Examensarbete

Device-aware Adaptation of Websites

av

Milad Barsomo och Mats Hurtig

LIU-IDA/LITH-EX-G--14/064--SE

2014-06-16
Examensarbete

Device-aware Adaptation of Websites

av

Milad Barsomo och Mats Hurtig

LIU-IDA/LITH-EX-G--14/064--SE

2014-06-16

Handledare: Niklas Carlsson
Examinator: Nahid Shahmehri
Students in the 5 year Information Technology program complete a semester-long software development project during their sixth semester (third year). The project is completed in mid-sized groups, and the students implement a mobile application intended to be used in a multi-actor setting, currently a search and rescue scenario. In parallel they study several topics relevant to the technical and ethical considerations in the project. The project culminates by demonstrating a working product and a written report documenting the results of the practical development process including requirements elicitation. During the final stage of the semester, students create small groups and specialise in one topic, resulting in a bachelor thesis. The current report represents the results obtained during this specialization work. Hence, the thesis should be viewed as part of a larger body of work required to pass the semester, including the conditions and requirements for a bachelor thesis.
Device-aware Adaptation of Websites

by

Milad Barsomo and Mats Hurtig

ABSTRACT

The use of handheld devices such as smart phones and tablets have exploded in the last few years. These mobile devices differ from regular desktops by having limited battery power, processing power, bandwidth, internal memory, and screen size. With many device types and with mobile adaptation being done in many ways, it is therefore important for websites to adapt to mobile users. This thesis characterise how websites currently are adapting to mobile devices. For our analysis and data collection, we created a tool which sends modified HTTP GET requests that makes the web server believe the GET requests were sent from a smart phone, tablet, or a regular desktop. Another tool then captured all the HTTP packets and let us analyse these for each platform. We chose to analyse the top 500 most popular websites in the world and the top 100 websites from 15 different categories fetched directly from www.alexa.com. Among other things, we observed that of the total HTTP objects fetched to render an average website, mobile or non-mobile, more than half of the objects were images. Another conclusion is that a website fetched by an iPhone 4 device is more heavily reduced in amount of images than a Nexus 7.

Department of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden
## Contents

1 Introduction ................................................. 1
   1.1 Motivation ........................................... 1
   1.2 Goal .................................................. 2
   1.3 Problem formulation ................................. 2

2 Theory ....................................................... 3
   2.1 Web content adaptation ............................... 3
   2.2 Techniques of adaptation ............................. 3
      2.2.1 Remove formatting ............................... 4
      2.2.2 Filter media .................................... 4
      2.2.3 Simple redirection ............................... 4
      2.2.4 Full detection and adaptation .................. 4
   2.3 HTTP .................................................. 5
      2.3.1 GET requests .................................... 5
      2.3.2 User agent ...................................... 6
      2.3.3 GET response ................................... 7
   2.4 Related work .......................................... 7

3 Method ....................................................... 9
   3.1 Data collection ....................................... 9
   3.2 Filtering .............................................. 10

4 Analysis of results ......................................... 11
   4.1 Website composition .................................. 11
   4.2 Device comparison .................................... 13

5 Discussion ................................................... 17
   5.1 Results ............................................... 17
   5.2 Methods .............................................. 18
   5.3 Impact on user ....................................... 18

6 Conclusions and future work .............................. 19
Chapter 1

Introduction

1.1 Motivation

The difference between a desktop and a mobile client is vast. During web browsing differences like battery life, bandwidth, network conditions, storage capacity, processing power and screen size affects mobile clients such as mobile phones and tablets. Non-adaptive websites may therefore waste scarce bandwidth to download a full size image belonging to a web page for a mobile phone, even though the client will need to resize it to fit the screen. Even if the mobile client downloads the entire non-mobile web page, the screen size might prevent the user from reading for example a news text comfortably as one would have to adapt the view manually on the touch screen. That is why some websites have mobile versions of their websites which are made to fit and to be easily managed with a mobile client.

In many cases when the developers make a mobile version of a website, they take away some features that is not supported by the tablet or mobile phone. It is also likely that the content is cut down to smaller size. A news website on a desktop show large and detailed images of a disaster where a mobile client will get a picture of lower resolution. They would also want to make interface elements larger so that a user with touch screen will not waste time zooming or pressing the wrong item.

