Received: 1 May 2021 | Revised: 27 September 2021 | Accepted: 8 October 2021

DOI: 10.1002/spe.3046

RESEARCH ARTICLE

WILEY

Transcoding web pages via stylesheets and scripts for saving energy on the client

Hüseyin Ünlü[1,2] | **Yeliz Yesilada**[1]

[1]Computer Engineering Program, Middle East Technical University Northern Cyprus Campus, Mersin, Turkey

[2]Computer Engineering Department, Izmir Institute of Technology, Izmir, Turkey

Correspondence
Hüseyin Ünlü, Computer Engineering Department, Izmir Institute of Technology, Izmir, Turkey.
Email: huseyinunlu@iyte.edu.tr

Abstract

Mobile devices and accessing the web have become essential in our daily lives. However, their limitations in terms of both hardware such as the battery, and software capabilities can affect the user experience such as battery drain. There are some best practices for the web page design that are shown to affect the downloading time of web pages. In this study, we report our experience in applying these practices to see their effect on energy saving. We propose two techniques: (1) concatenating external script and stylesheet files and (2) minifying external script and stylesheets that can be used to transcode web pages to improve energy consumption on the client-side and therefore improve the battery life. We present our experimental architecture, implementation, and a systematic evaluation of these two techniques. The evaluation results show that the proposed techniques can achieve approximately 12% processor energy-saving and 4% power saving in two different client types, 13% improvement in a typical laptop battery life, and 4% improvement in a typical mobile phone battery life.

KEYWORDS

browsing, energy-saving, software implementation, software tools, transcoding, web

1 | INTRODUCTION

In today's world, mobile devices have an important role in our lives, and people prefer to use their mobile devices instead of desktop computers.[1,2] Indeed, the number of mobile devices operating worldwide is more than the population now, and according to a forecast, it is expected to reach 17.72 billion by 2024.[3] When we look at the web access, the trend is also the same—the mobile web overtook desktop web.[4] In parallel to this trend, web pages have also become more complex, and the technologies used also changed. Nowadays, web pages include more images and videos in high resolution and thus, their size has also increased. According to some studies, while a typical web page size was 929 KB in 2011, this page was 3034 KB in 2017, and the expectation is that the average size will increase similarly in the future.[5] However, mobile devices have some limitations such as limited screen size, the limited bandwidth of the connection, small device memory, limited processing power, and limited battery.[6,7] These limitations can easily affect users' experience (UX).[8,9] Mobile phone hardware and software try to reduce energy consumption as much as possible letting the phone have longer battery life or use a small battery, but if the web interaction is more demanding on the device the battery life will be much shorter.[10,11] This cannot only affect UX and in the long run, can affect the lifespan of batteries and sustainability of the environment.[12]

In an experimental study, the energy consumption of the web page components including source code written in HTML, scripts, stylesheets, images is measured.[13] This study shows that the energy consumption of images, JavaScript

(JS), and cascading style sheets (CSS) is high. Therefore, for saving energy on web page access, these elements should be optimized. Furthermore, even though there are different guidelines for performance improvement in web pages, most of these are not empirically confirmed to save energy while improving performance[6,7,14–18] (see Section 2).

This article reports our experience which aims to achieve energy saving during web access on the client-side via transcoding web pages without modifying their look and feel and without adding extra load to the client or the server.[19] In this article, we mainly focus on two transcoding techniques: (1) concatenating external JS and CSS files to reduce the HTTP request and (2) minification of external CSS and JS files to reduce the size of data transferred (see Section 3). To demonstrate the effect of these techniques, here we present an architecture and also an experimental prototype. We use this architecture to measure the energy consumed with and without transcoding on real web pages. The implemented services are evaluated over both a desktop and a mobile client with a sample set of web pages chosen from Alexa Top 100* (see Section 4). These web pages are stored and tested locally to eliminate the external factors that can affect the measurements. Results show that our services achieved statistically significant energy savings. Our three services (concatenation, minification, and concatenation + minification) reduced the cumulative processor energy (mWh) by 12.26%, 7.57%, and 9.87% over the desktop client and reduced the average power (mW) by 3.94%, 2.50%, and 3.34% over the mobile client in the average of our evaluation set respectively (see Section 5). We also analyzed the impact of this improvement on a typical battery, and the result shows that our three services can achieve 13%, 7%, and 9% improvement in a typical laptop battery life and 4.1%, 2.5%, and 3.5% improvement in a typical mobile phone battery life on the client-side, respectively (see Section 6). On the other hand, to provide overall energy saving, the energy consumption of the proxy can also be considered. For this purpose, we developed a model between the energy consumption of the proxy server and the energy-saving on the client-side (see Section 6.1). The main contributions of this work can be summarized as follows (see Section 7):

(1) With this work, we *empirically* provide evidence that applying the two guidelines proposed previously for improving performance (i.e., concatenating external files and minimizing the size of external files) in web interaction, providing significant energy saving;
(2) Transcoding is not a novel approach but it is the first time that is considered for saving energy on the web. The proposed proxy-based approach does not put extra load on the client, or the server as all of the transcoding is done on the proxy. Even though this means energy consumption is moved to the proxy, the client does not need to install extra software and the developer does not need to know and apply the energy-saving guidelines. Therefore, with our approach we also remove the burden on developers to consider energy issues;
(3) Finally, our experimental work shows by reducing the energy consumption of the processor while accessing the web, the battery life can be increased, and thus it will contribute to sustainability when we consider the harmful effects of the battery waste in the environment.[20] The techniques are implemented on a proxy server to provide energy saving on the client-side. However, to provide overall energy saving, the energy consumption of the proxy can also be considered. In this study, we also developed a model between the energy consumption of the proxy server and the energy-saving on the client-side. Thus, we discuss how we contribute to sustainability.

2 | RELATED WORK

When the web architecture is considered, the energy consumption in web interaction can be reduced to three levels: hardware, network, and software. At the hardware level, energy-related studies focus on different areas such as reducing the energy consumption of the processor,[21] embedding renewable energy resources into mobile devices,[22] analyzing the role of the processor in the energy consumption of mobile web browser,[23] and reducing the energy consumption of the screen.[24] At the network level, there are different studies about energy consumption such as analyzing the energy consumption of the mobile network,[25] the impact of different communication technologies on mobile devices,[26–28] using localization[29] and caching[30] to improve energy saving in mobile networks. Moreover, the type of network technology is shown to have an impact on the energy consumption of mobile devices.[27,28] At the software level, different studies handle the software to reduce energy consumption. These studies handle the effect of the operating system and the programming language used,[31,32] propose code refactoring,[33,34] propose code obfuscation,[35,36] and introduce new communication protocols such as SPDY†. There are also some studies related to the energy consumption of websites[13] and the effect of different

*https://www.alexa.com.
†https://www.chromium.org/spdy/spdy-whitepaper

JS libraries on energy consumption.[37] Energy efficiency is also a very widely researched topic for the browsers.[38–41] Most browsers have settings to choose an energy-saving mode or plug-ins with different techniques. A recent analysis of the energy efficiency characteristics of web browsers can be found in Reference 42, which also explains these features in detail. Browser vendors are also doing regular energy efficiency tests and comparisons.[38–40] Middleware approaches are also used for energy management in different levels such as hardware, software, and operating system; however, its impact on user experience should be reduced by being invisible to the user and the system.[43]

Besides these studies, there have also been studies on transforming web pages into alternative formats to improve mobile user experiences which are known as transcoding. Transcoding is the process of transforming web pages into alternative forms to have improvements for different purposes such as providing web accessibility for disabled people, increasing the performance of web interaction, or improving the UX for web users.[19] Considering the web architecture, it can be done on the client, server, or proxy (which sits in between client and server). In client-side transcoding, adaptation is conducted after the web page is retrieved; therefore it is done on the client. However, since the transcoding is done on the device itself, the device should be powerful. When we consider the limitations of mobile devices (e.g., limited memory, processor), the client-side transcoding has some significant drawbacks.[44] In server-side transcoding, the adaptation is done on the server-side before it is served to the client, and therefore, the process is invisible to the client since all the process is done at the server-side.[19] However, one needs to be the owner of the content to do the transcoding. Compared with these approaches, the proxy is a specialized server that sits between the client and the webserver, and the transcoding can be done on the proxy. The overall transcoding process in the proxy is transparent to the client after the address of the transcoding server is set. When the client requests a web page, the client sends the request to the proxy, and then the proxy sends a request to the server, and the content is sent to the proxy. After transcoding, the web page content is sent to the client.[44] Proxy-side transcoding does not put any extra work on the client- or server-side but the connection speed between the client, proxy, and server could be the drawback.[44] When we look at the techniques used for transcoding, many techniques are utilized such as text magnification,[19] color scheme changes,[19,45] alternative text insertion,[19,45] page rearrangement,[19,46] simplification,[19,47,48] summarization,[19,49–51] image consolidation,[6,52,53] responsive image,[6,7,54] data compression,[6,55,56] reducing the number of redirects,[14,18,53] expiration header,[6] dynamic adaptation.[57] Although many of these techniques can achieve energy saving, only a very few techniques (such as alternative text insertion, simplification, reducing the number of redirects, expiration header, and dynamic adaptation) are validated for energy efficiency.

