MANCHESTER 1824
The University of Manchester

The University of Manchester Research

Twisting Web Pages for Saving Energy

DOI:
10.1007/978-3-319-60131-1_13

Document Version
Accepted author manuscript

Link to publication record in Manchester Research Explorer

Citation for published version (APA):
Köksal, E., & Harper, S. (2017). Twisting Web Pages for Saving Energy. In *International Conference on Web Engineering* (pp. 225-245) https://doi.org/10.1007/978-3-319-60131-1_13

Published in:
International Conference on Web Engineering

Citing this paper
Please note that where the full-text provided on Manchester Research Explorer is the Author Accepted Manuscript or Proof version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version.

General rights
Copyright and moral rights for the publications made accessible in the Research Explorer are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Takedown policy
If you believe that this document breaches copyright please refer to the University of Manchester's Takedown Procedures [http://man.ac.uk/04Y6Bo] or contact uml.scholarlycommunications@manchester.ac.uk providing relevant details, so we can investigate your claim.

OPEN ACCESS

Download date:26. Apr. 2024

Twisting Web Pages for Saving Energy

Eda Köksal[1], Yeliz Yeşilada[1], and Simon Harper[2]

[1] Middle East Technical University Northern Cyprus Campus,
Güzelyurt, Mersin 10, Turkey [edakoksal89@gmail.com||yyeliz@metu.edu.tr]
[2] Schoool of Computer Science, University of Manchester,
Manchester, United Kingdom [simon.harper@manchester.ac.uk]

Abstract. Battery capacity (energy density) is increasing at around 3% per year. However, the increasing requirements of the mobile platform is placing higher demands on this capacity. In this case, there are three options: decrease our expectations of the mobile platform, increase the capacity and therefore size and weight of our batteries, or create energy saving solutions to extend battery-life with minimal effect on platform performance. Here we present a system called Twes+ [3] which is inline with the last option and aims to transcode web pages for increasing battery-life when surfing-the-web without changing the look and feel. Our evaluation results show that there is a statistically significant energy saving when using our Twes+ transcoder. Our *redirect* service brings a 4.6% cumulative processor energy reduction, while *image* transcoding service, brings a 7% cumulative processor energy reduction. These savings equate to between a 40 to 60 minute saving depending on the mobile device.

Keywords: Transcoding, Energy, Green Computing, Mobile Web

1 Open Data

Twes+ is open sourced at https://github.com/EdaKoksal/Twest.git. This repository includes materials used, raw measurements and statistical test results.

2 Introduction

Mobile device ownership is increasing worldwide especially in developing regions. For example, in 2014, the number of mobile Internet users exceeded the number of desktop users and in 2015, the number of connected devices per person rose to 3.42. 2020s prediction shows that this ratio will be doubled [34]. Although the popularity of mobile devices increases, these devices still have constraints such as the limited bandwidth of connection, device memory, processing power, battery size, etc. [16, 28, 39]. Further, battery capacity (energy density) is only increasing at around 3% per year [34]. On the other hand, web technologies evolve quickly and these evolutions normally require increased energy capacity. Indeed, the total size and the total number of requests required between client and server are steadily increasing. For instance, total transfer size has increased from 831 KB to 2135 KB between 2012–2015 [3]. These changes and also

[3] Twes+ is pronounced as Twist

the limitations of mobile devices can make their everyday usage inconvenient. Mobile devices connect to the web usually through a cellular connection or a Wi-Fi. These connections consume a serious amount of energy. For example, the battery life of an iPhone 7 is up to 240 hours on standby, 12 hours on the cellular connection and 14 hours on Wi-Fi connection [46]. As a result of the constraints and the changes in the web, accessing the web from a mobile device can quickly drain the battery [19].

To address these limitations, this paper presents a system called Twes+ (***T***ranscoding ***W***eb pages for ***E***nergy ***S***aving) that is proposed to save energy on the client-side. Twes+ aims to systematically transcode the source code of web pages on the proxy so that, during browsing, less energy is consumed on the client. Transcoding here means the process of transforming and adapting the source code of web pages into alternative forms to improve the user experience (UX) [4]. While Twes+ transcodes web pages, it takes three principles into account: the look&feel of the web page should not be changed, the changes should not depend on server-side, and the changes should not add extra energy consumption or settings to the client-side. When a web page is visually designed, it includes implicit information that guides the user around the web page [42]. Therefore, we want to preserve this implicit information. Even though some web pages have specific mobile versions, these typically include the subset of what is available on the desktop, so there is information loss. This can be addressed with Responsive Web Design (RWD) but in practice not many sites are "fully responsive" and they can also remove functionality required on mobile devices [20]. Therefore, with this principle, we do not want to reduce the amount of information, and we do not want to change the look&feel. Regarding the second principle, even though there exist guidelines to develop web pages with energy awareness, very few pages take them into account [16, 28]. Furthermore, there are many non-technical/programmer web developers, so expecting them to address these guidelines might not be realistic. By keeping Twes+ independent from the web server, the proposed adaptations can be applied to any web page systematically. That is why Twes+ is proposed as a proxy-based system. Regarding the third principle, the focus is to improve the energy consumption on the client, so the developed system should not add extra energy consumption to the client and also the user should not be asked to make configurations, etc. as users tend not to use such advance settings [27]. Therefore, Twes+ is implemented as a proxy which does not modify the look&feel of the page, not require any extra workload from the server side, not require the user to make any configurations and not consume extra energy on the client.

In the literature, many transcoding techniques are proposed (§ 3). Some of these aims to improve the UX for mobile or disabled web users, and because of the principles introduced above they are not appropriate for our purpose. Techniques have also been introduced to save energy such as compression servers, and are proven to save energy on the client-side [5, 13, 16, 38, 44]. However, there are still techniques that can improve energy but have not been investigated which include reducing the number of redirects and consolidating images. These are already proposed as guidelines but their effect on energy saving have not been demonstrated [12, 16, 17, 28, 30, 44]. Therefore, Twes+ encodes two transcoding techniques: redirect transcoding and image transcoding service based on Internet Content Adaptation Protocol (ICAP) (§ 4). Therefore, the overall **contribution** of Twes+ is scientifically showing that these transcoding techniques can save energy as suggested by the unproven guidelines.