There are different techniques used for adapting to mobile users [1]. One is to completely ignore their needs and provide the full website. Another is to build a separate website especially for mobile clients. The redirecting of mobile users also differ. Some automatically redirect the mobile client to the mobile site and show a link to the full website. Another approach is to show the full website but to provide a link to the mobile site. These techniques will be described further in the theory chapter.

This thesis focus on analysing the adaptation of a website to a mobile client by observing the differences on the content types (e.g. images and CSS) between the websites fetched for different mobile clients.
1.2 Goal

The aim for this project is to clarify if and how websites adapt to mobile clients, focusing on server-side adaptation. In other words, the differences in the response from a web server, when a mobile client is detected. To do that we need to develop a methodology for measuring and capturing the content in a website from mobile clients and a computer. We will use a tool which sends GET requests, with modified user agents. The different clients we are going to imitate are a desktop using Chrome as browser, iPhone 4 using the standard browser, Safari and the tablet Nexus 7 also using standard browser, Chrome. We do not believe that different browsers have a big impact on our results hence we chose the popular ones for each client. In this thesis, when we talk about ”Chrome” we mean the desktop client. The reasons we chose to imitate iPhone 4 and Nexus 7 is because these clients differs in both screen size, where the iPhone 4 has a 3.5 inch screen and a Nexus 7 has a 7 inch screen. They also differ in operating system, the iPhone 4 uses a version of iOS and the Nexus 7 uses Android. We think these differences will give us clearly different results. For each device we first collect data from the 500 most popular websites on the web, this gives us a reference on adaptivity for mobile platforms (which is motivated more in chapter 3). Second, we collect the 100 most popular websites from all categories, excluding the Adult and World category. The selection of websites we made is motivated by the heavy skew of website popularity and because of similar choice made by related work [2] [3] [4] [5]. Using these data sets we can analyse how different websites in the categories adapt to mobile users. Comparisons are also made against the results from the top 500 websites.

In order to reach our goals, we will also contribute to the development of a methodology which compares the size, in bytes, and the amount of objects of a website’s content types with different user agents. We are then going to compare our result with other similar work and analyse the development and our methodology.

1.3 Problem formulation

To specify our aim in a more concrete way, we have identified the following questions that we answer in this thesis:

- How does a mobile website differ in size from one which is fetched for regular desktop?
- What content types are reduced in an adapted mobile website?

These questions led us to a few methodology related questions:

- What parameters does a GET request include? And which matters for the Mobile Web?
- How do you change the parameters in a GET request, to make the website think that we request it from a mobile client?
- How do we analyse multiple GET responses effectively?
- What methods for website adaptation are used today on different mobile clients?
Chapter 2

Theory

2.1 Web content adaptation

There are many different techniques to adapt the content of a website on a mobile [6]. The majority of the strategies and techniques used are build on a model called DAD mode. Three steps defines this model, the first step is Detection, second is Adaptation and the last step is Deliver.

The adaptation of the content can be performed in three levels. The first level is through the web content’s server, this simply stores content that will be delivered depending on what device request it. From the needs of the requesting device, the server can then adapt by converting and tailor the web content.

The second level, which normally is out of the developer’s control, is at the proxy server that acts as an intermediator between the mobile client and the server. It is usually the web content’s server who control and implement it, however a third party device can also control it.

The third and last level is adaptation through the client-side. First of all, the process adaptation can be determined by the user through definitions for preferences, scales and type of the adaptation. Secondly, media selectors, associated with Cascading Style Sheets (CSS), have a big and important role in this level of adaptation. The downside of client-side adaptation is that the user’s air time might be consumed unnecessarily and results in additional cost for the content that ends up not being displayed. This is because the same XHTML markup for desktop browsers is being downloaded for mobile browsers.

In this thesis we will focus on the server-side optimisations.

2.2 Techniques of adaptation

When adapting the content of a website for mobile devices there are multiple techniques to be used at each of the three levels.

This technique is comfortable because it requires practically no effort from the host of the website. It requires much more work from the client, an example of a mobile browser which can manage this type of approach is Opera Mini. Opera Mini sends the web page requests through their own servers which compresses
and adapts the websites before sending it to the mobile devices. Nevertheless, the consideration of this kind of optimisation is not being looked into in this thesis. Other browsers on smart phones or tablets can do the optimizing of the web site on the client which can drain battery life [6]. Browsers that do not optimise the website for the mobile device might load a website which the user finds unmanageable.