In the literature, there are also different guidelines for performance improvement and energy saving.[6,7,14–18] Although most of these guidelines do not include specific suggestions about energy, some of them may achieve energy saving when we consider the main ideas behind reducing the energy consumption of a website: (1) to minimize HTTP requests and (2) to reduce the size of the components.[6,18] When we look at the guidelines for improving the speed of access to websites, we can see that these two principles are behind most of these guidelines. These guidelines are mainly related to caching, number of images, number of requests, redirects, scripts, and stylesheets. Unfortunately, few of these recommendations are validated, and some of them are implemented as tools that work on different sides such as client, proxy, or server. In an experimental study, the energy consumption of the web page components including source code written in HTML, scripts, stylesheets, images are measured,[13] and it is shown that scripts and stylesheets consume the most energy. Therefore, here we focus on CSS and JS related guidelines summarized in Table 1. As can be seen from this table, most of these techniques have not been empirically evaluated in terms of energy-saving or performance. Similarly, the number of research studies that attempt to reduce the energy consumption of websites is also very low. Uzair et al.[58] introduced a machine learning-based predictive modeling technique is introduced that automatically down-samples the web page for energy-efficient rendering. Their approach showed a minimum reduction of 24.6% energy consumption of the overall system by resampling images on the server-side. Their approach sacrifices the quality of service while maintaining the user experience. Zhang et al.[59] described an optimization framework for offloading policy to reduce the energy consumption on both the mobile device and the cloud considering the time delay for video transcoding. They proposed an online algorithm to dispatch transcoding tasks to service engines to reduce energy consumption. Their result shows that the proposed algorithm outperforms the other three alternative algorithms with lower time energy consumption and queue length. Lastly, Koksal et al.[53] attempted to reduce the energy consumption of websites based on proxy side transcoding. This system achieves battery life improvement without modifying the look&feel of the web page. With this work, two techniques are explored: (1) combining images with CSS sprites and (2) reducing the number of redirects. Their results show that redirect service can achieve 4.6% and image adaptation can achieve a 7% reduction in cumulative processor energy, which is equal to between 40 and 60 min saving in a battery of a mobile device.

TABLE 1 CSS and JS techniques

			Location				
Technique	**Implementation**	**Reference**	**Server**	**Proxy**	**Client**	**M.**	**B.**
Avoid CSS @imports	Google PageSpeed Module	60	✓			?	?
Avoid document.write	Google Lighthouse Tool	61	✓			?	?
Combine images with CSS sprites	Google PageSpeed Module	62	✓			?	?
	Twes+	53		✓		x	*
Combine external CSS and JS	Google Apache Module	18	✓			?	?
	Google Closure Compiler	63	✓			?	?
	Google PageSpeed Module	64	✓			?	?
	Mobify Jazzcat	65	✓		✓	?	?
	IBM Worklight Studio	66	✓			?	?
	Grunt contrib-concat	67	✓			?	?
Minify JS and CSS	YUICompressor	68	✓			?	?
	JSCompress	69	✓			?	?
	JSMin	70	✓			?	?
	Minify	71	✓			?	?
	CSSNano	72	✓			?	?
	csso	73	✓			?	?
	Microsoft Ajax Minifier	74	✓			?	?
Minification and concatenation	JS & CSS Script Optimizer	75	✓			?	?
	Merge+Minify+Refresh	76	✓			?	?
	Dependency Minification	77	✓			?	?
	Minqueue	78	✓			?	?
	Combine and Minify	79	✓			?	?
	Granule	80	✓			?	?
	Jawr	81	✓			?	?
	The Asset Pipeline	82	✓			?	?
	Codekit	83	✓			?	?
	Prepros	84	✓			?	?
Remove unused CSS	Firefox Dust-me selectors	85	✓			?	?
	Chrome Developer Tools	86	✓			?	?
	Gtmetrix	87	✓			?	?
	unused-css.com	88	✓			?	?
	mincss	89	✓			?	?
	uncss	90	✓			?	?
	CSS remove and combine	91	✓			?	?
	Who killed my battery?	13	✓			?	*
Order scripts	JS and CSS Script Optimizer	76	✓			?	?
Make unique	Remove duplicates: uniq.js	92	✓			?	?
Efficient JS	Who killed my battery?	13	✓			?	*

Note: *, shown scientifically; x, irrelevant; ?, not shown scientifically; M., modify the look and feel; B., battery life improvement.

In brief, there has been extensive work in improving energy saving at network communication, at very low-level hardware components, and the main software components used. There has also been a broad set of guidelines proposed for developers to save energy in the developed web pages. However, not many of those are empirically evaluated. Similarly, limited research studies (such as resampling images on the server-side, offloading the computation on mobile devices to a mobile cloud system, and transcoding websites over a proxy by combining images and reducing the number of redirects) attempt to reduce the energy consumption of websites by transcoding. However, none of them attempted to save energy by transcoding over the proxy focusing on JS and CSS-related guidelines. We explain our focus, method, and evaluations in the following sections.

3 | METHODOLOGY AND ARCHITECTURE

We propose to transcode web pages on a proxy server based on the following two techniques:

(1) **Combining external CSS and JS**: Given the web architecture each external resource means an extra HTTP request. Therefore, the more external files mean more HTTP requests, and to reduce this, external files can be concatenated.[6,15,18] It is proposed that an ideal web page should have one external JS and one CSS file.[18] To ensure this, concatenation is done at the build time, but there are some services to make this process at the runtime using the content delivery network (CDN). For example, Google Apache Module concatenates files dynamically at run-time. It uses a special URL format to download multiple files using a single request.[18] There are also various tools to concatenate external files. However, most of them work on the server-side (see Table 1). Thus, only the owners of websites can use these tools to concatenate external files. Also, the primary objective of these tools is to reduce the loading time of the websites and to reduce energy is typically not considered. Therefore, in our work, we propose to apply this technique at the proxy and also target evaluating energy saving.

(2) **Minification of CSS and JS**: Minification refers to removing unnecessary characters, such as white spaces and comments, from the code to reduce the size of a file. This is typically done to improve the load time of a page. According to a survey, in 10 top U.S. websites, 21% size reduction was achieved by minification.[14] According to Reference 16, minification typically achieves a 20% size reduction for JS. There are different tools to minify JS and CSS. The YUI Compressor[‡]is a JavaScript and CSS compressor which, in addition to removing comments and white-spaces, obfuscates local variables using the smallest possible variable name. JSMin[70] is another JS minifier that works on the server-side. There are also many different minification tools, and some of them support both concatenation and minification at the same time (see Table 1).

3.1 | Architecture and implementation

Our system architecture is illustrated in Figure 1. Server services the page, proxy requests the page from the server and transcodes and returns it to the client, and client requests a web page via a browser.

To implement our experimental platform, we use Squid Caching Proxy which is an open-source proxy[§]. There are different adaptation mechanisms in Squid and our implementation; we use ICAP[¶] with GreasySpoon[#]as the content adaptation mechanism since it is open source and also can modify both the header and body part of the response. Moreover, non-HTTP contents can be modified by the ICAP server. When an HTTP message reaches the ICAP client, it sends the message to the ICAP server, and the transformation is done here. An ICAP client may have many ICAP servers, and an ICAP server may have many ICAP clients. We created two services that work over the ICAP server (see Figure 1) as follows:

(1) **Consolidation service:** This service concatenates external JS and CSS files to one JavaScript and one CSS file. Once the client requests a web page and the HTTP message comes to the proxy, it is redirected to our ICAP server. First, the ICAP server forwards the call to our consolidation service which finds all JS and CSS files and downloads them. Each

[‡]yui.github.io/yuicompressor.
[§]http://www.squid-cache.org/.
[¶]wiki.squid-cache.org/Features/ICAP.
[#]greasyspoon.sourceforge.net.

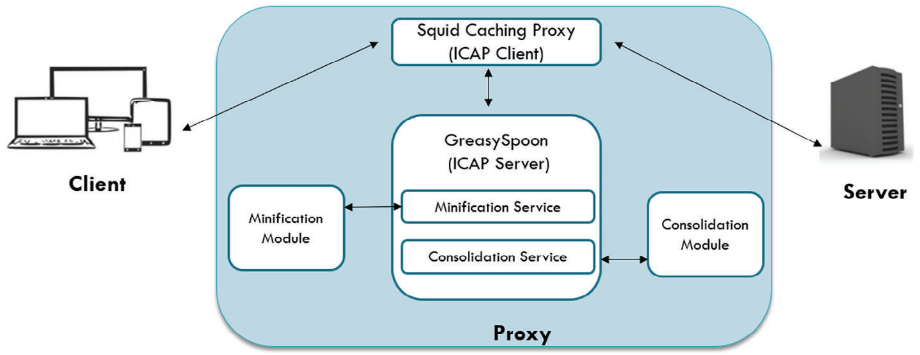

FIGURE 1 The software architecture

downloaded external JavaScript is appended to a combined JS file called last.js file and each external CSS downloaded is appended to a common file called "last.css" file on our proxy. The underlying HTML is then transcoded by updating all CSS and JS calls to this common file. To illustrate, if our HTML has the following JS calls:
<script type="text/javascript" src="script1.js" language="javascript">
<script type="text/javascript" src="script2.js" language="javascript">
<script type="text/javascript" src="script3.js" language="javascript">
The consolidation module removes all of the above external script calls and puts one external script call to request the last.js file from the IP address of the proxy as follows:
<script src="http://194.170.10.2/last.js">
With these transcoding, the end-user requests one concatenated .js and .css file from the IP address of the proxy instead of the server. If the number of external CSS or script files is less than or equal to one, no transcoding is done by the module.