In order to assess the impact of Twes+, we conducted an experiment which investigates the following research questions: 1. "Does Twes+ save energy by applying redirect transcoding to reduce the number of redirects?" 2. "Does Twes+ save energy by reducing the number of requests/ responses between client&server to retrieve images?". For redirect service, a controlled web page is used, and three different redirection cases are created. Case 1 is designed without any redirection. When the web page is requested, the content can be accessed from the original URL. Case 2 is designed with five redirections, as more than five redirections indicate an infinite redirection loop [18]. The web page content starts loading after the first five redirection requests/responses. Case 3 is designed with infinite redirection where the web page content cannot be accessed.

For image service, we used Alexa top 100 pages and assess the energy consumption with and without Twes+ (§ 5). For comparison, we measured the number of requests/responses and the cumulative energy of the processor. The experiment illustrates that Twes+ saves energy on the client side by reducing the number of redirect round trips and the number of image requests/responses without modifying the look&feel of the web pages. Twes+ was able to eliminate the infinite temporary redirect round trips and with Twes+ 4.6% reduction in cumulative processor energy is observed. With image transcoding service, on average 7% reduction in cumulative processor energy is observed by reducing the number of request/response round trips between the client and the server. It is also observed that this percentage is more than 10% with pages that include many images (§ 6). Based on these results, we also look at battery life improvements with Twes+ and our estimations show that Twes+ can provide significant battery life improvements, especially when the page has a lot of images and the number of redirection is above the given limit in the HTTP specification (§ 7). For example, in average Twes+ provides 7.6% power reduction and if we consider specific pages, for example ebay, we can say that Twes+ provides 40.27 minutes battery improvements when the cumulative processor energy is equal to the entire system energy consumption and 24.99 minutes improvement when the cumulative processor energy is equal to the 70% of the system energy consumption (assuming that 30% is used by other processes, and only ebay is processed). Although there are some limitations in the current version of Twes+ (§ 7), the overall results show that Twes+ saves energy by systematically transcoding the source code of web pages (§ 8).

3 Related Work

Web pages can be accessed in different ways on mobile devices including native applications, mobile web applications, widgets and browsers. Among these, browsers are cross-platform, and they can be used to access any web page. Most of the mobile browsers provide ways to control energy saving on mobile devices. For example, they can be used to block accessing the flash content, compress data, and transcode the visual look&feel of web pages. For instance, Google Chrome browser has data saver [13] on mobile device browsers for compression, Apple Safari has power saver on desktop browsers of laptops for blocking the flash content and Opera Mini browser has data saving mode [32] on mobile device browsers for transcoding the web pages to reduce the content, etc. These are simple but yet very useful techniques for saving energy, however more complex techniques can be explored that can potentially save even more energy.

When we consider the web architecture, we can see that energy consumption can be reduced in three ways: on the hardware side, network interaction and the software/application side. On the hardware side, there are studies that consider the ways of using renewable energy sources in the mobile devices themselves to improve the battery life. For instance, methods are introduced to integrate thermoelectric generator to the Central Processing Unit (CPU) heat pipe to use the waste heat of the processor and integrating photovoltaic unit to the devices to generate power from environmental illumination [2]. Intel Corporation developed a mobile processor that uses Core-Multi-Processor micro-architecture to achieve high performance with low power consumption [21]. Even though these are very crucial for saving energy on mobile devices, software and network can also be as important as hardware for saving energy.

There are also studies to improve energy consumption on the network side. For instance, the energy consumption of a customer is examined, and the results show that 0.83Wh/day for a terminal is consumed whereas 120Wh/day for the mobile network is consumed [15]. There are also studies to reduce the energy consumption of localization services such as a system called Senseless. Instead of only using the Global Positioning System (GPS), it combines accelerometer, GPS and 802.11 access point in location-aware services. Their early results show that their design can extend the battery life of mobile devices from nine hours to 22 hours [6]. Another proposed optimisation for localisation is changing the host-centric network paradigms. The current host-centric approach is based on placing all intelligence at the host. By adopting an information-centric approach or by adapting a multi-access mobile networking, energy saving can be achieved. Briefly, the information-centric approach means the network connects the information consumers with information producers and distributors instead of nodes. The multi-access mobile networking approach means it uses surrounding networks together [34].

Compared to network and hardware based studies, there are limited studies on the software side. There are some studies on the protocol side. Google, for instance, has developed a protocol called SPeeDY Protocol (SPDY) which is proposed as a replacement of Hypertext Transfer Protocol (HTTP). It allows multiple requests in a Transmission Control Protocol (TCP) session, so the number of TCP sessions setup can decrease. In addition to this, the server and the client can push data to each other without a setup request. The main purpose of SPDY protocol is to reduce the latency [11]. In addition to studies in the protocol, there are also specific studies on mobile operating systems. For instance, for Android OS, one study shows that better compilers for Android can save energy [33]. There is also a study for energy saving on the mobile applications. This study gives guidelines to develop applications that considers the limitations of mobile applications by studying some micro-benchmarks. By writing the same micro-benchmark or common primitives or briefly code refactoring, the battery life improvement can be achieved [47]. In addition to these studies, the energy consumption by a browser to display components of a web page is also measured. For example, to download and display Apple page, the required energy is approximately 46 Joules – 12 Joules of this 46 Joule is to download and render CSS files. To decrease the energy consumption, non-used functions are removed in the CSS file and five Joules have been saved [31]. Even though these are important studies to show how the energy is consumed by the browser, the results of such studies can be only be used to develop guidelines for web developers.

Besides these studies on the software side, some transcoding techniques have also been introduced in the literature either as a browser plugin, a client-side application or as a proxy based applications. Table 1 shows these techniques and our examination of them according to the principles discussed in the Introduction section: do they modify the look&feel of the page?, do they do transcoding on the server, client or proxy side? are there any studies to demonstrate that they save energy in the client?

There are some techniques that focus on adapting the content of web pages to address the screen size limitation and therefore they modify the look&feel. For example, some techniques provide a summary of the page [4, 39, 1, 26], however even though this could be good for energy reduction, it also significantly reduces the content. Similarly, some techniques are introduced to simplify the page that can save energy [4, 7, 43]. Furthermore, there are also techniques that can potentially save energy, for example, techniques that rearrange the page content such that only the important parts are displayed to save energy [39, 4, 36, 35], techniques that automatically add alternative texts to images that can replace images for saving energy [4, 22, 40], and color scheme changes so that the client spends less energy for rendering [4, 40, 22].

