Mobile Web Performance

Version of 06 Dec 2011

Paul Picher, Matthias Rella, Farhan Sahito and Mario Zupan

Abstract

Web performance in general is a big issue, particularly on mobile devices the performance is even more important. The connection to the Internet is not the major reason for low performance on websites, the way how a website is implemented is more important than the internetconnection. This survey will give a short overview on Mobile Web and how it is used and where the mobile devices are limited. Furthermore a detailed view on the best practice rules for web performance will be given and discussed in detail with given examples. In the last section will be a detailed view on two tools to evaluate the mobile and the desktop version of the popular website Amazon.com.

Contents

Chapter 1

Introduction

Mobile Web is a paradigm in recent years with approximately 726 million users having access to it with a forecasted increase growth in it. The present period of mobile Web revolution provides a big opportunity for businesses to capitalize on the mobile marketing strategies to connect with large population and generate huge traffic. It also provides around-the-clock access to the Internet users regardless of location and in recent times, we have seen the tremendous power of the mobile Web as a tool for social change which became a modern revolution in the Middle East and North Africa. In spite of all benefits, mobile Web also has its share of challenges that present unique problems due to that reasons the performance is falling short of expectations. This survey paper will first explore the definition of mobile Web with its history. In the first chapter this paper will spot the benefits mobile Web can bring such as constant connectivity, limitless access and interactive capabilities and the revolution it has bring in the country like Egypt. The second part of this chapter will focus on the limitation and challenges related to mobile Web and explain the problems users are facing with this emerging technology. This chapter will finally discuss the performance and optimization and will define that how Performance is the crucial to the success of mobile Web and how it can meet with user expectations. Chapter 2 is concerned with Best Practices for Web Performance Optimization and have a look at issues with mobile where needed. The last chapter will deal with the evaluation and comparison of two web performance measurement tools.

Chapter 2

Mobile Web

The Mobile Web refers to accessing and interacting with Web content or the use of Internet-connected applications (connected to a wireless network) from mobile devices, for instance, tablet computer or Smartphone [W3Mobile, 2008]. Today it is a most popular, rapid and widespread personal technology allows Internet use away from a computer, completing the communication continuum and constitutes the entirety of the Internet [MogisticCondition, 2007]. In 1996, the first access to the mobile web was commercially offered in Finland on the Nokia 9000 Communicator phone on the Radiolinja and Sonera networks [NokiaFirst, 2011] and in 1999 the first commercial mobile-specific browser-based web service was launched when NTT DoCoMo introduced i-mode in Japan [Telecomvisions, 2011].

The growth of this technology is staggering, with approximately 726 million users having access to a mobile Web with a forecasted increase growth in it [Matt Murphy, 2011]. According to a recent study, of the top 100 Web sites most visited on the internet, it is interesting to see that 71 of them have content specifically designed for mobile devices. The remaining 29 either prompt the user about a device specific application or do not support mobile devices [QuantcastTop, 2011]. This study further reveals that within the top million most visited sites on the Internet, approximately 39,000 sites (3.9%) reported the content as non-scaleable. It is a strong indicator that the Web content is designed specifically for a mobile device and only 6% of the sites reported that they are at a minimum, aware of mobile users [QuantcastTop, 2011]. These numbers illustrate the potential of the mobile platform as the right solution compared to other existing and emerging technologies and the increasing usage is predicted more websites will support mobile devices in the years to come [BuildwithMobile, 2011].

Mobile Web is a paradigm shift from how the web has been built in recent years and website developers and designers are beginning to spot the benefits it can bring [BuildwithMobile, 2011]. According to Maximilien, today, there are more mobile computers than there are any other forms of computing devices [Maximilien, 2008] and Morgan [2009] further argue that, due to the convergence of five advanced trends mobile Web is nowadays ramping faster than desktop Web such as: 3G Internet connection, video, VoIP, social networking, and high-end mobile devices. According to him it is expected that within 5 years mobile devices will become the number one gate to access the Web due to these trends and the number of devices such as smart phone or tablet computer will be over 10 times the number of fixed devices accessing the Web [Morgan, 2009]. The recent growth and production of devices such as: the BlackBerry, Apple iPhone, Microsoft Windows Phone and Google's Android Stack, supporting new methods of interaction with website with high resolution [BuildwithMobile, 2011]. In spite of many challenges, limitations and hardware constraints the mobile Web suffer, the revolutionary Web browser developed by Apple for the Safari mobile, iPod Touch, iPhone has been proven to be a user-friendly way of surfing Web [Maximilien, 2008].

2.1 The Benefits of the Mobile Web

Some of the more important benefits of the Mobile Web are:

Business and Marketing: The present period of mobile Web revolution provides a big opportunity for businesses to capitalize on the mobile marketing strategies to connect with large population and generate huge traffic. The rise of Smartphone and tablets devices has also made a multi-faceted web presence to reach wider audience and its usage has made mobile web presence a competitive necessity for every business in the world [BusinessWeek, 2008; NubiqGrowing, 2011].

Constant Connectivity: Mobile Web provide around-the-clock access to the Internet regardless of location. Devices like smartphone provide a continual link to the wealth of information such as users can verify on the status of a flight while on road to the airport, check traffic reports, and even access a map and directions if they decide to take an alternate route [Kroski, 2009].

Limitless Access: The mobile Web encompasses the whole Web and online resources that they would find via their desktop computer, not only those Web sites which have been designed for mobile browsing [Kroski, 2009].

Interactive capabilities: The world of mobile web holds immense potential as it is an easy and convenient way to get access to information on the run such as looking at the price of stocks, reading news and accessing email are typical examples of the kind of activities that users perform while they are mobile. Users can also tag resources, write blog posts, and form connections on social networks [Kroski, 2009].

Web 2.0 and Mobile Web: Web 2.0, the next generation web is considered to be as the next big wave riding in the field of mobile computing that use web as platform to provide users a rich experience as they enjoy on desktop with all characteristics available on personal Web. This is coincides with W3C vision for the web-The Web Anywhere, for Everyone, at Anytime, on Everything [Hiren, 2008].

Revolution and the Mobile Web: In recent times, we have seen the tremendous power of the mobile Web as a tool for social change which became a modern revolution in the Middle East and North Africa – in countries such as Egypt, Tunisia, and now Libya. More and more users are becoming accustomed to the mobile Web as their primary news source to get up-to-date information in hot spots around the world and organizing themselves in collective action or communicating their message at large, and garnering support from people in far away nations. According to a survey there was a sharp uptake in Egypt the use of the mobile Web and a sustained increase in traffic to certain sites due to the circumstances in that country during the month of February. According to Ayman Shokr, "The key success for the revolution was the way we connected, via Facebook and other social media via Internet, to mobile phones," The mobile Web provided a platform for those voices in the midst of conflict that might otherwise not have been heard [OperaMobile, 2011].

Performance has always been crucial to the success of Mobile Web sites. Users on mobile Web expect performance to be good as what they experience on their work or home computer. However, with its series of benefits, mobile Web also has its share of challenges that present unique problems due to that reasons the performance is falling short of expectations [OperaMobile, 2011]. Studies have indicated that even small delay in page load times lead to less ad revenue, less stickiness, and less customer satisfaction. According to Zhang [2005] the issues associated with mobile devices are a function of the devices and standard usability issues raise challenges for Web designers [Zhang, 2005]. Challenges such as fragmentation of mobile devices, lower bandwidth, higher latency, smaller memories, and lower processing power of mobile devices have imposed a heightened need to optimize site performance to meet consumer expectations [Zhang, 2005].

2.2 Limitations and Challenges

Fragmentation of Mobile Devices: It is evident that mobile Web access still suffers today from usability and interoperability issues. Usability issues are focused around the physical size of the mobile devices such as limits on display resolution and user input/operating. Interoperability problems are centered on the platform fragmentation of mobile devices such as browsers and mobile operating systems [VisionMobile, 2011]. On the other side the performance of mobile Web is a consequence of fragmentation in the market of mobile devices. For example, a number of different operating Systems available these days fro smartphones, such as Android from Google, Windows Mobile from Microsoft, Symbian and Maemo from Nokia iOS for Apple's iPhone, iPod and iPad, WebOS from Palm (now HP), Bada from Samsung and Blackberry. These different Application Frameworks incompatible between each other and unfortunately any application develop in one platform does not work in any of the other platforms [Cristobal, 2010].

Due to device fragmentation, mobile Web developers encounter challenges when creating content for these devices but usually these new channel are confronted with numerous development standards and with varying functional capabilities, distinct mobile browsers, sizes as well as screen resolutions. This lack of uniformity cause universal compatibility a near impossibility [Kroski, 2009]. According to Passani [2010], due to this device diversity the mobile Web development has become a very complex issue as different devices have different features and different dimensions such as hardware characteristics, network characteristics, multimedia formats and behavior of the browser. He also argued that device fragmentation is the consequences of several aspects such as:

- Rapid innovation, spread and advancement in underlying mobile technologies and standards.
- Mobile device manufacturers' drive to differentiate from market competitors.
- Network operators' request to have mobile devices that are mostly customized to their networks.
- Device software is typically not upgraded and poorly performing software stays around longer.
- Mobile devices require an average 18 months to change the first device in the business market [Passani, 2010].

Small Screen: Mobile devices have smaller screen so navigating with a mobile device is not very pleasant experience and this is the most visible difference between a mobile device and a desktop. Mobile phone screen sizes have also been increasing since few years, but even the screen of the iPhone 4 is still small in comparison to a standard 1024x768 desktop that inadvertently break the user experience by making zoom in and out. Similarly, most mobile displays have fewer pixels than desktop ans users spend more time attempting to locate information rather than simply browsing. An iphone retina display is 960x640 and iMac with its smallest screen has 1920x1080 display [Passani, 2010].

Limited Input Capability: One important aspect of mobile devices is touch based input rather using mouse or keyboard to get access to mobile Web. While this is sufficient for dialing phone number or entering text but with Web it is an annoying activity, when users have to type on a fiddly little plastic keyboard or tapping a minuscule on screen keyboard. Most modern touch devices allow the user to perform gestures using one or more fingers, such as using pinching, swiping and so on. For instance, users swipe left to right to move between folders in a gallery. jQuery mobile with JavaScript framework can generate events for such gestures to support mobile Web [Maximilien, 2008; Passani, 2010].

Slow processors: Mobile devices generally have less processing power and smaller caches than desktop computer. Thus Web browsers take longer to render pages and payloads. Mobile browsers are slower to parse HTML and JavaScript intensive pages can run very slowly [Cristobal, 2010; Passani, 2010].

Less Bandwidth: Mobile devices have little bandwidth available when compared with desktop. While cellular network speeds are improving and with the advent of 3G mobile devices (EDGE, UMTS, HSDPA) the situation is improving for some users, a typical 3G device will be lucky to get more than 1MB/s download rate: Some time lot of users can count on a speed down to just a few KB/s compared to between 1.5MB/s and 20MB/s for ADSL link and even faster for Wi-Fi access [Passani, 2010].

Limited multitasking: Multitasking describes user performance of being able to run more than one application at once or switching of application from one task to another. Unfortunately many devices still cannot multitask as desktop. Desktop allow you to see several app windows at the same time. Some mobile devices such as iPad and iPhone 4 allow you to swap between running application by double clicking the Home button, but sometimes a limited number and not multiple windows in the same screen; it is also not as quick as desktop [Morgan, 2009].

No, or Poor, Flash Support: Flash is almost ubiquitous on the desktop with more than 90% browser installed with Flash player. However, the story is different with mobile Web as no iOS devices run Flash. Devices like Android with version 2.2 or later can run Flash but it causes performance and stability problems and many users choose to turn it off as it is usually not a pleasant experience [ElatedTen, 2011].