2.2.1 Remove formatting

A large problem for the browsers on mobile devices is to parse CSS files, the complex formatting rules require a lot of operations which is limited by battery power, processing power and internal memory [6]. A fast solution to this is to remove all the formatting from the layout page so that only links and text is sent to the mobile user. The benefit for the mobile user is that the amount of data downloaded will be very low which will save battery life but the user might end up with a website that makes no sense since the navigation for the website can be done through images.

2.2.2 Filter media

To keep the HTML file same for all but still adapt the website for mobile devices, it is reasonable to have separate CSS files for mobile devices and desktop users [6]. The HTML file either contains the CSS rules or links to a CSS file on the server. The CSS file makes media queries to check if the screen size of the current device is equivalent to a mobile device and the layout of the page will then follow the rules written for small screens. A JavaScript-function may also be called on when the content is loaded to check the screen resolution and get the appropriate CSS file. The conclusion from this part is that a mobile device and a desktop might get the same HTML-file but when the web page is rendered on the different screens they can appear different due to the CSS rules applied to the different screens.

2.2.3 Simple redirection

This approach is redirecting a user to a mobile version or desktop version of a web site, by using a technique that detects the user’s device type and (not necessarily) other features such as the supported Markup Language [6]. This approach is good to use, for example, when the content does not have to be fully optimized for the device it is accessed from.

There are different ways of the detection of a device, the easiest way is through a script that uses the information from HTTP headers. Since the HTTP header have information about the user agent and Markup Language, it can use this data to redirect the user to the correct version. But a simple script like this have a few downsides, e.g. poor accuracy because of the limitation of more useful capabilities that could be used to detect the correct device. (Can be done by using WURFL file, which is explained in the next subsection).

2.2.4 Full detection and adaptation

Many variables are needed to implement the full detection and adaptation approach in the framework. However, the most important variable is the user agent,
since it contains many important characteristics. Screen size, the existence of the touch screen, preferred Markup Language and support for graphic formats, multimedia files, Java and styles, are some of these characteristics [6]. This approach requires multiple versions of the website, mainly because if the detection of some variables (e.g. the preferred language) fails, then the user should be redirected to another version. There should also be different options available for the content generation, such as selecting language. The downside for this approach is that it might require many resources on your server and should only be used if you have more than enough resources available.

This approach would need to use a more advanced detection technique than mentioned in the previous section. This can be done using the WURFL file. WURFL stands for Wireless Universal Resource File, which is a file that contains information about mobile devices. This file is regularly updated and information about the device’s capabilities can be accessed. Using this file would result in a significantly much higher accuracy to find the correct device, than using a simple script. But this method would require more space on the server and have slower execution time.

2.3 HTTP

Focusing on the server-side optimizations, it is important to understand the requests seen by the server and how they can be used to identify different mobile devices. In this section we provide a brief background on HTTP [7]. HTTP is short for Hyper Text Transfer Protocol and can shortly be described as a communication protocol between clients and web servers. It is located in the application layer and usually runs over TCP. To get a clear understanding of how the protocol works we can demonstrate with a small example: Say that someone wants to checkout CNN’s news website and types in http://www.cnn.com in the address bar of the browser (a client) and it sends a HTTP request to to the server via a TCP connection on port 80 on the server. When the server have received this request, it looks up the file which was requested and it proceeds to send back a response which contains the content of the website.

2.3.1 GET requests

A GET request is one of the HTTP request methods and it is also the majority of request method used to the Web servers [7]. The GET request should only retrieve the data requested from a specific resource, with no other effects. However a GET request can be cached, bookmarked and remain in the browser history which is an important way to save time and performance. This way, when the browser sends a request, the content will load faster because it can used the cached copy. The disadvantages with a GET request are that they have length restrictions and is not secure when requesting sensitive data such as passwords [8]. This is because a GET request adds data on the URL in the address bar, which makes it restricted to 2048 characters. Also if you send a GET request for a web page which needs login information, then the user name and password would be added to the URL and be visible to everyone.

The parameters, also called request headers, of a GET request are:
• **User Agent** - This information are used by server to know about the browser and platform used. This is very important since different browsers are implemented differently and companies can have different design or layout for their website depending on the browser and platform used. The string format of the user agent header can contain multiple comments and tokens (which are keywords) that identify the agent. The string is ordered by the importance of the listed tokens for identifying the application.

• **Accept** - This header contains information about the type of pages the client are allowed to handle (e.g., text/html).