When we look at the location of the transcoding implementations exist in the literature, we see that there are some techniques already implemented as a proxy such as data compression and concatenation of external files. These studies focus on compressing data so to reduce the amount of data transferred [5, 16, 13, 8, 23] and they focus on concatenating external files such as stylesheets and scripts to reduce the amount of HTTP requests [16, 44, 38]. These studies demonstrate to improve energy consumption on mobile devices. There are also some techniques implemented on the proxy that affect the battery life negatively. For example, [29] proposes image inlining but this requires both encoding on the server side and decoding of these images on the client side. This was demonstrated to be six times slower than external linking on mobile devices.

Finally, when we look at Table 1, we see that there are still three techniques that can potentially improve the battery life and do not modify the look&feel of the page but they have not been technically investigated. These are image consolidation, responsive image and reducing the number of redirects. They are proposed as guidelines for developers to make their pages faster, but they have not been implemented as transcoding techniques. Although in Twes+, these promising techniques are implemented on a proxy, this paper only covers the image consolidation and the number of redirects reduction.

4 Twes+

Figure 1 shows the software architecture of Twes+. Squid Caching Proxy is used as a proxy server (A in Figure 1) which is a popular open source proxy server to cache the frequently accessed web pages. It can reduce the response time and the payload. Moreover, Squid Caching Proxy supports different content adaptation mechanisms. The ICAP mechanism is adopted because it can modify both the request and response content. Moreover, it is not dependent on a single proxy server project or vendor [14].

The ICAP requires an HTTP proxy as a client (A) and an ICAP-server (B) to reside the adaptation algorithm [14]. The relation between the ICAP-server and the ICAP-client is many-to-many. The content of the web page comes from the ICAP-client to the ICAP-server for adaptation. In the Squid Caching Proxy-ICAP section [4], some

[4] http://wiki.squid-cache.org/Features/ICAP

Table 1: Transcoding methods [§P/wSN: People with Special Needs, S: Server-Side, C: Client-Side, P: Proxy-Side, *: Reverse-Proxy].

Technique	Reference	Location			Modify the Look & Feel	Adaptation for P/wSN §	Screen Size Adaptation	Battery Life Improvement
		S	P	C				
Text Magnification	[4]	✓	χ	✓	✓	✓	✓	?
Color Scheme Changes	[4, 40, 22]	χ	✓	✓	✓	✓	χ	?
Alternative Text Insertion	[4, 22, 40]	χ	✓	✓	✓	✓	✓	✓
Page Rearrangement	[39, 4, 36, 35]	✓	✓	χ	✓	✓	✓	?
Simplification	[4, 7, 43]	✓	✓	χ	✓	χ	✓	✓
Summarization	[4, 39, 1, 26]	χ	✓	✓	✓	χ	✓	✓
Image Consolidation	[24, 16, 12, 17]	✓	✓*	χ	χ	χ	χ	?
Responsive Image	[20, 16, 28, 12, 30]	✓	χ	✓	χ	χ	✓	?
Image Inlining	[28, 29]	✓	χ	χ	χ	χ	χ	χ
Data Compression	[5, 16, 13, 8, 23]	✓	✓	✓	χ	χ	χ	✓
Concatenation of external files	[16, 44, 38]	✓	✓	χ	χ	χ	χ	✓
Reducing the number of Redirects	[44, 41, 37]	✓	✓*	χ	χ	χ	χ	?
Expiration Header	[16]	✓	✓	✓	χ	χ	χ	✓

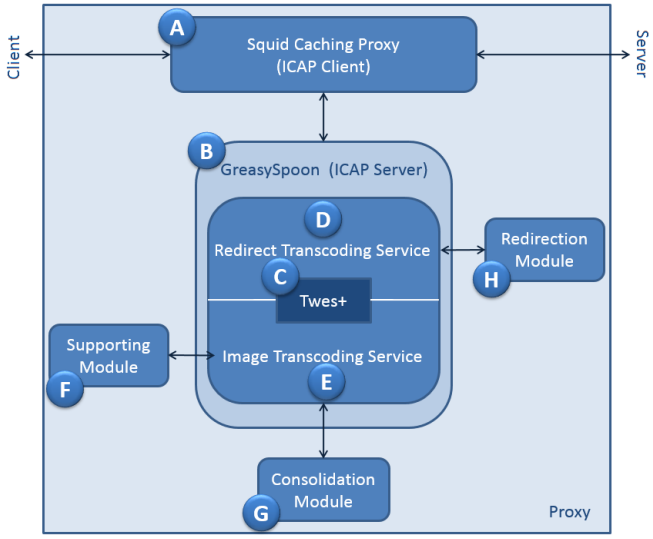

Fig. 1: Software architecture of Twes+.

ICAP-servers are suggested such as C-ICAP, ICAP-Server, POESIA, Traffic Spicer and GreasySpoon [14]. GreasySpoon (B) is adopted for Twes+ as it is an extensible open-source project [25]. Twes+ (C) resides on the GreasySpoon ICAP-server.

Twes+ mainly transcodes the response of a web page. When a web page is requested by the client and when the web server replies with a response, the data passes through the Squid Caching Proxy. Squid Caching Proxy forwards the response to the ICAP-server. Twes+ (C) resides on ICAP-Server (B) and contains two different adaptation services: (1) redirect transcoding service (D) and (2) image transcoding service (E). After the adaptation, the adapted response goes back from ICAP-Server to ICAP-Client and then from ICAP-Client to the client.

4.1 Redirect Transcoding Service

The goal of this service is to handle the temporary redirects. The content of an HTTP message is the status line, header and body. The status line shows the HTTP protocol, the status code and the explanation of the status code. The status code is a three-digit integer and the most significant digit gives the classification of this code [18]. 3xx status codes are the redirection status codes. There are several redirection status codes. From all the redirection, "Temporary Redirect 307" is the one that does not have any positive impact on the client-side [18]. This redirection indicates that the requested content address is changed to another temporarily. There is also another redirection called infinite redirect loop and usually occurs because of a missing file. According to the HTTP specification, the client side should detect the redirects which are more than five as this indicates an infinite loop [18]. Although this limit is adjustable on the browsers, infinite loops create significant amount of network traffic. Moreover, each redirection carries the overhead of a request/response but without the content of the page. More importantly, the battery of mobile devices drains while the device is connected and trying to resolve temporary redirects.