(2) **Minification service:** In our minification service, we follow the guideline that recommends minifying CSS and JS to reduce the size of the external components. Thus, this service minifies external JavaScript and CSS files. This service works very similarly to the consolidation module. HTTP message reaches our ICAP server, and the call is forwarded to our minification service. The minification service first finds all .js and .css file calls from the HTML and downloads these external files into our proxy. These files are then minimized by using the following open source tools. YUI Compressor is used for CSS and JSMin[70] is used for JS minification. Both of these tools work similarly, and they mainly remove unnecessary white spaces and comments from these external files. After these minification processes, we replace the calls in the HTML file with these minified local files. To illustrate:
<link rel="stylesheet" href="c1.css" type="text/css" />
is replaced with
<link rel="stylesheet" href="http://194.170.10.2/c1.css" type="text/css"/>
Therefore, when the user requests the file, they will retrieve the minimized files from the proxy rather than retrieving the full external files from the server.

4 | EVALUATION

To evaluate our proposed transcoding services, we ask the following two research questions, and in this section, we detail our evaluation process that investigates these questions.

(1) *Does our system allow energy saving on the client-side by concatenating external JS and CSS files to reduce the number of HTTP connections?*: Here, we aim to provide evidence to show that by concatenating external resources such as JS and CSS, we can provide energy savings to the clients.
(2) *Does our system allow energy saving on the client-side by minifying external JS and CSS to reduce the size of the components?*: By investigating this, we aim to provide evidence to show that by minimizing the size of external resources such as JS and CSS, we can save energy on the client.

4.1 | Materials

To test our services with real web pages, the evaluation materials were selected from the Alexa Top 100 list on October 8, 2018.[93] However, to eliminate the external factors (such as the latency in the network) that may affect our measurements, these web pages are downloaded to the local server to be used in our measurements. For downloading web pages, we used WebScrapBook[||] and in case of failures, we used Wget[**]. Unfortunately, we could not use some of these pages and had to eliminate them from our dataset. For example, some web pages have versions in different languages such as *google.com* and *google.com.tr*, so we eliminate the alternatives and only keep the main version. Web pages with inappropriate content are also eliminated. After these two steps, 70 web pages remained from the top 100 list. Since we used the locally stored versions for measurements, pages that could not be downloaded properly by WebScrapBook and Wget and did not function properly were also eliminated. After this step, 35 pages remained which were tested using Google Chrome DevTools to ensure that there were no modifications on the local versions. To do that, missing files that are served from other servers are downloaded, and the code is adjusted, respectively.

To also see the effect of our concatenation service, the web page needs to include at least two external CSS or JS files, but since eight of these web pages include one or fewer external JS or CSS files, they were removed from the test set. Lastly, eight of remained web pages were modified after transcoding. Thus, 19 web pages remained for the evaluation of concatenation or concatenation + minification services. Similar to this, to evaluate our minification service, a web page need to include at least one external CSS or JS file and three pages do not include any external JavaScript or CSS files, they were not used in the minification service evaluation. Seven of the remained web pages were modified after minification. Finally, 25 web pages were kept to evaluate the minification service. Table 2 shows the final dataset used in our evaluations.

4.2 | Equipment

The evaluation is done over two types of clients: desktop and mobile. To eliminate external factors and differences in server/proxy and client components that could affect the measurements, three identical desktop computers are used for Desktop measurements. The processor of these computers is Intel Core i7 model 4770 with a base frequency of 3.4 GHz and the turbo frequency of 3.9 GHz. The processor consists of 4 cores, and the size of the installed memory is 16 GB. The operating system of the server and the proxy is Ubuntu 16.04, and the desktop client is Windows 10. The screen size is 18.5 inch and the resolution is 1920×1080. We have set the brightness of the screens to 50% during the tests. Cisco 2901 Integrated Services Router is used to provide a connection between these computers.

For testing with a mobile client, the router and the client computer are replaced. The router is replaced with a switch which is Cisco Catalyst 2960-X, and Netgear Wireless-N 300 Access Point (WN802T) is connected to the switch to provide a wireless network. The client desktop is replaced with the Samsung Galaxy S3 Mini (GT-I8200Q). Its processor is a 1.2 GHz dual-core Cortex-A9, and it has 1 GB RAM. The installed operating system is Android 4.2.2. The screen resolution is 480×800 pixels, and the brightness is set to 50% during the tests. No external microSD card is installed, and only the PowerTutor application is installed to be used in our tests. The mobile device connects to the local network via Wi-Fi over the wireless access point and the switch.

As we used Intel Power Gadget, the desktop devices computers' processors should be 2nd Generation Intel Core Processor or later and our computers have Intel Core i7. On the other hand, we could not find a mobile device that PowerTutor's power model was built on but we chose one of the devices used in a study[94] with PowerTutor.

4.3 | Clients and measurement tools

We use two types of clients for measurements as we wanted to see the overall effect on both a device with limited resources, that is, a mobile device, and with a device that has much more powerful resources. Also, we do not want the external factors, that is, any fluctuation in network speed affect our measurements. We used a local network for the mobile client to minimize the fluctuation but we also used a desktop client to ensure that the measurements are performed in a stable environment.

[||]https://chrome.google.com/webstore/detail/webscrapbook/.
[**]https://www.gnu.org/software/wget/.

TABLE 2 Evaluation set for our services[93]

Rank	Website	Number of JavaScript files	Number of CSS files	Conc.	Min.	Conc + min
3	facebook.com	1	10	✓	✓	✓
4	baidu.com	1	0	X	✓	X
5	wikipedia.org	2	0	✓	✓	✓
6	qq.com	8	2	✓	✓	✓
9	yahoo.com	14	13	✓	✓	✓
10	sohu.com	8	1	✓	✓	✓
14	twitter.com	1	3	✓	✓	✓
15	amazon.com	0	5	✓	✓	✓
19	360.cn	14	3	✓	✓	✓
25	blogspot.com	2	2	X	✓	X
32	alipay.com	1	0	X	✓	X
43	imdb.com	7	3	✓	✓	✓
45	microsoft.com	2	1	✓	✓	✓
52	tumblr.com	13	9	X	✓	X
55	naver.com	7	6	✓	✓	✓
56	office.com	5	5	✓	✓	✓
59	wordpress.com	1	5	✓	✓	✓
66	paypal.com	5	2	✓	✓	✓
69	microsoftonline.com	2	1	✓	✓	✓
70	soso.com	1	0	X	✓	X
72	stackoverflow.com	2	2	✓	✓	✓
74	github.com	3	3	✓	✓	✓
89	roblox.com	10	2	✓	✓	✓
95	bbc.com	6	6	X	✓	X
100	quora.com	2	1	✓	✓	✓

Note: X, irrelevant; ✓, relevant.

Mobile users can access the web in different ways: native applications, web applications, hybrid applications, web sites, and widgets.[95–99] Widgets and native applications use the Internet in their background, but they are task-specific. They are not suitable for cross-platforms. Mobile web applications and hybrid applications are cross platforms, but again they are designed to perform a specific task. Web sites can be accessed with all the browsers. Responsive websites are the ones that deliver different versions of a web page depending on the requested client type. Since web applications can have security restrictions on access, in our study, we focus on mobile clients that access web pages via browsers.

Different tools are used on the desktop and mobile to measure the power or energy consumption, the size of the components, and the number of requests. We used the following tools for measurements:

PowerTutor is used for the measurements done on the mobile client which is an Android application that displays the power consumption of major system components such as a display, CPU, network interface, and different applications. PowerTutor is chosen as it is widely used in the literature. For instance, in their study, Wu et al.[100] indicated that software-based tools to measure the energy consumption of mobile applications such as PowerTutor give reasonable results and provide useful values because they provide detailed energy consumption information for each hardware component. In another study, Chang et al.[101] suggested PowerTutor that Android developers can take advantage of real-time power estimation feature from the application to make their code more energy-efficient.

PowerTutor's power model was built on HTC G1, HTC G2, and Nexus One. In the literature, several studies use PowerTutor to measure energy consumption. Some studies[102,103] used the

mobile devices that its power model built on, while some other studies[94,104] used with other mobile devices, even in,[94] they used the same mobile device as we used in our study. On the other hand, Rizk and Youssef[105] used both the mobile device that the power model was built on and other mobile devices for their measurements. Therefore, based on these in the literature, we use PowerTutor in our study.
The application has already been withdrawn from the Android Market but it can be downloaded as APK from its web page[††]. There are three main reasons that we preferred PowerTutor in our evaluation. Firstly, it has been used widely in the literature. Secondly, it was the only free measurement application for mobile phones. Thirdly, it produces text file output as well to see detailed results.

Intel Power Gadget is a tool to monitor power usage, and we used it for the measurements done on the desktop client. Cumulative processor energy (mWh) data is extracted from the log file generated by the tool[‡‡].

Google Chrome DevTools is used to record some requests, loading time, JS and CSS files, and the size of the components on the desktop client. It is a web developer tool that helps developers to edit web pages on the fly. It is built directly into Google Chrome and developers may record the process of the loading process of the web pages via this tool[§§].

4.4 | Procedure

To see the impact of the two services implemented, they are evaluated both separately and together. Firstly, the web pages are evaluated with our minification service, then they are evaluated with concatenation service, and finally, the impact of both concatenation and minification is evaluated together. In our evaluation, we first concatenate pages and then minify the concatenated page. Our metrics are the number of requests, page load duration, and power/energy consumption for concatenation service. For minification service metrics, a number of the requests is replaced with the size of the components. The evaluation is done over two different architectures to see the impact of our services. The first architecture represents the case where is no transcoding proxy and consists of only the client and the server. In the second architecture, proxy sits between the server and the client, and web pages are transcoded before being served to the client.