• **Accept-Charset** - Simply the character set acceptance by the client (e.g., ISO-8859-5).

• **Accept-Encoding** - This one tell the client which page encoding and compression methods the client handle (e.g., gzip).

• **Accept-Language** - This data gives information about which languages the client accept (e.g., English).

• **If-Modified-Since** - If the GET request is cached, this header is used to check the time and date of the last time it was used.

• **If-None-Match** - If the GET request is cached, this header checks freshness from the previously sent tags.

• **Host** - Simply tells the DNS name of the server.

• **Authorization** - This header is used for the cases when the client need to prove it can access protected pages.

• **Referer** - The misspelled header says which URL that was used to request the new URL.

• **Cookie** - In this header, we find information about the history and path of the client’s Web browsing.

### 2.3.2 User agent

Since our focus mainly will be on User Agents, we have researched it more carefully. The format of the string of an User Agent is [9]:

```
"User-Agent" : 1*( product — comment )
```

```
product = token ["/" product-version]
product-version = token
```

An example of such User-Agent:

```
CERN-LineMode/2.15 libwww/2.17b3
```

Today, most browsers are using Mozilla rendering engine, a typical user agent string looks something like [7]:

```
Mozilla/5.0 (iPad; U; CPU OS 3_2_1 like Mac OS X; en-us) AppleWebKit/531.21.10
(KHTML, like Gecko) Mobile/7B405
```

The components of this user-agent string would be:

- **Mozilla/5.0**: The product and version of the rendering engine.
- **(iPad; U; CPU OS 3_2_1 like Mac OS X; en-us)**: Information and details about the system and browser the client is running
- **AppleWebKit/531.21.10**: The platform and version of the platform the browser is using
- **(KHTML, like Gecko)**: The details of the platform used
- **Mobile/7B405**: An extension to indicate specific enhancements
2.3.3 GET response

The GET response is naturally what the server returns after a GET request has been received. The first line of a GET Response is the status line which contains a three-digit code and gives information about how the execution of the GET request went on the server. The first digit of the status code tells if the request encountered any errors or if it executed successfully, Table 2.1 shows a few examples. Like the

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Information</td>
<td>100 = server agrees to handle client’s request</td>
</tr>
<tr>
<td>2xx</td>
<td>Success</td>
<td>200 = request succeeded; 204 = no content present</td>
</tr>
<tr>
<td>3xx</td>
<td>Redirection</td>
<td>301 = page moved; 304 = cached page still valid</td>
</tr>
<tr>
<td>4xx</td>
<td>Client error</td>
<td>403 = forbidden page; 404 = page not found</td>
</tr>
<tr>
<td>5xx</td>
<td>Server error</td>
<td>500 = internal server error; 503 = try again later</td>
</tr>
</tbody>
</table>

Table 2.1: HTTP status codes

GET request carries request headers, the GET response carries response headers. Some headers are the same as for the GET request and some are exclusive for the GET response. Two headers which are used in both request and response are Date and Range, the first one tells the exact time and date when the request or response was sent and the second one tells the range of data which is requested or sent as a response.

The headers which are exclusive for GET response are [7]:

- **Set-Cookie** - A HTTP Cookie for the user to store.
- **Server** - Gives information about the server (e.g., nginx).
- **Content-Encoding** - Gives information about the type of encoding used on the data sent.
- **Content-Language** - Tells which language the content is in.
- **Content-Length** - The size of the GET response in bytes.
- **Content-Type** - Tells what type of data which is in the GET response.
- **Content-Range** - If a portion of the web page is in the GET response, this header identifies its range.
- **Last-Modified** - Information about when the date and time when the web page was last changed.
- **Expires** - Information about the date and time when it loses its validity.
- **Location** - Tells the client where it should send its requests.
- **Accept-Ranges** - Tells the client if the server accepts requests where portions of a web page is requested. The headers are followed by the actual content and if the Content-Type value was ”text.html”, the content would be a html file made up by a string of ASCII characters.

2.4 Related work

There have been a lot of research done in the area of HTTP-traffic and website complexity [10] [11]. There were some papers that were interesting and relevant
to us in our study. For example, Butkiewicz et al. [3] which was measuring the amount of objects per content-type from websites in different categories. Measurements of origin and non-origin content, images provided from a content delivery network for example, was also performed. It was made to tell how much non-origin content a website consisted of and how much it affected the load-time of the website. They fetched 1700 websites from www.alexa.com and analysed them from four geographically different positions.