Normally, when there is a redirect, the server response with a new address for the content. The client browser extracts this new address and sends another request to this new address. Statistics show that the number of web pages that includes redirection increased from 61% to 78% between December 2010 and January 2016 [3]. With Twes+, our goal is to eliminate these temporary and infinite redirects to save energy. When Twes+ redirect transcoding service receives the response, it checks the status code. If the status code indicates that there is a redirect, it starts the adaptation. The redirect transcoding service extracts the new address from the location field of the header and forwards this address to the redirection module (H in Figure 1). There can be three cases: 1. there is no redirection; 2. the number of redirection is less than the limit which is recommended as five; 3. the number of redirection is more than the limit.

For case 1, redirect transcoding service forwards the responses without modification. For case 2, redirection module returns the new address to redirect transcoding service. The content of the web page is requested from this new address and forwarded to the client. For case 3, redirection module stops tracking after the fifth redirection. It returns negative to the redirect transcoding service. Redirect transcoding service updates the header and the body of the response as a warning message. This response is forwarded to the client. The results of the redirect transcoding module indicates that there is a reduction in the number of redirect round trips.

4.2 Image Transcoding Service

The image content of web pages are external resources except when they are included as data URI. External resources require request and response roundtrips to be added to a web page. The overall goal of image transcoding service is to combine images in order to reduce the number of requests and responses required to add images to web pages. In order to achieve this, this service includes a number of steps. The first request of the client passes through the proxy and reaches to the server. The server forwards the content and the HTML source is selected by Twes+ image transcoding service (E in Figure 1). If the HTML source code of the page does not include an image tag, this file is transferred to the client without any modification. On the other hand, if there is an image tag, the transcoding process starts. On image transcoding service, the image tags are tracked to derive the essential data of the image. Image tags require minimum two attributes which are the location <src> and an alternate text <alt>. Although there are other attributes, the dimension attributes are controlled by Twes+. In addition to the individual image tags, there are arranging structures like frame tags that can include image tags inside them. These tags are processed as well. To sum up, Twes+ extracts image tags and retrieve the source, the width and the height. The default value of the width and the height are set zero in case they are not defined. After extracting these three attributes, they are forwarded to the supporting module

(a) The First Image

(b) The Second Image

(c) The Third Image

Fig. 2: Images before concatenation

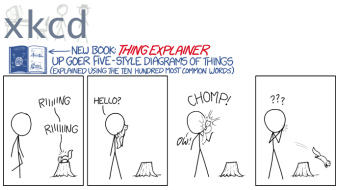

Fig. 3: Concatenated version of all images in Figure 2 [Width:679 pixels Height:371 pixels]

(F in Figure 1) of Twes+. This module downloads the image from the web server by using the given source attribute. The dimensions of the downloaded image are found out by ImageMagick[5]. The supporting module compares the required and the original dimensions. This two values might be equal or it might be the case that the second one is not specified. Therefore, we made an assumption that both the required and the original dimensions are the same. There is also a possibility that the required dimensions are defined but they are not the same as the original image. It means that the browser of the client needs to resize the image. However, this resizing can consume the battery of the client. Therefore, the resizing process is also done by the supporting module. When the required and the original dimensions are not equal, the supporting module applies the resizing to the original image. This way, the server-side does not change the original image while the client-side does not apply resizing.

In the current version of Twes+, the following image formats are supported PNG, JPEG/JPG, and GIF [3]. Since the goal is not to modify the look&feel of the web pages,

[5] http://www.imagemagick.org

the format of the images in the original web page is preserved. To be able to consolidate images, the name of the images on the web page are altered. The new name carries the name of the web page, the format of the image and a number based on the order of this format appears in the HTML. There are some possibilities that the supporting module returns negative. These cases are when the downloaded file is not an image or it is an animated GIF image. In order, not to have problems with the image formats that are not supported, only supported formats are considered in our module. Animated GIF images are not modified because they are the combination of several images and the dimensions do not have to be fixed. Responsive image technique is also applied for mobile devices by the supporting module.

After all the image tags are examined, the consolidation module (G in Figure 1) is called. Consolidating all the images into one image is not a practical idea. If a web page contains too many images, this makes the concatenated image size too large and sending a large image is not safe in case of any network failure. In Twes+ implementation, every ten images of the same format are concatenated. The limit is ten because WildCards are used during the consolidation process and WildCards works on base ten. According to these concerns, the consolidation module combines all the downloaded images. According to the order and the format of the image, the consolidation process is performed (see Figure 2 for individual images and Figure 3 for consolidated image from xkcd-Squirrelphone [6]). The consolidation module generates a decoded name with a keyword.

Images are concatenated vertically. The CSS is used to split images to access individual images. The cumulative height is used to determine the specific position of the images. Image tags are replaced with the new ones. The source is defined as a spacer.gif to hold a place in the layout for the image. The CSS styles with the location of the source are injected into the HTML file so an extra request/response is prevented. The modified HTML file is delivered to the client. The client-side requests consolidated image from the proxy. The *url_rewrite_program* feature of Squid Caching Proxy is used. When the requests are passing through the proxy, the keyword is tracked. If a request comes from the client that includes the keyword, the response is searched inside the proxy. Otherwise, the request is processed as it is.

5 Evaluation

In order to experimentally evaluate whether or not Twes+ can save energy, we conducted a study. This study aims to investigate the following two research questions:

1. *Redirect Transcoding Service:* The goal is to validate whether or not Twes+ redirection transcoding service can reduce the number of redirects and Twes+ can save energy. Therefore, the research question is *"Does Twes+ save energy by applying redirect transcoding to reduce the number of redirects?"*.
2. *Image Transcoding Service:* In order to validate whether or not Twes+ can save energy by reducing the number of image requests/ responses with image consolidation technique. Thus, the research question is *"Does Twes+ save energy by reducing the number of requests/ responses between client and server to retrieve images?"*.

[6] http://xkcd.com/1578/

To address these questions, we followed different evaluation strategies for each of them. Regarding the first question, we mainly used a controlled web page and introduced different cases for measurement. Regarding the second question, we used real web pages and we did measurements with original and modified pages by Twes+.