During the evaluation over the desktop client, regular 4G (4 Mb/s download speed, 250 kb/s upload speed, and 20 ms RTT) is simulated by Google Chrome DevTool. The cache is disabled, and the history is deleted over the browser. Also, the cache and the data on the proxy and the cache on the server are cleared to see the real impact of the services. All of the other applications apart from the browser and Intel Power Gadget are stopped during the evaluation over the desktop client. The duration for the energy measurement is set to a total of 30 s. The first 10 s are idle, and the browser is launched in this period. Then the web page is requested and waited for 10 s to load the page. Just after, we switch to Intel Power Gadget, and after waiting 10 s, the measurement is done. Indeed, requesting web pages are less than 10 s for most of the web pages, but this duration is constant. This duration is set after the tests are done to see the load time of the web pages.

During the evaluation of the mobile client, the cache and the history of the browser are cleared the same as the evaluation over the desktop client. The mobile device is connected to the network via Wi-Fi as we do not want the Internet to affect our tests. No other application apart from PowerTutor is installed on the mobile device. The duration of the test lasts 90 s. In the first 30 s, the browser is launched. Then, the web page is requested and waited for 30 s. After this duration, we switch to PowerTutor, and the test duration finishes after 30 s. The duration over the mobile client is longer than the duration over the desktop client as the load time of the web page is increased, and switching between the applications (browser and PowerTutor) takes more time compared with the desktop client. The tests are repeated if any unexpected behavior (such as fluctuation) is observed and we only keep the last stable measurement.

The evaluation is performed manually for both client types by one experimenter who is the first author of this article. Each web page is measured three times and the average is taken for both mobile and desktop. The device is restarted before each test. The test duration differs for mobile and desktop clients. There are two main reasons behind it: load time and fluctuation observation. As we used the same desktop version of the website for both types of clients, the load time

[††]http://ziyang.eecs.umich.edu/projects/powertutor/.
[‡‡]https://software.intel.com.
[§§]https://developers.google.com/web/tools/.

of web pages is different in mobile and desktop (higher in mobile). First, we set a safe duration more than the longest load time of our evaluation set. Secondly, we observed the fluctuations in measurements (from GUI of Intel Power Gadget and PowerTutor) during measurements. We saw that no fluctuation occurs if there is no fluctuation during the first 10 s on desktop and 30 s on mobile. Thus, we set the timing for taking the measurements as 10 + 10 + 10 for desktop and 30 + 30 + 30 for the mobile client.

As explained above, the evaluation of our services is done with real web pages to see the real impact of the services; no mock-up page is used. As the cache is disabled and the history is cleared to eliminate the factors that may affect our results, the results may differ (page load time may be shortened) in real-life usage with enabling cache. However, in our tests, we have tried to provide a test environment with minimum external influence on the measurements. Thus, we evaluated our services with real web pages in a test environment.

5 | RESULTS

We have our results in three cases: concatenation, minification, and concatenation + minification over two clients: desktop and mobile. Here we present our results for each client and also show our investigation of the impact of these transcoding techniques on the battery life of a typical laptop and a mobile phone battery.

5.1 | Results on desktop client

The mean of the number of requests, total size in KB, and the load time in are calculated before and after transcoding (see Table 3). The results show that the number of requests is decreased after the transcoding with concatenation and concatenation + minification and the total size of the components is decreased after the transcoding with minification and concatenation + minification services. The mean value of the number of requests decreased from 76.80 to 69.35 after concatenation and concatenation + minification. The total size decreased from 899.50 to 663.53 KB by minification service. On the other hand, the load time of the web pages increased after transcoding. The average load time increased to 2.41 s from 2.19 s after concatenation, to 3.77 s from 1.90 s after minification, and to 3.49 s from 2.19 s after concatenation + minification. We have a total of 19 web pages to test concatenation and concatenation + minification and 25 web pages to test minification service. Thus, the before values in these tables are different as the number of samples is slightly different.

The next step is to see the impact of the decrease in the number of requests and the total size on the energy consumption of the device. To see that, Intel Power Gadget is used to record cumulative processor energy over the desktop client. Intel Power Gadget records the cumulative processor energy in terms of milliwatt-hour (mWh) where x mWh represents that x milliwatts of electricity used continuously for one hour. The results show that our services reduced the cumulative processor energy on average of our evaluation set (see Table 4). The cumulative processor energy is decreased from 33.19 to 29.12 mWh after concatenation, from 32.21 to 29.77 mWh after minification, and from 33.19 to 29.91 mWh after concatenation + minification which means approximately 12.26%, 7.57%, and 9.87% reduction, respectively in an average of our test materials. Our concatenation and concatenation + minification services reduced the cumulative processor energy in all web pages from our evaluation set. On the other hand, our minification service could the cumulative processor energy in 20 web pages and increased the energy in five of the web pages from our evaluation set.

These results are also evaluated statistically to determine whether or not the difference between before and after transcoding over our services is significant. Firstly, Shapiro-Wilk (SW) test is applied to see whether or not the difference between the results is a normal distribution or not. Then, paired-dependent *t*-test or Wilcoxon signed rank test is

TABLE 3 Mean value of each sample of the number of requests, total size (KB) and the load time (s)

	Number of requests		Total size (KB)		Load time (s)	
Service	**Before**	**After**	**Before**	**After**	**Before**	**After**
Concatenation	76.80	69.35	1035.32	1035.37	2.19	2.41
Minification	55.44	55.44	899.50	663.53	1.90	3.77
Concatenation + minification	76.80	69.35	1035.32	913.65	2.19	3.49

TABLE 4 Mean value of each sample of the cumulative processor energy (mWh) and statistical results over desktop client

	Cumulative processor energy (mWh)						
Service	Before	After	Test	N	df	*t* or *Z* value	Effect size
Concatenation	33.19	29.12	Wilcoxon	19	–	−3.823	0.620
Minification	32.21	29.77	Wilcoxon	20	–	−3.619	0.572
Concatenation + minification	33.19	29.91	Wilcoxon	19	–	−3.783	0.614

TABLE 5 Mean value of each sample of the average processor power (mW) and statistical results over mobile client

	Average processor power (mW)						
Service	Before	After	Test	*N*	df	*t* or *Z* value	Effect size
Concatenation	654	628	*t*-test	–	18	5.502***	0.627
Minification	652	636	Wilcoxon	21	–	−3.560***	0.549
Concatenation + minification	654	632	Wilcoxon	19	–	−3.825***	0.620

Note: *T*-test effect size: 0.01—small, 0.06—moderate, 0.14—large. Wilcoxon effect size: 0.1—small, 0.3—moderate, 0.5—large.
$^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.0001$.

applied based on the *p*-value that comes from the Shapiro-Wilk test. Paired-dependent *t*-test is for normal distribution ($p > 0.05$), and Wilcoxon signed rank test is for not-normal distribution. Based on the *p*-values, Wilcoxon signed rank test is applied for the difference in cumulative processor energy before and after transcoding for concatenation, minification, and concatenation + minification (see Table 4). The result of the Wilcoxon signed rank test shows that there is a significant difference between before and after transcoding. To see the effect size, the *r*-value is calculated for the Wilcoxon signed rank test. For the *r*-value of Wilcoxon signed rank test, 0.1 is small, 0.3 is medium, and 0.5 is large effect size.[106] Thus, our statistical test results show that the effect size is large for each service as the effect size is greater than 0.5 in the output of the Wilcoxon signed rank test of our services.

5.2 | Results on mobile client

PowerTutor is used to trace average power on the mobile client in a total 90 s interval (30 s for the request). PowerTutor records the average power in terms of milliwatt(mW) and it is used to quantify the rate of energy transfer. Results show that our concatenation, minification, and concatenation + minification services achieved 3.94%, 2.50%, and 3.34% average power saving over the mobile client, respectively (see Table 5). The average power of the mobile device is decreased from 654 to 628 mW after concatenation, from 652 to 636 mW after minification, and from 654 to 632 mW after concatenation + minification. We achieved a reduction in average power for all web pages from our evaluation set with our concatenation and concatenation + minification services. On the other hand, our minification service achieved a reduction in 21 web pages from our evaluation set; in four web pages, the average processor power increased after transcoding.

These mobile results are also evaluated statistically to determine whether or not the difference between before and after transcoding over our services is significant. Paired-dependent *t*-test is applied to the result of the concatenation service and the Wilcoxon signed rank test is applied to the result of minification and concatenation + minification services based on p values of the Shapiro-Wilk test. Paired-dependent *t*-test shows that there is a significant difference in concatenation service over the mobile client for the average power (mW) before and after transcoding. To find the magnitude of the effect, eta squared statistics is used for the paired-dependent *t*-test. Interpreting the eta squared value is as follows; value 0.01—small effect, value 0.06—moderate effect, value 0.14—large effect.[106] Thus, our statistical result shows that the effect size is large (0.620) for our concatenation service. Similarly, the result of the Wilcoxon signed rank test shows that there is a significant difference between before and after transcoding with minification and concatenation + minification services where the effect size is large for both (0.549 and 0.620, respectively).

