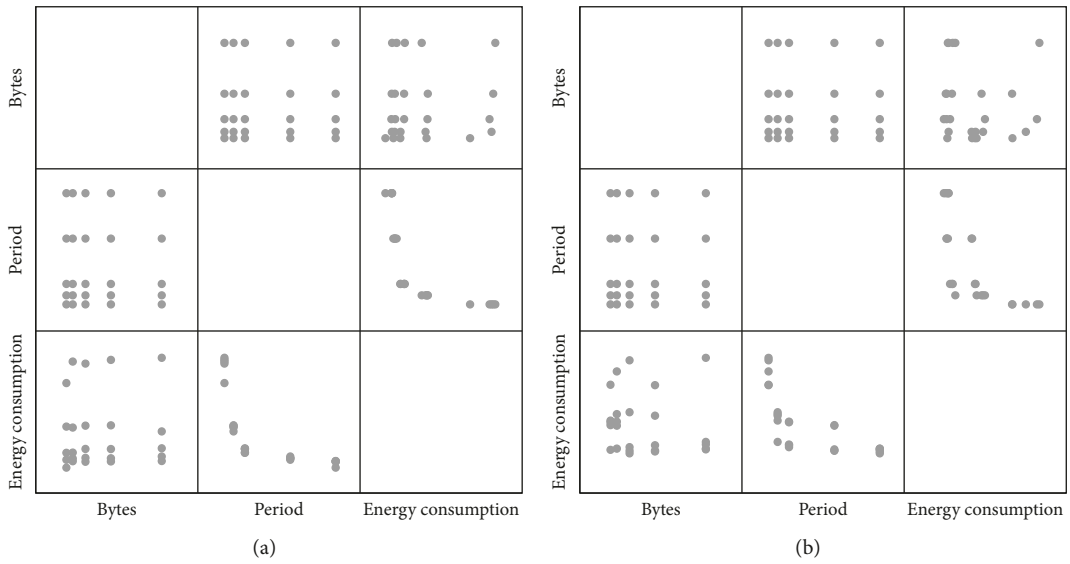

Figure 13: Scatterplots for WebSocket in Chrome (a) and Firefox (b) in 5-minute experiments.

Table 2: Spearman's rank correlation coefficient for 1-minute experiments.

		Chrome		Firefox	
		1 minute	5 minutes	1 minute	5 minutes
Polling	r_s	−0.833	−0.871	−0.698	−0.788
	p value	0.000	0.000	0.000	0.000
Long Polling	r_s	−0.957	−0.718	−0.757	−0.657
	p value	0.000	0.000	0.000	0.000
WebSocket	r_s	−0.945	−0.705	−0.672	−0.689
	p value	0.000	0.000	0.000	0.000

in the third section of this paper, the difficulties of reproducing experiments made by hardware solutions by third parties, and the precision demonstrated by software tools are the main reasons to select software solutions. Additionally, we are not interested in reporting absolute energy values, but to give recommendations to developers based on comparative results.

Another internal validity is related with the energy consumption of other elements of the Web applications in the experiments. In order to mitigate this threat, all the JavaScript clients present the same user interface, which equate the energy consumption of rendering in all the experiments. Internally, the three JavaScript-based client implementations are based in minimal examples provided by tutorials from Netbeans and university courses. The client does not use CSS, and the JavaScript functions used to implement asynchronous mechanisms are from standard JavaScript libraries and fully supported by mobile browsers used in our experiments.

Table 3: Descriptive statistics for energy consumption in joules.