5.1 Materials

For the redirect transcoding service experiments, we used the xkcd-Squirrelphone[6] web page as a controlled medium for measurement. In fact, we could use any web page for measurement as the content here is not relevant but since this was used in the literature [9] for other purposes, we decided to reuse this web page. We controlled the number of redirects by mainly creating different versions of this page by injecting controlled number of redirects. We mainly created three cases:

- Case 1: The control web page can be reached without any redirect.
- Case 2: After five redirects, the recommended action to the client-side is to take an action as infinite redirection. According to this suggestion, by using same web page and five redirects this case is designed [18]. When the client requests the web page, after five redirects web page content can be accessible.
- Case 3: With the same web page, over the server infinite redirection loop is designed. In this case, the client does not receive the content of the web page. After the fifth redirection, Twes+ changes the content to a warning message about the infinite redirect to the client.

Regarding the image transcoding service, we used web pages selected from the top 100 list of Alexa. We had to download these pages and serve them locally in order to do measurements. Some web pages we could not use because of the following reasons. Some sites have duplicates in different countries and more than one of those can be included in the top 100 list, therefore we eliminated replicated ones and kept only the first version in the list. For example, Google has duplicates for different countries. We could not do measurements for sites that require logins as we deleted the caching of browsers so we also eliminated those from the list downloaded. This was also a concern from the security perspective. Finally, when we downloaded web pages locally, we checked to see if their look&feel is modified, if it did, we also eliminated those. As a final check, we also wanted to make sure that we test pages that can be modified by Twes+ based on the supported image formats so if the page did not include the image formats supported by Twes+, we also eliminated those. At the end, we did the measurements for 16 web pages which are listed in Table 3.

5.2 Equipment

In order to perform the measurements, we need to control the overall architecture. Therefore, we used three identical computers as a client, a proxy and a server (see Figure 1). Their processor is Intel Core i7, model: 4770. The base and the turbo frequencies are 3.4 GHz and 3.9 GHz, respectively. Installed memory is 16 GB. The brightness of the screen is 50% and unified over all the measurements. The switch between these three desktops, Cisco Catalyst 2960-X Series is used and its speed is 80 Gbps (Gigabits per second).

5.3 Tools

The evaluation of Twes+ is done by measuring some key metrics with and without Twes+. These metrics are the number of requests/ responses and the cumulative processor energy. Measurements are performed with Chrome Devtool and Intel Power Gadget. Devtool is used to measure the number of requests/ responses. There are other browser base tools but the network throttling feature of Devtool is the main reason we preferred it over others. The network system is connected via wire. To be able to simulate regular 3G network, Devtool network throttling is used. The 3G network is preferred because it is more common for mobile devices and also it has a wider coverage. Moreover, it is able to give the number of requests, the load duration of the web page and the network timeline of the elements during the page load [10].

Intel Power Gadget is used to measure the energy consumed by the processor. It estimates power data by reading the data of registers and energy counters of the processor. It supports Intel Core processors from 2nd Generation up to 6th Generation Intel Core processors. It is able to measure the current package power consumption and the limit of the package power. In addition to power consumption, it gives the frequency of the package and the base frequency of installed processor.

5.4 Methodology

To be able to know the reduction in the number of redirects and in the number of image requests/ responses, the overall number of requests/ responses is measured. To observe the impact of Twes+, the energy consumed by the processor is measured with and without Twes+. During the measurements, the cache and the history of all the components (client/server/proxy) are cleared. Entire system works locally to prevent the influence of the Internet traffic. The system is tracked by using Intel Power Gadget and when there are no changes, the measurements are started and the measurements are repeated minimum of three times to ensure that there is no external effecting factors. While measuring the number of requests/ responses, on the client only Chrome Devtool is left active. While measuring the power and the energy, on the client only Intel Power Gadget and the browser are left active. Although there is no limitation for the duration of the test for the number of requests/ responses measurements, the duration of the power and the energy measurements is set as 30 seconds so the impact of starting and stopping the tools to measure is eliminated.

6 Results

We present our results based on the research questions introduced above.

"Does Twes+ save energy by applying redirect transcoding to reduce the number of redirects?"

For each of the three cases, the number of requests/responses, the load duration and the cumulative processor energy are measured with and without Twes+. These measurements are done when the web page is requested by the client. Table 2 shows the overall results. The required number of requests/ responses for the control web page is seven. For case 1, the number of requests and responses are the same because there is no redirect. For case 2, the content is loaded after five redirects. Without Twes+, the number

of requests/ responses includes these redirects. With Twes+ it remains same as case 1. For case 3, the content cannot be reached because of the infinite number of redirects. The browser is prevented from interfering the redirects so the redirects never stop without Twes+. On the other hand, with Twes+ redirects are stopped. The client received a warning message from Twes+. By reducing the number of requests/ responses, a reduction in the cumulative processor energy is observed for three cases. For case 1, this was dropped from 63.55 to 61.13mWh, which means 3.8% decreased was observed on the original case. In case 2, this was dropped from 80.39 to 61.7, therefore 23.3% decrease was observed on the original case. In case 3, this was dropped from 76.02 to 72.5 which means 4.6% decrease observed (see Table 2). Finally, regarding the load duration, in case 1, we did not observe any difference (2.06 vs. 2.04) and in case 3, we could not measure for without Twes+ case and with Twes+, the loading finished in 0.576s. In case 2, we observed a small increase in the load duration because of the process over Twes+. This process is detecting redirects, extracting the new location, tracking the remaining redirects and retrieving the content from the web server.

Table 2: Total number of requests/responses and cumulative processor energy [w/:with, w/o:without]

	Number of requests/ responses		**Load duration (s)**		**Cumulative processor energy (mWh)**	
	w/o Twes+	w/ Twes+	w/o Twes+	w/ Twes+	w/o Twes+	w/ Twes+
Case 1	7	7	2.06	2.04	63.55	61.13
Case 2	12	7	1.73	2.08	80.39	61.70
Case 3	69	2	*	0.576	76.02	72.50

"Does Twes+ save energy by reducing the number of requests/ responses between client and server?"
For each web page used in the experiment, the number of image requests/ responses, the load duration and the cumulative processor energy are measured with and without Twes+. The mean, median and the standard deviation of these results are also calculated. Table 3 shows that with Twes+, although Twes+ increases the page load duration, the number of total image requests/ responses and the cumulative processor energy are lower than without Twes+. To determine whether or not the difference between with and without Twes+ is significant, the Paired-Dependent T-Test or its non-parametric alternative Wilcoxon Signed Rank Test is applied since same web pages are measured with and without Twes+ and the design is repeated measures. The normality of the difference between the results with and without Twes+ are checked by using ShapiroWilk (SW) test to decide either Paired-Dependent T-Test or its non-parametric alternative Wilcoxon Signed Rank Test is needed to be applied.