5.3 | Results on battery life with a typical laptop battery

To illustrate the impact of these transcodings on battery life, we also investigated the effect on two different device batteries: laptop and mobile phone. To do this, we use the ASUS N580VD laptop battery with 47 Wh capacity and Samsung Galaxy S3 Mini (GT-I8200Q) battery with 5.7Wh capacity. As the average power of the result does not show the average total system power, two different assumptions are used: the average processor power is 100% of the total system power and 70% of the total system power. The mean values of each service are used to calculate the battery duration to analyze the improvement in the battery duration in an overview. We applied the power reduction on the desktop client for laptop battery calculations and the power reduction on the mobile client for mobile phone battery calculations.

The result of the laptop battery calculation shows that our concatenation, minification, and concatenation + minification services can achieve 13%, 7% and 9% improvement in the laptop battery life on the client-side with the assumption of processor power equals to total system power (see Table 6) and 9%, 5% and 7% improvement in the laptop battery life on the client-side with the assumption of processor power equals to 70% of the total system power (see Table 7).

As we observed less power reduction over the mobile client, the mobile phone battery shows less improvement compared with the laptop battery. The result of the mobile phone battery calculation shows that our concatenation, minification, and concatenation + minification services can achieve 4.1%, 2.5%, and 3.5% improvement in the mobile phone battery life on the client-side with the assumption of processor power equals to total system power (see Table 8) and 2.9%, 1.8%, and 2.4% improvement in the mobile phone battery life on the client-side with the assumption of processor power equals to 70% of the total system power (see Table 9).

TABLE 6 Laptop battery life analysis with the assumption of total system power

		Battery duration (h)			
100%	**%Power[a] reduction**	**Before**	**After**	**Improvement (h)**	**Improvement (%)**
Concatenation	11.35	11.93	13.47	1.54	13
Minification	6.39	12.27	13.13	0.86	7
Concatenation + minification	9.01	11.93	13.09	1.16	10

[a]Average processor power.

TABLE 7 Laptop battery life analysis with the assumption of 70% of total system power

		Battery duration (h)			
70%	**%Power[a] reduction**	**Before**	**After**	**Improvement (h)**	**Improvement (%)**
Concatenation	11.35	11.93	12.97	1.04	9
Minification	6.39	12.27	12.86	0.59	5
Concatenation + minification	9.01	11.93	12.72	0.79	7

[a]Average processor power.

TABLE 8 Mobile phone battery life analysis with the assumption of total system power

		Battery duration (h)			
100%	**%Power[a] reduction**	**Before**	**After**	**Improvement (h)**	**Improvement (%)**
Concatenation	3.94	8.716	9.076	0.361	4.1
Minification	2.50	8.742	8.962	0.220	2.5
Concatenation + minification	3.34	8.716	9.019	0.303	3.5

[a]Average processor power.

TABLE 9 Mobile phone battery life analysis with the assumption of 70% of total system power

70%	%Power[a] reduction	Battery duration (h) Before	Battery duration (h) After	Improvement (h)	Improvement (%)
Concatenation	3.94	8.716	8.968	0.253	2.9
Minification	2.50	8.742	8.896	0.154	1.8
Concatenation + minification	3.34	8.716	8.928	0.212	2.4

[a]Average processor power.

6 DISCUSSION

This study aims to save energy on the client-side by concatenating and/or minifying external JS and CSS files to reduce the number of HTTP connections and/or reduce the size of the components. Thus, two services are implemented on the proxy server: (1) concatenation and (2) minification. The evaluation of these services is done together. Our results show that the services implemented achieved statistically significant energy savings. Our three services (concatenation, minification, and concatenation + minification) reduced the cumulative processor energy (mWh) by 12.26%, 7.57%, and 9.87%, respectively in an average of our evaluation set. The statistical tests show that there is a significant improvement and the effect size is large.

The results show that the highest improvement is achieved with concatenation service, without minification. Although concatenation + minification reduces the number of requests and the size of the components, by only concatenating the external files, more energy saving is achieved. There may be different factors that affect this, but our major finding is the load time. The average load time of the concatenation service increased by 10.2% while this rate is 59.49% for concatenation + minification and 98.67% for minification. As the concatenation service does not increase the load time so much, it achieved the highest energy saving.

When we analyze the load time results, it can be seen that concatenation is the fastest service and minification is the slowest service. Concatenation + minification has a moderate speed although it manages both concatenation and minification and its speed is better than minification only. The reason is that the minification service starts a different process for each external JavaScript and CSS file. On the other hand, concatenation + minification first concatenates the external files, and then it minifies the concatenated files. Thus, the number of minification processes is decreased over the concatenation + minification service. Multithreading techniques could improve these results. However, further research could be conducted to investigate the effects of multithreading techniques on load time for our minification service.

The impact of the services is also evaluated over a mobile client. The average power measurement results over the mobile client show that our concatenation, minification, and concatenation + minification services achieved 3.94%, 2.50%, and 3.34% average power saving, respectively. The percentage of energy-saving over the mobile client is less than the energy-saving over the desktop client. One reason for this may be that different evaluation intervals are used for the mobile and the desktop client. The interval for the measurements over the mobile client is 90 s (30 s idle, 30 s for the request, and 30 s idle again) and over the desktop client is 30 s (10 s idle, 10 s for the request, and 10 s idle again). This interval is changed for the desktop, and the mobile client as switching between PowerTutor and the browser takes more time than launching the browser on the desktop client. Another reason for different intervals is that the loading time of the web page is increased over the mobile client, therefore to observe better the time difference of the services is obtained less over the mobile client as the time interval is increased.

The guidelines mentioned in Section 3 states that minification can achieve around 20% size reduction. In our study, the minification service achieved up to a 26% reduction in the total size of the web page. Thus, our study also provides evidence for the effect of this guideline. Previously, we mentioned that[53] applies a similar approach with our study to reduce the energy consumption of websites based on two techniques: (1) combining images with CSS sprites and (2) reducing the number of redirects. Their results show that redirect service can achieve 4.6% and image adaptation can achieve a 7% reduction in cumulative processor energy. The results of our study show that we achieve more reduction compared with this previous study. However, they applied different techniques therefore we cannot do direct comparisons. However, future studies can be conducted to investigate the effect of combined techniques.

Finally, our study is not without limitations. First, we could not use all the pages in Alexa top 100. However, we believe the dataset we used is still realistic and does provide evidence for the energy-saving improvements on the client.

Here, we present our evaluation methodology in full detail so that it can easily be replicated such as using a larger testbed or working under different conditions such as a large number of devices and different distributions of requests to also emulate overload conditions, and so on. Our evaluation methodology can be replicated with rigorous analyses on other edge clients as well. Another further study can be related to the effect of the operating system, browser, and encryption in energy reduction. To illustrate, the literature review showed us that browsers play a role in energy consumption.[38–41] Also, our evaluation methodology can be replicated by investigating specific and popular content types such as video.

Second, we performed our tests manually for both mobile and desktop clients. The reason is that we observed if any fluctuation occurs before and during the tests from GUI of both Intel Power Gadget and PowerTutor. This limitation led us to perform tests manually. However, further studies can be conducted to automate the tests.

Third, for illustrating battery life improvement data, we use a sample battery, but a much more detailed study could be conducted on that.

Fourth, the power model of the tool that we used for the measurements over mobile phone (PowerTutor) was built on three specific devices and they state that when used with phones other than the given phone models, power consumption estimates will be rough. However, we had no other choice for measurements over mobile devices. To decrease the inconsistency in measurements, we tried our best to find a mobile phone to be used in our tests that has similar specifications to the mobile phones that PowerTutor's power model was built on. There are also some studies[94,104] that use different mobile devices to measure energy consumption with PowerTutor. Furthermore, in our study PowerTutor measurements were used for comparative purposes, therefore, if there are any errors or biases in measurement, that would occur in both cases. Therefore, this would not affect the comparison. Also, we performed our tests on the desktop client. We believe that if there is an improvement over the desktop which has a powerful processor and has a limitless source, there will be an improvement over the mobile device which has a less powerful processor and limitations.

Fifth, in our study, we provide evidence for Desktop and web pages displayed on a mobile client. However, we might also have responsive web pages that are served differently on mobile devices. Further studies can be conducted to investigate the effect of responsive web pages. This would improve the results of the mobile client.

Finally, when we concatenate external files into a single file, this has the disadvantage of increasing the load time, and also we do not take into account if these external files are used or not. Therefore, further research could be conducted to investigate techniques to reduce the load time while saving energy and also improve the smartness of concatenating external file processes such that asynchronous processing would be possible (files are concatenated when they are needed or eliminated if they are not needed at all).

In summary, our results show that the implemented services may achieve energy saving on the client-side and this energy saving is statistically significant. Our battery life example analysis also illustrates that these services improve the battery life on the client-side.

6.1 | Modeling sustainability

To consider the impact of our work on sustainability, we need to consider that the proxy also consumes energy for transcoding which means when the client saves energy proxy server spends energy. However, the proxy can be used for large-scale systems to answer a lot of clients, and to cache on the proxy may be applied to reduce the number of requests between the proxy and the main server. Once the content is transcoded on the proxy, it can be served to many clients, and this may decrease the energy consumption of the proxy and the load time of the web page. Here we introduce a basic model and its result over our concatenation service to see whether our system achieves overall energy efficiency or not.

Our approach for modeling the sustainability is as follows: In the first request of the web page, all of the web page sources are downloaded to our proxy server and the transcoding is done. The source with the transcoded files is cached over the proxy. After the first request of the web page, if any client requests the same web page, the sources are served from the proxy instead of the server. Here, we assume that there is no change in the web page during this process. Our model eliminates the server after the first request of the web page. Thus, the energy consumption of the server is ignored after the first request of the web page.

