

MICHAEL CALE
CODING
<for>
SPEED

A Hacker's Guide to a Faster Web

Coding For Speed

A Hacker's Guide to a Faster Web

Michael Cale

© 2014 - 2015 Michael Cale

A jug fills drop by drop

– Bhudda

Table of Contents

[Important Information](#)

[Your free gift](#)

[The aggregation of marginal gains](#)

[Building for speed](#)

[Book layout](#)

[Why Page Speed Matters](#)

[Three speed limits](#)

[Fast = a better user experience](#)

[Mobile is the internet](#)

[How slow is the web?](#)

[If search matters, speed matters](#)

[Slow equals less revenue](#)

[How the web communicates](#)

[Some definitions](#)

[Under the hood of the web browser](#)

[The journey of a packet request](#)

[Caching for fun and profit](#)

[Font rendering](#)

[Major Actions for Speed](#)

[Cache](#)

[Use ETags](#)

[Avoid redirection](#)

[Minimize data transfer](#)

[Replace Apache with Nginx](#)

[Avoid iframes](#)

[Hardware matters](#)

[Images](#)

[Image Techniques](#)

[Keep the image, lose the stuff](#)

[File type overview](#)

[What's out there?](#)

[Use png for art and jpg for photos](#)

[Use compression](#)

[Use cacheable components](#)

[Use progressive jpg](#)

[Replace images with CSS3 gradients](#)

[Use CSS sprites](#)

[Don't store images in a database](#)

[Use preloading appropriately](#)

[Use lazyload images](#)

[Avoid scaling images](#)

[Use base64 encoding](#)

[Don't forget your SVG](#)

[Page Techniques](#)

[DOM Architecture](#)

[CSS techniques](#)

[JavaScript techniques](#)

[DOM techniques](#)

[Server techniques](#)

[Wordpress](#)

[Cache](#)

[Optimize images automatically](#)

[Keep the database tidy](#)

[LazyLoad images](#)

[Remove gravatar images](#)

[Eliminate bandwidth leaks](#)

[Deactivate unused plugins](#)

[Optimize the homepage](#)

[Remove unnecessary PHP functions](#)

[Wordpress optimization](#)

[A Wordpress case study](#)

[Moving my personal blog \(wordpress\)](#)

[First look](#)

[Step 1 - image size](#)

[Step 2 - homepage streamlining](#)

[Second Look](#)

[Step 3 - General cleanup](#)

[Third Look](#)

[Conclusion](#)

[Further improvement](#)

[Simplify your CSS](#)

[Remember the basics](#)

[General layout](#)

[User anchor elements](#)

[Group similar CSS](#)

[Use hex colors](#)

[Browser-specific styles](#)

[Validate your CSS](#)

[Compress your CSS](#)

General PHP tips

[Variables](#)

[Output](#)

[String Operations](#)

[Avoid Regex \(in general\)](#)

[Files](#)

[Logic and Iteration](#)

Geeking out on TCP

[Characteristics of a TCP connection](#)

[Optimizing TCP](#)

[Further Reading](#)

Never stop learning

[Web Performance Today](#)

[Mobilewebbestpractices.com](#)

[Mobilexweb](#)

[Zoompf web performance blog](#)

Webmaster Tools

[Speed](#)

[File Compression](#)

[CSS](#)

[Image Compression](#)

[Base64 Encoding](#)

[Validation](#)

[Minification](#)

[SVG Optimization](#)

[Sprites](#)

[Browser developer tools](#)

[Validation](#)

[Server Tools](#)

[Analytics](#)

Free Gift

[Blog](#)

Important Information

Your free gift

As a way of saying *thank you* for your purchase, I'm offering a free checklist to the readers of this book. To get your Checklist <for> Speed, send an email to ****checklist@michaelcale.com****.</for>

Disclaimer

While all reasonable attempts have been made to verify the information provided in this work, the author does not assume any responsibility for errors, omissions, or contrary interpretations of the subject matter. The views expressed in this book are those of the author alone and should not be taken as expert instruction. The reader is responsible for their own actions.

A Special Thanks

Several tables and images in Chapter 8 are courtesy of webpagetest.org.

The aggregation of marginal gains

Dave Brailsford was asked to do what had never been done.

As the new General Manager of Great Britain's professional cycling team (Team Sky), he was asked to lead the team to a victory in the Tour de France.

Inspired by the book *Moneyball* by Michael Lewis, Brailsford took a clean review of cycling's standard measurements and thought processes. He developed a new and innovative approach.

Initially, there were no grand, sweeping changes to the team training program. But there were hundreds of tiny, seemingly trivial changes. His goal was a 1 percent improvement in every area. And he meant *every* area.

He analyzed and focused on areas that other teams considered meaningless, or had never considered. He theorized that the combination of these tiny improvements would, in the end, be larger than their sum.

When he was hired, he was told the team goal was to win the Tour de France in 5 years. The team won it in two.

“People often associate marginal gains with pure technology, but it is far more than that. It is about nutrition, ergonomics, psychology. It is about making sure the riders get a good night's sleep by transporting their own bed and pillow to each hotel. It is about using the most effective massage gel. Each improvement may seem trivial, but the cumulative effect can be huge.” – [Dave Brailsford](#)

Yes, he said transporting their own bed to each hotel. Actually, it was just the mattress, duvet, sheets, and pillows. And Team Sky staff cleaned each hotel room before arrival to make sure the athletes were comfortable.

Even the team's bus was designed with a focus of making life easier for the athletes. It has custom seats, a cinema, and showers to ease the pre and post-race routines. It has a meeting room for strategy sessions whose glass becomes opaque at the touch of a button. The large black bus is nicknamed the Death Star.

Team Sky went on to dominate the 2012 Summer Olympics. Great Britain won 12 total medals in Cycling including 8 Gold, more than double the medal count of any other nation. Team Sky followed that up with a back-to-back win of the Tour de France in 2013 as well.

Building for speed

The reason we digress into cycling is to introduce this valuable concept.

The aggregation of marginal gains is the exact approach that will dramatically improve performance of web sites and services.

Taken alone, many of the approaches in this book may not provide notable (or even *noticeable*) improvements. But taken on the whole, there is room to significantly improve the speed of web services.

For more on incorporating this principle into your personal life, please read *This Coach Improved Every Tiny Thing by 1 Percent and Here's What Happened* by James Clear <http://jamesclear.com/marginal-gains>.

Book layout

In general, the techniques discussed in the book are presented with major techniques first. That is, the techniques presented in the early chapters will, in general, give you more improved performance than the techniques listed in later chapters.

Now, before we get into the specific techniques, let's review the importance of speed in web design and why it matters.

Why Page Speed Matters

Three speed limits

The [research of human perception](#) has identified three important speed limits for any user interface.

- 0.1 seconds - the upper limit where the user feels the application is responding immediately. For example, touch events on a touchscreen.
- 1.0 seconds - the upper limit where the user feels that the application is working, but does not feel 'sluggish'. The user notices the delay, but does not lose their sense of focus.
- 10 seconds - the upper limit for users to retain interest in the task. Best practices for delays in excess of ten seconds are a progress indicator and a method for the user to abort the task.

These are thresholds of human perception that have been repeatedly verified by decades of research. This is one reason why Google [recommends](#) that web pages load in less than one second on mobile devices.

Fast = a better user experience

Users love speed.

- 74% of [mobile users](#) will abandon a site after a 5-second wait.
- 69% of tablet users expect a site to load in 2 seconds or less.

Mobile *is* the internet

According to Akamai's [State of the Internet](#), mobile data volume is doubling year-over-year. It won't be long before desktop web use is going to be dwarfed by mobile use.

All these mobile devices are *radios*. When they connect over a cellular network, they must establish a radio connection before any data can be sent or received. Establishing a radio connection can take **several seconds**. Once established, each radio connection has a *timeout* period. After this time period, the channel goes idle and a new radio channel must be established.

The causes **latency** when mobile devices connect over cellular networks. In the US, round-trip transmissions on 3G networks are normally 100-450 ms. On 4G networks, they are usually 60-180 ms. In the UK 3G network latency is 289-2,653 ms for 3G and 35-187 ms for 4G.

This latency is unavoidable, so web developers must structure their work to be as fast as possible to deliver the best user experience.

How slow is the web?

According to [Google](#), the average page load time for mobile is just over 7 seconds. The median page load time is slightly more than 3 seconds, so there are some extremely slow web sites that are skewing the average.

The majority of web pages load in between 1 and 7 seconds on a mobile device. Less than 10% of web pages load in 1 second or less on mobile. That should be your goal - one second load time or less on a mobile device. That will put you in the top 10%.

According to Google, average web page size has increased 56% since last year.

If search matters, speed matters

In 2010, Google [announced](#) they would start using page load times as a factor in how they rank web pages.

So if search engine results are important to you, then speed should be important to you as well.

Slow equals less revenue

Amazon [reported](#) that every 100ms *decrease* in page load time *increased* revenue by 1%.

[Mozilla](#) reduced the loading time for their Firefox download page from an average of 7 second to 4.8 seconds. In an A/B test between the two sites, the faster site had 15% more download conversions.

Google [discovered](#) that increasing latency by 400ms reduced the number of searches by 0.2%.

Your users demand speed **and** a rich online experience regardless of what device they are using and the quality of their connection. It's your job to deliver and to deliver quickly.

How the web communicates

Before we get into the specific techniques, let's cover some basics of the browsers and the Document Object Model (DOM).

Some definitions

bandwidth - [noun] - *amount of data transferred per unit of time.*

latency - [noun] - *delay in data transfer time, specified as the amount of time from sending a data request and receiving the response (round trip time).*

render - [verb] - *to cause to become; to make.*

parse - [noun] - *the result, or act of obtaining by processing text.*

Under the hood of the web browser

All web browsers have the same basic purpose. They request a resource from the server and display it to the user or client. Most often this resource is a web page in the form of HTML, but it could also be an image, a PDF, or other kind of content.

The major browsers in operation today are Chrome, Firefox, Safari, Internet Explorer (IE), and Opera on the desktop. On mobile devices, there are iPhone, Android, Chrome, Opera (Mini and Mobile), UC Browser, and Nokia. All of these browsers are now webkit-based with the exception of Chrome (versions 28+) and Opera (versions 15+) which use the Blink rendering engine.

WebKit is the piece of software that powers the layout engine. This is the part of the browser that takes the content (HTML, images, etc.) and the formatting information (CSS) and displays it properly on the screen.

The Document Object Model (DOM) defines the logical structure of HTML and XML documents. The DOM identifies the objects in a page and defines their behavior and attributes.

In the simple DOM below, the DOM defines the header and paragraph in the HTML document.

```
1 <html>
2     <head>
3     </head>
4     <body>
5         <h1>This page is amazing!</h1>
6         <p>This is my favorite paragraph</p>
```

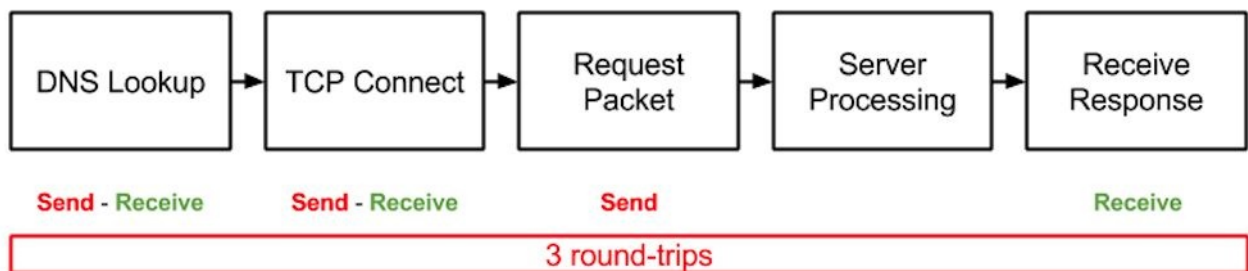
```
7     </body>
8 </html>
```

This page is amazing!