The result of the statistical tests presented in Table 4 shows that there is a significant difference between with and without Twes+ for the number of image requests/ responses, the load duration and the cumulative energy. Interpretation of these results is that although there is a minor increase in the load duration of the web pages, there is a statistically significant reduction in the number of image requests/ responses and this leads to energy saving with Twes+. To find the magnitude of effect, r value is calculated for the number of image requests/ responses and the results indicate a large effect size. Eta Squared statistics applied for the load duration and the cumulative processor energy and the results indicate respectively moderate and large effect size.

Table 3: Number of image requests/responses, the load duration(finish) and the cumulative processor energy [w/:with, w/o:without, SD: Standard Deviation]

		Number of image requests/ responses		**Load Duration(s)**		**Cumulative processor energy (mWh)**	
Rank	Name	w/o Twes+	w/ Twes+	w/o Twes+	w/ Twes+	w/o Twes+	w/ Twes+
17	Ebay.com	165	28	25.57	30.15	76.18	66.75
28	Microsoft.com	30	5	17.92	20.6	93.72	89.02
29	AliExpress.com	192	31	84	96	76.23	77.03
33	Ask.com	6	5	3.4	3.68	67.22	63.62
43	Imgur.com	70	13	9.19	10.27	78.23	75.58
47	Imdb.com	61	30	10.46	12.49	73.22	67.58
50	Fc2.com	11	10	4.08	8.5	79.33	74.24
55	Stackoverflow.com	9	8	4.08	4.23	78.04	75.94
61	Odnoklassniki	56	18	21.43	23.4	77.34	75.37
72	Booking.com	37	26	7.52	8.36	73.74	63.96
75	NicoVideo.jp	43	16	7.43	21.61	80.65	72.74
76	Flipkart.com	149	29	21.48	28.95	83.38	72.18
85	BBC.co.uk	59	14	17.03	16.43	77.14	66.03
95	DailyMotion.com	45	9	10.93	12.29	75.43	67.32
96	Wikia.com	33	14	28.72	33.25	66.35	65.63
97	ChinaDaily.com.cn	108	87	36.39	29.65	89.29	80.80
	Mean	**67.13**	**21.44**	**19.35**	**22.49**	**77.84**	**72.11**
	Median	**50.50**	**15.00**	**13.98**	**18.52**	**77.24**	**72.46**
	SD	**56.90**	**19.69**	**19.78**	**21.83**	**6.92**	**6.92**

Table 4: Statistical test differences with and without Twes+

	Number of image requests/ responses	**Load duration (s)**	**Cumulative Processor Energy (mWh)**
Test	Wilcoxon	T-Test	T-Test
N	16	16	16
df		15	15
p	$<.0005$	.023	$<.0005$
t or z	-3.5198	2.532	-6.019
Eta Square or r	.6221	.3	0.71

7 Discussion

Twes+ is a novel system that combines two different transcoding techniques to improve the energy consumption on mobile devices while browsing the web. These techniques are previously recommended as guidelines for developers but they are not implemented as Twes+ does. To our knowledge Twes+ is the only system that implements these techniques on the proxy. The experiments presented here also demonstrate that Twes+ saves energy, in particular it reduces the number of requests/responses between client and server and consequently it saves processor energy. Therefore, the overall contribution of Twes+ is to scientifically show that these suggested techniques as guidelines do save energy. Here we first discuss the results based on the two services included in the Twes+, and then we discuss the impact of energy saving on the battery life.

Twes+ redirect transcoding service is experimentally evaluated with three cases. The results show that the cumulative processor energy is almost the same with and without Twes+ while there is no redirect. On the other hand, Twes+ can eliminate the finite and infinite redirects of the web page and it leads to energy saving on the client-side. This reduction is because of Twes+ is removing the unnecessary requests/responses.

Every redirect carries the overhead of the request and response without the content of the web page. Even though the browsers have a limit to prevent the infinite redirects, the round trip still occurs on the client-side. With Twes+, all the temporary finite or infinite redirects are removed.

To estimate the impact of this energy saving on the battery life of mobile devices, we made the following assumptions about the battery of the client device. The battery specification of a laptop (TOSHIBA Satellite P70) has the capacity of 4400 mAh and the voltage is 10.8 V. We estimated the battery life improvement for two cases: the cumulative processor energy is equal to the entire system energy consumption and we denote this as 100% and 70% of the entire system (in this case, we assume that 30% is used by other processes). Table 5 shows the battery life improvement estimations for the three cases of this transcoding service. The battery life of mobile devices varies from the tasks done over the device. With Twes+, this duration is extended for browsing by eliminating the redirects. If we assume that the cumulative processor energy is equal to the entire system energy, in case 2 the battery of the mobile device can stay 90.5 mins longer. In real system, there are other components that consume energy such as screen, ethernet, etc. Therefore, if we assume that the processor energy is the 70% of the entire system energy consumption, in case 2 the battery life of the mobile device can stay 48.6 mins longer.

Table 5: Battery life analysis for redirect transcoding service results

	Power Reduction (%)	**Battery Life Extension (100%) (mins)**	**Battery Life Extension (70%) (mins)**
Case 1	2.83	10.9	7.4
Case 2	23.23	90.5	48.6
Case 3	4.33	14.3	9.6

We used a different strategy to evaluate Twes+ image transcoding service. We can say that we tested it in the wild. We picked actual web pages from Alexa 100. Instead we could also have a controlled page and do the measurements based on that, however we wanted to make sure we demonstrate the impact of Twes+ on real web pages. For some web pages as can be seen in Table 3, the reduction in the cumulative processor energy is low. This is mainly because the number of images is low. For example, StackOverflow page includes nine images requests/responses and with Twes+ this number reduces to eight. The number of transcoded images in these web pages is low because some of the images are inline images or background images or requested by the CSS files or the format of the image is not supported. However, in most pages, we observe 50% reduction in the number of requests/responses and also reduction in the cumulative processor energy. For example, the number of image requests/responses for Flipkart reduced from 149 to 29 images and the cumulative processor energy reduced from 83.38 mWh to 72.18 mWh (i.e. 13.4%).