2.3 Performance and Optimization

Performance is the crucial to the success of mobile Web and above limitations have imposed a heightened need to optimize site performance to meet user expectations. Studies have indicated that small improvement in page loads lead to more customer satisfaction and more sales, and no matter how beautiful, interesting and interactive Web pages are, users become impatient if Web page is taking more than 2 or 3 seconds to render [StrangeloopMobile, 2011]. Users expect performance as good as desktop computer, despite the slower networks and lower-powered hardware and they are less likely to convert from browsing to buying in delay and may even close the browser before the page ever loads [StrangeloopMobile, 2011]. The Harris Interactive Mobile Transactions Survey in 2011, reported that (mobile site optimization) 85% of users who had made a mobile transaction in the last year expected the mobile experience to be better or equal to shopping online, and 63% reported that they would be less likely to buy from the same Website if they experienced a transaction problem on their mobile devices [StrangeloopCustomers, 2011]. Another study published by Equation Research in February 2011 stated that 46% of users reported that websites on their phone load more slowly than expected and nearly 60% mobile users expect load pages in 3 seconds or less. Almost 74% said they would leave a site if a Website took 5 seconds or more to load. This research also reveals that 77% of the top companies have mobile page load times of more than 5 seconds.

Eric Schurman (Bing) and Jake Brutlag (Google Search) also co-presented results from latency experiments in Velocity conference-2009 in San Jose [OReillyFast, 2009]. Eric Schurman from Bing found that a 2 second delay changed searches/user by -1.8% and revenue/user by -4.3%. Jake Brutlag from Google found that a 400 millisecond slowdown resulted in a -0.59% change in queries. Google further revealed after the delay was solved, users still had -0.21% fewer queries that indicates long term behavior in case of slower user experience [OReillyFast, 2009].

Phil Dixon, from Shopzilla presented a case study in Velocity 2009 that provides real world numbers that show the benefits of making mobile Web faster. Phil Dixon presented several suggestions about the impact of performance on the bottom line with most takeaway statistics. He concluded that a 5 second speed up (from 7 sec to 2 sec) was resulted after a yearlong redesign performance that further resulted win-win of performance improvements with 25% increase in page views, a 50% reduction in hardware and 7-12% increase in revenue [OReillyFast, 2009].

Chapter 3

Mobile Web Performance Optimization

This chapter will explain best practices for optimizing web performance and pay additional attention to mobile issues where needed. Most of the ideas and approaches presented here originate from Souders [2007], Souders [2009] and from a bunch of Google Page Speed optimization rules [GoogleBestPractices, 2011]. The chapter is divided into three sections: Roughly spoken the first is concerned with the reduction, the second with the parallelization and the last with the size of requests. Tackling these three parameters affects client-side web performance most effectively.

3.1 Making few (HTTP) requests

A web page usually is built up of several more or less independent components. The bare HTML document may be the most important one but effectively not the first to concern with when it comes to web performance optimization. To mention only the most common components, there may be images, stylesheets and script files included into the page by referencing an external sources. Downloading many instances of these through separate HTTP requests will likely result into a slowly building up web site. Especially on mobile devices network latency usually is quite high and the overhead for each of many requests can become a dominating factor in the total downloading time.

For example assuming a round-trip time (RTT)[1] of 100 ms for a web site with 20 external resources (images, stylesheets and script files) 2000 ms of the total downloading time will be spent on network overhead without a single byte representing content being transferred. Assuming a bandwidth of 1 Mbs and an average size of 30 KB per resource will result into the content being downloaded in estimated 5 seconds. Hence, nearly a third of the total transmission time is "lost" for not being used for content. Therefore the probably most important point in speeding up page loading is the reduction of overhead by making as few HTTP requests as possible.

However, requests for components are not the only ones which cost loading times. Redirects and DNS lookups also affect client-side requesting behaviour on different levels of network communication. Before focussing on these, this section deals with performance techniques on the application level, i.e. concerning static resources.

3.1.1 Combining Images

There are three ways of optimizing requests for images: image maps, CSS sprites and inline images. This subsection will summarize these approaches according to Souders [2007, chap. 1].

Image maps can be used for parts of a web page which contain several links near to each other with each being decorated by an image. Navigation bars are the most prominent examples for such page components (see **??**).

[1]The round-trip time is the time needed for sending a request and receiving a response without taking data transfer of the payload into account.

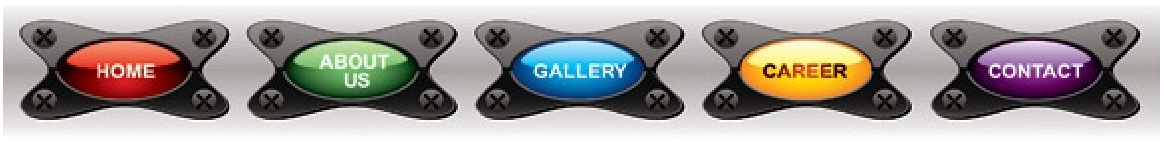


Figure 3.1: Several adjacent images in a navigation bar for linking purposes [YouToArtNavBar, 2011].

Instead of using separate images for each of the navigational buttons and hence producing separate HTTP requests, one could transmit the images as a whole and apply the HTML `MAP` tag to it. Usually the map will associate certain areas within the image to links or Javascript event handlers.

```
<img usemap="#map1" border=0 src="/images/navbar.jpg"/>
<map name="map1">
   <area shape="rect" coords="0,0,116,70" href="home.html"/>
   <area shape="rect" coords="116,0,232,70" href="about.html"/>
   <area shape="rect" coords="232,0,348,70" href="gallery.html"/>
   <area shape="rect" coords="348,0,464,70" href="career.html"/>
   <area shape="rect" coords="464,0,580,70" href="contact.html"/>
</map>
```

This technique can be leveraged to map circular and polygonal areas too by `shape="circle"` or `shape="poly"` respectively.

CSS sprites function in a similar way but without being restricted to adjacent images. In fact, sprites can combine all types of presumably non-photographic images together, for instance decorative shades, background tiles and icons. Leveraging width, height and offset attributes in the CSS styling of an element web designers can cut out pieces from the sprite and display it separated at any location of the web page (see 3.2).

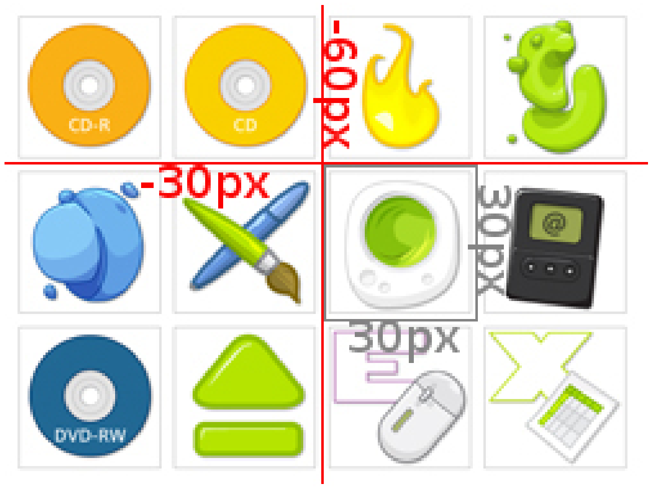
Besides the reduction of HTTP requests and a more flexible markup with CSS sprites compared to image maps Souders [2007, chap. 1] mentions the additional advantage that sprites typically need less file size than the sum of the combined images as such. This results from the fact that a single image contains a more or less fixed amount of compression specific meta-data in the header field of the file. As with sprites several images can share this overhead without reduplication.

Furthermore GoogleRTT [2011] recommend to combine images to sprites, if they

- are loaded together on the same page, which will result in the most efficient use of the image.
- can be compressed efficiently using GIF or PNG for the borders between the pieces of images will not be blurred as with JPEG.
- are cacheable and do not change dynamically with each page view.
- rely on similar 256 color palettes because truecolor PNG will result in a larger file size of the sprite image.
- are small for the per-request overhead for each of them will be eliminated most effectively.

Inline images relieve the web designer from referencing the resource by an external URL with an `http:` scheme. Instead the `data:` scheme can be applied which contains the raw data of the image, for example:

Nowadays most browsers support this type of URL scheme including Internet Explorer (IE) 8 and 9. IE browsers up to version 7 do not. Inlining is well suited for dynamic and/or small images which must not be cached. However, one can enforce caching the data by writing the `data:`-URL into a cachable CSS file like in the following example and leveraging both performance benefits of caching and request reduction.



```
<style>
#icon {
  height: 30px;
  width: 30px;
  background-image:
    url("mysprite.png");
  background-position:
    -60px -30px;
}
</style>

<span id="icon"></span>
```

Figure 3.2: Icons being combined in a single sprite image [YouToArtSprite, 2011] and displayed by CSS.

```
<img src="data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAAUA
AAAFCAYAAACNbyblAAAAHElEQVQI12P4//8/w38GIAXDIBKE0DHxgljNBAAO
9TXL0Y4OHwAAAABJRU5ErkJggg==" alt="Red dot"/>
```

Figure 3.3: An image of a red dot displayed using a `data:`-URL [WikipediaDataURL, 2011].

```
#image {
   background-image: url(data:image/png;base64,JDKFIENCKDFO ...);
}
```

One considerable drawback concerning performance is mentioned by WikipediaDataURL [2011]. Base64 encoding maps binary data to a subset of URL safe characters which results in a data stream which is approximately one third larger. Nevertheless, this can be overcome by using GZip-compression.

3.1.2 Combining Scripts and Stylesheets

As with images externally referenced Javascript and CSS files can be combined together into one file to minimize the amount of HTTP requests. While being good development practice to split up code into separate modules clients usually do not care for the modularity of a stylesheet or a script. Web developers therefore should implement techniques to concatenate modules to one file on the server side. Souders [2007, chap. 1] exemplifies the gain in loading times through an experimental setup and states a 38% faster response.

Furthermore GoogleRTT [2011] recommends to split up scripts and stylesheets into two files: one for code which is inevitably needed at startup for rendering the page and one for the rest of the logic which may not be needed immediately.

3.1.3 Caching

After combining scripts and stylesheets the next step in speed-up web sites may be the adoption of reasonable caching techniques. Some static resources like images or other external files which do not change often can be taken into account for caching, i.e. storing and retrieving them locally instead of kicking off additional requests.

As for the HTTP header there are several ways to induce client-side caching. Since HTTP/1.0 there exists the `Expires` field accompanied by a fixed future date denoting until when the cache of the requested resource will be valid. However, for being limited to a fixed value the expiration date has to be reset to a new future date as soon as the component is retrieved again. A server side module like `mod_expires` can be leveraged to generate such dates automatically:

```
<FilesMatch "\.(gif|jpg|js|css)">
   ExpiresDefault "access plus 10 years"
</FilesMatch>
```

Figure 3.4: Automatic generation of future dates through mod_expires directives [Souders, 2007, chap. 3].

Beneath a field like `Expires: Sun, 5 Dec 2021 23:18:33 GMT` this directive will also add a `Cache-Control: max-age=315360000` to the HTTP header. This field has been introduced in HTTP/1.1 and provides additional cache control. In this example it overcomes the fixed date restriction of `Expires` and sets the number of seconds relative to the time of requesting until when the resource should be cached [Souders, 2007, chap. 3].

With the coming of HTML5 more flexible caching techniques are available on the client side. On the one hand web programmers can make use of a cache-manifest where cacheable resources can be listed and on the other hand different forms of persistent storages can be leveraged like the so-called WebStorage, the Web SQL database and the IndexedDB.

3.1.4 Avoiding redirects

Especially with high latency times avoiding redirects can be an effective way to enhance web performance. Redirects take a whole RTT but do not serve any content to the user. Therefore removing them will not affect the appearance of a web site. Souders [2007, chap. 11] states that the most useless redirect is the one caused by many web servers which add missing trailing slashes to URLs to match the root of a given directory and hence support auto-indexing. However, in many cases this feature is not needed and can be turned off using the appropriate directives in the configuration of the server.