This is my favorite paragraph

The journey of a packet request

Requesting a data packet

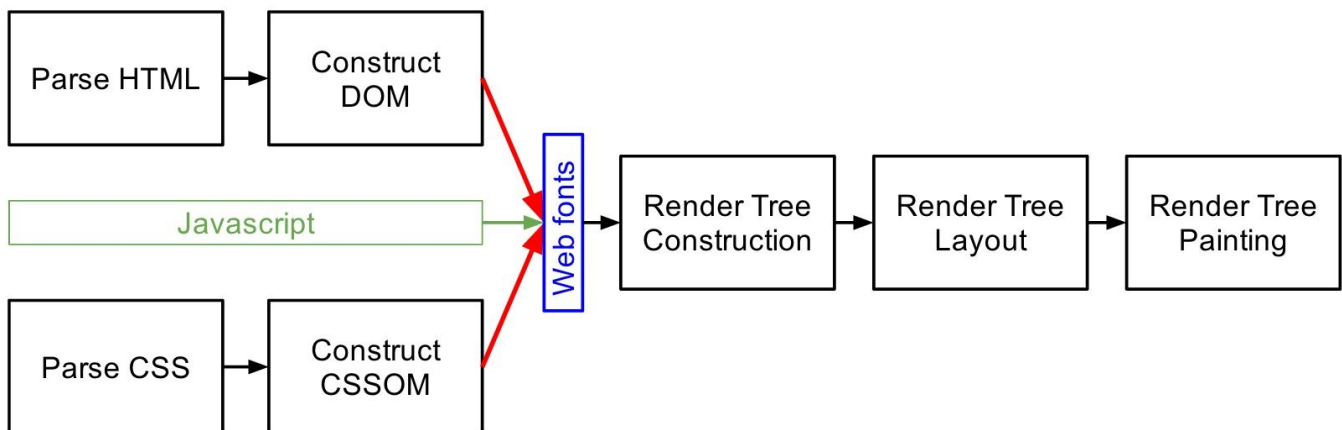


Packet Data Flow

Before we can start building the DOM, there are a few steps that must be accomplished. First, we have to do a DNS lookup to translate the URL to an IP address. That's one round-trip to a server and back. Second, we have to accomplish a TCP connect. We'll cover an especially geeky chapter on the TCP process later. This TCP connection is a second round-trip. Once these two steps are complete, we can finally request some actual HTML. So we send a third request and once that round-trip is complete, we may actually receive our first data packet that has something we can show to the user.

Rendering steps

Rendering engine basic flow



Once the data packets are flowing to the browser, it begins parsing the HTML and CSS. As the browser is parsing the text, it begins constructing the Document Object Model and the CSS Object Model (CSSOM).

Both the DOM and CSSOM must be complete for the displayed elements before the render tree can be constructed. The render tree is the combination of the content with style information from internal and external locations.

Render tree construction can become blocked by DOM construction, CSS construction, or javascript. Parsing stops when Javascript is encountered and only continues after the script is completed. If the script is external, the browser must retrieve the script over the network. Exceptions to this occur if the script is tagged defer or async (HTML5). We'll cover those in more detail later.

Prerequisites for building the render tree for above-the-fold display:

- HTML parsed
- CSS parsed
- DOM complete
- CSSOM complete
- Synchronous javascript complete

The render tree is built in a series of 'chunks' with style attributes, like color and dimensions. These chunks are built in the proper order to layout on the screen. The layout process assigns each piece of the DOM with the exact screen coordinates for where it should appear.

The browser's rendering engine will attempt to display contents to the screen as soon as possible. It will start building the render tree and layout before the total page HTML is fully parsed if possible.

Caching for fun and profit

A cache is a location for storing data received from a server. It can be HTML, images, files, etc. The purpose of using a cache is:

1. Reduce latency
2. Reduce network traffic

These also increase speed and provide an improved user experience.

The cache may reside on the users hard drive, or the cache may be on a proxy server between the client and the host web server. These may be at an Internet Service Provider, or part of a corporate network, or part of a Content Delivery Network (CDN).

When a visitor comes to your site for the first time, everything has to be loaded from the your server. All images, content, scripts, style sheets, etc. But much of that can be stored in the browser's cache, so when the user returns, much of the site can be loaded from the cache, which is much faster. There's no reason for the user to download the images for your navigation buttons when they are right there on the user's hard drive.

You want to make sure that your site is set up properly for caching because you don't want to deliver *stale* or out-of-date information to your users.

Planning your site to use caching properly will provide a speedy experience for you users while reducing the load on your server.

Every cache has certain rules to determine whether information is *fresh* or *stale*. *Fresh* content is valid to be sent to the client while content that is *stale* will normally be refreshed from the host. Under some circumstances, the cache may serve stale items to the user, such as if the host server cannot be reached.

The two primary tools to control how your site is cached are the HTTP headers Expires and Cache-Control.

Sample HTTP Header

```
1 HTTP/1.1 200 OK
2 Last-Modified: Sat, 22 Feb 2014 10:44:00 GMT
3 Expires: Sat, 01 Mar 2014 10:44:00 GMT
4 Content-Type: application/ocsp-response
5 content-transfer-encoding: binary
6 Content-Length: 1443
7 Cache-Control: max-age=489464, public, no-t\
8 ransform, must-revalidate
9 Date: Sun, 23 Feb 2014 18:46:30 GMT
```

The Expires header allows you to set a specific date in Greenwich Mean Time (GMT) for the item to expire. This technique relies on the system time of both the server and the cached system. If one is significantly off, results can be unexpected. Once set, there is no way to force a cached object to refresh before the expiration. In such a situation, one technique would be to rename the object, which would force the cache to fetch the *new* object.

Introduced in HTTP version 1.1, the Cache-Control header offers a more flexible approach for controlling how your site is cached. There are several settings that can be

specified using `Cache-Control`.

`max-age` = specifies the maximum number of seconds item is considered *fresh* (from the time of request)

`must-revalidate` = forces the cache to refresh item when the item becomes stale

`no-cache` = may not be cached

`no-store` = instructs the cache not to store the request or any response to it

`private` = allows user-specific caches (such as a user's browser)

`public` = may be cached

For a complete list of `Cache-Control` settings, see the [HTTP 1.1 protocol](https://tools.ietf.org/html/rfc7234).

Setting a Cache-Control header in PHP

```
1 <?php
2     Header("Cache-Control: must-revalidate");
3     $expiration = 60 * 60 * 24 * 7;
4     $expirationstring =
5     "Expires: " .
6     gmdate("D, d M Y H:i:s", time()
7     + $expiration) . " GMT";
8     Header($expirationstring);
9 ?>
```

Setting Cache-Control header in HTML

```
<meta http-equiv="Cache-Control" content="public">
```

When both `Expires` and `Cache-Control` headers are present, the `Cache-Control` settings take priority

Set extremely large time ranges on items that rarely change, like navigation buttons or other static images. According to the specification, the maximum you should set is one year, or 31,536,000 seconds, although that is probably extreme. Setting cache parameters for specific file types can be done in the `htaccess` file (for Apache servers).



Avoid using the `Pragma: no-cache` header. It's behavior is unspecified and implementation varies greatly.

Etags

While not a replacement for proper `Cache-Control` use, etags can be a useful resource, especially if using a CDN that supports them.

Entity-tags, or etags are are unique identifier for the resource being requested. Often, they are set to a hash of a timestamp that the resource was last updated. These allow proxies (or CDNs) know that if the etag matches what they have stored, then they still have a valid resource.

Make caching work for you

First and foremost, identify the largest resources on site that are served the most often. These will likely be images or other files. If they are static, make them cacheable with a large max-age value.

If one of these resources changes, make sure to use a different name for the new version. This will prevent the old cached version from being served to your users.

For files and images, create a common storage location so that the address is consistent even if you have several references to the asset on different pages.

Be consistent in your URL addresses. Ideally, multiple users accessing the same page should see the same URL. This will enable caching to consistently serve up the appropriate resources from the most appropriate location.

Font rendering

Web fonts have become extremely popular. Google [reports](#) that its [fonts](#) are serving up a billion font-views each day across over 100 million web pages.

Could this be a problem? What if the font takes a long time to load? It depends on the browser.

Firefox holds rendering for up to 3 seconds to allow the font to load. After 3 seconds, it uses a fallback font and then re-renders when the download is complete.

Safari and Chrome both hold rendering until the font is loaded. Chrome is being modified to mimic the Firefox behavior in the future.

Internet Explorer renders with the fallback font and re-renders once the font is loaded.

Both the DOM and the CSSOM must be complete before any web fonts are downloaded. The font(s) will only be downloaded if they are needed for the current page.

Font loading is easily found on the waterfall charts since the URL will be included in the event name.

Major Actions for Speed

Cache

The first time a user visits your site, everything must load. That means the maximum amount of requests and responses. Theoretically, that is the slowest it will be.

If your slow site didn't scare them away completely, they may come back.

When they do, their browser will likely have saved much of your site in the browser's cache. So it will load more quickly.

There are a few things you can do on site to assist in this process.

Set a large Expires header for static components. This explicitly makes these components cacheable and tells the browser how long they are valid. If you tell the browser an image expires in 5 or 10 years, the browser will use the cached version when your user returns until that time runs out. After the expiration date, the browser will reload that component from the server.

If you do need to change a component that has a far-future expiration, then make sure the replacement has a different name and it will load from the server.

For dynamic components, add a Cache-Control header.

Make your AJAX cacheable

Remember that browsers cannot cache anything with a query string, so if your links contain a ?, then try to find a dynamic link to that resource (one that doesn't require the ?).

Use ETags

An ETag is a unique identifier you can assign to a specific web address. When a browser has a cached version of that asset, it will check the ETag. If the ETag is the same, the server will respond with a 304 response (Not Modified). The browser will use the cache.

If the ETag has changed, the browser loads the latest version from the server.

When using ETags, make sure you have the correct procedures in place to make sure they are updated as your site assets are updated.

Avoid redirection

Redirection is a very processor-intensive activity that is often unnecessary.

In addition, the header responses 301 moved permanently and 302 moved temporarily are not cached unless additional headers require it. Expires and Cache-Control would require caching.

For Apache users, make sure to use `Alias` or to escape trailing slashes in `mod_rewrite` so your server won't redirect `www.url.com/title` to `www.url.com/title/`. This is an automatic 301 redirect that occurs frequently.

Also, make sure that any redirects go directly to the desired location. If pages A and B both are required to redirect to page C, you never want to redirect from page A to page B (then to page C). Just have each page redirect straight to page C.

Minimize data transfer

Anything that requires the web server to access disk storage or anything across the network (like accessing the database server) is going to be costly.

When you need to hit the database, do it. Just make sure you aren't making two trips to the database when it's possible to make a single trip.

The same principle applies for file access.

Replace Apache with Nginx

Nginx (pronounced Engine X) is an open source replacement for Apache. Since it is event-driven instead of process-driven it uses much less memory. This makes it extremely fast.

Nginx excels at serving static pages.

It doesn't have nearly the number of features as Apache, but if you want to serve static pages with lightning speed, consider the switch.

For more on Nginx, see wiki.nginx.org

Avoid iframes

iFrames are a convenient technique to embed content from another web site, but you should avoid using them whenever possible.

iFrames are one of the most expensive DOM elements to load. They are far more expensive than scripts and style sheets. Most browsers will at least *download* scripts in parallel with other assets (stylesheets, other scripts, images). Most browsers will *not* download scripts in parallel with iFrames.

In addition, loading iFrames generally block the window's `onLoad` event, which is when user sees that page loading is complete. This event doesn't fire until all resources, including iFrames, are fully loaded. In Safari and Chrome, you can set the `SRC` property using javascript, which will prevent this blocking behavior.

To determine whether to use an iframe, consider the content. Which is most important - the content in the iframe or the content on the page? If the iframe content is a priority, then include it and suffer the performance reduction.

Hardware matters

Suppose you sign up with a new host. They put you on a decent server that is say, two years old. Time goes by and you stay with the same hosting company.

Now it is five years later. Is your site now running on a seven-year-old machine? You may not know until you ask, or until something goes awry.

If you aren't getting the hardware upgrades you think you deserve, then find a new host. Having a solid migration plan always ready makes this much easier to accomplish.

Images

Images are probably the single most critical element in your page loading process. It's so important that we'll dedicate an entire chapter to image techniques.

Image Techniques

Overall, image compression is likely to provide the most speed increase for level of effort than any other technique. If you aren't optimizing your images, this is probably where you want to begin.

Keep the image, lose the stuff

Image files come with all kinds of information that are not the image itself.

There is often embedded thumbnails, color palettes, and metadata. These are useful to graphics designers, but they bloat the file size. This extraneous information can be easily removed through compression, which we'll cover shortly. First, a bit about image file types.

File type overview

GIF files use a table of 256 colors and several algorithms to find the optimum set of 256 colors to represent the colors in the image. Limiting the number of colors is a form of compression. Another method of compression is that the file format uses an abbreviated notation when there are large areas of uniform color. **GIF** files retain the transparency from the original image.