In our model, we attempt to find how many clients should request the same web page to achieve energy efficiency in the overall system. Thus, the comparison is based on the total system energy of n clients with and without transcoding. The total energy consumption is calculated with two components (server and client) for the normal case (without transcoding), three components (server, proxy, and client) for the first request (caching), and two components (proxy and client) for

TABLE 10 Measurement inputs for the sustainability model

	B (mWh)	A (mWh)	C (mWh)
Client	33.542	24.431	55.964
Proxy	0	22.366	40.963
Server	19.974	0	19.974
Total	53.516 (B)	46.797 (A)	116.302

Note: A, energy with transcoding and the source is already cached; B, energy without transcoding; C, energy during first request for transcoding.

the case after data is cached on the proxy. These measurements are done in 30 s period, similar to the tests done for the evaluation of the services.

In our model, another assumption is that there is no traffic in our server. The scenario for our model is as follows: Client(n) requests the web page and after 30 s passed, the client(n+1) requests the same web page. No idle time is passed between these 30 s. Thus, the inputs of our model is as follows:

A: Total Energy with Transcoding and the Source is Already Cached
B: Total Energy without Transcoding
C: Total Energy of First Request for Transcoding
A = Proxy Energy + Client Energy
B = Server Energy + Client Energy
C = Server Energy + Proxy Energy + Client Energy
n = Number of Clients

Based on the above explanations, the formula should be as follows to have overall energy efficiency with *n*clients:

$$n * B > C + n * A.$$

The results of the above formula for the concatenation service is shown in Table 10. The values in the table are the mean values for 17 web pages out of 19. No energy efficiency is obtained for 2 of the web pages and this is why 17 web pages instead of 19 web pages are used to calculate the average values. The average number of clients is calculated as 18 to achieve overall system energy efficiency (see Table 10 and Figure 2). In Figure 2, the energy consumption of the overall system for n clients with and without transcoding is showed. According to the figure, it can be seen that if the number of clients is more than 18, the energy consumption of the system with transcoding is less than the energy consumption of the system without transcoding. Therefore, after 18 clients request the same web page, energy efficiency is achieved.

The power measurements over the client are done with Intel Power Gadget and s-tui[¶¶] is used over the proxy and the server. The reason for the different tools used during the evaluation is that the operating system of the client is Windows10 and the proxy and the server is Ubuntu 16.04. We could not find the same tool that measures the power on both of the operating systems. To see the consistency between these tools, we have done some benchmarking. Same benchmarks are tested on Windows and Ubuntu using Intel Power Gadget and s-tui and the standard error between these tools is calculated as 5.2% on average.

For testing purposes, the caching process for our model is not done automatically based on an algorithm. The web page source is saved to the proxy manually. We assumed that the source is cached over the proxy after the first request and when a client requests the same web page, the source is sent over the proxy instead of the server. Here, the manually saved source is delivered to the client by proxy. As no caching mechanism is structured over our proxy, the average power of the proxy and the client during the first request (caching) is based on some calculations. The formulas to calculate the average power of the proxy and the client during the first request is as follows:

Total Energy Consumption during First Request (Proxy) = Total Energy Consumption to Transcode the Web Page (Proxy) + Total Energy Consumption to Request the Web Page without Transcoding (Client) − Total Idle Energy Consumption (Client),

[¶¶]https://github.com/amanusk/s-tui/.

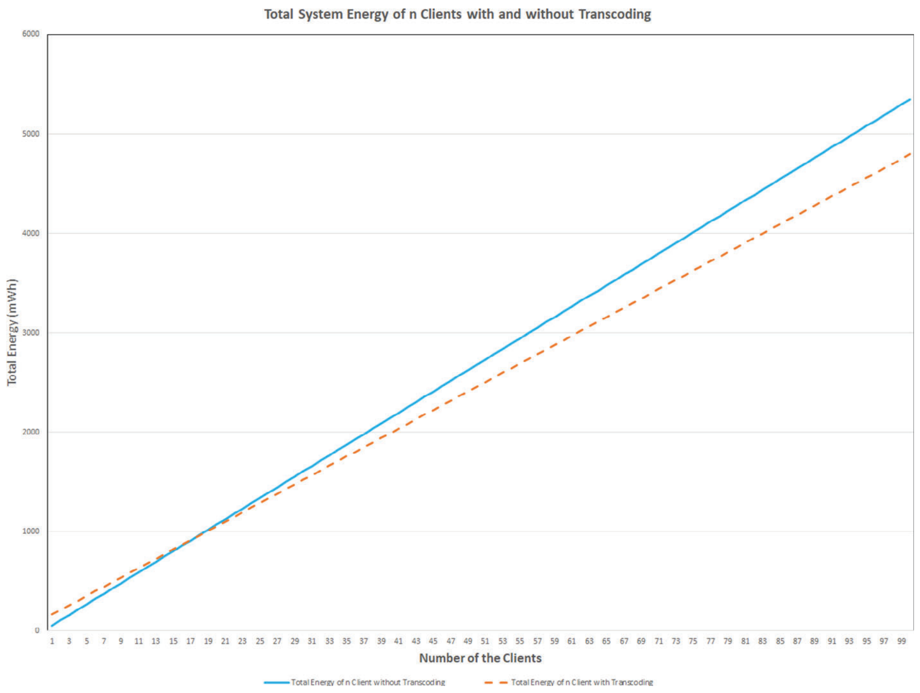

FIGURE 2 Total system energy of n clients with and without transcoding

*Total Energy Consumption during First Request (Client) = Total Idle Energy Consumption (Client) + 2 * (Total Idle Energy Consumption (Client) − Total Energy Consumption to Request the Transcoded Web Page without Caching(Client)).*

We have described our basic model to see whether energy efficiency in our overall system can be achieved or not. This model is evaluated with concatenation service and the result shows that overall system energy efficiency can be achieved in 17 of 19 websites and the mean client number that is needed to request the web page is calculated as 18.

We have constructed our model for experimental purposes to evaluate the overall energy efficiency of the system. However, there can be large-scale effects when deploying our solution on physical deployments. Indeed, Squid Caching Proxy has been deployed with real-life case studies[##]. To illustrate, Wikimedia operated more than 50 Squid servers in three locations around the world serving almost three gigabits of Wiki traffic at peak times. In the second example, Flickr used Squid as a reverse proxy accelerator in front of its photo storage service. In their deployment, they kept 3–5 h of hot objects in each cache where each Squid server handles approximately 1000 requests a second. However, there is a need for new studies to explore the large scale-effects in today's digital world.

Further studies can be performed with our basic model. Firstly, we did not integrate a caching mechanism over the proxy and this model can be improved based on real measurements. Caching mechanism may also reduce the load time. Secondly, we only evaluated concatenation service with the model. The overall efficiency of the other services can be modeled. The last possible future work for our model is integrating the network traffic as our model ignores the network traffic between the components.

7 | CONCLUSION AND FUTURE WORK

This article presents a transcoding-based approach to save energy on the client-side while browsing web pages without modifying the look&feel and without adding extra load to the client or the server. This article also proposes an architecture and an experimental platform centered around a proxy server with two techniques implemented: (1) concatenating external JavaScript and CSS files to reduce the number of HTTP requests and (2) minifying JavaScript and CSS to reduce the size of the components. This architecture and also the experimental platform is used to evaluate these two techniques with a set of real web pages from Alexa Top 100. This systematic evaluation shows that the proposed three services (concatenation, minification, and concatenation + minification) can reduce the cumulative processor energy (mWh) in

[##]http://www.squid-cache.org/Library/.

average by 12.26%, 7.57%, and 9.87% over the desktop client and can reduce the average power (mW) by 3.94%, 2.50%, and 3.34% over the mobile client respectively. This article also shows the impact of this improvement on a typical battery, and the result shows that these three services can achieve 13%, 7%, and 9% improvement in a typical laptop battery life and 4% improvement in a typical mobile phone battery life. on the client-side respectively.

In brief, the experimental study presented in this article provides evidence that two guidelines proposed: concatenation and minification can reduce energy consumption over the mobile client and can increase the battery life duration. This study also reveals some future work. The literature review shows that many guidelines could improve the energy efficiency of web pages. This article investigates only two of those, but many others can be investigated that can potentially contribute to sustainability. Our proposed architecture can easily be extended to investigate other techniques and the methodology used here can easily be facilitated to measure the impact of such other possible techniques.

DATA AVAILABILITY STATEMENT

Our experimental prototype is open-source, and the codes are available in our external online repository at https://github.com/unluh/transcoding.