Both Souders [2007, chap. 11] and GoogleRTT [2011] mention the common use of redirects for logging page views. In this case users are accessing a resource which actually performs some logging or tracking logic and afterwards redirects the user to the actual content. This request flow can be avoided either by implementing the logging function within the application logic on the server side or inserting a Javascript snippet at the end of the page which will request the logging resource asynchronously.

When it comes to mobile, GoogleMobile [2011] draws attention to the practice of many web sites to redirect mobile clients to designated versions of the page, for example www.example.com to m.example.com. One could work around this redirect by either removing it and serving the content of the mobile version instead or by making the redirect cacheable. GoogleMobile [2011] recommends to add `Vary: User-Agent` and `Cache-Control: private` to the HTTP header of the redirect. The next time the mobile agent will try to access the desktop version of the web site its client should retrieve the hostname of the mobile one from cache redirect the agent internally.

3.1.5 Minimizing DNS lookups

In the retrieval of a web site DNS lookups show up as an additional request with the full RTT being involved. Therefore avoiding the DNS request may be a feasible approach to reduce overall response times. However, according to Souders [2007, chap. 9] caching of DNS queries resides outside of a web designer's control. It are the Internet Service Provider (ISP), the operating system and the browser which decide upon the Time-To-Live (TTL) of DNS lookups. "The maximum TTL values sent to clients for 10 top U.S. web sites range from one minute to one hour" [Souders, 2007, p. 65].

However, there are two ways of controlling DNS requests from the application side. The first is to reduce DNS lookups simply by reducing the number of different hostnames appearing on a page and referring to external resources. GoogleMobile [2011] recommends further to use URL paths instead of hostnames, for example www.example.com/developer instead of developer.example.com. The second technique Souders [2007, chap. 9] proposes is to leverage the `Connection: keep-alive` HTTP header field which advices clients to keep sockets open for further connections and avoid repeating overhead procedures like DNS lookups. According to Souders [2009] however a caveat is to be concerned: `keep-alive` needs to be set in HTTP/1.0 only for HTTP/1.1 uses persistent connections by default. Additionally a `Content-Length` header field must be set but its value may be unknown for dynamic content.

The next section will be concerned with another paradigm of speeding up web performance: parallelizing downloads. One technique presented there will do the contrary as the last one presented in this section, namely using multiple different hostnames. However, the situations in which such an approach would be useful will be discussed in the next section.

3.2 Parallelizing downloads

Older browsers (for instance IE up to 7 and Firefox up to version 2.x) adhere to the RFC-HTTP [1999, sec. 8.1.4] and do not allow more than two parallel connections per hostname whereas newer browsers already use up to six. In the worst case, a web page consisting of several components (i.e. more than two) will be adding up RTTs per every two components requested for downloading. Souders [2009, chap. 11] coined the term "sharding" domains for the practice of serving resources from several different (sub-)domains.

3.2.1 Sharding domains

As the restriction of RFC-HTTP [1999, sec. 8.1.4] is defined on a per-hostname basis using different CNAME records for the same domain will fulfill the goal of parallelizing downloads. The IP address a hostname resolves to is not taken into account in this policy. GoogleParallelize [2011] additionally suppose to ask the Content Delivery Network (CDN) one's web site may be using for support of this technique.

Nevertheless the restriction to two connections per hostname is well-grounded on thoughts of careful server treatment. As Souders [2009, chap. 11] states persistent connections (`keep-alive`) being used in HTTP/1.1 by default cause higher workload for servers. Forcing multiple concurrent connections by sharding domains may result in a denial of service (DOS) especially with newer browser which do not stick with the RFC. Souders [2009, chap. 11] therefore proposes to adapt the sharding behaviour depending on the User-Agent and hence balance connections over older and newer browsers.

As already mentioned in the previous section there is a tradeoff between the number of DNS lookups and the number of parallel connections. The question in concern here is about the so-called "critical path": "We define the critical path as the code and resources required to render the initial view of a web page." [GoogleDNS, 2011]. According to this term these are the resources which should be loaded first and fastest. Knowing the number of components which are needed for displaying the web page one can easily determine the optimum between performance loss caused by DNS lookups and performance gain caused by sharding domains.

However, mobile versions should not use domain sharding anyway for mobile browsers are limited to a low absolute maximum of around five parallel connections. Splitting requests over several domains therefore will not lead to more connections and the additional RTTs for the DNS lookups could take down the overall performance instead of rising it [SoudersVelocity, 2011, 22:23].

3.2.2 Loading scripts asynchronously

When it comes to Javascript (JS) files there is caveat for parallelizing downloads. GoogleParallelize [2011] puts forward that "many browsers do not download JavaScript files in parallel, so there is no benefit from serving them from multiple hostnames." Other approaches need to take place in setting up concurrent script downloads which rely on re-ordering of external components, JS snippets or HTML5 attributes. Downloading asynchronously (i.e. in parallel) is the one thing tackled in this section. The other is controlling for the parsing and execution of scripts which usually blocks the rendering of the page and becomes an important issue especially for mobile web performance.

First to mention is the easiest and most straight forward way of avoiding blocking script downloads. Souders [2007, chap. 6] advices to put scripts at the bottom of a page in order to let the browser fetch the presentational content first and render it before arriving at the external script source. However, this re-ordering will not solve the problem of sequential script downloads.

Therefore Souders [2009, chap. 4] favours several feasible asynchronous approaches. One is to use an asynchronous `XmlHttpRequest` (XHR) to retrieve the code and consecutively pass it to Javascript's `eval()` function or inject it into a dynamically created `<SCRIPT>` tag. Both of it will parse and execute it finally. From a browser's perspective the response of the XHR does not appear as a script and by this cannot cause any blocking behaviour. The main disadvantage may be that due the *Same Origin Policy* [WikipediaSOP, 2011] request URLs of XHRs must reside within the same domain as the application. This can be overcome by defining a `src` of the dynamically created `<SCRIPT>` tag instead of retrieving the code through XHR. Sources in this tag may be located on any domain.

Another `<SCRIPT>` attribute is `DEFER` which defers the download and execution of a script to the end of the rendering process. Furthermore, with HTML5 an additional attribute is made available: `<SCRIPT ASYNC>` will induce the browser to load the script in parallel to other components and execute it as soon it has arrived. However, again during the execution of the script the browser will block rendering. With fast processing client nodes this issue may not narrow the user experience but mobile devices with generally low processing power are more likely to cause a noticeable rendering lag.

GoogleCodeMobile [2009] proposes some kind of *Lazy Loading* approach for slow processors. Javascript code is downloaded asynchronously using an XHR for instance but as a string literal instead of plain code. This type of text is kept in storage and as soon as the user demands for certain functionality only a part of the text is passed to `eval()` to parse and execute the desired module. This may reduce response times on mobile devices especially if there is a good deal of code which may not be needed and hence needs not to be parsed.

3.3 Reducing size

The last section of the best practices presented here deals with the reduction of the size of the request and of the payload to enhance loading performance. For the first GoogleRequest [2011] recommends to minimize the request overhead by trying to keep HTTP packages as small as possible so it fits in a TCP package of usually 1500 bytes of size. One can achieve this by avoiding unnecessary header fields which may be set automatically by frameworks in use or default configurations of web servers. Cookie headers may be reduced to a minimum by sending an identifier for a server-side cookie data storage only instead of sending a trail of cookie data through the HTTP header.

Moreover cookie header can be removed from requests for static content like images or scripts. Clients can be induced to not sending cookie data by serving static content from a designated domain. Cookies which may be set for the dynamic part of a web site will not be sent in requests for the static domain for cookies being associated with domain names.

For the second, Souders [2007] mentions several good practices for minimizing payload data, for instance to use Gzip compression of textual resources and to “minify” Javascript, CSS and HTML content. To minify means to remove insignificant whitespaces and linebreaks and to shorten identifier for variables, functions and DOM elements.

The third thing to mention here is the usage of the `Last-Modified` and the `ETag` HTTP header fields as described by Souders [2007, chap. 13]. As a resource being cached with an expiration date will expire finally these fields deal with requests for a renewal of the component. Setting a `Last-Modified` header along with the request for a resource the denoted date will be cached too. After expiration (set through an `Expires` field) the client can send a conditional GET request using the `If-Modified-Since` field and asking whether the resource has been modified since the earlier given `Last-Modified` value. The server then may response either with an HTTP 304 code (`Not Modified`) or serve the full payload of the modified component. This technique will also reduce loading times as cached stuff may not having been changed even after expiration.

On the contrary `ETag` can be used to associate some kind of unique hash value with a cached resource. Instead of an `If-Modified-Since` request the client may ask for `If-None-Match` accompanied by the cached ETag. Depending on whether the ETag of a resource has changed the server serves the new resource or not. Unfortunately, by default ETags are server-specific. This means that one and same resource will be associated with different ETags when located on different machines. However, there are ways to configure ETags to rely on more generic file attributes like the file size and a timestamp. Souders [2007, chap. 13] also supposes web programmers to consider removing ETags completely and leveraging the simpler `Last-Modified` header instead.

Chapter 4

Evaluation of Tools

In this section two performance measurement tools will be presented and evaluated. The first section deals with *Google Page Speed*, the second with *Fiddler* and the last section will compare the two and draw a conclusion.

4.1 Google Page Speed

There is a tool from Google to Test the speed of websites. This is called Google Page Speed. Google provides basically three types of services.

1. Apache - Module
2. Google Page Speed Service
3. Google Page Speed Online / Browserplugin

These services have advantages and disadvantages which will all be discussed in more detail in the following pages but the main focus will be on the Google Page Speed Online tool. This tool is used to evaluate a popular website:amazon.com

4.1.1 Google Page Speed Service

[Google, 2011d] The basic idea of this Service is, that a website provider does not have to deal with the speed issue at all because Google will take total care of this issue by hosting the website. This is done in the following way. The provider of the Website signs up with the domain, verifies the ownership of the domain by a DNS-Entry and redirects the requests set a CNAME DNS-Entry to the Google-servers. The Google-Servers gets all the content from the signed domain and all the HTML-sites are rewritten to increase the load speed. After setting up the Google Page Speed Service the provider has access to the Google APIs Console, where several settings can be made or changed on the Rewrite Service. The provider can enable or disable the following settings:

- Combine CSS
- Move CSS to Head
- Combine Javascript
- Optimize Javascript
- Optimize Images
- Resize Images

- Proxy CSS
- Proxy Images
- Proxy Javascript

Advantages

- Provider does not have to know anything about best practice on web performance
- Google-Servers automatically fetch content and optimize it
- Google-Servers serve requests
- Google-Servers are more fail-safe than private servers

Disadvantages

- Google has all content of the website.
- Depending on Google

4.1.2 Google Page Speed - Apache Module

[Google, 2011a] Google implemented a loadable module for the Apache Webserver. The idea is that the server rewrites the HTML sites on-the-fly to improve the speed of the page. The module is open-source and uses filters to rewrite the resources. The webmaster only has to load the module on the Server and change some settings in a special configuration file. The benefit on this technique is that the web developer does not have to think or even have to know the best practice on web performance. The developer only implements the website and the server does the rest. The module implements the following filters to increase web performance:

- JavaScript optimization
- HTML optimization
- CSS optimization
- Image optimization
- Caching optimization

Advanteges

- The Web developer does not need to know web performance best practice
- The Server does all the optimization with filters

Disadvantages

- The Webmaster has to load the module and configure it
- Only available for Apache Webservers