Use **GIFs** for small uses like buttons and for web animations.

JPG files maintain high image quality in a compact format. This file type is optimized for photographs. Use when you are willing to sacrifice some image quality in order to minimize file size. **JPG** files do *not* retain transparency.

If you have images that contain text, simple shapes, or large areas of uniform color, do not use **JPG**.

PNG files provide high image quality and small file size. **PNG** file compression is reversible so the original image can be recovered.

TIFF images are high quality pixel-based images. They often have very large file sizes. Avoid using **TIFF** images whenever possible.

WebP is an image type developed by Google. They are routinely smaller than **PNG** and **JPG** images and they support transparency. Google is beginning to use these on their properties (Google Play store and YouTube). So far, Chrome is the only browser that has native support, so probably best to avoid for now. You can read more about **WebP** at Google's [developer site](#).

For most web images, use **PNG** wherever possible. Exceptions are **JPG** for photos or images with a lot of colors, and **GIF** for animation.

What's out there?

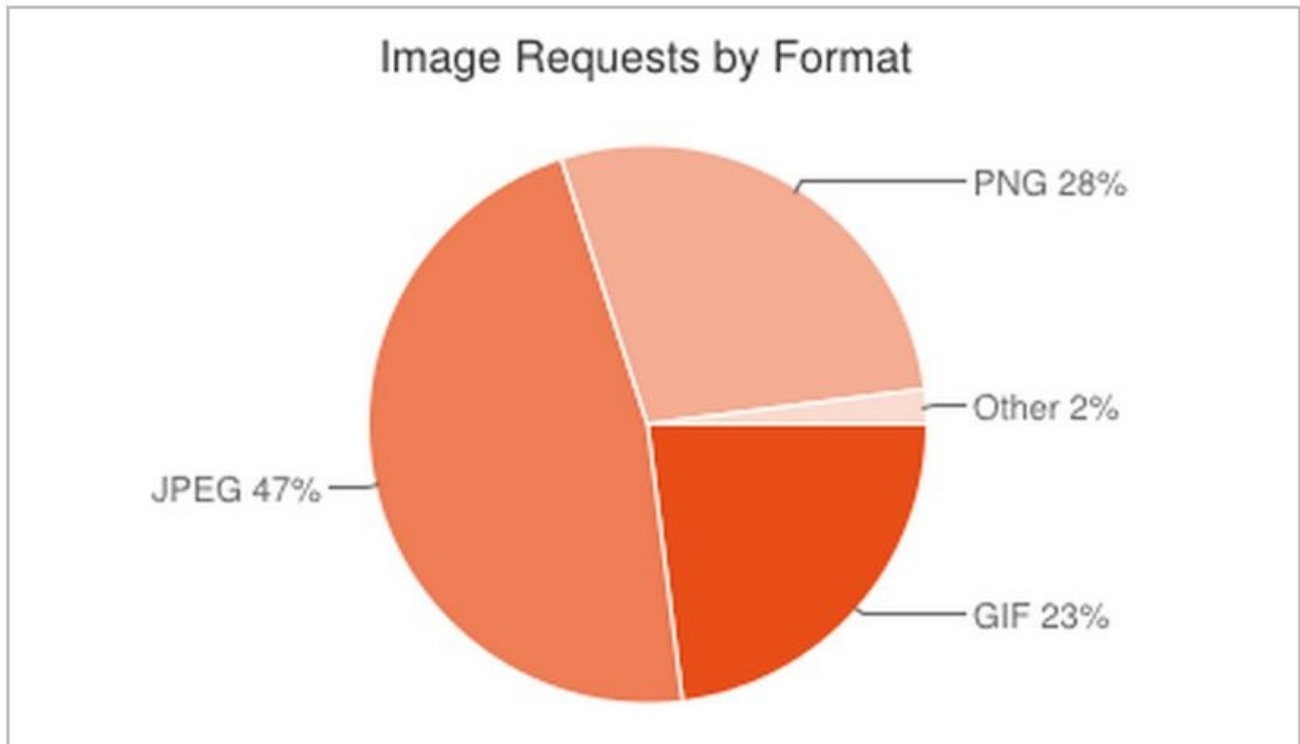


Image source: <http://httparchive.org>

Use png for art and jpg for photos

PNG is the preferred image file type for speed, unless you require the use of animated images.

JPG is normally the best format for photographs.

Compress your **GIF** files and see if there is any file size improvement to using **PNG**.

Use compression

Compression allows you to transfer data in a smaller file size. Lets look at a simple compression scheme.

Data before compression - xxxxxxxxyyyzzzzz

Data after compression - x7y4z5

Knowing how the data is 'compressed', we could receive the *after-compression* string and be able to reconstruct the original data perfectly. This would be an example of **lossless** compression since we could transfer the data with absolutely no loss of information.

Always keep a copy of your original images before you compress them.

The second type of compression is known as **lossy**. Lossy compression does allow some data degradation.

For example:

Data before compression - {3.129, 4.021, 9.713}

Data after compression - {3.1, 4.0, 9.7}

Using lossy compression for image will result in degradation of the image quality. But **do not automatically disregard lossy image compression**. Use both types and compare image quality. If there is minimal (or no) perceptible distinction between the two images, then use the one with the smallest file size.

Use [Smush.it](#), [PunyPNG](#), [ImageOptim](#), [Optimizilla](#) , or a similar service to reduce image size as much as possible without sacrificing image quality.

These tools can reduce image size by near 40%. Do that for all the images on your site and you can make a huge impact on loading time.

PNG files can be lossy compressed with <http://pngquant.org>

Other image compression tools include [pngcrush](#) and <http://compresspng.com>.

Use the utility jpegtran on all your **JPG** images. It executes lossless compression and removes superfluous information like comments and EXIF data. It is available at <http://jpegclub.org>.

Use cacheable components

For any images, scripts, and stylesheets that are over 25kb in size, the iPhone will not cache them.

Keeping these items under this 25kb speed limit will greatly increase speed for return users.

Use progressive jpg

A baseline JPG image is comprised as one top-to-bottom scan of the image. A progressive JPG builds the image with a series of separate scans so the image quality improves with each scan.

The image quality and file size are roughly equivalent in both types of JPG (progressives are often slightly smaller in size), but the progressive scans provide a better user experience. Instead of watching the complete image load from top to bottom, the image size is readily apparent and the image quality improves quickly. This is particularly beneficial for users with slow network connections.

Progressive JPGs are just a different way of transferring the image data. Indeed, with a fast enough data connection, you may not be able to notice the difference between the two.

If you're using an image optimization tool like [Smushit](#), it will convert your JPGs to progressive.

Other image tools such as [Photoshop](#) or [ImageMagick](#) will give you the option to use progressive JPG.

Replace images with CSS3 gradients

A straightforward method is to remove the image request entirely and replace the image with CSS3 gradients. They can be overlaid over the background color and they handle transparency well.

This will add more overhead to your CSS load, but it should be a faster option than the image request.

Use CSS sprites

Another technique is to combine all your (required) images into a single http request with CSS sprites.

This actually combines several (or all) your images into one image with defined X and Y coordinates and uses the CSS property background-position to put everything in the proper place.

This is particularly good when there are a large number of small images, such as for menu icons.

There are two approaches here. Using a modular sprite approach would use a CSS sprite for part of the page, like for a navigation menu. An uber sprite approach would combine all images for a single page into one CSS sprite.

If your site has a large number of pages, using CSS sprites can become maintenance-intensive. Using sprites is much more complex than straightforward image use.

Google's search results page uses the uber sprite approach.

CSS Sprite Generator

<http://spritegen.website-performance.org>

Tips for CSS sprites

1. Optimize the images before creating the sprite.
2. Avoid images with a large amount of whitespace.
3. Arrange images horizontally to make the sprite as small as possible.
4. Organize images of similar color themes together.

Don't store images in a database

Some sites store their images in a database as a [blob] data type. This is inefficient.

Instead, store the image file name (or a portion of it) in the database if you must, and build the link dynamically including any required information (e.g. user id). Then let the server load the file directly. This is faster than storing the image in a database.

Use preloading appropriately

Only pre-load images when:

- the image isn't shown by default, but may be needed later. Hover and click state images, for example.
- the image isn't used on the current page, but will be used on the next page (or there is a high probability it will be required).

Use lazyload_images

If you are using the `mod_pagespeed` module (more on that later), you can load the images lazily. `lazyload_images` is a server filter that defers the loading of images that are not yet viewable by the client.

It accomplishes this by inserting javascript that uses a beacon to report to the server which images are viewable (and therefore need to be loaded). The default behavior is to only load images above-the-fold. Images that are below-the-fold are only requested after they are visible in the client's viewport.

If there is a particular image that you need to load on a page, you can force `lazyload_images` to load it with `pagespeed_no_defer`.

```
1 <img pagespeed_no_defer="" src=.../>
```

Images below-the-fold can be set to load after the `onload` event is triggered or as the user scrolls down the page.

Avoid scaling images

If your image is not the same size that you want to display, avoid this -

```
1 
```

If your image must be 500px by 500px, then change the image size instead of using the browser to scale it up or down.

Use base64 encoding

Another technique is to encode your image into a base64 string.

```
1 
```

The disadvantage of this technique is that the base64 string may be significantly larger in size than the original image. Typically, they are about 10% larger if using `gzip` and 30%+ without compression. The advantage is that you eliminate the server request-response cycle for that image.

This technique normally works particularly well for large gradient background images. Or if your site has several small-size images, it may be beneficial to encode them and eliminate all those `http` requests.

PHP has a convenient function where you can do the encoding on the fly.

```
1 base64_encode(file_get_contents("/path/to/i\  
2 mage"));
```

Internet Explorer 7 and earlier do not support this technique.

Don't forget your svg

While you will get far more improvement out of your efforts toward your image optimization, Scalable Vector Graphics (SVG) share some similarities with images and benefit from some of the same techniques.

SVGs are simple XML files, but many commercial SVG-creation software packages load them up with unnecessary information that, while useful to the program, they are irrelevant for proper display. Adobe Illustrator and Inkscape are common offenders.

Similar to images, these can be validated and cleaned of this extraneous data which can reduce file size by more than 50%.

SVG Optimiser by Peter Collingridge <http://petercollingridge.appspot.com/svg-optimiser>

T Just as with images, save your originals before making changes

Page Techniques

In this chapter, we'll look at techniques to structure your pages for maximum performance. We'll go into deep detail on the Document Object Model (DOM), stylesheets, JavaScript, and some server-side methods as well.

DOM Architecture

All stylesheets and scripts must be downloaded, parsed, and executed before the browser can render the web page. This can dramatically slow response time, especially for mobile users.

For many script-heavy applications, like many AJAX-type interfaces, there could be kilobytes of data to download and process.

The bulk of javascript operations are used for events, like user inputs and navigation, that occur after the page has completed loading.

By structuring your page properly, you can significantly speed up the page load and still load all the necessary script for your application.

Sequence matters

Loading external javascript causes further page processing to stop until the script is loaded. In the example below, the browser loads the external script, then loads the external CSS.

JS Blocks Loading

1. external.js



2. external1.css



3. external2.css



150 ms

The external CSS files are able to load concurrently. Simply by changing the sequence that these files are loaded, the browser can load the three external assets simultaneously, dramatically reducing page load time.

Parallel Loading

1. external1.css



2. external2.css



3. external.js



→ 75 ms

Note: Most modern browsers will *download* external scripts in parallel, but they will only *parse* and *execute* the code in sequence. This is necessary to retain any code dependencies between scripts.

Code Separation

Make sure split your pages into two sections:

1. Required for initial render
2. Everything else

This means you may *not* be putting all your javascript and css in the page header. These should be split to increase page speed.