	Chrome			Firefox		
	Polling	Long Polling	WebSocket	Polling	Long Polling	WebSocket
1 minute						
Mean	137.49	127.41	121.259	222.56	201.84	218.056
Standard deviation	31.946	39.728	11.255	21.164	14.89	1.152
Minimum	114.93	99.417	111.102	203.57	184.53	212.53
Maximum	203.79	216.08	143.659	279.18	236.46	229.628
5 minutes						
Mean	498.73	511.06	437.162	616.19	580.95	572.31
Standard deviation	155.33	179.97	58.551	55.447	50.952	37.24
Minimum	346.49	344.95	381.722	552.15	539.69	538.99
Maximum	795.87	865.93	555.983	723.39	678.6	656.863

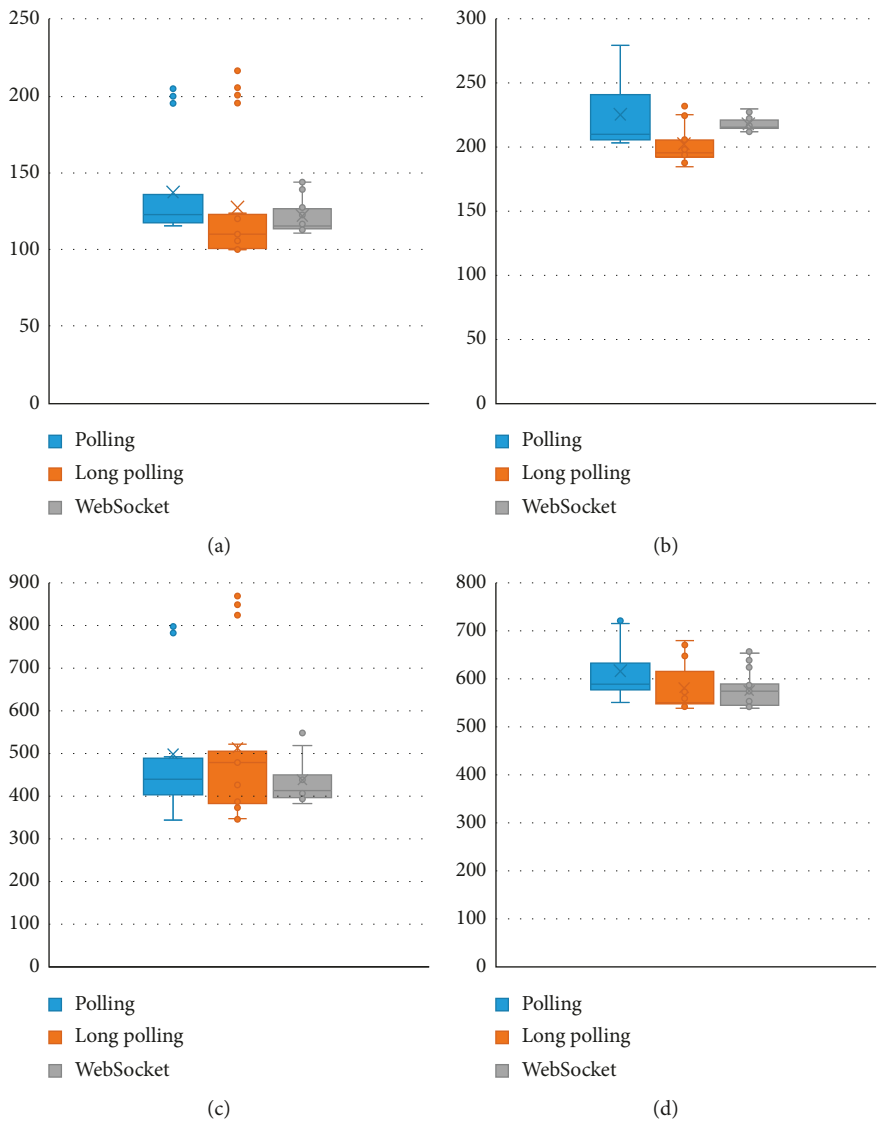

Figure 14: Box plots of the energy consumption for Polling, Long Polling and WebSockets in Chrome for 1-minute experiments (a), Firefox for 1-minute experiments (b), Chrome for 5-minute experiments (c), and Firefox for 5-minute experiments (d).

Another internal threat may come from using a set of parameters in the experiments that could not be considered exhaustive. According to the literature, the main factors involved in power consumption during asynchronous communications are the polling period and the amount of information transmitted. The range of values selected

Table 4: Statistically significance in the different scenarios (Wilcoxon sum rank test and Cohen d).