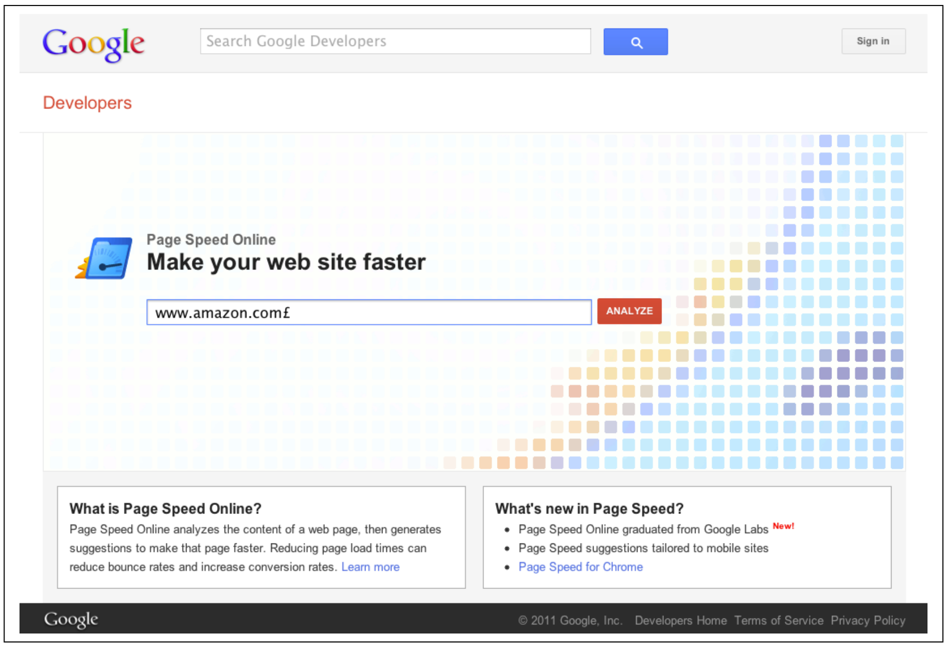
4.1.3 Google Page Speed Online

[Google, 2011b] Google Page Speed is a tool which evaluates a website. It can be used in an online version or as a plug-in for Google Chrome or Mozilla Firefox. Here the online version will be looked at in more detail. Google Page Speed Online is basically a stopwatch. It measures the time it takes to load the page. The tool returns a Score between zero and one-hundred according to the HTML-code of the site. It analyses the HTML-code and gives recommendations to improve the performance. The tool can work in two different modes. First in the Desktop-Mode which means that the tool tests the desktop version of the Website. Second it can work in Mobile-Mode; in this mode the mobile version of the Website is tested, which is rather useful since almost every website has a mobile version.

Userinterface

The Userinterface of the Google Page Speed Online is very simple. Basically, the Userinterface consists of a text field where the URL is entered. A simple click on ”Analyze” and the tool starts to evaluate the Website.

This takes a few moments and after the Website was evaluated the User gets the results and recommendations of how to improve it. If one wants to switch between the mobile and the desktop mode, the analysis has to be done since it is only possible to switch between the modes in the analysis summary.


Google
Search Google Developers
Sign in
Developers
Page Speed Online
Make your web site faster
www.amazon.com£
ANALYZE
Google
© 2011 Google, Inc. Developers Home Terms of Service Privacy Policy

What is Page Speed Online?

Page Speed Online analyzes the content of a web page, then generates suggestions to make that page faster. Reducing page load times can reduce bounce rates and increase conversion rates. Learn more

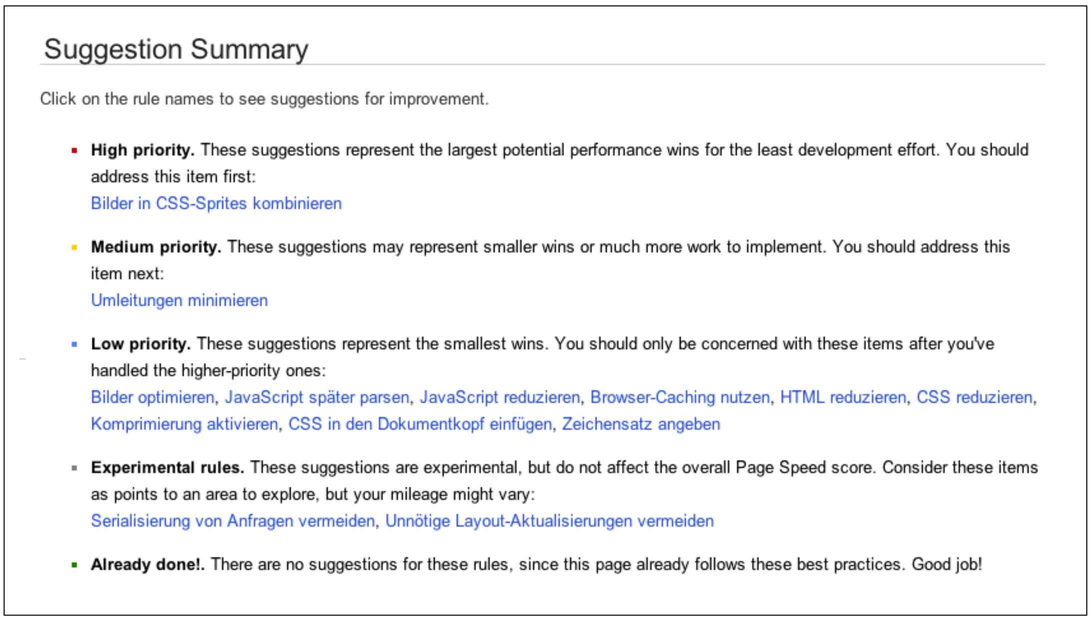
What's new in Page Speed?

- Page Speed Online graduated from Google Labs New!
- Page Speed suggestions tailored to mobile sites
- Page Speed for Chrome

Figure 4.1: Google Page Speed - Userinterface

Recommendation

After the analysis the tool shows the recommendations separated in high, medium and low priorities. There is also a section for ”Experimental rules”; in this section rules are displayed which are not yet approved. The recommendations with the highest priority will increase the performance best and in the ”Already done” section the tool shows rules which have already been applied to the website. All the suggestion will increase the performance of the webpage in different way. To get detailed information on how this is done, it is possible to click on each of the single recommendations.

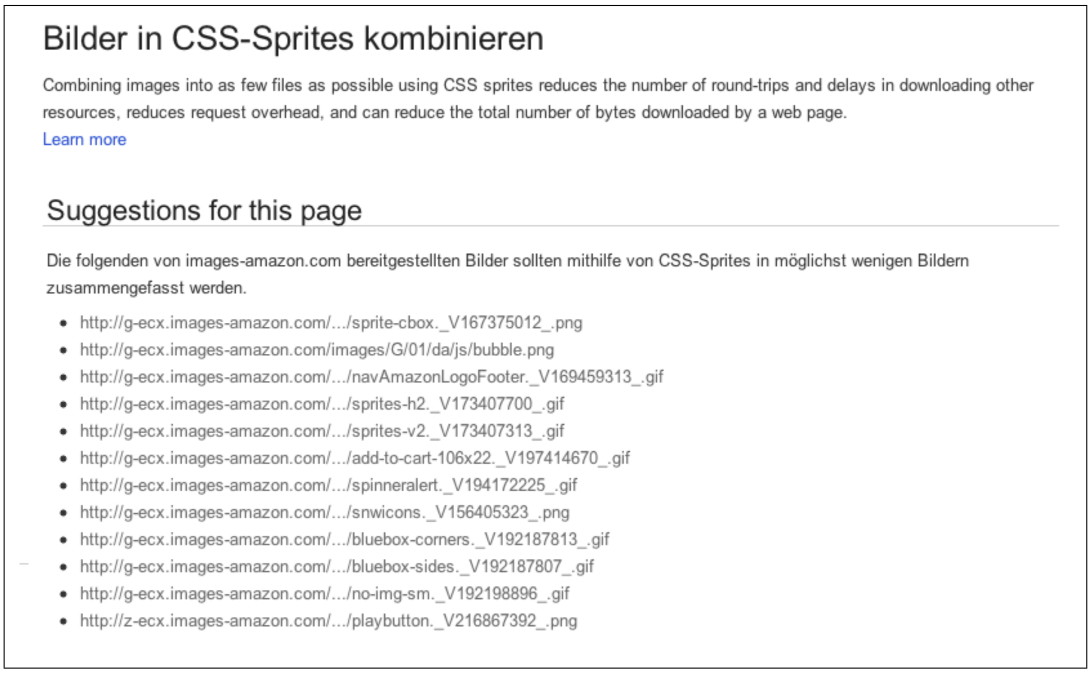


Suggestion Summary

Click on the rule names to see suggestions for improvement.

- **High priority.** These suggestions represent the largest potential performance wins for the least development effort. You should address this item first:
  Bilder in CSS-Sprites kombinieren
- **Medium priority.** These suggestions may represent smaller wins or much more work to implement. You should address this item next:
  Umleitungen minimieren
- **Low priority.** These suggestions represent the smallest wins. You should only be concerned with these items after you've handled the higher-priority ones:
  Bilder optimieren, JavaScript später parsen, JavaScript reduzieren, Browser-Caching nutzen, HTML reduzieren, CSS reduzieren, Komprimierung aktivieren, CSS in den Dokumentkopf einfügen, Zeichensatz angeben
- **Experimental rules.** These suggestions are experimental, but do not affect the overall Page Speed score. Consider these items as points to an area to explore, but your mileage might vary:
  Serialisierung von Anfragen vermeiden, Unnötige Layout-Aktualisierungen vermeiden
- **Already done!.** There are no suggestions for these rules, since this page already follows these best practices. Good job!

Figure 4.2: Google Page Speed - Suggestion Summary



Bilder in CSS-Sprites kombinieren

Combining images into as few files as possible using CSS sprites reduces the number of round-trips and delays in downloading other resources, reduces request overhead, and can reduce the total number of bytes downloaded by a web page.
Learn more

Suggestions for this page

Die folgenden von images-amazon.com bereitgestellten Bilder sollten mithilfe von CSS-Sprites in möglichst wenigen Bildern zusammengefasst werden.

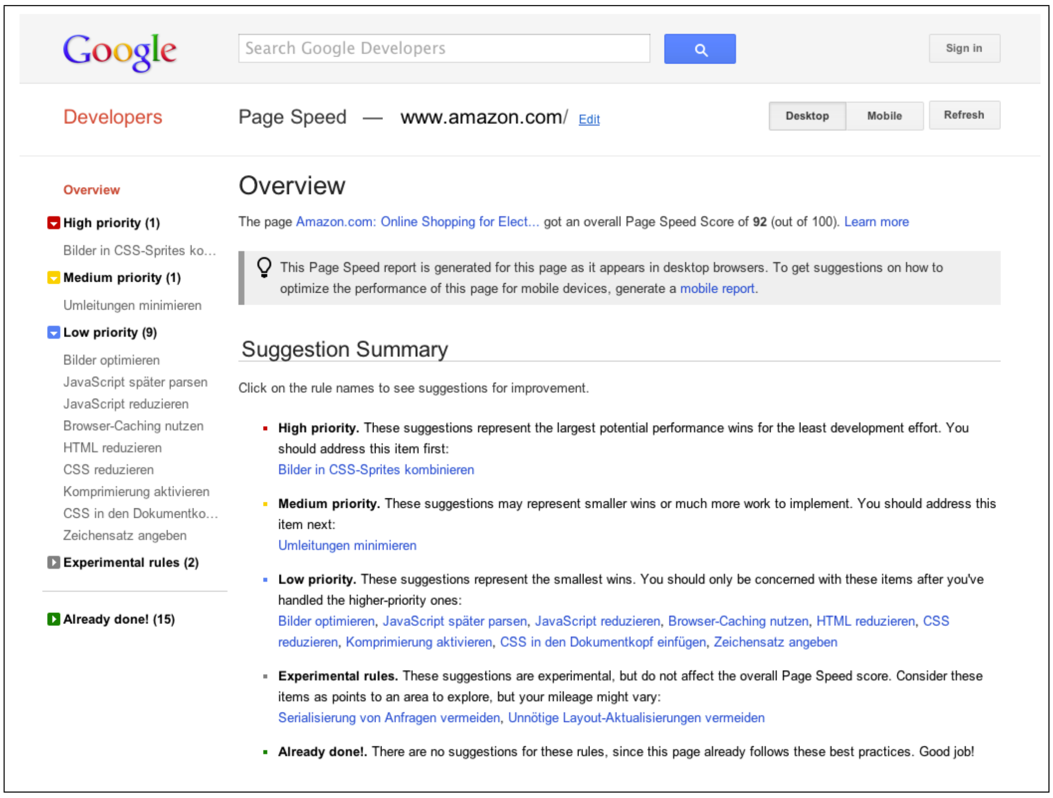
- http://g-ecx.images-amazon.com/.../sprite-cbox._V167375012_.png
- http://g-ecx.images-amazon.com/images/G/01/da/js/bubble.png
- http://g-ecx.images-amazon.com/.../navAmazonLogoFooter._V169459313_.gif
- http://g-ecx.images-amazon.com/.../sprites-h2._V173407700_.gif
- http://g-ecx.images-amazon.com/.../sprites-v2._V173407313_.gif
- http://g-ecx.images-amazon.com/.../add-to-cart-106x22._V197414670_.gif
- http://g-ecx.images-amazon.com/.../spinneralert._V194172225_.gif
- http://g-ecx.images-amazon.com/.../snwicons._V156405323_.png
- http://g-ecx.images-amazon.com/.../bluebox-corners._V192187813_.gif
- http://g-ecx.images-amazon.com/.../bluebox-sides._V192187807_.gif
- http://g-ecx.images-amazon.com/.../no-img-sm._V192198896_.gif
- http://z-ecx.images-amazon.com/.../playbutton._V216867392_.png