Start by identifying which of your script functions that are used before the onload event is triggered.

```
1 <html>
2 <head>
3
4     <style type="text/css">
5         .main_nav{...}
6         .sidebar{...}
7         .content{...}
8         /* Place any other styles here if require\
9 d for initial render */
10    </style>
11
12    <script type="text/javascript">
```

```

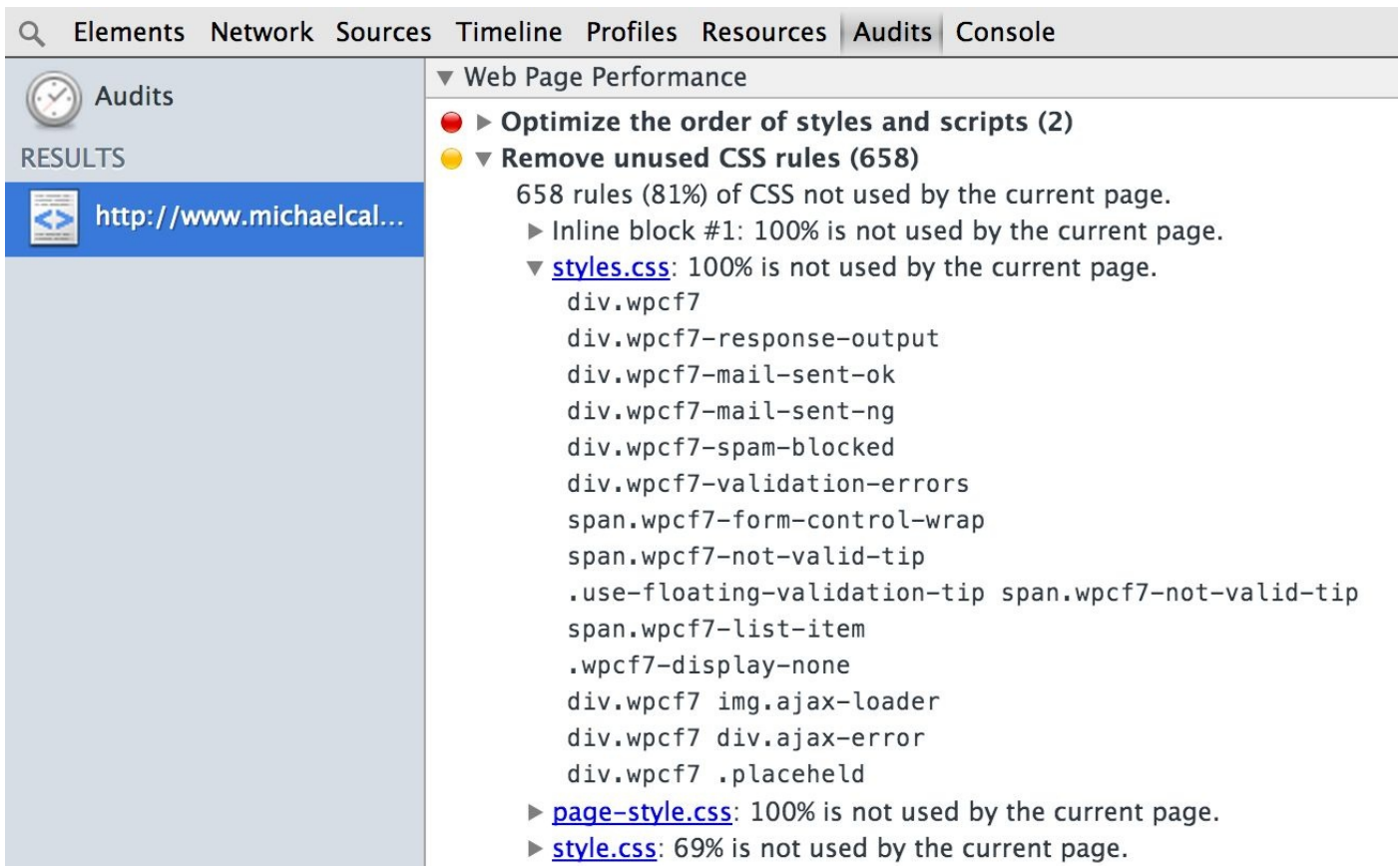
13             /*
14                 Place only the script necessary for init\
15 ial render here
16                 Optimally, no script would be required i\
17 n this location
18             */
19         </script>
20
21 </head>
22 <body>
23     <div class="content">
24         Your content.
25     </div>
26     <div class="sidebar">
27         Sidebar here
28     </div>
29
30     <script type="text/javascript">
31         function run_after_onLoad() {
32             Load('stylesheet', 'remainder.css')
33             Load('javascript', 'remainder.js')
34         }
35     </script>
36
37 </body>
38 </html>

```

Identify critical CSS

If you're building a new site from the ground up, you can plan your CSS so it is divided up in above-the-fold (we'll call it *critical* CSS) and below-the-fold. But if you are working on a completed site, there are tools to help you identify the critical CSS. For Apache servers, there is even a tool we will cover later that can do this automatically.

Use the Chrome DevTools Audit function to identify specific CSS elements that aren't used on each page. This can be found in View -> Developer -> Developer Tools -> Audits -> Web Page Performance.



The screenshot shows the Chrome DevTools Audit panel. The left sidebar has tabs for Elements, Network, Sources, Timeline, Profiles, Resources, Audits, and Console. The Audits tab is selected, showing a list of results for the URL <http://www.michaelcal...>. The main panel displays the 'Web Page Performance' section. Under 'Optimize the order of styles and scripts (2)', there is a red dot icon. Under 'Remove unused CSS rules (658)', there is a yellow dot icon. The text indicates that 658 rules (81%) of CSS are not used by the current page. A list of unused CSS selectors is provided, including `div.wpcf7`, `div.wpcf7-response-output`, `div.wpcf7-mail-sent-ok`, `div.wpcf7-mail-sent-ng`, `div.wpcf7-spam-blocked`, `div.wpcf7-validation-errors`, `span.wpcf7-form-control-wrap`, `span.wpcf7-not-valid-tip`, `.use-floating-validation-tip`, `span.wpcf7-not-valid-tip`, `span.wpcf7-list-item`, `.wpcf7-display-none`, `div.wpcf7 img.ajax-loader`, `div.wpcf7 div.ajax-error`, and `div.wpcf7 .placeheld`. At the bottom, it notes that `page-style.css` is 100% unused and `style.css` is 69% unused.

Another great resource for identifying critical CSS is the article [Detecting Critical Above-the-fold CSS](#) by Paul Kinlan. He has a couple of excellent tools for solving this problem. [This code](#) can be used as a bookmarklet or DevTools snippet to identify the critical CSS on your site. It was developed as a proof of concept, so be sure to read the caveats in the article.

Load external CSS before external JS

External script files load serially in most browsers. So loading an external script file causes the browser to block subsequent loading until the script file is completely loaded.

In contrast, loading of external CSS and images occurs in parallel, up to a limit.

Therefore, always load external CSS files first, and then any external script files so that the CSS file will continue loading in parallel - any loads already in progress will continue.

The key here is to make sure that your scripts do not depend on any of the styles in the external CSS to be able to execute properly.

A good technique is to place any script that is not required for page rendering at the bottom of the page.

CSS techniques

Keep CSS in the <head>

It may be tempting to put your critical CSS in the head and put your non-critical CSS below the above-the-fold content, but this can cause problems.

CSS is required to be in the head, according to the [HTML specification](#).

Furthermore, a few browsers (like Internet Explorer) will prohibit rendering if it encounters stylesheets outside of the header until the stylesheets are loaded. This is to prevent the browser from having to redraw page elements if their style changes. In this case, the user will see a blank white page while loading.

If you are seeing this behavior on your pages then comply with the specification and put your stylesheets in the <head>.

Avoid @import

Using @import in your stylesheet allows you to load additional CSS files. The problem with this approach is that the processing of the main stylesheet stops while the @import operation is loading.

Instead, copy the necessary CSS into your main stylesheet, or use multiple <link> tags.

Use proper CSS selectors

IDs are the most efficient CSS selector.

After ID, the most efficient are class, tag, and universal, in that order.

```
1 content {...} /* ID ->Fastest */
2 .content {...} /* Class */
3 ul {...} /* Tag */
4 * {...} /* Universal ->Slowest */
```

Also, avoid descendant selectors. Don't do this if you can help it:

html body li a {...}

Now it may not be practical to have every element have its own CSS id. That would maximize CSS efficiency, but would probably create insanity in you or your team.

Just be aware of how to code CSS efficiently and find a balance of efficiency and practicality.

Inline or external JS and CSS

From a speed perspective, it is better to use external CSS and javascript rather than inline, or on-page, CSS and javascript. The primary advantage to using external files is that those will normally be cached by the browser and can be used for subsequent page loads. The disadvantage is you increase HTTP requests when you use external files.

Inline script and CSS must be loaded each time with the HTML. It will never be cached. But if you must using inline script and styles to deliver a better user experience, then use it.

Validate and minify

Minimizing your CSS (and JS and HTML) files removes an extraneous characters including comments and whitespace. This reduces the file size, and thereby reduces load time.

Many frameworks, like [Twitter Bootstrap](#), provide both normal (human-readable) and minimized CSS and script files.

Validate your stylesheets with W3C CSS Validator <http://jigsaw.w3.org/css-validator/>.

Tools for minimizing CSS:

1. YUI <http://refresh-sf.com/yui/> (also Javascript)
2. Slimmer <https://pypi.python.org/pypi/slimmer/> (also Javascript)
3. CSS Minifier <http://cssminifier.com>
4. CSS Compressor <http://www.csscompressor.com>
5. Minify CSS <http://www.minifycss.com>

JavaScript techniques

Javascript carefully

As we've mentioned, scripts can block page rendering on either the CSS object model (CSSOM) or the Document object model (DOM).

Avoid changing CSS elements via script like `elem.style.*`.

Avoid script commands like `document.write` that will block DOM construction.

Use asynchronous script whenever possible to allow simultaneous loading.

Minimize Javascript

Ideally, you want to build your page so that **no** javascript is required for the initial render. If some is going to be required, make sure it is as little as possible.

Load script asynchronously

To avoid blocking the page render, load scripts asynchronously. This attribute is new in HTML5.

```
<script src="/js/functions.js" async></script>
```



`async` is supported in Chrome, Safari, Internet Explorer 10+ and Firefox 4.0+.

Each `async` script is executed after it is downloaded and before the window's `load` event. They are **not** necessarily loaded in the order they occur.

Google Analytics has been upgraded to run asynchronously. If you have sites older than a year or more, it's probably a good idea to make sure you are using the `async` script. To check your tracking code, make sure it contains `ga.async=true` in the code.

Defer script loading

To load your javascript only when the page has finished loading, use the `defer` attribute.

```
<script src="/js/functions.js" defer></script>
```

Scripts with the defer attribute **will** be loaded in the sequence they occur on the page. This occurs after parsing is completed and before the DOMContentLoaded event.

The defer attribute is specified in HTML4 and HTML5.



If a script is deferrable, it could also be placed at the bottom of the page.

Avoid loading content with JS

It's not a good idea to load content using JS because some search engines won't see your content, just your script. This is more of a good-practice for SEO and not so much for increasing speed.

For more on this topic see this [step-by-step guide](#) or read [Google's proposed specification](#).

Use GET for AJAX requests

POST is implemented as two-step process. First, it sends the headers, then it sends the data. GET, on the other hand, is normally a single-step process.

Different browsers have different data limits for GET, so if you are sending a lot of data, you may have to use POST instead.

URL lengths of over 2000 characters have been known to [cause problems](#) for Internet Explorer, so if your data transfer is nearly 2k, you probably shouldn't be using GET if you are concerned about users with this browser.

Validate and minify

Validate your Javascript with tools like [JSHint](#) or [JSLint](#).

Minimizing your Javascript with:

1. Javascript Compression Tool <http://jscompress.com>
2. Minify Javascript <http://www.minifyjavascript.com>
3. Ajax Minifier <http://aspnet.codeplex.com/releases/view/40584>
4. UglifyJS <https://github.com/mishoo/UglifyJS>

DOM techniques

Besides streamlining your styles and scripts, there are several steps you can take on your pages themselves to maximize the user experience.

Avoid Flash

Flash is a speed-killer. Avoid it whenever possible.