Another related article analysed web traffic of 70,000 users spread over almost 200 countries from year 2006 to 2010 [12]. One interesting result from the article was that popular URLs are getting even more popular and that the client-side interactions with web pages are making up for almost half of the traffic rather than the initial page-load.

Articles on the subject of mobile web browsing helped us understand what was important for mobile web browsing. An example is one where tests of load times are performed with smart phones on mobile optimised websites and non optimised websites [13]. They saw that a smart phone with better processing power had a much shorter browser delay on the same network conditions. They also concluded that the network bandwidth does not affect the load time as much as the network RTT (round-trip-time). Wang et al. [14] also identifies RTT as a key factor in resource loading time and it discusses client-side optimisations such as caching, browser parallelization and web page prefetching. Cloud-based optimisations were also discussed where the mobile web browser uses the cloud to perform computation-intensive and energy draining tasks. Such offloading is effective in saving of the scarce battery resources at the mobile client.

The big difference with this thesis and the related work is that this thesis focus on analysing of websites fetched from different user agents.
Chapter 3

Method

3.1 Data collection

This section describes the method used to modify the HTTP requests to fool the web servers to believe that the requests were made from different devices.

The method used contains mainly of two tools. The first one is GNU Wget (version 1.11.4 for windows), which is a program that downloads files from the web without any interaction from the user [15]. It supports both HTTP and HTTPS and can follow and download links from HTML files. This software is free and easy to use by either using scripts or terminal. This tool satisfy our requirements for downloading a website using a list of an arbitrary number of URLs as input which is the reason we chose to use GNU Wget.

For our setup we added some flags when running Wget, just so Wget would do what we wanted it to do. Flags are options to set for fetching a site. There are many flags available and Wget is able to use as many flags as needed in one running session. The first flag we added was ”-p” which causes Wget to get all the files including images and scripts (etc.) which is needed to view the web page like a web browser would. To change user agent to our three different platforms in Wget we add the flag ”-user-agent=<string>“, the default is ”Wget/version”.

To make sure Wget does not fetch data from a local cache we add the flag ”-no-cache” and to make Wget load URLs from a list we add the flag ”-i”. When we are downloading web pages in large scale we do not want to let Wget retry to download a web page which is inaccessible for multiple reasons so we limited Wget to retry five times as maximum. Downloading the top 500 web pages and 100 pages for each category by three different platforms will take up quite a lot of space on our test machine which is why we added the flag ”–delete-after”. It makes Wget remove the files after downloading.

The second tool that was used is Wireshark (version 1.10.6), which captured the packets generated from Wget while it was sending GET requests to all the web servers. It allows us to analyse and filter HTTP packets. One of the reasons we chose it was because it had been a main tool for the laboratory work in a previous network course we took (TDTS11 at Linköping University). Wireshark also allows filtering on content type to distinguish if the HTTP packet was getting an image or a text for example. Since Wireshark captured all HTTP packets while
Wget was running we had to make sure that our test machine was not running any other programs that sent HTTP packets which would make the traces infected with non-relevant HTTP packets.

Before starting the data collection, we had to fetch the top 500 websites from www.alexa.com and since it only showed 25 URLs per page, it was needed to fetch the HTML files from index 0-19 of www.alexa.com/topsites/global/index and merge the downloaded HTML files to one [16]. We then made a regular expression to get all the URLs from the merged HTML file. The same method was used to fetch the top 100 websites from the categories: Art, Business, Computers, Games, Health, Home, Kids and Teens, News, Recreation, Reference, Regional, Science, Shopping, Society and Sports. The use of top-500 websites and top-100 websites per category is motivated by the heavy skew of website popularity [2]. It tells us that popular websites are getting more and more popular and takes up more of the HTTP-traffic and affects more users from around the world. Our choice of websites is also similar to related work [3] [4] [5]. This was performed from the 17th of April 2014 to 22nd of April 2014.

### 3.2 Filtering

For each category and user agent we had a trace of captured packets. For all the captured traces, it was needed to decide what information we wanted to analyse. Related work [3] has been looking in to certain content types. These were images, JavaScript, CSS and Flash. Therefore we filtered our traces using WireShark’s own filtering function on these content types as well, to be able to compare some of our results with other work. We are also excluding HTTP packets which have other response code than 200, so the focus would be on the packets which are successfully downloaded.