Figure 4.3: Google Page Speed - Recommendation in Detail

Amazon.com in Desktop-Mode

[Google, 2011c] To show the features of the tool an analysis on the Amazon.com Website was done. The analysis of Amazone.com ended with the result of 92 points out of 100. The good result leads to a small amount of recommendations.

- **High priority:**
  - Combine Pictures in CSS-Sprites
- **Medium priority:**
  - Minimize Redirections
- **Low priority:**
  - Compress Images
  - Minimize Javascript, HTML, CSS
  - Use Browser-Caching
  - Activate Compression
  - Move CSS to Head
  - Define Encoding
- **Experimental rules:**
  - Avoid Serialized Requests
  - Avoid unnecessary Layout-updates



Overview

The page Amazon.com: Online Shopping for Elect... got an overall Page Speed Score of **92** (out of 100). Learn more

This Page Speed report is generated for this page as it appears in desktop browsers. To get suggestions on how to optimize the performance of this page for mobile devices, generate a mobile report.

Suggestion Summary

Click on the rule names to see suggestions for improvement.

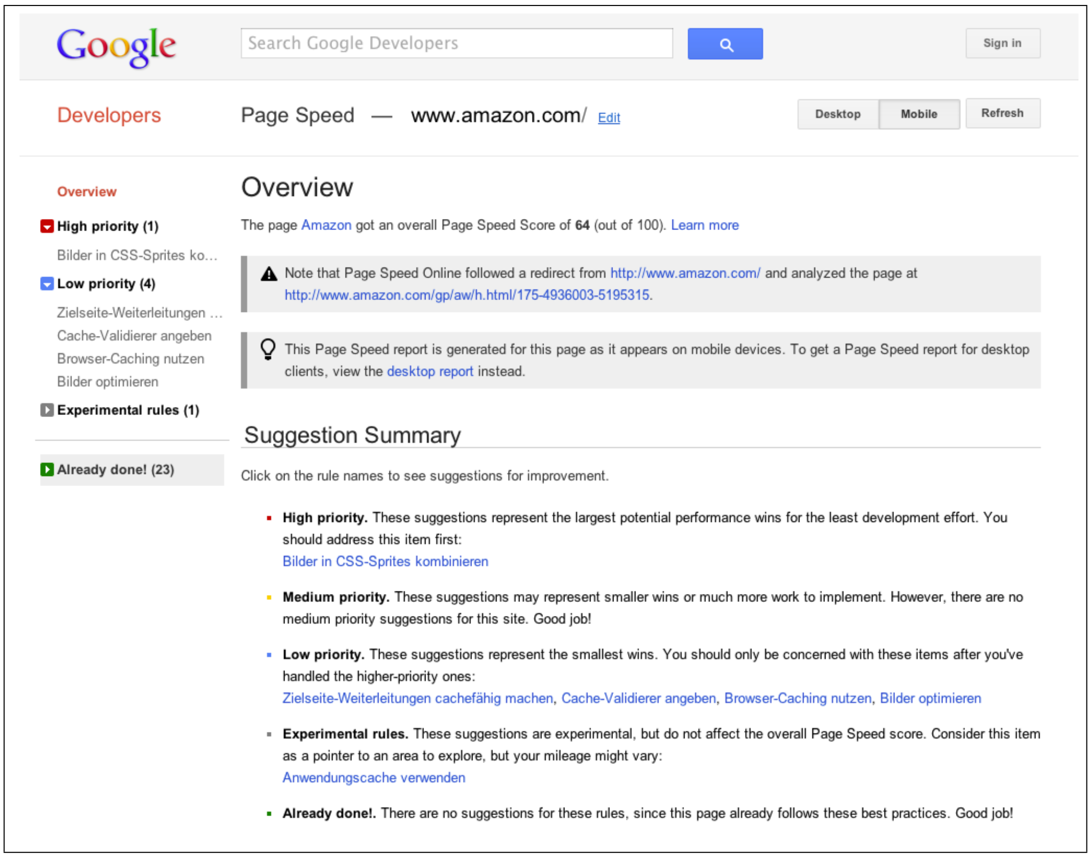
- **High priority.** These suggestions represent the largest potential performance wins for the least development effort. You should address this item first:
  Bilder in CSS-Sprites kombinieren
- **Medium priority.** These suggestions may represent smaller wins or much more work to implement. You should address this item next:
  Umleitungen minimieren
- **Low priority.** These suggestions represent the smallest wins. You should only be concerned with these items after you've handled the higher-priority ones:
  Bilder optimieren, JavaScript später parsen, JavaScript reduzieren, Browser-Caching nutzen, HTML reduzieren, CSS reduzieren, Komprimierung aktivieren, CSS in den Dokumentkopf einfügen, Zeichensatz angeben
- **Experimental rules.** These suggestions are experimental, but do not affect the overall Page Speed score. Consider these items as points to an area to explore, but your mileage might vary:
  Serialisierung von Anfragen vermeiden, Unnötige Layout-Aktualisierungen vermeiden
- **Already done!.** There are no suggestions for these rules, since this page already follows these best practices. Good job!

Figure 4.4: Google Page Speed - Desktopmode: Amazon.com

Amazon.com in Mobile-Mode

[Google, 2011c] The major difference in the Mobile-Mode is that, the analysis is made on the mobile version of the website. These websites differ to the desktop versions in many ways for instance in the screen resolution. On mobile devices is even more important that the website works at its best due the high Round-Trip-Time and the low CPU performance. The analysis on Amazon.com returned 64 Points out of 100. This result leads to a list of recommendations.

- **High priority:**
  - Combine Pictures in CSS-Sprites
- **Medium priority:** NO Suggestions
- **Low priority:**
  - Cacheable Redirections
  - Use Cache-Validator
  - Use Browser-Caching
  - Optimize Images
  - Remove ? from Static Requests
- **Experimental rules:**
  - Use Applicationcache



Overview

The page Amazon got an overall Page Speed Score of **64** (out of 100). Learn more

Note that Page Speed Online followed a redirect from http://www.amazon.com/ and analyzed the page at http://www.amazon.com/gp/aw/h.html/175-4936003-5195315.

This Page Speed report is generated for this page as it appears on mobile devices. To get a Page Speed report for desktop clients, view the desktop report instead.

Suggestion Summary

Click on the rule names to see suggestions for improvement.

- **High priority.** These suggestions represent the largest potential performance wins for the least development effort. You should address this item first:
  Bilder in CSS-Sprites kombinieren
- **Medium priority.** These suggestions may represent smaller wins or much more work to implement. However, there are no medium priority suggestions for this site. Good job!
- **Low priority.** These suggestions represent the smallest wins. You should only be concerned with these items after you've handled the higher-priority ones:
  Zielseite-Weiterleitungen cachefähig machen, Cache-Validierer angeben, Browser-Caching nutzen, Bilder optimieren
- **Experimental rules.** These suggestions are experimental, but do not affect the overall Page Speed score. Consider this item as a pointer to an area to explore, but your mileage might vary:
  Anwendungscache verwenden
- **Already done!.** There are no suggestions for these rules, since this page already follows these best practices. Good job!

Figure 4.5: Google Page Speed - Mobilemode: Amazon.com

4.2 Fiddler

4.2.1 Origins

[Fiddler, 2011a]
The Fiddler Web Debugger is a Proxy Server which logs HTTP and HTTPS traffic. The tool allows the user to extend the provided functionality with a powerful event-based scripting subsystem using any .NET language. The name of the software derives from ”fiddling” with incoming and outgoing data, because the user can manipulate requests and responses as well as any other information that goes through the proxy.

[Eric Lawrence, 2009]
Fiddler is the brainchild of Eric Lawrence, a Program Manager at Microsoft, responsible for Internet Explorer security. He first released Fiddler in 2003 with very limited functionality, but it was the best option at the time, as HTTP-Request debugging back then usually consisted of using a packet-sniffer and manually looking through all the headers and the packets. Lawrence worked with the ClipArt team at Microsoft at the time and was looking for a more efficient way to analyse and debug HTTP-Traffic and thus Fiddler was born.

Six years after the first release of the tool in 2003, Lawrence released Fiddler2, a very sophisticated web debugging software written completely in .NET. It was designed for extensibility, allowing users to write their own filters and such, so that he, because Lawrence was working on the project alone, wouldn’t have to implement a routine for every possibility.
Today there is a huge community behind the tool and there are very complex third party extensions as well such as neXpert [neXpert, 2011].

4.2.2 Features

[Eric Lawrence, 2009], [John Hrvatin, 2009], [Eric Lawrence, 2011]

Traffic Monitoring and Archiving

Fiddler can capture traffic, save it, reuse it and compress it into different formats in order to send it to another location. The tool is a proxy server, and thus can be used from every possible endpoint such as mobile phones, desktops or other servers. It uses the WINInet Stack, which is used by every commonly used browser except Mozilla Firefox, but there is an addon for that as well. The user can basically set every application to the proxy and let the generated traffic run through Fiddler.

Once some HTTP-traffic is captured, it can also be archived and saved into various binary formats and especiall into the SAZ, the Session Archive ZIP, which is a very efficient way to distribute traffic.
For example, for testing in an external Intranet, one could capture the traffic internally, save it and then send it to the responsible IT-engineers for further analysis using Fiddler.

Traffic Analysis and Manipulation

The main part of the functionality revolves around the analysis and manipulation of the captured traffic. Users can filter HTTP traffic in every imaginable way (because of the high extensibility) and even create automated routines to filter out certain aspects. The traffic can also be easily compared, for example to see what changed between two identical requests when executed at different times.

To further improve the efficiency in finding errors, Fiddler provides many features to manipulate the active traffic, such as breakpoint-debugging, making it possible to edit every request before it leaves and every response before it returns, thus opening endless possibilities to test different settings of headers, hardware and software, without actually changing any of these things.

The Requestbuilder and Autoresponder are two other features which can be used for automated manipulation of the traffic, with these tools you could, for example, change every picture on a website to a certain picture you define. The timeline gives a nice overview of what takes how long and where the bottlenecks appear to be.