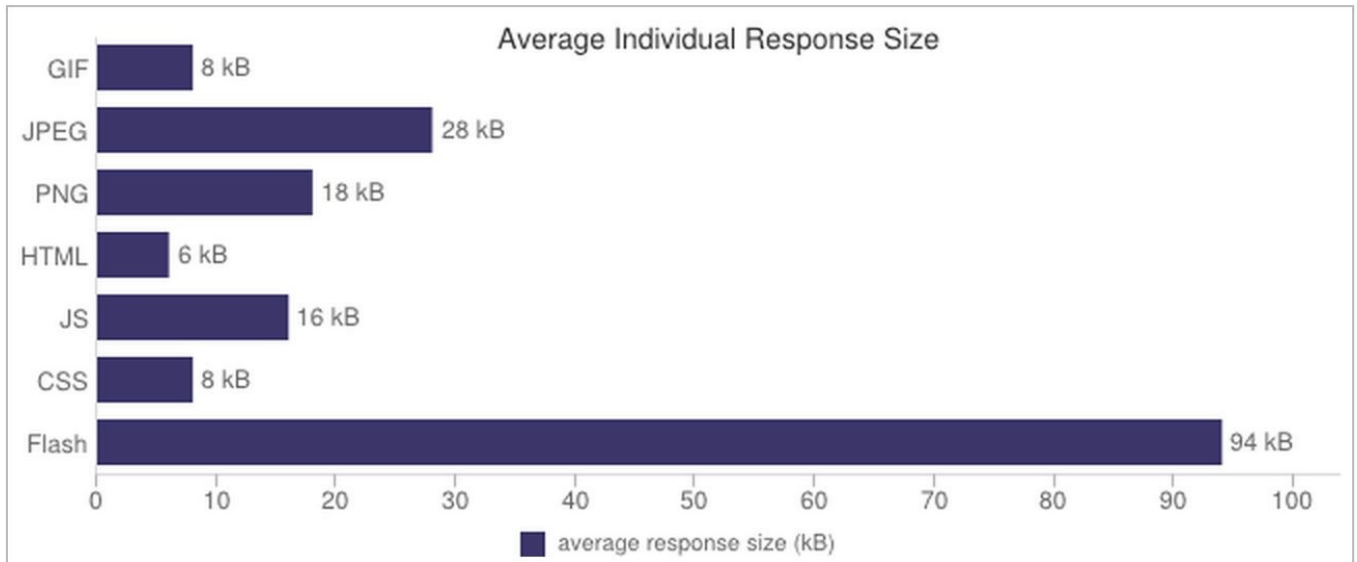


Image source: <http://httparchive.org>

Page size

Ideally, we want **all** of our above-the-fold content, including the necessary CSS to be less than 14 kB in size. This is the maximum amount of data that can be transferred in a single HTTP request (assuming there are no redirects). This could be as large as ~40 kB if using compression like gzip. We'll cover gzip in more detail later.

Pages over this size require more send-receive cycles to and from the server.

Remove unnecessary DOM elements

Each element of the DOM has some cost associated with loading it.

```
1  <body>
2      <div>
3          <div>
4              <p>This content is incredible</p>
5          </div>
6      </div>
7  </body>
```

versus

```
1  <body>
2      <div>
3          <p>This content is incredible</p>
4      </div>
5  </body>
```

Use preloading

Preload components based on where you believe the user is headed next.

For example, if the user begins typing in a form, you could begin loading the script that will be necessary on the next page, assuming the user hit the submit button on the form.

Or you can automatically pre-load scripts or other objects after the current page has loaded to take advantage of idle browser time.

```
<link href="//url.com/images/file.png" rel="prefetch" />
```

Prefetching with javascript

```
1 (function($) {  
2     $.ajax({ url: "/js/script_A.js", cache: true,  
3     dataType: "text" });  
4     $.ajax({ url: "/js/script_B.js", cache: true,  
5     dataType: "text" });  
6 })(jQuery);
```

Reduce DNS Lookups

The Domain Name System (DNS) matches the IP address of your server to your URL.

Assuming the client's DNS cache is empty, the browser must make a DNS call for all the hostnames for the page's URL, script files, external stylesheets, images, jQuery objects, etc. Putting all of these assets at the same URL reduces the number of DNS lookups.

For example, it would be better to store your product assets in `www.example.com/products` than to place them in `products.example.com`. This would reduce the number of DNS calls required. Similarly, you would want to run CSS, javascript, and other assets from the same domain as much as is practical.

Share the heavy load

If your site is hosting a lot of images, or video, or other large assets, consider hosting these on dedicated servers designed for such content. Put images on Flickr, video on YouTube or Vimeo, and let their servers share the load.

By doing so, when the user requests a page, the load is distributed across multiple servers. Your server is providing the content (HTML, CSS, etc.) while the other servers are responding with the image or video or whatever you have linked.

The advantage of this approach is that you get to take advantage of the expertise and scale of these other specialized services. The disadvantage is that in the event of a problem, it is out of your hands and you have to rely on a third party to correct the issue.

Validate and minify

Make sure your html is current and doesn't contain any deprecated tags such as `` or `<center>`. Removing deprecated html is another step in optimizing everything. The **W3C Markup Validation Service** makes this easy.

W3C Markup Validation Service <http://validator.w3.org/>

Using web fonts

Web fonts can be significant in size. One of the web most popular web fonts, Google's Open Sans is 217K and supports over 20 languages. It contains Latin, Cyrillic, and Vietnamese character sets. Limiting it to only Latin drops the size down to 36K, a more than 80% improvement. <http://www.igvita.com/2012/09/12/web-fonts-performance-making-pretty-fast/#optimizing>

A few tips on using web fonts:

Limit the font to a specific subset with `subset=latin`

```
1 <link href="http://fonts.google.com/css/?
2 family=Gafata&subset=latin" rel="stylesheet"
3 " />
```

Only load the font styles you actually use

If you're not using Bold Italic on your page, don't bother loading it.

Consider loading only the specific characters you need

If you only need certain letters, such as for a headline, you can only load the font for those letters. At the time of publication, this is a beta feature.

```
1 <link href="http://fonts.google.com/css/?
2 family=Gafata&subset=latin&text=MyHeadline"
3 rel="stylesheet" />
```

Server techniques

Enable text compression

Text compression can significantly reduce the size of the HTTP response.

For most sites, the transfer is largely images and text (html, css, and scripts). The images are already compressed, so let's look at how to compress the text as well.

Browsers indicate their support for compression in the HTTP request. Two common formats for compression are gzip and deflate.

gzip is much more prevalent than deflate.

gzip is supported by PHP, Apache, and Google App Engine. In Apache 2.0+, the module `mod_deflate` enables gzip. In earlier versions back to Apache 1.3, it is in the module `mod_gzip`.

Accept-Encoding: gzip, deflate

On most servers, the *compression level* of gzip is set to level 6. Level 1 offers the fastest speed but the lowest compression ratio. Level 9 provides the slowest compression speed and the highest compression ratio. This is configured using the *DeflateCompressionLevel* directive.

Use mod_pagespeed (apache)

The good folks behind Google's PageSpeed tool have developed a module named `mod_pagespeed` to automate much of the techniques we've discussed.

<https://developers.google.com/speed/pagespeed/module>

mod_pagespeed is an open-source Apache module that automatically optimizes web pages, including CSS, images, and Javascript. It is currently free, but Google notes that they [may charge](#) for it at some point in the future. If, or when the pricing changes, you will have 30 days notice, according to Google.

Dreamhost was one of the first hosting companies to offer mod_pagespeed. I use Dreamhost for some of my sites, but I have not used mod_pagespeed.

See how easy it is to enable mod_pagespeed on Dreamhost. [{Video}](#)

mod_pagespeed will take care of several speed improvements automatically. If you have multiple CSS files, the module will combine them into one file and minimize them. It will also minimize your script files as well.

These can be customized through the settings for mod_pagespeed.

For more on what mod_pagespeed can do, go to https://developers.google.com/speed/pagespeed/module/config_filters.

Use output buffering (PHP)

PHP has a useful technique. You can turn on output buffering, which will send all html to memory instead of to the browser.

Normally (without buffering), the html is transferred to the browser piecemeal. By enabling output buffering, it is sent to the browser all at once, which decreases loading time.

```
<html> <head><! – css, js —></head> <body> <p>This is amazing content!</p> </body>
</html>
```

You can also compress the buffer by using the command `ob_start('ob_gzhandler');` to reduce HTML size even further.

Using output buffering also makes it simple to make dynamic changes to the web content, like adding a user's name, or changing a login/logout button depending upon the user's status.

Flush it out (PHP)

If any of the pages on your site require a large amount of back-end processing, use `flush()` to send at least a portion of the data to the browser. This allows the browser to begin fetching page components while the back-end continues processing.

A good technique is to do this immediately after the `</head>`.

```
1      <!-- css, js -->
2  </head>
3      <?php flush(); ?>
4  <body>
```

Use a CDN

If your site receives a large amount of traffic, consider using a Content Delivery Network (CDN).

This is where service providers have a distributed network of servers around the world so they can push your content to your users from a server near the user's part of the globe. CloudFlare is one such provider with a large set of features.

Amazon Cloudfront is a similar service. <http://aws.amazon.com/cloudfront/>

Other CDN service providers:

Akamai <http://www.akamai.com>

CloudFlare <http://www.cloudflare.com/features-cdn>

EdgeCast <http://www.edgecast.com>

LiquidWeb <http://www.liquidweb.com/services/cdn.html>

MaxCDN <http://www.maxcdn.com> <!-- Get AFFILIATE LINKS

<http://www.liquidweb.com/cn/c/refer/index.html>

<http://www.maxcdn.com/company/affiliates/>

—>

Wordpress

Nothing in this world is free. Using Wordpress makes a lot of things simple, but it also makes a lot of sites slow. Here's a few tips on making your Wordpress site a bit faster.

Cache

Just like a normal site, cache your content to speed loading for returning users. There are several great caching plug-ins that will do this for you.

WP-Cache <http://wordpress.org/extend/plugins/wp-cache/>

W3 Total Cache <https://wordpress.org/plugins/w3-total-cache/>

WP Super Cache <https://wordpress.org/plugins/wp-super-cache/>

Quick Cache <https://wordpress.org/plugins/quick-cache/>

Varnish

Varnish is a server-side cache that stores a copy of your pre-built web pages and rapidly serves them up to users. This enables the pages to load without calling on Wordpress, PHP, and MySQL to build the page for each user request.

For more go to <http://www.varnish-cache.org>.

If you're using Amazon Web Services, Jeff Reifman offers an excellent installation guide at <http://jeffreifman.com/detailed-wordpress-guide-for-aws/install-varnish/>.

Optimize images automatically

If you use a lot of images, compressing them all manually can be extremely time consuming. Automate this process with the plugin **WP-Smushit**. It will automatically optimize your images as you upload them.

WP-Smushit <http://wordpress.org/extend/plugins/wp-smushit/>

WP-Smushit has an upper limit of 1 MB. It will not compress files larger than 1 MB.

Keep the database tidy

There are a few simple things you can do to keep the database size to a reasonable level.

Draft Control - Each time you save a draft, Wordpress stores a copy of that revision in the database. If you save it twenty times, you probably don't need all twenty revisions - normally the last 2 or 3 is enough. The plugin **Revision-Control** allows you to control the number of revisions that get stored in your database.

Revision-Control - <http://wordpress.org/extend/plugins/revision-control/>

Database Optimization - it's good practice to periodically repair your database. This allows it to cleanup any outdated data elements and keep size to a minimum. You can do this through the plugin **WP-Optimize**. An additional plugin, **WP-DB-Manager** allows you to automatically schedule when your database will be optimized.

WP-Optimize - <http://wordpress.org/extend/plugins/wp-optimize/installation/>

WP-DB-Manager - <http://wordpress.org/extend/plugins/wp-dbmanager/>

LazyLoad images

Previously, we learned about lazy-loading images, where the images only load when they are in-view or are about to be in-view.

The plugin **jQuery Image Lazy Load** will do this for you. It will only load images that are above-the-fold and will load other images as the user scrolls.

jQuery Image Lazy Load - <http://wordpress.org/extend/plugins/jquery-image-lazy-loading/>

Remove gravatar images

Gravatar images appear next to each comment that a user makes. If the user isn't logged in, a default image is displayed.

In *Settings* -> *Discussion*, set the default Gravatar image to nothing.

Personally, I prefer to remove them everywhere, even if the user is logged in.

Eliminate bandwidth leaks

If you have a lot of images on your site, it is possible that others may be hotlinking to them. If I want to display *your* image on *my* site all I have to do is link to it.

```
<img src='http://yoursite.com/your-awesome-picture.png'>
```

This forces *your* server to bring up your image to display it on my site. How nice of you to provide the bandwidth for me.

Keep any such leaching attempts by adding this to your root `.htaccess` file.

```
1 RewriteEngine on
2 RewriteCond %{HTTP_REFERER} !^$
3 RewriteCond %{HTTP_REFERER} !^http(s)?://(w\
4 ww\.)?yourURL.com [NC]
5 RewriteCond %{HTTP_REFERER} !^http(s)?://(w\
6 ww\.)?google.com [NC]
7 RewriteCond %{HTTP_REFERER} !^http(s)?://(w\
8 ww\.)?feeds2.feedburner.com/yourRSSfeed [NC]
9 ]
10 RewriteRule \.(jpg|jpeg|png|gif)$ - [NC,F,L]
```

This will still allow your site, Google, and your RSS feed (assuming FeedBurner above), to continue to serve images normally, but will prevent any other bandwidth links.