To see the impact of this energy saving on the battery life of mobile devices, we followed the same assumption that the client works on battery with the specification given above. With the same ratios between the entire system energy and the processor energy consumption, Twes+ is able to extend the battery life (see Table 6). We see 7.6% power reduction and the mean value of the battery life extension is 26.04 minutes in case the cumulative processor energy is equal to the 100% of the entire system energy and 16.20

minutes in case the cumulative processor energy is equal to the 70%of the entire system energy. For example, although the number of images are low in StackOverflow page, with Twes+ there is 4.7% power reduction and 14.90 mins (the cumulative processor energy consumption is equal to the entire system) or 9.94 mins (the cumulative processor energy consumption is equal to 70% of the entire system) extension can be achieved. For Flipkart page, the battery life extension is 53.23 mins for 100% case and 31.42 mins for 70% case, and observed power reduction is 15.7%.

Table 6: Battery life analysis for image transcoding service results

		Power Reduction (%)	**Battery Life Extension (100%) (mins)**	**Battery Life Extension (70%) (mins)**
17	Ebay	11.4	40.27	24.99
28	Microsoft	7.1	19.49	12.67
29	AliExpress	0.9	2.79	1.93
33	Ask	4.9	18.01	11.99
43	Imgur	0.9	2.66	1.85
47	Imdb	7.4	26.11	16.92
50	Fc2	5.0	15.98	10.62
55	Stackoverflow	4.7	14.90	9.94
61	Odnoklassniki	3.5	11.16	7.54
72	Booking	17.2	65.97	38.24
75	NicoVideo	9.2	30.08	19.13
76	Flipkart	15.7	53.23	31.42
85	BBC	14.8	53.88	32.13
95	DailyMotion	9.8	34.37	21.70
96	Wikia	1.5	5.54	3.82
97	ChinaDaily	7.5	22.21	14.39
	Mean	**7.6**	**26.04**	**16.20**
	Median	**7.3**	**20.85**	**13.53**
	SD	**5.14**	**19.10**	**11.04**

Besides these estimations, we also looked at the impact of image transcoding service of Twes+ on some popular mobile devices. The effect of 7% reduction is investigated for Samsung Galaxy S4-5-6-7 and iPhone 5-6s by using their battery specifications. On average, Internet usage time is estimated to be extended by 40.6 mins(3G) and 48.3 mins(Wi-Fi), and specifically for Samsung Galaxy S4 (33.6 - 3G, 42 - Wifi), S5 (46.2 - 3G, 50.4 - Wifi), S6 (42 - 3G, 50.4 - Wifi), S7 (46.2 - 3G, 58.8 - Wifi) and iPhone 5 (33.6 - 3G, 42 - Wifi) and 6s (42 - 3G, 46.2 - Wifi).

In overall, our results are promising but the current Twes+ implementation has some limitations that can be improved in the future. Firstly, in its current implementation of Twes+, the CSS properties for the image dimensions are not taken into account. This is mainly because processing external files can consume energy and delay the loading time of resources. Therefore, further work is required to investigate how to handle external CSS files without increasing the loading time of resources. Secondly, Twes+ only supports popular image formats and it does not consolidate animated GIF files because they can be the combination of several images with different dimensions. Finally, Twes+ also has some limitations with respect to dynamic web pages. When a web page modifies images with JavaScript, it creates several variations into the image

tags. These cases are not currently handled by Twes+. However, even though Twes+ has these limitations, we believe by addressing these, Twes+ would have even more impact on the energy saving and battery life.

Finally, in this study, the evaluation of Twes+ is done with 16 web pages. Although our findings show the status with real websites, a further study can be conducted with more web pages. Due to space limitations, in this paper we only reported our measurements with a particular device but more measurements with other mobile devices can be found in [45]. In this paper, we also mainly looked into the energy saving on the client. However, with caching Twes+ can also contribute to green computing and especially to sustainability. In particular, if pages that are used commonly can be transcoded once by the proxy and can be accessed by many, this could potentially affect the sustainability and energy consumption by mass.

8 Conclusions

This paper presents our proxy-based system called Twes+. Twes+ transcodes and adapts the source code of web pages to save energy. Twes+ currently has two transcoding techniques: redirect transcoding and image transcoding. Redirect transcoding aims to improve temporary redirects, and image transcoding aims to reduce the number of image requests to the server by consolidating images. This paper also presents our systematic evaluation of these two techniques. The results of our studies show that Twes+ saves energy on the client side by reducing the number of redirect round trips and the number of image requests/ responses without modifying the look&feel of the web pages. Twes+ was able to eliminate the temporary redirect round trips and in cumulative processor energy for infinite redirects 4.6% reduction is observed with Twes+. With image transcoding service, on average 7% reduction in cumulative processor energy is observed by reducing the number of request/response round trips between the client and the server. It is also observed that this percentage is more than 10% with pages that include many images.

We also investigated the battery improvements with Twes+ and our estimations shows that with the image transcoding service of Twes+, in average Twes+ provides 7.6% power reduction and if we consider specific pages, for example Ebay, we can say that Twes+ provides 40.27 minutes battery improvements when the cumulative processor energy is equal to the entire system energy consumption and 24.99 minutes improvement when the cumulative processor energy is equal to the 70% of the system energy consumption (assuming that 30% is used by other processes, and only Ebay is processed). This also gives us on average 40 to 60 minutes of battery life extension on typical mobile devices. With redirection service of Twes+, we observe the most improvement in case 2 where redirection module returns the new address to redirect transcoding service and the content of the web page is requested from this new address and forwarded to the client. Of course, these are our estimations on a specific configuration but we believe Twes+ is a promising system in saving energy while users are browsing the web and it can be further extended and improved by adding new transcoding services to Twes+.