		Chrome		Firefox	
		1 min	5 min	1 min	5 min
Polling-Long Polling	Wilcoxon	0.009	0.000	0.000	0.002
	Cohen d	0.389	0.1	1.55	0.83
Polling-WebSocket	Wilcoxon	0.006	0.184	0.233	0.003
	Cohen d	0.709	—	—	0.997
Long Polling-WebSocket	Wilcoxon	0.045	0.000	0.677	0.839
	Cohen d	0.217	0.574	—	—

illustrate minimal interactions such as the delivery of a message in Twitter (140 bytes) and more complex cases that even require fragmentation at the IP level (2240 bytes). So, considering five different sizes of data, we think we have covered a great variety of situations. Regarding the frequency that new data are available, we have considered the energy consumption for long interactions or updates of webs that do not require interaction by the user. This situation can occur when the user requires to be notified of recent publications of social networks or specialized webs (e.g., trading, gambling and sport results webs, to mention a few).

The threats to construct validity analyse the extent to which the energy measurement tool measures the theoretical construct it is designed to measure. We have identified the election of a single software tool to measure the energy consumption as a threat to construct validity. In a previous section, we have already discussed that using only one energy measurement tool cannot be considered a serious construct threat since we have tested different widely accepted software tools in previous experiments, and the results showed that they provide similar energy measures.

Regarding the external validity, we have taken into consideration the influence of using a concrete mobile browser in the experiments. There are only two mobile browsers supporting the three asynchronous communication mechanisms of this study, Chrome and Firefox. We are aware that the use of one browser or other can affect to the energy consumption of the device. So, we have opted to use the two most extended web browsers in our experiments, Google Mobile Chrome and Mozilla Firefox for Android, which use a different browser engine (i.e., Gecko). Other mobile browsers use internally Blink, the same browser engine that includes Mobile Chrome.

Finally, we consider as an external threat the generalization of the results to all mobile phones and Android versions. Here the limitation is to have reliable energy measurement tools available for enough devices. In order to mitigate this limitation, we have opted for using GreenOracle as measuring tool (as stated before, GreenOracle and Trepn Profiler produce similar results for different devices).

6. Related Work

The energy consumption in mobile devices has been studied in different works. The experimental studies cope with energy consumption in different contexts and with different purposes [12, 27, 29–31, 39–51]. These studies are conducted to help developers about being concerned of energy consumption and contribute to provide alternative choices more energy saving. The study in [31] analyses how symptoms of poor design or implementation choices (namely, code smells) affect energy consumption. The study highlights that methods affected by some code smell types consume up to 87 times more than methods affected by other code smells and propose refactoring to reduce energy consumption in all of the situations. Code smells energy consumption is also studied in [29] in order to assess the benefits of a tool that automatically correct code smells and evaluate their impact on energy consumption. The study in [40] provides detailed profiles of the energy consumed by common operations performed on common data structures such as Java List, Map, and Set abstractions, showing that the alternative data types for these abstractions differ significantly in terms of energy consumption depending on the operations. Developers can use the usage context of a data structure and the measured energy profiles to decide between alternative collections implementations. The energy impact of logging in different Android mobile apps using GreenMiner is assessed in [30]. This study shows that logging has a negligible effect on energy consumption for most of the mobile applications tested. The approach in [40] combines empirical measurements of different machine learning algorithm implementations with complexity theory to provide concrete and theoretically grounded recommendations to developers who want to employ machine learning on smartphones in terms of energy consumption and accuracy. The study concludes that some implementations of algorithms generally perform better than others and indicates which other factors and parameters can affect which machine learning algorithms and what implementations will provide the best results.

The mobile device communication functions have also been considered in different experimental studies. In [39], the energy consumption of the main functions of a mobile phone, including data transfer with 2G and 3G, were researched. Later, the same authors analysed the energy consumption of different communication components like Bluetooth, WLAN, 2G, and 3G in more detail [44]. This study, conducted on a Nokia N95, concludes that 3G communication is more energy consuming than GSM (2G) communication, for using different application and services requiring the data connection.