Deactivate unused plugins

Plug-ins can be great, but if you don't absolutely need them, then deactivate **and delete them**. They are unnecessary overhead.

Yes, I have just mentioned a half-dozen or so plugins and then just told you plugins are bad. There is no free lunch with plugins. If you find them helpful, use them. Just remember that there is at least some usage cost for each plugin.

Optimize the homepage

There are a few small simple steps you can take to make the home page a bit faster.

1. Reduce the number of posts that display on the home page.
2. Always show excerpts instead of the entire post.
3. Remove any unnecessary widgets.
4. Remove any *sharing* widgets from the home page. (only include them at the bottom of the post itself).

Remove unnecessary PHP functions

There are many calls to PHP functions that can be easily removed or replaced. Every server call we can remove is a small bit of speed we can add to our page loading. We'll go through several replaceable tags here.

These techniques probably offer the least amount of improvement at moderate to high level of effort. So these would only be useful after everything else is already fully optimized.

Important Safety Tips

1. Always backup your site before making any code changes.
2. Before making any changes to your theme's source files, create a copy of your theme (a Child theme). You can use a plugin to simplify this process.
3. Make sure to make a note of where you made changes and what changes you made so you can reverse them if something goes awry.

Removing your theme and adding it back to Wordpress will reload the original theme files (and remove any edits you made previously).

header . php

In the header.php file, these can be removed and replaced by fast-loading plain text:

language_attributes();

header . php has this: This

```
1 <html class="ie ie7" <?php language_attribu\
2 tes(); ?>>
```

But looking at the source code for my site shows this -

```
1 <html class="ie ie7" lang="en-US" prefix="o\
2 g: http://ogp.me/ns#">
```

So changing that line in header . php to match the second example removes that function

call with faster-loading plain text.

This call is made several times before the <head> tag.

bloginfo('charset');

Replace <meta charset="<?php bloginfo('charset'); ?>" />

with <meta charset="UTF-8" />

bloginfo('pingback_url');

Replace

<link rel="pingback" href="<?php bloginfo('pingback_url'); ?>" />

with

<link rel="pingback" href="http:// **your URL** /xmlrpc.php" />

get_template_directory_uri()

<script src="<?php echo get_template_directory_uri(); ?>/js/html5.js"
type="text/javascript"></script>

becomes

<script src="http:// **your URL** /wp-content/themes/ **some variation of your theme name** /js/html5.js" type="text/javascript">

body_class('')

<body <?php body_class(); ?>>

becomes

<body class="home blog custom-background custom-font-enabled">

(or something similar, depending on your specific CSS).

sidebar.php

bloginfo('rss2_url');

This may also be in footer.php.

bloginfo('comments_rss2_url');

This may also be in footer.php.

footer.php

<!-- queries. seconds. -->

This is an html comment that is only visible if you look at the source code. Unless you view this often, it is superfluous.

widgetized sidebar

The widgetized sidebar is only used if widgets are *not* enabled. So if you have widgets enabled, you are safe to delete these two lines and everything in between:

```
<?php /* Widgetized sidebar, if you have the plugin installed. */ if (
!function_exists('dynamic_sidebar') || !dynamic_sidebar() ) : ?>
```

and

```
<?php endif; ?>.
```

Wordpress optimization

For more on Wordpress optimization, use the GTMetrix tool to analyze your blog.

<http://gtmetrix.com/wordpress-optimization-guide.html>

A Wordpress case study

Moving my personal blog (wordpress)

I recently moved my personal blog to a new host. I've added a few new posts, but haven't looked at it from a speed perspective, so let's do that.

First look

| Document Complete | | | Fully Loaded | | |
|-------------------|----------|----------|--------------|----------|----------|
| Time | Requests | Bytes In | Time | Requests | Bytes In |
| 7.585s | 34 | 4,000 KB | 7.645s | 35 | 4,000 KB |

| Load Time | First Byte | Start Render | Visually Complete | <u>Speed Index</u> | DOM Elements |
|-----------|------------|--------------|-------------------|--------------------|--------------|
| 7.585s | 0.207s | 1.784s | 7.400s | 3848 | 652 |

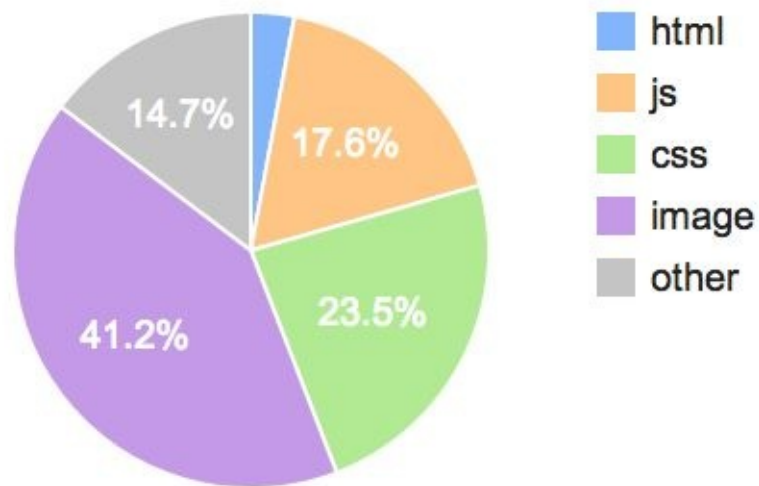
So we see from the first load, that this is extremely slow. It took 1.9 seconds for the first screen paint to occur. The document was complete after 7.58 seconds and fully loaded at 7.645 seconds.



Remember that anything over one second is a noticeable delay.

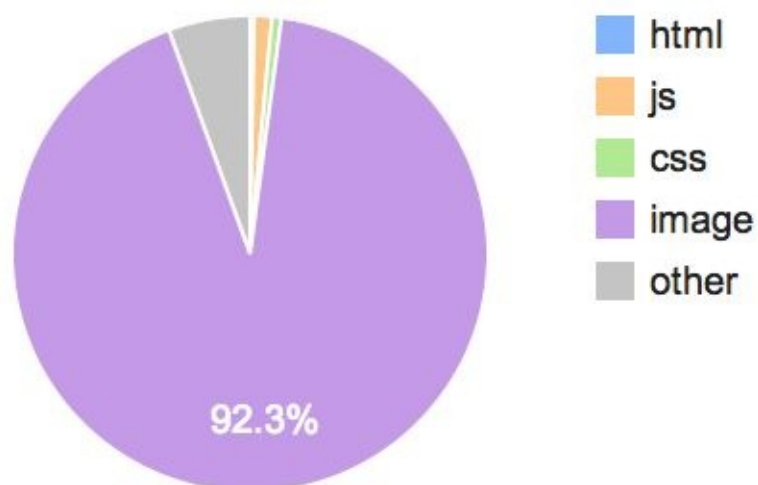
The test also shows us how many server requests we are using and how those are broken down by html, css, scripts, and images.

Requests



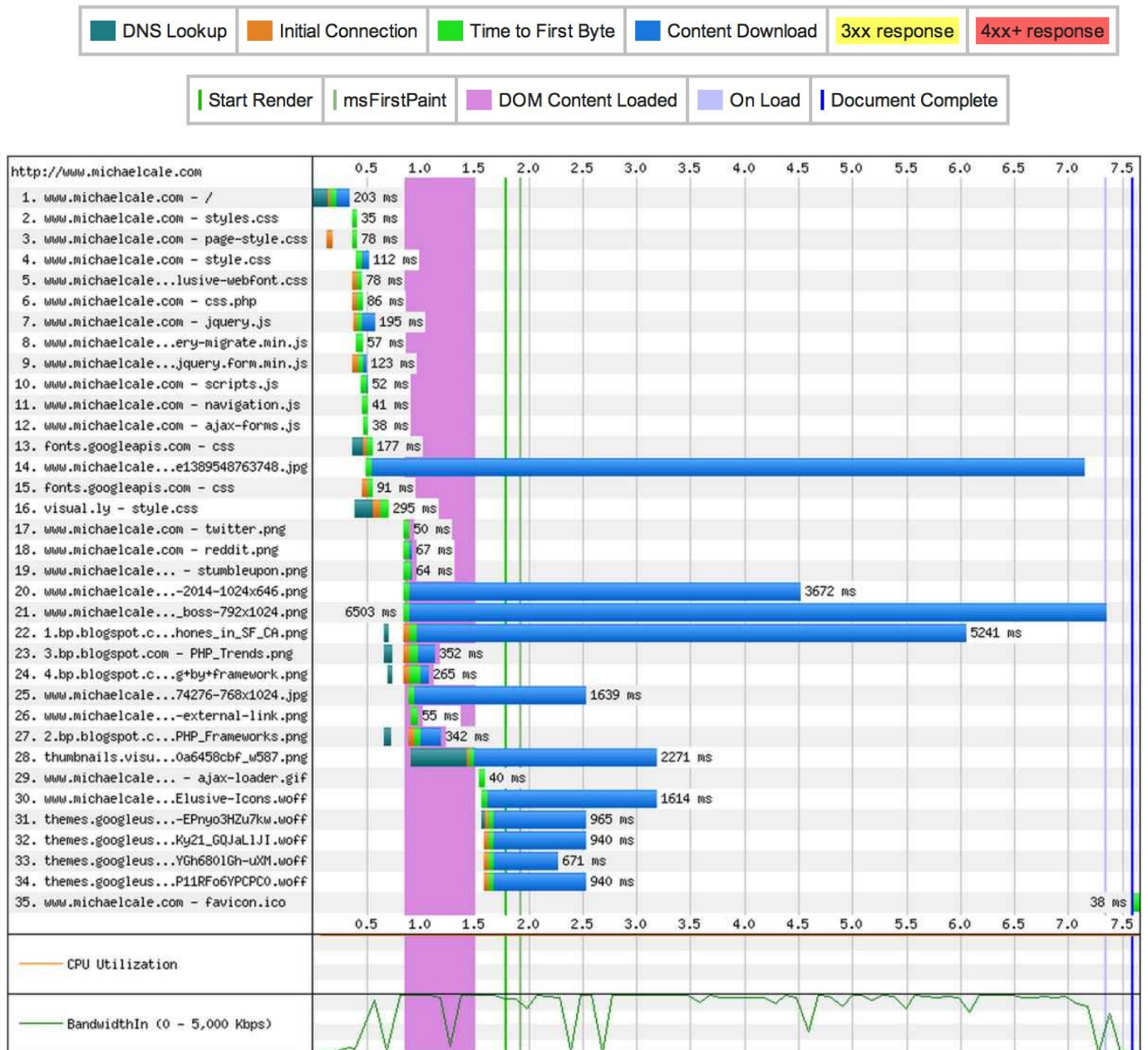
It also shows us a breakdown of how much data we are serving to the client. So right off, it looks like we have to work on these images.

Bytes



Let's take a look at the waterfall and see what's happening.

Waterfall View



Those long blue bars that stretch across the page are all images on the site that are taking a very long time to load. Some of them 6 to 7 seconds. We'll start there.

Step 1 - image size

The problem is with the image file in the first post (row 14 on the waterfall). The file is gigantic. It is a **JPG** that is 1.2 MB.

But it's not just the *file size*, it's the *image size*. It is a whopping 3k by 2k pixels. Reducing the size to 1500 x 1125 pixels brings the file size down to 395Kb. Incidentally, running the original image through Smush.it reduces the file size by an additional 5.2%.

Another option I could've done was to put the image in the post itself, so that it didn't show on the homepage, but I like having an image above the fold.

So for all the other images on the home page, I reduced their size and then compressed all images.

Step 2 - homepage streamlining

There is still a large image that is hitting the home page (row 21 in the waterfall). I don't want to remove it, but even compressed, it is over 1MB. So I am going to move it to the blog post, so users will only see it if they click the headline and go to [the post](#). That will keep it off the home page.

Now that we have the images cleaned up, let's keep reducing the size of the homepage. We currently have Wordpress configured to show 10 blog posts on the first page. Let's reduce that to five (under *Settings* -> *Reading*).