References

1. H. Ahmadi and J. Kong. Efficient web browsing on small screens. In *Proceedings of AVI '08*, pages 23–30, USA, 2008.
2. M. Ali, Y. Alex, and A. von Jouanne. Integration of thermoelectrics and photovoltaics as auxiliary power sources in mobile computing applications. *Journal of Power Sources*, 177:239–246, 2008.
3. H. Archive. Trends and statistics, September 2015. http://httparchive.org/.
4. C. Asakawa and H. Takagi. Transcoding. In Y. Yesilada and S. Harper, editors, *Web Accessibility*, pages 231–260. Springer, 2008.
5. K. C. Barr and K. Asanović. Energy-aware lossless data compression. *ACM TOCS*, 24(3):250–291, 2006.
6. F. Ben Abdesslem, A. Phillips, and T. Henderson. Less is more: energy-efficient mobile sensing with senseless. In *Proceedings of MobiHeld*, pages 61–62. ACM, 2009.
7. J. Chen, B. Zhou, J. Shi, H. Zhang, and Q. Fengwu. Function-based object model towards website adaptation. In *Proceedings of WWW '01*, pages 587–596, USA, 2001.
8. C.-H. Chi, J. Deng, and Y.-H. Lim. Compression proxy server: Design and implementation. In *USENIX Symposium on Internet Technologies and Systems*, 1999.
9. P. Cobbaut. Introduction to squid, http://linux-training.be/networking/ch09.html, May 2015.
10. Chrome DevTools overview, October 2014. https://developer.chrome.com/devtools.
11. The chromium project - SPDY: An experimental protocol for faster web, September 2014. http://www.chromium.org/spdy/spdy-whitepaper.
12. Pagespeed module - sprite images, August 2014. http://tinyurl.com/juc537v.
13. Data compression proxy, January 2015. https://developer.chrome.com/multidevice/data-compression.
14. R. C. Duane Wessels, Henrik Nordstrom and A. Jeffries. Squid: optimising web delivery, Squid-cache.org, 2013.
15. M. Etoh, T. Ohya, and Y. Nakayama. Energy consumption issues on mobile network systems. In *Proceedings of SAINT '08*, pages 365–368, USA, 2008. IEEE Computer Society.
16. T. Everts. Rules for mobile performance optimization. *Queue*, 11(6):40:40–40:51, June 2013.
17. L. Fainberg, O. Ehrlich, G. Shai, O. Gadish, A. Dobo, and O. Berger. Systems and methods for acceleration and optimization of web pages access by changing the order of resource loading, Aug. 2 2010. US Patent App. 12/848,559.
18. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc2616 hypertext transfer protocol http/1.1, 1999.
19. M. Firtman. *Programming the Mobile Web*, volume 1. OReilly Media, Inc., second edition, March 2010.
20. B. Frain. *Responsive Web Design with HTML5 and CSS3*. Packt Publishing, 2012.
21. S. Gochman, A. Mendelson, A. Naveh, and E. Rotem. Introduction to intel core duo processor architecture. *Intel Technology Journal*, 10(2), 2006.
22. Google. Chrome accessibility, November 2015. http://tinyurl.com/pgyrgf2.
23. R. Kothiyal, V. Tarasov, P. Sehgal, and E. Zadok. Energy and performance evaluation of lossless file data compression on server systems. In *Proceedings of SYSTOR 2009*, pages 4:1–4:12, USA, 2009.
24. D. Kumar. Image sprites - how to merge multiple images, and how to split them, June 2012. http://tinyurl.com/j56g6h8.
25. L3WS. Greasyspoon- open-source ICAP server factory for core network services, 2015. http://greasyspoon.sourceforge.net/.
26. P. P. Y. Lai. Efficient and effective information finding on small screen devices. In *Proceedings of W4A '13*, pages 4:1–4:10, USA, 2013.

27. Y. Liu and L. Guo. An empirical study of video messaging services on smartphones. In *Proceedings NOSSDAV '14*, pages 79:79–79:84, USA, 2014.
28. K. Matsudaira. Making the mobile web faster. *Queue*, 11(1):40:40–40:48, Jan. 2013.
29. P. McLachlan. On mobile, data uris are 6x slower than source linking (new research), July 2013. http://www.mobify.com/blog/data-uris-are-slow-on-mobile/.
30. Mobify. Image resizing with mobify.js, 2013. https://www.mobify.com/mobifyjs/docs/image-resizing/.
31. T. Narendran, A. Gaurav, N. Angela, B. Dan, and S. J. P. Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of WWW'12*, pages 41–50. ACM, 2012.
32. Opera. Faster browsing on slow networks with off-road mode, December 2015. http://help.opera.com/opera/Windows/1116/en/fasterBrowsing.html.
33. K. Paul and T. K. Kundu. Android on mobile devices: An energy perspective. In *Proceedings of CIT '10*, pages 2421–2426, USA, 2010. IEEE Computer Society.
34. K. Pentikousis. In search of energy-efficient mobile networking. *IEEE Communications Magazine*, 48:95–103, 2010.
35. R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma. Learning block importance models for web pages. In *Proceedings of WWW '04*, pages 203–211, USA, 2004.
36. H. Takagi, C. Asakawa, K. Fukuda, and J. Maeda. Site-wide annotation: Reconstructing existing pages to be accessible. In *Proceedings of ASSETS'02*, pages 81–88, USA, 2002.
37. A. T. S. Team. Apache traffic server, January 2015. http://tinyurl.com/pv63dxj.
38. M. team. Introducing jazzcat: A javascript and css concatenation service, August 2012. http://tinyurl.com/jf2v82u.
39. L. Thoba and T. Mamello. A transcoding proxy server for mobile web browsing. In *The Southern Africa Telecommunication Networks and Applications Conference*, volume 2011.
40. E. Ugo, I. Gennaro, M. Delfina, and S. Vittorio. Personalizable edge services for web accessibility. *UAIS*, 6:285–306, 2007.
41. Yahoo. Best practices for speeding up your web site, January 2015. https://developer.yahoo.com/performance/rules.html.
42. Y. Yesilada, C. Jay, R. Stevens, and S. Harper. Validating the use and role of visual elements of web pages in navigation with an eye-tracking study. In *Proceedings of WWW'08*, Beijing, China, 2008.
43. X. Yin and W. S. Lee. Using link analysis to improve layout on mobile devices. In *Proceedings of WWW '04*, pages 338–344, USA, 2004.
44. N. C. Zakas. The evolution of web development for mobile devices. *Queue*, 11(2):30:30–30:39, Feb. 2013.
45. E. Köksal Ahmed: Transcoding Web Pages For Energy Saving On the Client-Side. Middle East Technical University Northern Cyprus Campus, 2016.
46. Apple: Iphone7. http://www.apple.com/sg/iphone-7/specs/.
47. R. Ana, C. Mateos and A. Zunino. Improving scientific application execution on android mobile devices via code refactorings. Software: Practice and Experience (2016).