ORCID

Hüseyin Ünlü https://orcid.org/0000-0001-9906-6066

REFERENCES

1. Information and Communication Technology (ICT). Usage in households and by individuals. Turkish statistical institute (TurkStat) website; September, 2013. Accessed February 11, 2021. http://www.turkstat.gov.tr/PreTablo.do?alt_id=1028
2. Pew: U.S. Adult Gadget Ownership Statistics. RIT cross-media innovation center industry portal website; October 18, 2013. Accessed February 11, 2021. http://printinthemix.com/Fastfacts/Show/787
3. O'Dea S. Forecast number of mobile devices worldwide from 2020 to 2024. statista website; December 18, 2020. Accessed February 11, 2021. https://www.statista.com/statistics/245501/multiple-mobile-device-owner%25ship-worldwide/#:%7E:text=The%20number%20of%20mobile%20devices,devices%20compare%d%20to%202020%20levels
4. Mobile and tablet internet usage exceeds desktop for first time worldwide. Statcounter website; 2016. Accessed February 11, 2021. http://gs.statcounter.com/
5. Everts T. The average web page is 3MB. How much should we care? speedcurve website; August 9, 2017. Accessed February 11, 2021. https://speedcurve.com/blog/web-performance-page-bloat/
6. Everts T. Rules for mobile performance optimization. *Commun ACM*. 2013;56(8):52-59. https://doi.org/10.1145/2492007.2492024
7. Matsudaira K. Making the mobile web faster. *Commun ACM*. 2013;56(3):56-61. https://doi.org/10.1145/2428556.2428572
8. Cui Y, Roto V. How people use the web on mobile devices. Proceedings of the 17th International Conference on World Wide Web, WWW '08; 2008:905-914; Association for Computing Machinery, New York, NY.
9. Haan dE, Kannan P, Verhoef PC, Wiesel T. Device switching in online purchasing: examining the strategic contingencies. *J Market*. 2018;82(5):1-19. https://doi.org/10.1509/jm.17.0113
10. Dagon D, Martin T, Starner T. Mobile phones as computing devices: the viruses are coming! *IEEE Pervasive Comput*. 2004;3(4):11-15. https://doi.org/10.1109/MPRV.2004.21
11. Firtman M. *Programming the Mobile Web*. O'Reilly Media, Inc; 2010.
12. Banerjee N, Rahmati A, Corner MD, Rollins S, Zhong L. Users and batteries: interactions and adaptive energy management in mobile systems. In: Krumm J, Abowd GD, Seneviratne A, Strang T, eds. *UbiComp 2007: Ubiquitous Computing*. Springer; 2007:217-234.
13. Thiagarajan N, Aggarwal G, Nicoara A, Boneh D, Singh JP. Who killed my battery? analyzing mobile browser energy consumption. Proceedings of the 21st International Conference on World Wide Web, WWW '12; 2012:41-50. Association for Computing Machinery, New York, NY.
14. Best practices for speeding up your web site. Yahoo! developer website. Accessed February 11, 2021. https://developer.yahoo.com/performance/rules.html
15. Make the web faster. Google developer website. Accessed February 11, 2021. https://developers.google.com/speed/
16. Souders S. High-performance web sites. *Commun ACM*. 2008;51(12):36-41. https://doi.org/10.1145/1409360.1409374
17. Souders S. *Even Faster Web Sites: Performance Best Practices for Web Developers*. O'Reilly Media, Inc; 2009.
18. Zakas NC. The evolution of web development for mobile devices. *Commun ACM*. 2013;56(4):42-48. https://doi.org/10.1145/2436256.2436269
19. Asakawa C, Takagi H, Fukuda K. Transcoding. In: Yesilada Y, Harper S, eds. *Web Accessibility: A Foundation for Research*. Springer; 2019:569-602.
20. Slack R, Gronow J, Voulvoulis N. Hazardous components of household waste. *Crit Rev Environ Sci Technol*. 2004;34(5):419-445. https://doi.org/10.1080/10643380490443272
21. Gochman S, Mendelson A, Naveh A, Rotem E. Introduction to intel core duo processor architecture. *Intel Technol J*. 2006;10(2):89-97.

22. Muhtaroglu A, Yokochi A, von Jouanne A. Integration of thermoelectrics and photovoltaics as auxiliary power sources in mobile computing applications. *J Power Sources*. 2008;177(1):239-246. https://doi.org/10.1016/j.jpowsour.2007.11.016
23. Zhu Y, Halpern M, Reddi VJ. The role of the CPU in energy-efficient mobile web browsing. *IEEE Micro*. 2015;35(1):26-33. https://doi.org/10.1109/MM.2015.8
24. Sorber J, Banerjee N, Corner MD, Rollins S. Turducken: hierarchical power management for mobile devices. Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services, MobiSys '05; 2005:261-274; Association for Computing Machinery, New York, NY.
25. Etoh M, Ohya T, Nakayama Y. Energy consumption issues on mobile network systems. Proceedings of the 2008 International Symposium on Applications and the Internet, International Symposium on Applications and the Internet; 2008:365-368
26. Pentikousis K. In search of energy-efficient mobile networking. *IEEE Commun Mag*. 2010;48(1):95-103. https://doi.org/10.1109/MCOM.2010.5394036
27. Perrucci GP, Fitzek FHP, Sasso G, Kellerer W, Widmer J. On the impact of 2G and 3G network usage for mobile phones' battery life. Proceedings of the 2009 European Wireless Conference, European Wireless Conference; 2009. 255-259.
28. Balasubramanian N, Balasubramanian A, Venkataramani A. Energy consumption in mobile phones: a measurement study and implications for network applications. Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, IMC '09; 2009:280-293; Association for Computing Machinery, New York, NY.
29. Ben Abdesslem F, Phillips A, Henderson T. Less is more: energy-efficient mobile sensing with senseless. Proceedings of the 1st ACM Workshop on Networking, Systems, and Applications for Mobile Handhelds, MobiHeld '09; 2009:61-62; Association for Computing Machinery, New York, NY.
30. Sailhan F, Issarny V. Energy-aware web caching for mobile terminals. Proceedings 22nd International Conference on Distributed Computing Systems Workshops, Proceedings 22nd International Conference on Distributed Computing Systems Workshops; 2002:820-825.
31. Paul K, Kundu TK. Android on mobile devices: an energy perspective. Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology, 10th IEEE International Conference on Computer and Information Technology; 2010:2421-2426.
32. Oliveira W, Oliveira R, Castor F. A study on the energy consumption of android app development approaches. Proceedings of the 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), IEEE/ACM 14th International Conference on Mining Software Repositories (MSR); 2017:42-52.
33. Rodriguez A, Mateos C, Zunino A. Improving scientific application execution on android mobile devices via code refactorings. *Softw Pract Exper*. 2017;47(5):763-796. https://doi.org/10.1002/spe.2419
34. Pérez-Castillo R, Piattini M. Analyzing the harmful effect of god class refactoring on power consumption. *IEEE Softw*. 2014;31(3):48-54. https://doi.org/10.1109/MS.2014.23
35. Bunse C. On the impact of code obfuscation to software energy consumption. In: Otjacques B, Hitzelberger P, Naumann S, Wohlgemuth V, eds. *From Science to Society*. Springer International Publishing.; 2018:239-249.
36. Sahin C, Wan M, Tornquist P, et al. How does code obfuscation impact energy usage? *J Softw Evol Process*. 2016;28(7):565-588. https://doi.org/10.1002/smr.1762
37. Miettinen AP, Nurminen JK. Analysis of the energy consumption of javascript based mobile web applications. In: Chatzimisios P, Verikoukis C, Santamaría I, Laddomada M, Hoffmann O, eds. *Mobile Lightweight Wireless Systems*. Springer; 2010:124-135.
38. Cameron C. Chrome: faster and more battery-friendly. Google chrome website; September 6, 2016. Accessed February 11, 2021. https://blog.google/products/chrome/chrome-faster-and-more-battery-friendly/
39. Kaźmierczak B. Why we challenge Microsoft's battery test. opera blogs website; June 22, 2016. Accessed February 11, 2021. https://blogs.opera.com/desktop/2016/06/over-the-edge/
40. Weber J. Get more out of your battery with Microsoft edge. windows blogs website; June 20, 2016. Accessed February 11, 2021. https://blogs.windows.com/windowsexperience/2016/06/20/more-battery-with-edge/#MCyQIXD4topvxSdz.97
41. Bui DH, Liu Y, Kim H, Shin I, Zhao F. Rethinking energy-performance trade-off in mobile web page loading. Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, MobiCom '15; 2015:14-26; Association for Computing Machinery, New York, NY.
42. Choi Y, Park S, Cha H. *Optimizing Energy Efficiency of Browsers in Energy-Aware Scheduling-Enabled Mobile Devices*. Association for Computing Machinery; 2019.
43. Noureddine A, Rouvoy R, Seinturier L. A review of middleware approaches for energy management in distributed environments. *Softw Pract Exper*. 2013;43(9):1071-1100. https://doi.org/10.1002/spe.2139
44. Yesilada Y, Harper S, Eraslan S. Experiential transcoding: an EyeTracking approach. Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility, W4A '13; 2013; Association for Computing Machinery, New York, NY.
45. Iaccarino G, Malandrino D, Scarano V. Personalizable edge services for web accessibility. Proceedings of the 2006 International Cross-Disciplinary Workshop on Web Accessibility (W4A): Building the Mobile Web: Rediscovering Accessibility? W4A '06; 2006:23-32; Association for Computing Machinery, New York, NY.
46. Song R, Liu H, Wen JR, Ma WY. Learning block importance models for web pages. Proceedings of the 13th International Conference on World Wide Web, WWW '04; 2004:203-211; Association for Computing Machinery, New York, NY.
47. Chen J, Zhou B, Shi J, Zhang H, Fengwu Q. Function-based object model towards website adaptation. Proceedings of the 10th International Conference on World Wide Web, WWW '01; 2001:587-596; Association for Computing Machinery, New York, NY.
48. Yin X, Lee WS. Using link analysis to improve layout on mobile devices. Proceedings of the 13th International Conference on World Wide Web, WWW '04; 2004:338-344; Association for Computing Machinery, New York, NY.