Blog pages show at most

5



posts

Cache is working

The test also loads your site a second time. I am using the plugin [W3 Total Cache](#) (the free version), so caching is active.

| | Load Time | First Byte | Start Render | <u>Speed Index</u> | DOM Elements | Document Complete | | | Fully Loaded | | |
|-------------|-----------|------------|--------------|--------------------|--------------|-------------------|----------|----------|--------------|----------|----------|
| | | | | | | Time | Requests | Bytes In | Time | Requests | Bytes In |
| First View | 7.585s | 0.207s | 1.784s | 3848 | 652 | 7.585s | 34 | 4,000 KB | 7.645s | 35 | 4,000 KB |
| Repeat View | 1.276s | 0.000s | 0.849s | 936 | 652 | 1.276s | 0 | 0 KB | 1.276s | 0 | 0 KB |

So on the second load, with caching enabled, the fully loaded site only took 1.276 seconds. So repeat visitors would be spared that horrible load time (assuming they bothered to come back).

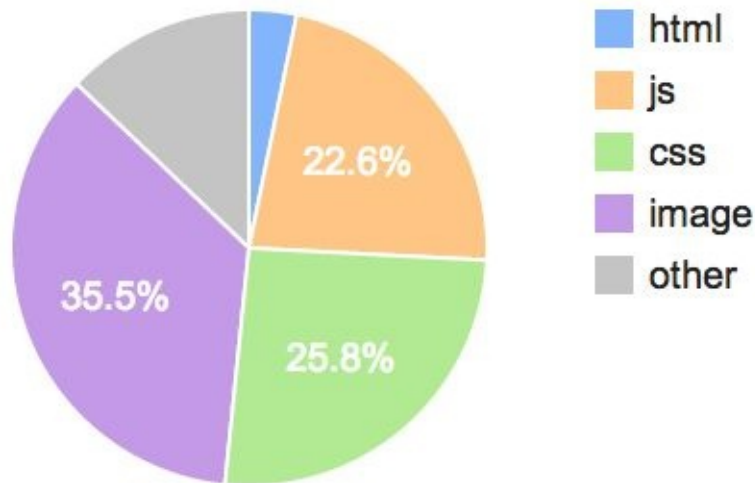
Now let's update our performance again.

Second Look

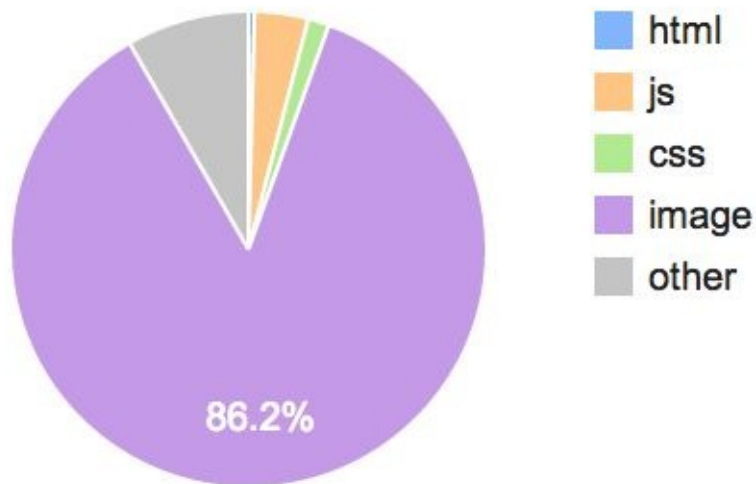
| Document Complete | | | Fully Loaded | | |
|-------------------|----------|----------|--------------|----------|----------|
| Time | Requests | Bytes In | Time | Requests | Bytes In |
| 3.596s | 31 | 1,715 KB | 3.751s | 32 | 1,715 KB |

| Load Time | First Byte | Start Render | Visually Complete |
|-----------|------------|--------------|-------------------|
| 3.596s | 0.126s | 1.382s | 2.800s |

Requests



Bytes



Our image work has some seriously good results. Our page load time went from almost 7.6 seconds down to 3.6 seconds. So we've cut load time by more than half.

Our *start render* time is much improved as well. It dropped from 1.78 seconds to 1.38 seconds. So getting closer to that 1-second perception threshold.

The number of requests for images is better, but the number of bytes retrieved is still dominated by images. That will probably always be the case and that's okay. But we should be able to do a bit more to speed things up.

Step 3 - General cleanup

Now that we've done the heavy lifting, let's do a bit more cleanup to get the site as fast as possible. So for this next test, here are a few more steps.

Remove share buttons from homepage.

Delete unused plugins (eight total).

Update header .php to remove the following and replace with plain text -

```
1 language_attributes()  
2 bloginfo('charset')  
3 bloginfo('pingback_url')  
4 get_template_directory_uri()
```

Install and activate WP Smush.it plugin (further image compression).

Third Look

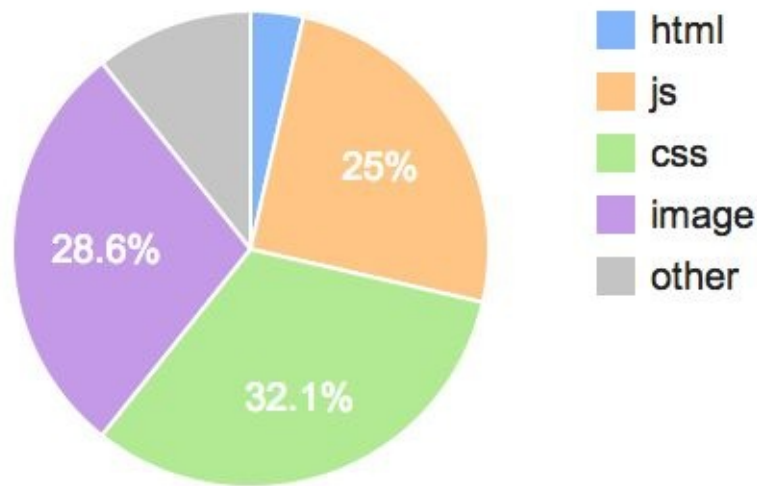
One more speed test shows further improvement. As far as the user experience, the start render now begins in under 0.7 seconds, compared with just under 1.4, so we've cut that by just about 50%.

The entire page was visually complete in 2.6 seconds compared with 2.8 seconds on the second round. So that's an improvement of 7%.

| Document Complete | | | Fully Loaded | | |
|-------------------|----------|----------|--------------|----------|----------|
| Time | Requests | Bytes In | Time | Requests | Bytes In |
| 3.152s | 28 | 1,667 KB | 3.204s | 29 | 1,667 KB |

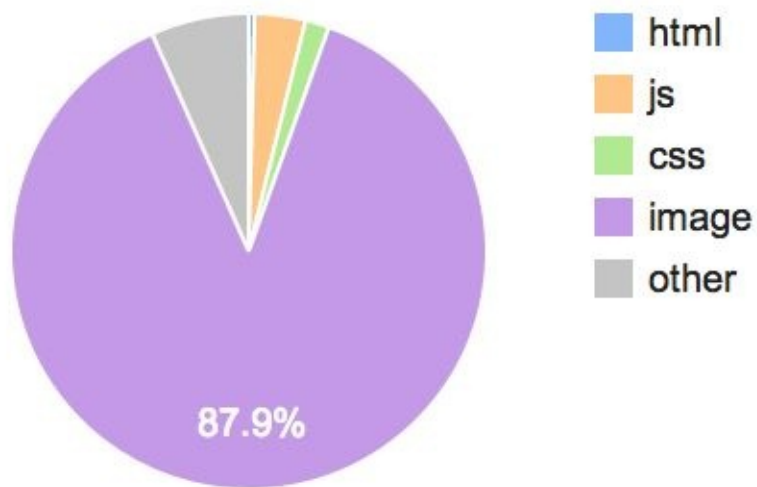
| Load Time | First Byte | Start Render | Visually Complete |
|-----------|------------|--------------|-------------------|
| 3.152s | 0.104s | 0.689s | 2.600s |

Requests

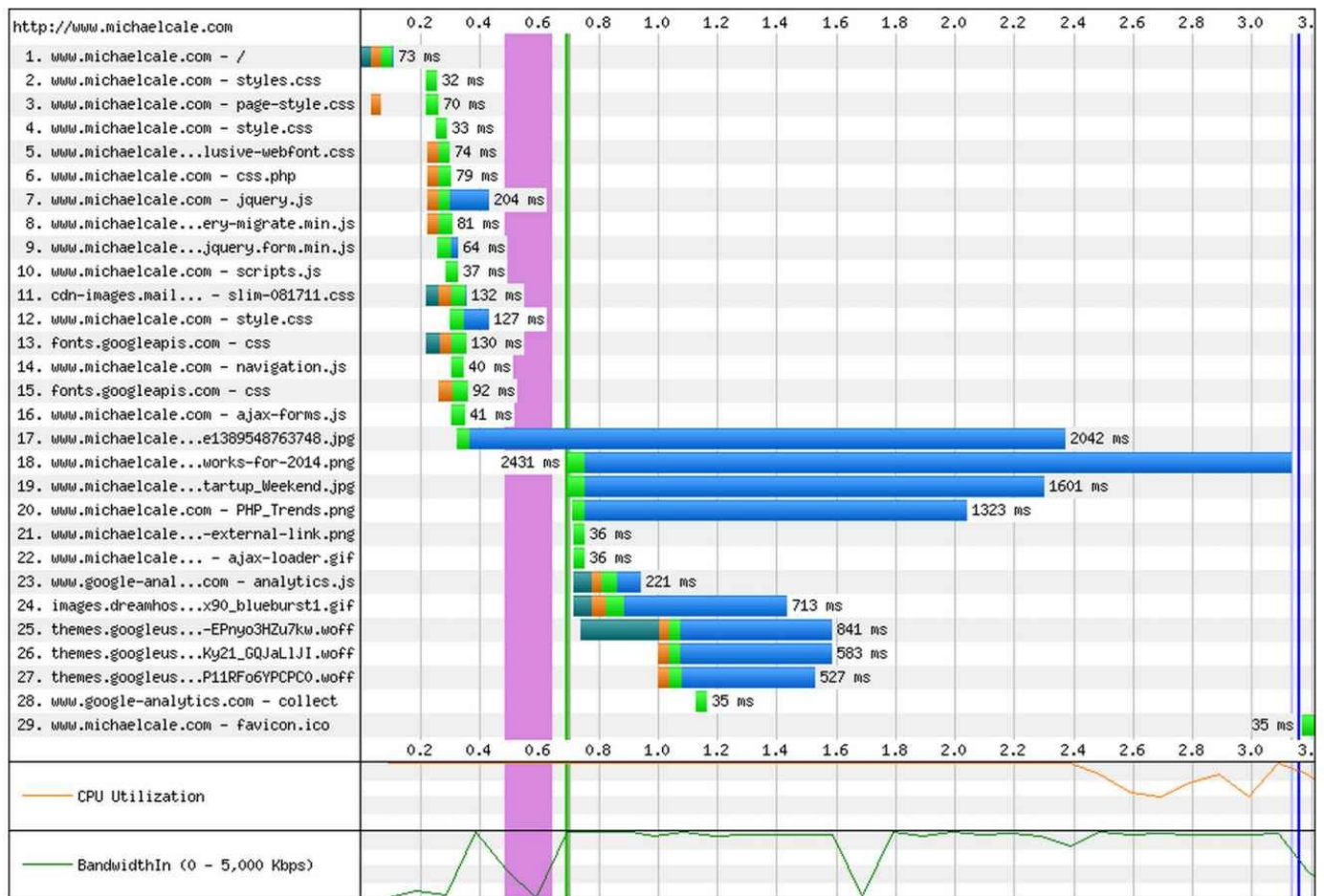


The number of server image requests are down a bit more, from 36% to 32% of the total. The total number of server requests are down to 29, from 35 when we first started.

Bytes



The images are still the major data component, up slightly from 86% of all data to 88%. We've done about all we can do, without removing them completely. The site is image-heavy, but we like images, so we'll live with it.



On this run, we've reduced the number of server requests a bit more, and cut out a few more bytes to load. Total load time dropped from 3.596 to 3.152 seconds (another 12% improvement).

Bug we had a big improvement in *start render* time. We went from 1.382 to 0.689 seconds. This puts us comfortably under our one-second perception limit.

Conclusion

We won't win any web-page-races with this site, but it's no longer horrible. We have the initial render beginning in well below our user-perception target of 1 second.

The total load time is about 3 seconds for everything including our heavy images.