As most of the mobile applications transfer data over the Internet, it is an important area to analyse the components needed for such data transfer at different layers. In [45], a comparison between WLAN and 3G with regard to their energy consumption is provided, showing that using WiFi as opposed to 3G is more energy efficient. This study also shows how the network activities (packet size and interval between packets sent/received) directly affect the energy consumption and ultimately battery life. Both works focus on the network access technology, and other studies try to optimize the energy consumption at the application level. In [46, 47], the differences between two current data interchange

formats (JSON and XML) are compared. The comparison analysed them with regard to the processing speed, overhead, and energy consumption. Results indicate that JSON format shows better performance in battery management. The comparison in [47] also test binary protocol buffers, which is most efficient when transferring big data volumes, showing better management energy for raw data.

Other works focus on providing a solution to make a more efficient use of battery for communications. An approach to extend the battery life by customizing the content was shown in [48]. The idea is that the server provides the content based on the battery status of the connected mobile phone. For instance, a mobile phone whose battery is nearly empty gets provided text and not video data. The server plays also an important role in the computation offloading approach, where battery intensive calculations get transferred to servers provided over the Internet (cloud computing) to save energy. Different existing researches show that communication and the data transfer is one of the important topics in regards to the energy consumption in mobile devices. Energy consumption at the transport layer including security issues is analysed in WLAN- and 3G-systems [49]. Close to our study, different works tackle with energy consumption of data transfer concepts in browsers of the mobile phones. In [50], WebSocket and AJAX are measured in regards to their energy consumption and performance for 3D graphic renderings in the browser of a mobile phone. A comparison between WebSockets and WebRTC as HTML5 connectivity methods was done by Mandyam and Ehsan [51]. They suggested some approaches how mobile web developers could reduce the power consumption on mobile devices such as the W3C battery API or the development of best practices. In [52], WebSocket protocol is compared to the Hypertext Transfer Protocol (HTTP) using the OpenPicus Flyport WLAN module. This work also studies the influence of the amount of data transferred and the transfer frequency in regards to the energy consumption. The work in [13] measures and compares HTTP/REST and WebSocket energy consumption using a mobile phone. The energy consumption between REST/HTTP and WebSocket is measured while using different access network technologies (Edge, 3G, and WLAN). The factors influencing the energy consumptions are identified by means of statistical analysis, and they conclude with the following results: (i) the use of REST consumes more energy than the use of WebSocket; (ii) the reason for the higher energy consumption of REST is not the overhead of the HTTP protocol; (iii) a continuous connect and disconnect consume more energy than a standing connection; (iv) the publish&subscribe model with WebSocket consumes less energy than the Long Polling with REST; (v) the standing connection with WebSocket has lower latencies than a REST connection; (vi) the publish&subscribe model with WebSocket consumes less energy than the Long Polling with REST; (vii) the used network (Edge, 3G, and WLAN) has influence on the result of the theses above.

Since energy is a critical resource for apps that run on mobile devices, several works cope with saving energy or optimizing battery duration as a problem to solve in the context of data transfer and communication. In [8], making HTTP requests is identified as one of the most energy consuming activities of a mobile phone among all operations. Based on previous studies, this work proposes an approach to reduce the energy consumption of HTTP requests in Android apps by automatically detecting and then bundling multiple HTTP requests.

7. Conclusions and Future Work

The experiments carried out in this paper provided interesting information for software developers about how different asynchronous communication mechanisms for Web applications and browser engines behave from an energy consumption point of view. According to the experimental data, we can state there is a significant difference between asynchronous solutions and the emulated one (i.e., simple polling) in the majority of the scenarios studied. However, between Long Polling and WebSocket, there is no scenario that can be considered the greenest one and their differences on power consumption are not statistically significant. The relevance of this study is also in the fine-grained information that it provides and can be used to make a reasoned decision about which is the best asynchronous technique for the requirements of each Web application. It is also very useful to find that software developers can increase the size of the pushed data without incurring in an increment in the energy consumption. However, software developers need to be careful because this is not the case if the frequency at which new data are pushed increases. The higher the update rate, the higher the energy consumption is. The pushing rate can be set by the back-end of the application attending the application requirements. The analysis in this paper helps in making that decision. In general terms, the use of WebSocket to push data shows the lowest energy consumption, although the energy consumption is very similar to Long Polling. According to our results, there is no single asynchronous mechanism clearly better in all the scenarios. Finally, another interesting conclusion is that Chrome for Android consumes less energy than Mobile Firefox, confirming our hypothesis that it is useful to explore and identify the most efficient browser engine.