For our analysis we compare the mean number of objects downloaded per website and the mean number of bytes downloaded per website for each category and user agent.
Chapter 4

Analysis of results

Other work came to the conclusion that load times are more affected by the amount of objects fetched per site rather than the amount of bytes fetched to render a website [3]. Hence, we will put more emphasis on the comparison between downloaded objects rather than downloaded bytes. We will also use the top 500 category as a reference for the other categories. This is due to the conclusions from related work which tells us that popular URLs are getting more popular and therefore makes up for an increasing fraction of the web traffic [12].

The structure of the results and analysis have been split up in to two sections. In the first section, we analyse the website composition, such as category comparison. In the second section, we analyse and compare the adaptation of the websites to the different devices.

4.1 Website composition

Consider first the file types that make up a website: Figure 4.1 shows how the average website, for each category, looks like in term of the total bytes of each of the major content types, as seen with the Chrome browser. For example we see that the average website in the Sports category consists of 7.14% CSS, 0.121% flash, 75.3% images and 17.4% scripts.

Figure 4.2 shows the average percentages of all categories for each device is shown. For example, for the Chrome experiments the average website (across all websites) consists of 8.78% CSS, 0.108% flash, 69.8% images and 21.3% scripts.
The figures clearly shows that the content type image, is the one which takes up most bytes from the analysed websites out of these four content types. The size of the content type scripts is the second largest, however usually it is less than 50% than the amount of bytes images make up of. The content type CSS is the third largest, and also there, CSS is usually less than 50% than the amount of bytes scripts make up of. Lastly, flash, have in many categories not been used at all and in some categories only couple of websites uses it which make it seem like nothing in the charts.

Noteworthy when looking at Figures 4.2 and 4.1 is how the website composition differs a lot between categories but when looking at the average of all categories, the composition does not differ much between devices.

In the same way as the figures above, we created two figures that shows the average amount of objects downloaded from a website. To continue with our example for the average website in the Sports category and observing the Figure 4.3, around 4 objects of CSS, 0.04 of flash, 41 of images and 10 scripts was downloaded per website. From the Figure 4.4 can the average amount of objects downloaded, per website, by each device be observed. To demonstrate with an example, a website contains 2.8 CSS, 0.03 flash, 22 images and 7 scripts objects, using Nexus 7 as device.
Figure 4.3: The average amount of objects downloaded per website by each category with Chrome as user agent

Figure 4.4: The average amount of objects downloaded by each device

From the Figure 4.3 we see that the average objects downloaded by top 500 websites is around 2 CSS, 1 Flash, 27 images and 5 scripts per site. Since top 500 is the reference category, the conclusion is made that if a category have similar numbers for their top 100 sites, the websites of the category has similar structure as the websites for top 500. A few of the categories have similar structure as the top 500. However two categories have much less images, the Computers category have only 8 images per website and the Reference category have 14.

Our results from the News category is similar to a related paper, where News was the category with the most downloaded objects from the content types images, scripts, CSS and flash between the categories Business, Games, Kids and Teens, News and Shopping [3]. That conforms with our results where News had the most downloaded objects from these four content types amongst the before mentioned categories.

The results show that Chrome as user agent, has the largest number of objects downloaded, followed by Nexus 7 and lastly iPhone 4. This is also another proof of the fact that the server actually adapt the web pages to different devices. The difference of the adaptation between devices will be discussed in next section.

4.2 Device comparison

The content type images is the main focus for our result analysis. Figure 4.4, in previous section, shows that the number of images is reduced most by user agent
iPhone 4. The amount of scripts and CSS is also lower, but not much at all. Flash, however, was only in very few cases less in iPhone 4 and Nexus 7 than Chrome since it already was close to zero. We will also take a closer look at images at a per-category basis. For that reason, the Figures 4.5 and 4.6 were created. These figures simply show the average amount of images a website consist of, in numbers of objects and bytes, downloaded from each device.

Figure 4.5: Average amount of images fetched by the categories

Figure 4.6: Average amount in bytes of the images fetched by the categories

Earlier we stated that this thesis will be focusing on the amount of objects downloaded rather than the amount of bytes downloaded, which Figure 4.6 shows. But the reason we made this figure is to show that the bytes downloaded is proportional to the amount of objects downloaded. Two categories which stands out are Sports and News which can be seen in Figure 4.5. These two categories clearly contains the largest amount of images of all categories. They both have a average higher than 40 images per website with the Chrome user agent. The difference between them are the other two user agents where Sports adapt more by containing just 22.89 and 26.82 images on average and News is slightly less adapting by containing 27.47 and 34.67 images on average for iPhone 4 and Nexus 7.