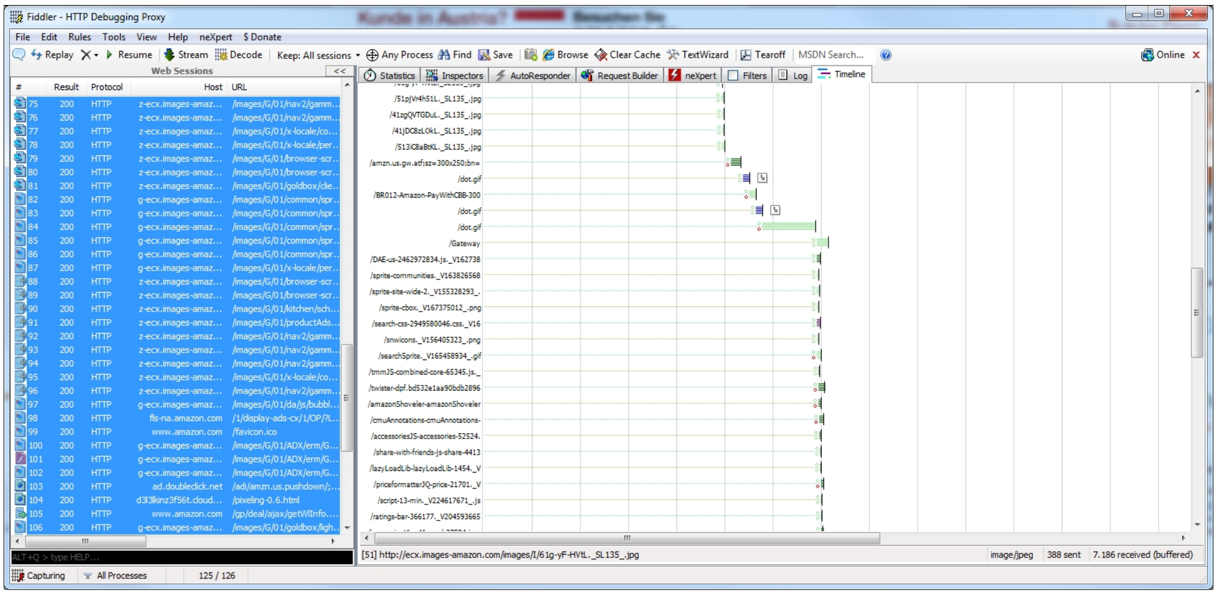
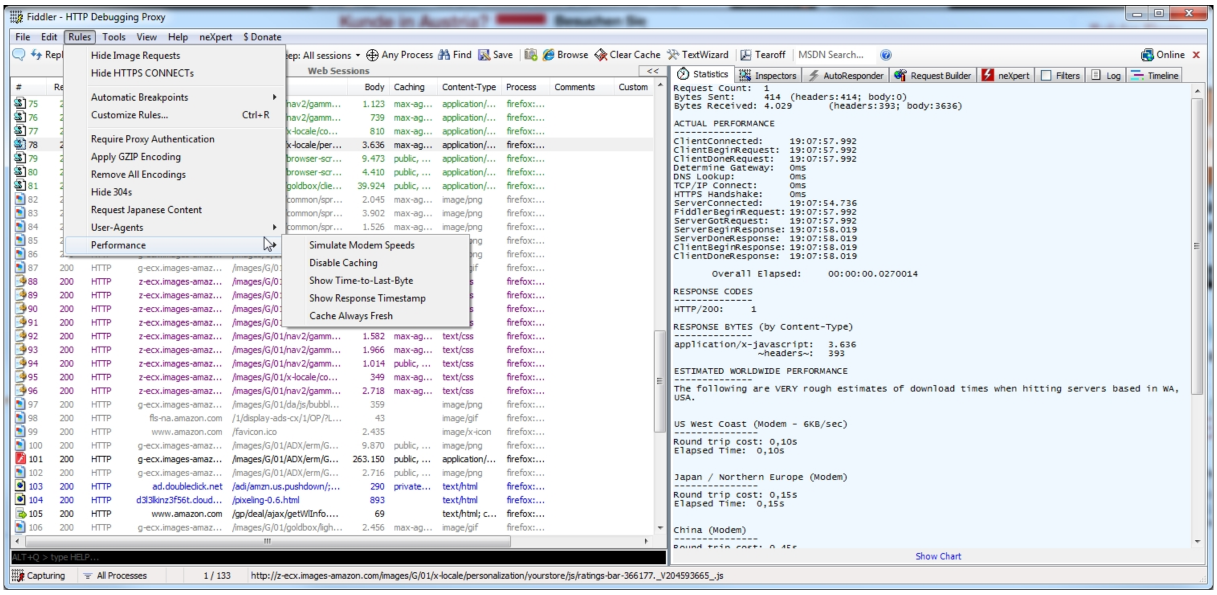

Fiddler - HTTP Debugging Proxy
File Edit Rules Tools View Help neXpert $ Donate
Replay Resume Stream Decode Keep: All sessions Any Process Find Save Browse Clear Cache TextWizard Tearoff MSDN Search... Online
Web Sessions
Statistics Inspectors AutoResponder Request Builder neXpert Filters Log Timeline
[51] http://ecx.images-amazon.com/images/I/61g-yF-HVtL._SL135_.jpg image/jpeg 388 sent 7.186 received (buffered)
Capturing All Processes 125 / 126

Figure 4.6: Fiddler built-in timeline

Performance (Mobile)

The tool provides some built-in functionality for performance-testing such as the simulation of slower network speeds, compression and caching manipulation. Furthermore there are many extensions dealing with the different facettes of performance testing, especially neXpert.


Fiddler - HTTP Debugging Proxy
File Edit Rules Tools View Help neXpert $ Donate
Hide Image Requests
Hide HTTPS CONNECTs
Automatic Breakpoints
Customize Rules... Ctrl+R
Require Proxy Authentication
Apply GZIP Encoding
Remove All Encodings
Hide 304s
Request Japanese Content
User-Agents
Performance
Simulate Modem Speeds
Disable Caching
Show Time-to-Last-Byte
Show Response Timestamp
Cache Always Fresh
Statistics Inspectors AutoResponder Request Builder neXpert Filters Log Timeline
Show Chart
Capturing All Processes 1 / 133 http://z-ecx.images-amazon.com/images/G/01/x-locale/personalization/yourstore/js/ratings-bar-366177._V204593665_.js

```
Request Count:   1
Bytes Sent:      414  (headers:414; body:0)
Bytes Received:  4.029      (headers:393; body:3636)

ACTUAL PERFORMANCE
--------------
ClientConnected:     19:07:57.992
ClientBeginRequest:  19:07:57.992
ClientDoneRequest:   19:07:57.992
Determine Gateway:   0ms
DNS Lookup:          0ms
TCP/IP Connect:      0ms
HTTPS Handshake:     0ms
ServerConnected:     19:07:54.736
FiddlerBeginRequest: 19:07:57.992
ServerGotRequest:    19:07:57.992
ServerBeginResponse: 19:07:58.019
ServerDoneResponse:  19:07:58.019
ClientBeginResponse: 19:07:58.019
ClientDoneResponse:  19:07:58.019

       Overall Elapsed:     00:00:00.0270014

RESPONSE CODES
--------------
HTTP/200:     1

RESPONSE BYTES (by Content-Type)
--------------
application/x-javascript:   3.636
               ~headers~:   393

ESTIMATED WORLDWIDE PERFORMANCE
--------------
The following are VERY rough estimates of download times when hitting servers based in WA, USA.


US West Coast (Modem - 6KB/sec)
---------------
Round trip cost: 0,10s
Elapsed Time:  0,10s


Japan / Northern Europe (Modem)
---------------
Round trip cost: 0,15s
Elapsed Time:  0,15s


China (Modem)
---------------
```

Figure 4.7: Fiddler built-in performance tools

Test Integration

Fiddler provides the user with FiddlerCore, a .NET library for integration into applications. Using this library makes it possible for web-applications to access Fiddler-Features and simulate trafic using all available filters and modifications available to the software. It can be integrated very easily.

Before FiddlerCore, this was only able through the ExecAction.exe tool provided by Eric Lawrence, which made it possible to use certain Fiddler features from externally.

List of popular Extensions

[Fiddler, 2011b]

- SyntaxView, Syntax-Highlighting
- neXpert, Performance
- stressStimulus, load testing
- Gallery, Pictures
- AnyWHERE, locale spoofer
- Ammonite / Watcher, security
- HTML Inspector, checks HTML responses for inefficiencies
- Request-to-code, translates a request to .NET code

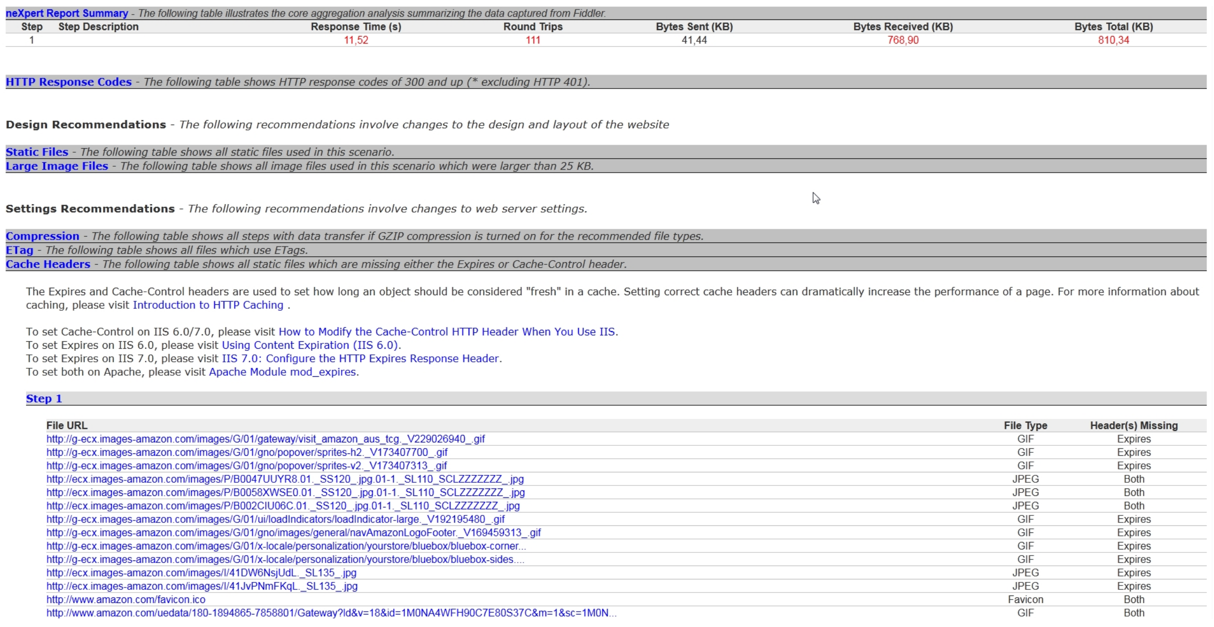
4.2.3 Test of amazon.com

Setup and Motivation

Finding an appropriate website to test a tool such as Fiddler is no easy task, there are many and more pages in the web in need of some serious optimization. By using a huge site like amazon.com, the tester can be fairly certain that, with millions of users visiting and actively using the application every hour of every day, the webteam of Amazon already did some testing on their own and probably removed all obvious problems which would otherwise lead to a laggy and slow experience for their customers.

Using Fiddler [Fiddler, 2011a] with the extension neXpert [neXpert, 2011] provides the tester with a lot of options. NeXpert itself includes the ability to divide the whole traffic into steps in order to split up the more and less performance-relevant parts of the timeline and thus having a better overview of the parts which are causing the problems. Through other extensions, self-made or downloaded, the tester can filter or manipulate some of the incoming traffic as well in order to search for very specific areas of problems (for example large images).