As part of our future work, we plan to use these results to analyse if the asynchronous interaction of most Web applications can be improved by monitoring and refactoring or reconfiguring the asynchronous communication functions without incurring a huge penalty in either energy consumption or user experience. Another goal is to study and compare the energy consumption in HTTP/2 now that is supported by mobile browsers. HTTP/2 offers better performance of websites and web applications. However, better performance does not always mean lower energy consumption. We plan to study whether HTTP/2 offers improved energy consumption performance achieving longer battery life in comparison with HTTP/1.1. As part of this study, we also plan to investigate how Transport Layer Security (TLS) (mobile browsers only support HTTP2 over TLS-https) influences the energy consumption of the mobile devices.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the projects Magic P12-TIC1814 and HADAS TIN2015-64841-R (cofinanced by FEDER funds) and the postdoctoral plan of the University of Málaga.

References

[1] "Number of mobile phone users worldwide from 2015 to 2020 (in billions)," August 2018, https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/.

[2] D. Li, S. Hao, J. Gui, and W. G. J. Halfond, "An empirical study of the energy consumption of android applications," in *Proceedings of 2014 IEEE International Conference on Software Maintenance and Evolution*, pp. 121–130, Victoria, Canada, September 2014.

[3] M. Tawalbeh, A. Eardley, and L. Tawalbeh, "Studying the energy consumption in mobile devices," *Procedia Computer Science*, vol. 94, pp. 183–189, 2016.

[4] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, IMC '09*, pp. 280–293, Chicago, IL, USA, November 2009.

[5] "Mobile and tablet internet usage exceeds desktop for first time worldwide," May 2016, http://gs.statcounter.com/press/2016.

[6] R. T. Fielding, J. Gettys, J. C. Mogul et al., *Hypertext Transfer Protocol–http/1.1*, Internet Requests for Comments, RFC Editor, RFC 2616, World Wide Web Consortium, Cambridge, MA, USA, 1999, http://www.rfc-editor.org/rfc/rfc2616.txt.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, *Hypertext Transfer Protocol–http/1.1*, Internet Requests for Comments, RFC Editor, RFC 2068, World Wide Web Consortium, Cambridge, MA, USA, 1997.

[8] D. Li, Y. Lyu, J. Gui, and W. G. J. Halfond, "Automated energy optimization of http requests for mobile applications," in *Proceedings of the 38th International Conference on Software Engineering, ICSE '16*, pp. 249–260, Austin, TX, USA, 2016.

[9] I. Fette and A. Melnikov, *The WebSocket Protocol*, Internet Requests for Comments, RFC Editor, RFC 6455, World Wide Web Consortium, Cambridge, MA, USA, 2011, http://www.rfc-editor.org/rfc/rfc6455.txt.

[10] V. Pimentel and B. G. Nickerson, "Communicating and displaying real-time data with WebSocket," *IEEE Internet Computing*, vol. 16, no. 4, pp. 45–53, 2012.

[11] P. Beverloo, M. Thomson, M. van Ouwerkerk, B. Sullivan, and E. Fullea, "Push API," Technical Report, W3C, Cambridge, MA, USA, January 2019, https://www.w3.org/TR/push-api/.

[12] I. Ayala, M. Amor, L. Fuentes, and D.-J. Munoz, "An empirical study of power consumption of web-based communications in mobile phones," in *Proceedings of IEEE 15th International Conference on Pervasive Intelligence and Computing*, pp. 861–866, Orlando, FL, USA, November 2017.

[13] V. Herwig, R. Fischer, and P. Braun, "Assessment of REST and Websocket in regards to their energy consumption for mobile applications," in *Proceedings of IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2015*, pp. 342–347, Warsaw, Poland, September 2015.