To be able to compare the differences between the devices and categories we made the Figure 4.7. This figure shows how much less images, in average per website and category, is fetched from iPhone 4 and Nexus 7, in percentages, compared to Chrome. For instance, if you visit a website in the Sports category using iPhone 4, the website contains almost 45% less images than if you would visit the same website using a computer with Chrome as browser. Likewise, using Nexus 7, you would get 35% less images.
The amount of images that is reduced varies, from the Figure 4.7 we observe that the maximum is close to 50% less than the amount we get from the Chrome user agent.

Two special categories are Computers and Kids and Teens. The Computers category looks like it have been adapted well by looking at the Figure 4.7 and compare it with top 500. But by observing the two Figures 4.3, we see that the Computers category have only less than 9 average images per website for Chrome. This means that the Figure 4.7 by itself is misleading, because of the small numbers of images downloaded.

The Kids and Teens category is special because as we can see in both Figure 4.5 and Figure 4.7, this is the only category where iPhone 4 have the biggest number of images downloaded compared to Chrome and Nexus 7. Nexus 7 had the least amount downloaded, however the difference between all three of the user agents is small. For this reason, the conclusion is that this category have been adapted least by any of the mobile clients.

Nexus 7: Figure 4.7 above, shows that the average numbers of images downloaded to Nexus 7 is around 16 – 17% less than Chrome. Since we are using the top 500 data as reference for the other categories, then if a category have reduced the amount of images downloaded from Nexus 7 is around 16 – 17% or more, then the category have adapted well to Nexus 7.

As we can see in the same figure, the categories Art, Computers and Reference have been adapted the least. the Art category have average 32 images downloaded per website, using chrome, (see Figure 4.5) and only 5% less than Chrome, which means that in average there is only slightly more than one image less per website using Nexus 7 than Chrome. Because of the small numbers of images downloaded for Computers, as mentioned in the section above, this category also only downloads slightly more than one image less per website using Nexus 7 than Chrome. The Reference category is 4% less than Chrome and have average almost 14 images downloaded per website for Chrome, meaning it’s less than one image difference compared to Chrome.

The categories Health, Home, Shopping and Society have also not adapted well compared to top 500 websites. Each of these categories have around 10 – 11% less images as average for Nexus 7 compared to Chrome.

The rest of the categories, except for the two special ones (Computers and Kids and Teens categories) as mentioned above, have been adapted in the percentages as top 500. The best one is the Sports category, which have more than 35% average
images less than Chrome. 41-42 images was downloaded by using Chrome as user agent which means that almost 27 images was downloaded by using Nexus 7 as user agent.

**iPhone 4:** Similar reasoning as for Nexus 7 tells us that if a category have reduced the amount of images downloaded to iPhone 4 is around 37% less than Chrome then the websites in these categories are well adapted to iPhone 4.

The least adapted category is the same as it is for Nexus 7, the Reference category. In fact this one is even worse than the Computer category since of only 14 images, iPhone 4 have only 11% less, which means that there is not even 2 images less per website. While the Computer category have a little more than 2.

Other categories that are a bit lower than our reference 37% are Art, Games, Health and Science (Figure 4.7) which have between 22 – 26% less images as average for iPhone 4 compared to Chrome.

The rest of the categories have been adapted well and the Shopping category have adapted the websites best to iPhone 4. There are over 50% less images in iPhone 4 compared to Chrome. An average of almost 27 images was downloaded per website for Chrome and therefore almost 13 images for iPhone 4.
Chapter 5

Discussion

5.1 Results

From the results we can clearly see that there is a significant difference between categories regarding how they adapted to both of the mobile clients. The categories which adapted their mobile websites more than top 500 are Business, News, Recreation, Regional and Sports. By overall we think that the Sports category have been adapting best. We believe it is because this category contains many popular sports sites with many images on its front page for desktop user but having the same amount of images on a mobile client would make website hard to get an overview of. Unlike e.g. the Kids and Teens category where most of the websites are created for young people and might use a lot of images on all platforms because the information is carried with images rather than text. Another reason could be that the Kids and Teens category does not have the same high traffic as Sports which leads to a discussion whether more popular websites adapt better than less popular websites or not. According to related work, the popularity of a website is less of an indicator of its composition than the category it belongs to [3].