49. Ahmadi H, Kong J. Efficient web browsing on small screens. Proceedings of the Working Conference on Advanced Visual Interfaces, AVI '08; 2008:23-30; Association for Computing Machinery, New York, NY.
50. Lai PPY. Efficient and effective information finding on small screen devices. Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility, W4A '13; 2013; Association for Computing Machinery, New York, NY.
51. Lam H, Baudisch P. Summary thumbnails: readable overviews for small screen web browsers. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '05; 2005:681-690; Association for Computing Machinery, New York, NY.
52. Fainberg L, Ehrlich O, Shai G, Gadish O, Amitay D, Berger O. Systems and methods for acceleration and optimization of web pages access by changing the order of resource loading; 2011. US Patent App. 12/848,559.
53. Köksal E, Yeşilada Y, Harper S. Twisting web pages for saving energy. In: Cabot J, De Virgilio R, Torlone R, eds. *Web Engineering*. Springer International Publishing; 2017:225-245.
54. Frain B. *Responsive Web Design with HTML5 and CSS3*. Packt Publishing Ltd; 2012.
55. Barr KC, Asanović K. Energy-aware lossless data compression. *ACM Trans Comput Syst*. 2006;24(3):250-291. https://doi.org/10.1145/1151690.1151692
56. Chi CH, Deng J, Lim YH. Compression proxy server: design and implementation. Proceedings of the USENIX Symposium on Internet Technologies and Systems; 1999.
57. Flinn J, Satyanarayanan M. Energy-aware adaptation for mobile applications. *SIGOPS Oper Syst Rev*. 1999;33(5):48-63. https://doi.org/10.1145/319344.319155
58. Uzair A, Beg MO, Mujtaba H, Majeed H. WEEC: web energy efficient computing: a machine learning approach. *Sustain Comput Inform Syst*. 2019;22:230-243. https://doi.org/10.1016/j.suscom.2018.08.005
59. Zhang W, Wen Y, Chen HH. Toward transcoding as a service: energy-efficient offloading policy for green mobile cloud. *IEEE Netw*. 2014;28(6):67-73. https://doi.org/10.1109/MNET.2014.6963807
60. Flatten CSS Imports. Google PageSpeed module website. Accessed February 11, 2021. https://developers.google.com/speed/pagespeed/service/FlattenCssImports
61. Uses document.write. web.dev website; May 2, 2019. Accessed February 11, 2021. https://web.dev/no-document-write/
62. Sprite Images. Apache PageSpeed website. Accessed February 11, 2021. https://www.modpagespeed.com/doc/filter-image-sprite
63. Google Closure Compiler. Google PageSpeed website. Accessed February 11, 2021. https://developers.google.com/web/tools/lighthouse/audits/document-write
64. Combine CSS. Google PageSpeed website. Accessed February 11, 2021. https://developers.google.com/speed/pagespeed/service/CombineCSS
65. Mobify Jazzcat. Mobify website. Accessed February 11, 2021. http://jazzcat.mobify.com/
66. Concatenation of JS and CSS files. IBM knowledge center website. Accessed February 11, 2021. https://www.ibm.com/support/knowledgecenter/en/SSZH4A_6.0.0/com.ibm.worklight.help.doc/devref/c_optimizing_apps_concatenation.html
67. Grunt contrib-concat. GitHub website. Accessed February 11, 2021. https://github.com/gruntjs/grunt-contrib-concat
68. Yui compressor; Accessed February 11, 2021. http://yui.github.io/yuicompressor/
69. JSCompress- the javascript compression tool. Accessed February 11, 2021. https://jscompress.com/
70. Crockford D. JSMin the JavaScript minifier; May 16, 2019. Accessed February 11, 2021. http://crockford.com/javascript/jsmin
71. Minify- JavaScript and CSS minifier. Accessed February 11, 2021. https://www.minifier.org/
72. CSSNano. Accessed February 11, 2021. https://cssnano.co/
73. CSSo. GitHub website. Accessed February 11, 2021. https://github.com/css/csso
74. AjaxMin. Nuget website. January 28, 2015. Accessed February 11, 2021. https://www.nuget.org/packages/AjaxMin/
75. Kotelnytskyi Y. JS&CSS Script Optimizer. Wordpress website; 2016. Accessed February 11, 2021. https://wordpress.org/plugins/js-css-script-optimizer/
76. Merge+Minify+Refresh. Wordpress website. Accessed February 11, 2021. https://en-ca.wordpress.org/plugins/merge-minify-refresh/
77. X-Team. Dependency minification. wordpress website; 2014. Accessed February 11, 2021. https://wordpress.org/plugins/dependency-minification/
78. Minqueue. Wordpress website. Accessed February 11, 2021. https://wordpress.org/plugins/
79. Combine and Minify. Nuget website; March 29, 2012. Accessed February 11, 2021. https://www.nuget.org/packages/CombineAndMinify/
80. Walsh J. Granule. GitHub website. March 29, 2012. Accessed February 11, 2021. https://github.com/JonathanWalsh/Granule
81. Jawr. GitHub website. April 23, 2018. Accessed February 11, 2021. https://j-a-w-r.github.io/
82. The Asset Pipeline. Railsguides website. Accessed February 11, 2021. https://guides.rubyonrails.org/asset_pipeline.html
83. CodeKit. Accessed February 11, 2021. https://codekitapp.com/
84. Prepros. Accessed February 11, 2021. https://prepros.io/
85. Firefox Dust-me selectors. firefox add-ons website. Accessed February 11, 2021. https://addons.mozilla.org/en-US/firefox/addon/dust-me-selectors/
86. What's New In DevTools (Chrome 59). Google developers website. Accessed February 11, 2021. https://developers.google.com/web/updates/2017/04/devtools-release-notes.
87. Lighthouse: remove unused CSS. Gtmetrix website. Accessed February 11, 2021. https://gtmetrix.com/remove-unused-css.html
88. UnusedCSS - Clean Your CSS. Accessed February 11, 2021. https://unused-css.com/
89. MinCSS. GitHub website; April 19, 2019. Accessed February 2021. https://github.com/peterbe/mincss

90. UnCSS. GitHub website; October 7, 2020. Accessed February 11, 2021. https://github.com/uncss/uncss
91. McArthur S. CSS remove and combine. chrome web store; February 3, 2014. Accessed February 11, 2021. https://chrome.google.com/webstore/detail/css-remove-and-combine/cdfmaaeapjmacolkojefhfollmphonoh?hl=en-GB
92. uniq.js. GitHub website. Accessed February 11, 2021. https://gist.github.com/telekosmos/3b62a31a5c43f40849bb
93. Alexa - Top Sites. Accessed October 18, 2018. https://www.alexa.com/topsites.
94. Cao Y, Chen S, Hou P, Brown D. FAST: a fog computing assisted distributed analytics system to monitor fall for stroke mitigation. Proceedings of the 2015 IEEE International Conference on Networking, Architecture and Storage (NAS); 2015:2-11.
95. Huy NP, van Thanh D. Evaluation of mobile app paradigms. Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia, MoMM '12; 2012:25-30; Association for Computing Machinery, New York, NY.
96. Jobe W. Native Apps vs. mobile web apps. *Int J Interact Mob Technol*. 2013;7(4).
97. Looper J. What is a webview? teleferik developer network; November, 2015. Accessed February 11, 2021. https://developer.telerik.com/featured/what-is-a-webview/
98. Saccomani P. Native, web or hybrid apps? whats the difference? MobiLoud website. Accessed February 11, 2021. https://www.mobiloud.com/blog/native%2010web%2010or%2010hybrid%2010apps/
99. Sin D, Lawson E, Kannoorpatti K. Mobile web apps - the non-programmer's alternative to native applications. Proceedings of the 2012 5th International Conference on Human System Interactions; 2012:8-15.
100. Wu H, Knottenbelt WJ, Wolter K. An efficient application partitioning algorithm in mobile environments. *IEEE Trans Parallel Distrib Syst*. 2019;30(7):1464-1480. https://doi.org/10.1109/TPDS.2019.2891695
101. Chang HC, Agrawal AR, Cameron KW. Energy-aware computing for android platforms. Proceedings of the 2011 International Conference on Energy Aware Computing; 2011:1-4.
102. Aly H, Basalamah A, Youssef M. Map++: a crowd-sensing system for automatic map semantics identification. Proceedings of the 2014 11th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON); 2014:546-554.
103. Vasilomanolakis E, Karuppayah S, Fischer M, et al. This network is infected: HosTaGe - a low-interaction honeypot for mobile devices. Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, SPSM '13; 2013:43-48; Association for Computing Machinery, New York, NY.
104. Azmoodeh A, Dehghantanha A, Conti M, Choo KKR. Detecting crypto-ransomware in IoT networks based on energy consumption footprint. *J Ambient Intell Humaniz Comput*. 2018;9(4):1141-1152. https://doi.org/10.1007/s12652-2010017-20100558-20105
105. Rizk H, Youssef M. MonoDCell: a ubiquitous and low-overhead deep learning-based indoor localization with limited cellular information. Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '19; 2019:109-118; Association for Computing Machinery, New York, NY.
106. Pallant J. *SPSS Survival Manual*. McGraw-Hill Education; 2013.

How to cite this article: Ünlü H, Yesilada Y. Transcoding web pages via stylesheets and scripts for saving energy on the client. *Softw Pract Exper*. 2022;52(4):984-1003. doi: 10.1002/spe.3046