[14] S. Loreto, P. Saint-Andre, S. Salsano, and G. Wilkins, *Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional http*, Internet Requests for Comments, RFC Editor, RFC 6202, World Wide Web Consortium, Cambridge, MA, USA, 2011, http://www.rfc-editor.org/rfc/rfc6202.txt.

[15] A. Russell, "Comet: low latency data for the browser," 2006, http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/.

[16] J. Postel, *Transmission Control Protocol*, Internet Requests for Comments, RFC Editor, STD 7, World Wide Web Consortium, Cambridge, MA, USA, 1981, http://www.rfc-editor.org/rfc/rfc793.txt.

[17] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao, and S. Tarkoma, "Modeling, profiling, and debugging the energy consumption of mobile devices," *ACM Computing Surveys*, vol. 48, no. 3, pp. 39:1–39:40, 2015.

[18] D. Li, S. Hao, W. G. J. Halfond, and R. Govindan, "Calculating source line level energy information for android applications," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis, ISSTA 2013*, pp. 78–89, Lugano, Switzerland, July 2013.

[19] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky, "GreenMiner: a hardware based mining software repositories software energy consumption framework," in *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014*, pp. 12–21, Hyderabad, India, June 2014.

[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, MobiSys '11*, pp. 335–348, Bethesda, MD, USA, June-July 2011.

[21] "Power monitor," April 2017, https://www.msoon.com/LabEquipment/PowerMonitor/.

[22] D. Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia, "Software-based energy profiling of android apps: simple, efficient and reliable?," in *Proceedings of IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Klagenfurt, Austria, February 2017.

[23] D. Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia, "Petra: a software-based tool for estimating the energy profile of android applications," in *Proceedings of the 39th International Conference on Software Engineering Companion*, Buenos Aires, Argentina, May 2017.

[24] S. Chowdhury, S. Borle, S. Romansky, and A. Hindle, "GreenScaler: training software energy models with automatic test generation," *Empirical Software Engineering*, pp. 1–44, 2018.

[25] A. Hindle, "Green software engineering: the curse of methodology," in *Proceedings of IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 5, pp. 46–55, Osaka, Japan, March 2016.

[26] S. A. Chowdhury and A. Hindle, "GreenOracle: estimating software energy consumption with energy measurement corpora," in *Proceedings of the 13th International Conference on Mining Software Repositories, MSR '16*, p. 4, Austin, TX, USA, May 2016.

[27] S. A. Chowdhury, S. Gil, S. Romansky, and A. Hindle, "Did I make a mistake? Finding the impact of code change on energy regression," PeerJ Preprints, Technical Reports 5:e2419v3, University of Alberta, Edmonton, AB, Canada, 2016.

[28] "Trepn power profiler a product of Qualcomm Technologies, Inc.," April 2017, https://developer.qualcomm.com/software/trepn-power-profiler.

[29] A. Carette, M. A. A. Younes, G. Hecht, N. Moha, and R. Rouvoy, "Investigating the energy impact of android smells," in *Proceedings of 24th International IEEE Conference on Software Analysis, Evolution and Reengineering (SANER)*, p. 10, Chicago, IL, USAs, February 2017.

[30] S. Chowdhury, S. Di Nardo, A. Hindle, and Z. M. Jiang, "An exploratory study on assessing the energy impact of logging on Android applications," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1422–1456, 2018.

[31] F. Palomba, D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia, "On the impact of code smells on the energy consumption of mobile applications," *Information and Software Technology*, vol. 105, pp. 43–55, 2019.

[32] K. Aggarwal, C. Zhang, J. C. Campbell, A. Hindle, and E. Stroulia, "The power of system call traces: predicting the software energy consumption impact of changes," in *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering, CASCON'14*, pp. 219–233, Markham, Canada, November 2014, http://dl.acm.org/citation.cfm?id=2735522.2735546.

[33] "Android debug bridge," April 2017, https://developer.android.com/studio/command-line/adb.html?hl=es-419.

[34] "Image color summarizer, RGB, HSV, LCH & Lab image color statistics and clustering simple and easy," April 2017, http://mkweb.bcgsc.ca/color-summarizer/.