The results also shows that a website fetched from iPhone 4 includes less objects than Nexus 7 and Chrome. Of course there can be many possible reasons for that. One reason could be that iPhone 4 have the smallest screen size followed by Nexus 7, and therefore less objects than Nexus 7 which have less objects than Chrome for desktop. Another reason is that the iPhone 4 could be considered more mobile than the Nexus 7 since it is a smart phone and the Nexus 7 is a tablet. This means that the tablet is probably going to be connected to a WiFi and/or a battery charger more often than the smart phone.

The accuracy of our results is mostly depending on Wget’s performance of fetching all content of the web pages. To improve the accuracy of the results further a real browser would be needed to ensure that the fetched resources are for the rendering of an entire web page.
5.2 Methods

The first method we planned to execute included a program which used a remote Chrome instance to visit a website and continue to another when the first one was complete. It could also give us a report per website with relevant information about what was fetched, how large these files were. Since it would give us reports about every single website, we would be able to compare our results with other work with greater confidence. However, when we started to scale up our tests this program was unable to load a list with large number of URLs [3].

We scrapped that method and looked for another script or program that could visit websites sequentially with a large list as input. We chose Wget because it had both a Windows and a Linux version. Wget would not do anything else but fetch the files required to load a website. We used Wireshark to fetch all HTTP GET requests made by Wget. Wget did not have a function to make a timeout between URLs, so it meant that we could not find a way to automatically sort the HTTP packets by website on the Wireshark traces. Therefore, we had to take the average on each trace which might have given us misleading results. Some websites might not have been available during the time of our tests which would lead us to believe there were zero images, flash files, javascripts and CSS files and give us a lower average result of these content types of the entire trace. If we used a method which let us get per site information we would know which websites were not reachable and we could present our results in median instead of mean.

5.3 Impact on user

Server side optimisations can of course impact the end users in various ways. The end users could save money on these optimisations because some of the users need to pay for the amount of data while using the Internet on 3G/4G and if there is less data to download from a website then the users can save money. But it is important not to reduce too much of the objects, because it may result to become harder for the end users to find the information they seek. For example if one person finds something important while surfing on the laptop, then visit a friend and want to show him this information using his smart phone, it is important that the information still is easy to find.

The server side optimisations can also make the smart phone’s or tablet’s battery last longer since downloading data via 3G/4G or WiFi uses a fair amount of energy. Fetching a website which has been adapted on the server side can also make the loading times shorter for the mobile devices compared to an unadapted website that may contain some heavy files which may take some for the mobile devices to download.

To optimise websites might also attract more users than non adapted websites. Because people prefer to be able to navigate easy, with not to big images or too much text. On a computer it is normal that a website contains much text and big images because of the size of the screen. But by using smaller sized mobile clients, the users don’t want to see massive text walls or pictures that would require a lot of scrolling vertically and horizontally.
Chapter 6

Conclusions and future work

The increasing use of mobile devices has led to a need for websites to adapt for different platforms. This thesis presents a reasonably large scale measurement study of how different websites, from different categories, adapt to mobile platforms. We have concluded that different categories can differ a lot in how they adopt their websites for mobile platforms. We also saw that there was a difference in website complexity between the categories. A possible reason to this can be the target audience which the website creators are developing for but also what type of information the website is trying to convey.

For future work we would want to improve our method of capturing the HTTP packets to allow per-website analysis. Also, instead of using Wget for fetching all the website content, we would like to write a script that makes a remote Google Chrome instance visit a list of web pages. Instead of using WireShark for the data collection it is possible to use Google Chrome’s developer tools can be used to emulate other devices and has the ability to create HAR files for a visited web page. HAR files are easy to analyse which is necessary when dealing with large amounts of data where a WireShark trace might be less easy to analyse because of all the unwanted packets which also are captured by WireShark.

Server-side optimizations at each website does not have to be the only focus of future studies. Other inputs, such as caching also plays a major role in the mobile web. It can make a significant difference for the end-user performance and may vary a lot depending on location, content provider and their CDN-infrastructure. Future work can therefore consider testing from various locations [17].

Other instruments would also be an option to consider. Performing these tests on the actual devices rather than changing user agents could tell us the difference in load time of the websites for different locations and/or networks. Issues like that the tools might drain the batteries of the mobile devices would require to be treated.
Bibliography


På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© [Milad Barsomo, Mats Hurtig]