Once everything is set up and ready, the user can just create a report using the neXpert extension, which is then saved on the local filesystem. This report is in HTML-format and provides a well structured overview of the areas neXpert found ways to improve upon.



neXpert Report Summary - *The following table illustrates the core aggregation analysis summarizing the data captured from Fiddler.*

Step	Step Description	Response Time (s)	Round Trips	Bytes Sent (KB)	Bytes Received (KB)	Bytes Total (KB)
1		11,52	111	41,44	768,90	810,34

HTTP Response Codes - *The following table shows HTTP response codes of 300 and up (* excluding HTTP 401).*

Design Recommendations - *The following recommendations involve changes to the design and layout of the website*

Static Files - *The following table shows all static files used in this scenario.*
Large Image Files - *The following table shows all image files used in this scenario which were larger than 25 KB.*

Settings Recommendations - *The following recommendations involve changes to web server settings.*

Compression - *The following table shows all steps with data transfer if GZIP compression is turned on for the recommended file types.*
ETag - *The following table shows all files which use ETags.*
Cache Headers - *The following table shows all static files which are missing either the Expires or Cache-Control header.*

The Expires and Cache-Control headers are used to set how long an object should be considered "fresh" in a cache. Setting correct cache headers can dramatically increase the performance of a page. For more information about caching, please visit Introduction to HTTP Caching .

To set Cache-Control on IIS 6.0/7.0, please visit How to Modify the Cache-Control HTTP Header When You Use IIS.
To set Expires on IIS 6.0, please visit Using Content Expiration (IIS 6.0).
To set Expires on IIS 7.0, please visit IIS 7.0: Configure the HTTP Expires Response Header.
To set both on Apache, please visit Apache Module mod_expires.

Step 1

File URL	File Type	Header(s) Missing
http://g-ecx.images-amazon.com/images/G/01/gateway/visit_amazon_aus_tcg._V229026940_.gif	GIF	Expires
http://g-ecx.images-amazon.com/images/G/01/gno/popover/sprites-h2._V173407700_.gif	GIF	Expires
http://g-ecx.images-amazon.com/images/G/01/gno/popover/sprites-v2._V173407313_.gif	GIF	Expires
http://ecx.images-amazon.com/images/P/B0047UUYR8.01._SS120_.jpg.01-1._SL110_SCLZZZZZZZ_.jpg	JPEG	Both
http://ecx.images-amazon.com/images/P/B0058XWSE0.01._SS120_.jpg.01-1._SL110_SCLZZZZZZZ_.jpg	JPEG	Both
http://ecx.images-amazon.com/images/P/B002CIU06C.01._SS120_.jpg.01-1._SL110_SCLZZZZZZZ_.jpg	JPEG	Both
http://g-ecx.images-amazon.com/images/G/01/ui/loadIndicators/loadIndicator-large._V192195480_.gif	GIF	Expires
http://g-ecx.images-amazon.com/images/G/01/gno/images/general/navAmazonLogoFooter._V169459313_.gif	GIF	Expires
http://g-ecx.images-amazon.com/images/G/01/x-locale/personalization/yourstore/bluebox/bluebox-corner...	GIF	Expires
http://g-ecx.images-amazon.com/images/G/01/x-locale/personalization/yourstore/bluebox/bluebox-sides....	GIF	Expires
http://ecx.images-amazon.com/images/I/41DW6NsjUdL._SL135_.jpg	JPEG	Expires
http://ecx.images-amazon.com/images/I/41JvPNmFKqL._SL135_.jpg	JPEG	Expires
http://www.amazon.com/favicon.ico	Favicon	Both
http://www.amazon.com/uedata/180-1894865-7858801/Gateway?ld&v=18&id=1M0NA4WFH90C7E80S37C&m=1&sc=1M0N...	GIF	Both

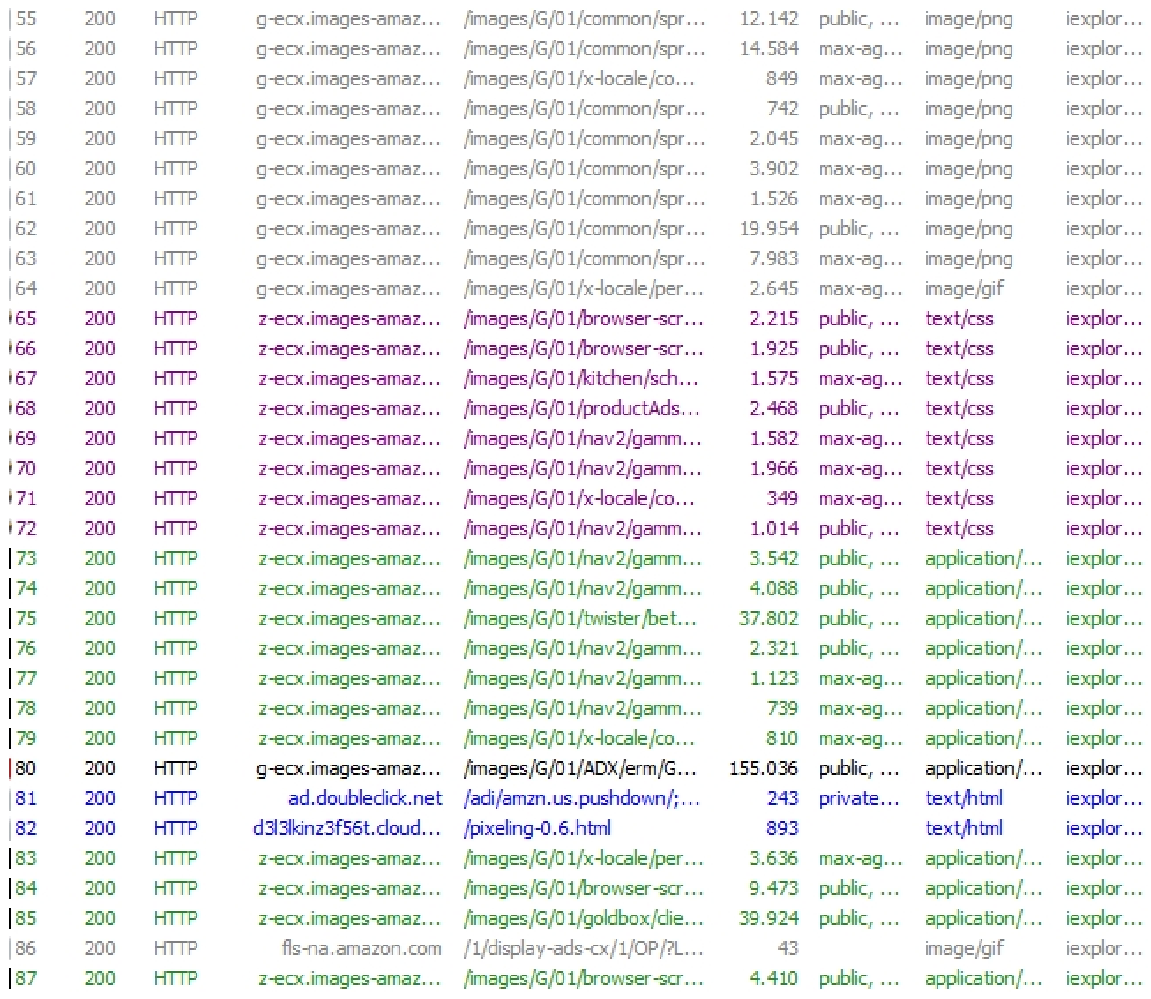
Figure 4.8: neXpert-Report of the amazon.com mainpage

Traffic Analysis

Fiddler, or rather neXpert does this by analysing the recorded traffic and checking the data for known problems such as missing expires-headers or large images. The tester could basically do this by him / herself manually by just browing through the traffic generated by the opening of amazon.com.

The traffic can be filtered, manipulated and displayed in many different ways, making it easier for the user to see possible problems. The in-built timeline function of Fiddler is very helpful as well, especially when trying to figure out where to set neXpert's Step-Markers. For example, users can iteratively move the step-markers further forward if there are no problems in the first, for example, 30 milliseconds of the page-request, thus eliminating unneccessary checking everytime.

Once captured, the traffic can be reused over and over again, so that a tester doesn't have to rely on the maybe varying performance of his / her ISP or Computer.



55	200	HTTP	g-ecx.images-amaz...	/images/G/01/common/spr...	12.142	public, ...	image/png	iexplor...
56	200	HTTP	g-ecx.images-amaz...	/images/G/01/common/spr...	14.584	max-ag...	image/png	iexplor...
57	200	HTTP	g-ecx.images-amaz...	/images/G/01/x-locale/co...	849	max-ag...	image/png	iexplor...
58	200	HTTP	g-ecx.images-amaz...	/images/G/01/common/spr...	742	public, ...	image/png	iexplor...
59	200	HTTP	g-ecx.images-amaz...	/images/G/01/common/spr...	2.045	max-ag...	image/png	iexplor...
60	200	HTTP	g-ecx.images-amaz...	/images/G/01/common/spr...	3.902	max-ag...	image/png	iexplor...
61	200	HTTP	g-ecx.images-amaz...	/images/G/01/common/spr...	1.526	max-ag...	image/png	iexplor...
62	200	HTTP	g-ecx.images-amaz...	/images/G/01/common/spr...	19.954	public, ...	image/png	iexplor...
63	200	HTTP	g-ecx.images-amaz...	/images/G/01/common/spr...	7.983	max-ag...	image/png	iexplor...
64	200	HTTP	g-ecx.images-amaz...	/images/G/01/x-locale/per...	2.645	max-ag...	image/gif	iexplor...
65	200	HTTP	z-ecx.images-amaz...	/images/G/01/browser-scr...	2.215	public, ...	text/css	iexplor...
66	200	HTTP	z-ecx.images-amaz...	/images/G/01/browser-scr...	1.925	public, ...	text/css	iexplor...
67	200	HTTP	z-ecx.images-amaz...	/images/G/01/kitchen/sch...	1.575	max-ag...	text/css	iexplor...
68	200	HTTP	z-ecx.images-amaz...	/images/G/01/productAds...	2.468	public, ...	text/css	iexplor...
69	200	HTTP	z-ecx.images-amaz...	/images/G/01/nav2/gamm...	1.582	max-ag...	text/css	iexplor...
70	200	HTTP	z-ecx.images-amaz...	/images/G/01/nav2/gamm...	1.966	max-ag...	text/css	iexplor...
71	200	HTTP	z-ecx.images-amaz...	/images/G/01/x-locale/co...	349	max-ag...	text/css	iexplor...
72	200	HTTP	z-ecx.images-amaz...	/images/G/01/nav2/gamm...	1.014	public, ...	text/css	iexplor...
73	200	HTTP	z-ecx.images-amaz...	/images/G/01/nav2/gamm...	3.542	public, ...	application/...	iexplor...
74	200	HTTP	z-ecx.images-amaz...	/images/G/01/nav2/gamm...	4.088	public, ...	application/...	iexplor...
75	200	HTTP	z-ecx.images-amaz...	/images/G/01/twister/bet...	37.802	public, ...	application/...	iexplor...
76	200	HTTP	z-ecx.images-amaz...	/images/G/01/nav2/gamm...	2.321	public, ...	application/...	iexplor...
77	200	HTTP	z-ecx.images-amaz...	/images/G/01/nav2/gamm...	1.123	max-ag...	application/...	iexplor...
78	200	HTTP	z-ecx.images-amaz...	/images/G/01/nav2/gamm...	739	max-ag...	application/...	iexplor...
79	200	HTTP	z-ecx.images-amaz...	/images/G/01/x-locale/co...	810	max-ag...	application/...	iexplor...
80	200	HTTP	g-ecx.images-amaz...	/images/G/01/ADX/erm/G...	155.036	public, ...	application/...	iexplor...
81	200	HTTP	ad.doubleclick.net	/adi/amzn.us.pushdown/;...	243	private...	text/html	iexplor...
82	200	HTTP	d3l3lkinz3f56t.cloud...	/pixeling-0.6.html	893		text/html	iexplor...
83	200	HTTP	z-ecx.images-amaz...	/images/G/01/x-locale/per...	3.636	max-ag...	application/...	iexplor...
84	200	HTTP	z-ecx.images-amaz...	/images/G/01/browser-scr...	9.473	public, ...	application/...	iexplor...
85	200	HTTP	z-ecx.images-amaz...	/images/G/01/goldbox/clie...	39.924	public, ...	application/...	iexplor...
86	200	HTTP	fls-na.amazon.com	/1/display-ads-cx/1/OP/?L...	43		image/gif	iexplor...
87	200	HTTP	z-ecx.images-amaz...	/images/G/01/browser-scr...	4.410	public, ...	application/...	iexplor...