[35] V. R. Basili, "Software modeling and measurement: the goal/question/metric paradigm," Techical Report, UMIACS TR-92–96, University of Maryland at College Park, College Park, MD, USA, 1992.

[36] N. van Berkel, C. Luo, T. Anagnostopoulos et al., "A systematic assessment of smartphone usage gaps," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, CHI '16*, pp. 4711–4721, San Jose, CA, USA, May 2016.

[37] D. Ferreira, J. Goncalves, V. Kostakos, L. Barkhuus, and A. K. Dey, "Contextual experience sampling of mobile application micro-usage," in *Proceedings of the 16th International Conference on Human-computer Interaction with Mobile Devices & Services, MobileHCI '14*, pp. 91–100, Toronto, Canada, September 2014.

[38] D. Hintze, P. Hintze, R. D. Findling, and R. Mayrhofer, "A large-scale, long-term analysis of mobile device usage characteristics," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 2, pp. 1–21, June 2017.

[39] W. Waybe and Daniel, *Applied Non-Parametric Statistics*, Houghton Mifflin Harcourt, Boston, MA, USA, 1978.

[40] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle, "Energy profiles of java collections classes," in *Proceedings of the 38th International Conference on Software Engineering*, pp. 225–236, Chicago, IL, USA, May 2016.

[41] A. McIntosh, S. Hassan, and A. Hindle, "What can Android mobile app developers do about the energy consumption of machine learning?," *Empirical Software Engineering*, pp. 1–40, 2018.

[42] E. A. Santos, C. McLean, C. Solinas, and A. Hindle, "How does docker affect energy consumption? Evaluating workloads in and out of docker containers," *Journal of Systems and Software*, vol. 146, pp. 14–25, 2017.

[43] 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, EuroSys '11*, pp. 153–168, Salzburg, Austria, April 2011.

[44] G. Perrucci, F. H. P. Fitzek, G. Sasso, W. Kellerer, and J. Widmer, "On the impact of 2g and 3g network usage for mobile phones' battery life," in *Proceedings of European Wireless*, pp. 255–254, Aalborg, Denmark, May 2009.

[45] G. P. Perrucci, F. H. P. Fitzek, and J. Widmer, "Survey on energy consumption entities on the smartphone platform," in *Proceedings of VTC Spring*, pp. 1–6, Budapest, Hungary, May 2011.

[46] G. Metri, A. Agrawal, R. Peri, and W. Shi, "What is eating up battery life on my smartphone: a case study," in *Proceedings of ICEAC*, pp. 1–6, Guzelyurt, Cyprus, December 2012.

[47] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of JSON and XML data interchange formats: a case study," in *Proceedings of the ISCA 22nd International Conference on Computer Applications in Industry and Engineering*, pp. 157–162, San Francisco, CA, USA, November 2009.

[48] B. Gil and P. Trezentos, "Impacts of data interchange formats on energy consumption and performance in smartphones," in *Proceedings of the 2011 Workshop on Open Source and Design of Communication, OSDOC '11*, pp. 1–6, Lisboa, Portugal, 2011.

[49] M. W. Kim, D. G. Yun, J. M. Lee, and S. G. Choi, "Battery lifetime extension method using selective data reception onsmartphone," in *Proceedings of ICOIN*, Y. Kim, C. Kim, and P. Tantatsanawong, Eds., pp. 468–471, Bali, Indonesia, February 2012.

[50] P. Miranda, M. Siekkinen, and H. Waris, "TLS and energy consumption on a mobile device: a measurement study," in *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*, pp. 983–989, Corfu, Greece, June-July 2011.

[51] K. Kapetanakis and S. Panagiotakis, "Evaluation of techniques for web 3d graphics animation on portable devices," in *Proceedings of International Conference on Telecommunications and Multimedia, TEMU 2012*, pp. 152–157, Crete, Greece, July-August 2012.

[52] B. D. Mandyam and N. Ehsan, "Mobile systems IV," Technical Report, World Wide Web Consortium, Cambridge, MA, USA, 2012.