Figure 4.9: Fiddler traffic of the amazon.com website

Results

During our extensive testing Fiddler and neXpert yielded some very interesting results. The report showed the following areas:

- HTTP Response Codes
  If a request returns a non-HTTP 200 response code, that can be very costly, so in this section, these occurences are shown with their respective files. The tested website amazon.com showed two of these problems, both coming from external advertising-sites.

- Static Files
  The Static-Files section shows all Javascript and CSS files that could be merged together to improve performance. As expected there are quite a lot of these, especially from dynamic sources.

- Large Image Files
  This section is fairly self-descriptive, it just lists images over a certain threshold size the user can configure. In this case, amazon.com had two images with a size over 25 KB, which was the threshold we used for the test.

- Compression
  Here are some recommendations on which files were not compressed with a detailed statistic on how much bytes could have been saved by using some kind of compression. In the amazon.com example, the decrease would have been 0,6 %, so it was hardly relevant.

- ETag
  The Etag section covers the validation header for objects in a user's cache. There were three objects that stood out and there is some helpful information on how to configure Etags on Apache Webserver.

- Cache Headers
  This section deals with missing headers on files concerning the caching of objects. Some of these headers will usually be missing on purpose, for some kind of dynamic content, but caching is a very important part of web-performance, so the tester should browse through the shown files very carefully and decide for each and everyone, if it really shouldn't be cached.

- Connection Header
  Two objects on the amazon.com page included a Connection header set to ”close”, which closes the TCP connection, thus leading to performance issues.

- Cookies
  Here a table is created showing each cookie, where it comes from and their respective size

Some of these results are to be expected of almost any website, such as the Static Files and some missing Cache Headers, others, like weird Connection Headers or ETag settings are not as common. NeXpert provides the user not only with a detailed statistic of the problems it found, there are all sorts of links to recommendations and problem-solving possibilities for the problem at hand for each of the mentioned sections, making it rather easy to get rid of a problem once it has been isolated and found.

Overall though, as expected, the amazon.com website passed with flying colors, the suggested changes were all pretty minor and are, in some cases unavoidable while retaining the same level of functionality on the site.

4.3 Comparison

4.3.1 Google Page Speed

Google Page Speed is a light-weight optimization tool which provides the user with a numerical rating between 0 and 100. 0 being the worst and 100 being the best score.

The tool runs in two Modes: Desktop and Mobile. These modes don't differ in straight functionality, but the recommendations differ at some points. The main focus of Google Page Speed is to make it a fast and very easy to use tool, not only for professional web-developers, but also for less experienced users who look for an efficient way to improve the speed of their web applications.

Google Page Speed is neither easily extensible nor does it provide extended functionality beyond just measuring the time a certain page loads and and giving recommendations on how to speed it up, but users do not have to download or install anything or read documentation at all, which makes it widely usable.

4.3.2 Fiddler

Fiddler is a well documented, feature- and extension rich Tool which provides nearly endless functionality for users who are not afraid to play around with it and read some of the well maintained documentation. Especially neXpert, being the leading performance-optimization extension, provides a whole plethora of detailed information and recommendations for improving the speed of a website.

NeXpert does not provide a numerical rating or anything of that kind, but rather focuses on very granular statistics and details about the problems found by the tool. The generated report includes recommendations and links leading to best practices and concrete suggestions to solve the problems at hand.

Besides neXpert, Fiddler also provides some built-in features such as a method to simulate slow network speeds and ways to spoof almost any relevant information, making it very well suited for in-depth analysis of where there are problems to be found.

4.3.3 Conclusion

To summarize, Google Page Speed is definitely the choice if the user wants some straightforward, easy to use and fast way to improve the performance of a website. It gives a well compressed overview and the suggestions provided by the tool are rock solid and will definitely have an improving effect.

Fiddler on the other hand isn't as simple and users have to actually download and install the core program itself and any needed extensions themselves. If the user manages to get the program working, it provides a large amount of functionality and fairly easy to use extensions like nExpert which will provide a detailed report with recommendations and the exact locations of the concerned files.

References

BuildwithMobile [2011]. *Mobile Web Technology Report 2011.* http://trends.builtwith.com/Reports/Mobile-Web-Technology-2011/Mobile-Web-Technology-2011.html.

BusinessWeek [2008]. *BusinessWeek.com.* http://www.businessweek.com/innovate/content/jun2008/id20080623_013282.htm.

Cristobal, V. [2010]. *Mobile Web Mashups: The long tail of mobile applications.* http://www.mobilemashups.com/Mobile_Web_Mashups.pdf.

ElatedTen [2011]. *10 Ways the Mobile Web is Different.* http://www.elated.com/articles/10-ways-the-mobile-web-is-different/.

Eric Lawrence [2009]. *Become a Web Debugging Virtuoso with Fiddler.* http://www.microsoftpdc.com/2009/CL25.

Eric Lawrence [2011]. *Fiddler Wiki.* http://fiddler.wikidot.com/.

Fiddler [2011a]. *Fiddler Blog.* http://blogs.msdn.com/b/fiddler/.

Fiddler [2011b]. *Useful Extensions for Fiddler.* http://www.fiddler2.com/Fiddler2/extensions.asp.

Google [2011a]. *mod-pagespeed Overview.* http://code.google.com/speed/page-speed/docs/module.html.

Google [2011b]. *Page Speed Online.* http://code.google.com/speed/page-speed/docs/online.html.

Google [2011c]. *Page Speed Online.* https://developers.google.com/pagespeed/.

Google [2011d]. *Page Speed Service - Google.* http://code.google.com/speed/pss/docs/overview.html.

GoogleBestPractices [2011]. *Web Performance Best Practices.* http://code.google.com/intl/de-DE/speed/page-speed/docs/rules_intro.html.

GoogleCodeMobile [2009]. *Gmail for Mobile HTML5 Series: Reducing Startup Latency.* http://googlecode.blogspot.com/2009/09/gmail-for-mobile-html5-series-reducing.html.

GoogleDNS [2011]. *Minimize DNS lookups.* http://code.google.com/speed/page-speed/docs/rtt.html#MinimizeDNSLookups.

GoogleMobile [2011]. *Optimize for mobile.* http://code.google.com/intl/de-DE/speed/page-speed/docs/mobile.html.

GoogleParallelize [2011]. *Parallelize downloads across hostnames.* http://code.google.com/speed/page-speed/docs/rtt.html#ParallelizeDownloads.

GoogleRequest [2011]. *Minimize request overhead.* http://code.google.com/speed/page-speed/docs/request.html.

GoogleRTT [2011]. *Minimize round-trip times.* http://code.google.com/speed/page-speed/docs/rtt.html.

Hiren, N.; Billy B.L. [2008]. *Mobile Computing With Web 2.0: Current State-Of-The-Art, Issues and Challenges.* http://www.iacis.org/iis/2008_iis/pdf/S2008_1104.pdf.

John Hrvatin [2009]. *Building High Performance Web Applications and Sites.* http://channel9.msdn.com/Events/MIX/MIX09/T53F.

Kroski, Ellyssa [2009]. *On the Move with the Mobile Web: Libraries and Mobile Technologies.* http://eprints.rclis.org/bitstream/10760/12463/1/mobile_web_ltr.pdf.

Matt Murphy, Mary Meeker [2011]. *Top Mobile Internet Trends.* http://www.techweb.com.cn/special/download/kpcb.pdf.

Maximilien, M.E. [2008]. *Mobile Mashups: Thoughts, Directions, and Challenges.* http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04597253.

MogisticCondition [2007]. *The Condition, Challenges and Opportunities of the Mobile Web.* http://www.mogistic.com/assets/mogistic/pdfs/WhitePaper-TheCondition,ChallengesandOpportunitiesoftheMobileWeb-v07252007.pdf.

Morgan, S. [2009]. *The Mobile Internet Report.* http://www.morganstanley.com/institutional/techresearch/pdfs/Theme_8_Regulatory_Impact.pdf.

neXpert [2011]. *neXpert Blog.* http://blogs.msdn.com/b/nexpert/.

NokiaFirst [2011]. *Nokia.com.* http://www.nokia.com/NOKIA_COM_1/About_Nokia/Sidebars_new_concept/Nokia_firsts/Firsts.pdf.

NubiqGrowing [2011]. *Growing Use of Mobile Websites.* http://www.nubiq.com/white-papers/GrowingUseofMobileWebsites.pdf.

OperaMobile [2011]. *State of the Mobile Web, February 2011.* http://media.opera.com/media/smw/2011/pdf/smw022011.pdf.

OReillyFast [2009]. *Making Your Site Fast.* http://radar.oreilly.com/2009/07/velocity-making-your-site-fast.html.

Passani, Luca [2010]. *Global Authoring Practices for the Mobile Web.* http://www.passani.it/gap/.

QuantcastTop [2011]. *Quantcast Top Sites.* http://www.quantcast.com/top-sites-1.

RFC-HTTP [1999]. *RFC Hypertext Transfer Protocol – HTTP/1.1 – section 8.* http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html#sec8.1.4.

Souders, Steve [2007]. *High Performance Web Sites.* O'Reilly, CA, USA. ISBN 0596529309.

Souders, Steve [2009]. *Even Faster Web Sites.* O'Reilly, CA, USA. ISBN 9780596522308.

SoudersVelocity [2011]. *Steve Souders: High Performance Mobile.* http://www.youtube.com/watch?feature=player_detailpage&v=5ENYA_RCCjM.

StrangeloopCustomers [2011]. *Mobile Site Performance is Turning Customers Away.* http://www.strangeloopnetworks.com/resources/product-information/whitepaper-strangeloop-mobile-website-optimization/mobile-site-performance-is-turning-customers-away/.

StrangeloopMobile [2011]. *Mobile Site Optimization.* http://www.strangeloopnetworks.com/assets/PDF/downloads/Whitepaper-Mobile-Site-Optimization.pdf.

Telecomvisions [2011]. *Explaining the success of NTT DOCOMO's i-mode wireless internet service.* http://www.telecomvisions.com/articles/pdf/fransman_imode.pdf.

VisionMobile [2011]. *Developer Economics 2011 – Winners and losers in the platform race.* http://www.visionmobile.com/blog/2011/06/developer-economics-2011-winners-and-losers-in-the-platform-race/.

W3Mobile [2008]. *Mobile Web for Social Development Roadmap.* http://www.w3.org/2008/MW4D/docs/mw4d_roadmap_color.pdf.

WikipediaDataURL [2011]. *Data URI scheme.* http://en.wikipedia.org/wiki/Data_URI_scheme.

WikipediaSOP [2011]. *Same origin policy.* http://en.wikipedia.org/wiki/Same_origin_policy.

YouToArtNavBar [2011]. *YouToArt.com.* http://www.youtoart.com/html/Vector/Icon_Logo/7999.html.

YouToArtSprite [2011]. *YouToArt.com.* http://www.youtoart.com/html/Icon/Website/12245.html.

Zhang, D.; Adipat B. [2005]. *Challenges, methodologies, and issues in the usability of testing mobile applications. International Journal of Human-Computer Interaction*, 18(4), 293–308. doi:10.1207/s15327590ijhc1803_3.