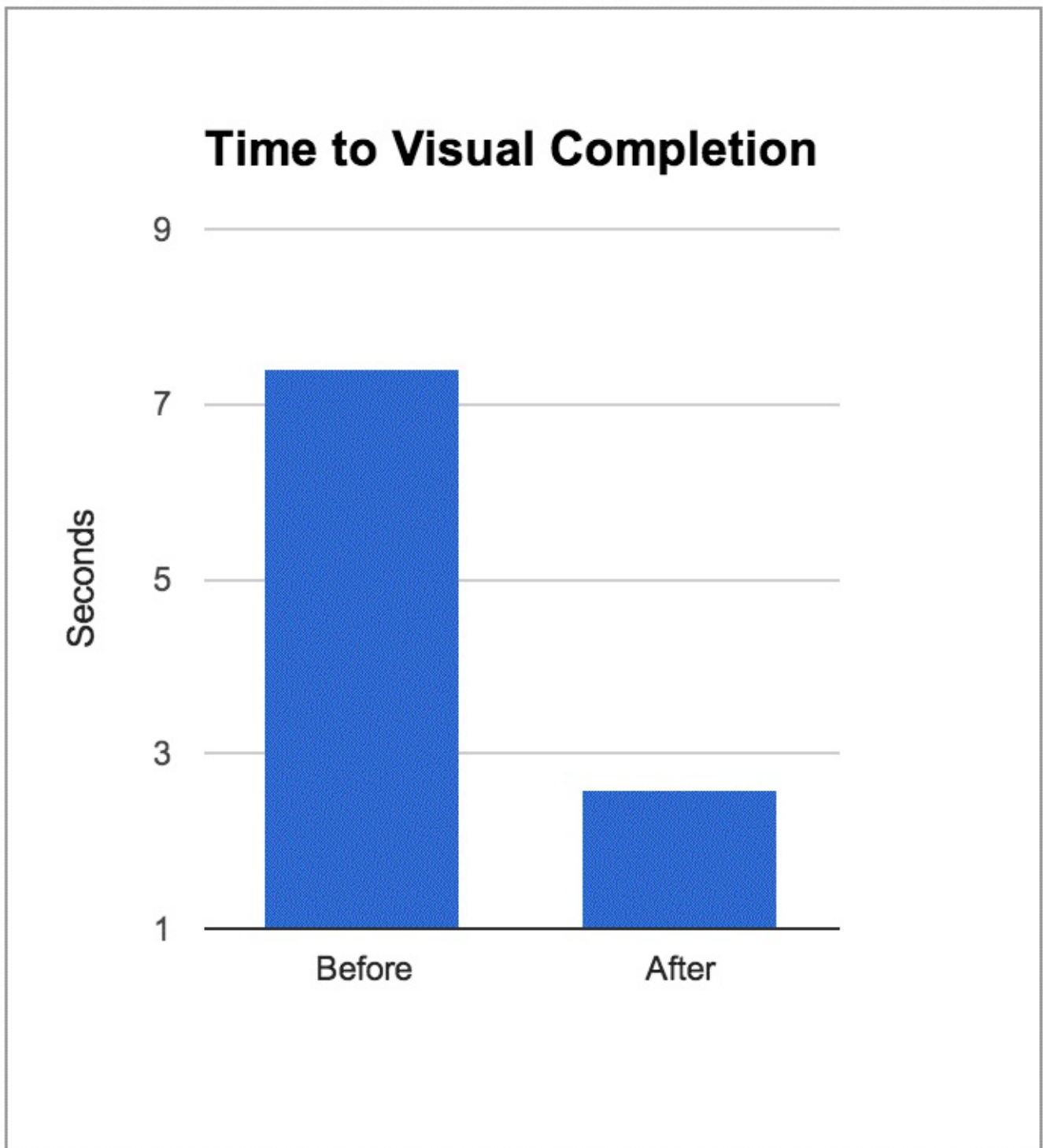
The image compression did work quite well. In our first test, the images were taking around 4-6 seconds or more to load. Now they are down to all less than 3 seconds. Using lossy compression would be another technique we could employ to allow some more speed.

Over time, as more posts are added, we can put those images in the posts themselves so they won't hit the home page.

Think back to the **aggregation of marginal gains** approach.

We didn't have the ability to build our page architecture with above-the-fold rendering optimized. We could have to some extent, but since it's a Wordpress site, there are some limitations.

Yet even so, we were able to use several simple techniques to dramatically speed up our user experience.



Further improvement

Further steps we could do on a Wordpress site to continue to improve speed would be to minimize the number of plugins.

Simplify your CSS

Remember the basics

Remember the basic rules of CSS - IDs are unique and classes are not.

Each element can only have one ID and it must be unique. If you use an ID on more than one element, your CSS will fail validation.

Any styling that needs to be applied to more than one object on a page should be done with a class.

To keep it straight, think of a physical product, your phone. Your phone has a bar code (or it had one on the packaging). Every phone of the same type has the exact same bar code. That's your CSS class.

Every phone also has a serial number on it somewhere (probably inside the case). Every serial number is different and unique to your phone. That's your CSS ID. Take two iPhones of the same model. Same bar code, different serial numbers. Same class, different IDs.



Remember that elements can have both an ID and a class.

If you don't have unique IDs the JS function `getElementbyID` will not work properly

IDs also have a special navigation function. If a user goes to

`http://www.yourwebURL.com#somecssid`

then the browser will take them directly to that CSS element, similar to an anchor link.

General layout

A valuable best practice for your CSS is to use a thoughtful layout.

```
1 /* --- General Styles --- */
2
3 html{}
4 body{}
5 h1{}
6 h2{}
7 h3{}
8 p{}
9 a{}
10
11 /* --- Header Styles --- */
12
13 #header{}
14
```

```
15 /* --- Nav Styles --- */
16
17 #nav{}
18
19 /* --- Other Special Styles --- */
20
21 #hero-unit{}
22
23 /* --- Footer Styles --- */
24
25 #footer{}
```

User anchor elements

Another best practice is to use the anchor elements (<h1> <p> <a>, etc.) whenever possible instead of creating a new class. For example if you have a element ID of #sidebar, you could further define #sidebar h1{} and #sidebar p{} instead of creating #sidebarheader{} and #sidebartext{}.

Group similar CSS

Group similar CSS

```
1 /* instead of this */
2 p {font-family: Verdana, sans-serif;
3     font-size: 14px;
4     color: #FFFFFF;}
5 span {font-family: Verdana, sans-serif;
6     font-size: 14px;
7     color: #FFFFFF;}
8
9 /* do this */
10 span, p {font-family: Verdana, sans-serif;
11     font-size: 14px;
12     color: #FFFFFF;}
13
14
15
16 /* instead of this */
17 #container{margin-top: 20px;
18     margin-right: 10px;
19     margin-left: 10px;
20     margin-bottom: 20px;}
21
22 /* do this */
23 #container{margin: 20px 10px 10px 20px;}
```

Use hex colors

Use hex color codes instead of named colors to give your CSS a slight speed boost.

Browser-specific styles

Beginning with CSS3, each browser has its own specification.

| Browser | CSS prefix |
|----------------------------|------------|
| Chrome, Safari, iOS, Opera | -webkit- |
| Firefox | -moz- |
| Internet Explorer | -msie- |

until version 15.0, Opera used the prefix -o-

These exist because the browsers use different rendering engines. If you use CSS for some features using -moz-, the other browsers will ignore it and you will need to find a way to use similar styling in the other browsers.

For more on cross-browser tips, check out [Simple Yet Important Cross-Browser Styling Tips Everyone Should Know](#) by Sam Norton.

Validate your CSS

Validating your CSS using a free tool can tip you off to improperly structured CSS.

[Validate your CSS](#)



The W3C CSS Validation Service

W3C CSS Validator results for TextArea (CSS level 3)

[Jump to](#)

W3C CSS Validator results for TextArea (CSS level 3)

Congratulations! No Error Found.

This document validates as [CSS level 3](#) !

Valid CSS



The W3C CSS Validation Service

W3C CSS Validator results for TextArea (CSS level 3)

[Jump](#)

W3C CSS Validator results for TextArea (CSS level 3)

Sorry! We found the following errors (1)

URI : TextArea

1

Invalid ID selector [#3para]

Invalid CSS

The example above is invalid for having an ID that begins with a number.

Compress your CSS

After validating, compress and minify your CSS using the free tools such as [Minify CSS](#) or [CSS Compressor](#). This will remove any unnecessary characters and extraneous whitespace.



Make sure to validate your CSS before you minify it

General PHP tips

Use language constructs over functions

Language constructs don't have the overhead that functions contain, and therefore, operate much faster.

Functions require specific PHP extensions be compiled. A call to `phpinfo()` or `get_loaded_extensions()` will show you all the extensions that are loaded into your version of PHP.

You can often tell the difference between the two because many language constructs don't require parentheses. `echo` and `isset` are two examples of commonly-used language constructs. Other examples include `print`, `unset`, `empty`, `include`, and `require`.

Variables

Use `isset` instead of `strlen`

Since `isset` is not a function, it performs much faster than `strlen` when checking a variable.

```
1 if (!isset($c)) {echo 'C is Speediful';}  
2     // is better than  
3 if (strlen($c)>0) {echo 'C is draggin, but \  
4 has data';}
```

Remember that `isset` evaluates `null` and `false` as **not** set.

Helpful Tip

If your function is dependent on having multiple variables set, you can check them all in the same `isset` statement.

```
1 $code = $for = $speed = 1;  
2 isset ( $code, $for, $speed ); // true
```

Use `===` when evaluating variables that may be false or null. This is faster than using `is_bool` or `is_null`.

Output

`printf()` is for newbs

Unlike `echo` and `print`, `printf` is a function and therefore carries the normal function overhead. `printf` supports several formatting structures that are probably unnecessary for simple output.

// Do not use `printf` if you want to code for speed

For more on the formatting options for `printf` see [PHP Manual sprintf](#)

Use echo instead of print

`print()` uses more overhead than `echo()`. This is because the `print()` function returns a status on whether it was successful or not.

`echo()` does not.

It's very important to use `echo` when the string contains a dollar-sign symbol.

```
1 echo 'code for speed' ;
2      // is faster than
3 print 'code for speed' ;
```

Use multiple parameters with echo()

Multiple parameters can be sent as output with a single `echo` command. Using multiple parameters is faster than concatenating output.

```
1 echo 'code ', 'for ', 'speed';
2      // is faster than
3 echo 'code' . ' ' . 'for' . ' ' . 'speed';
```

Static Text

When you have a substantial amount of static text, it is often best to put it outside of *PHP*.

String Operations

Use dot concatenation (mostly)

As long as there are no variables or dollar-signs in the strings, dot concatenation is about 200% faster than comma. If there are variables or dollar-signs, then using comma concatenation is slightly faster ~20%.

```
1 $c = 'Code ';
2 $f = 'For ';
3 $s = 'Speed';
4
5 echo $c . $f . $s ;
6
7      // is better than
8
9 echo $c , $f , $s ;
```

Avoid Regex (in general)

For most string operations, regular expressions are going to use more overhead than operations that do not contain `Regex`.

For example, `explode` is about 20% faster than `preg_split` when `preg_split` contains a regular expression.

Files

Always use complete path

Using the complete path significantly speeds up performance. This is because the server does not have to resolve the path for you.

```
1 include ( '/var/public_html/assets/function\
2 s.php' );
3         // is faster than
4 include ( '../../../functions.php' )
```

Logic and Iteration

switch or nested if else if

switch is better than multiple if else if statements.

Also, your case statements should be in the order of probability of occurrence. You want your most likely scenario to be at the top and your most remote possibility at the end.

```
1 switch ( $boolean )
2 {
3     case TRUE:
4         // code for speed
5         break;
6     case FALSE:
7         // Login Failed
8         break;
9     default:
10        // this is a problem
11 }
```

Use && and || instead of AND and OR

&& and || can be used interchangeably with AND and OR respectively, but && has precedence over AND and || has precedence over OR, so use the symbols.

Define size of loop before iteration

It is always best to calculate the maximum number of iterations outside the loop. Having to calculate the maximum in each pass dramatically increases processing time.

```
1 $c = count($data);
2 for ( $i = 0; $i < $c; $c++ ) {
3     // this is coding for speed
4 }
5
6 for ( $x = 0; $x < count( $data ); $x++ ) {
7     // this is going to run sloooooow
8 }
```



for is faster than foreach and while

Geeking out on TCP

The modern web is based on two data transmission protocols. The Internet Protocol (IP) allows us to move data through the network, or a series of networks. It contains the destination address and allows for data to move through almost any network to reach the desired destination. IP is the bus driver. It's going to map the route to the destination and deliver. It's not necessarily concerned with who or what is on the bus.

Each IP packet contains the destination address and a checksum. Think of it has a rough headcount of the number of passengers. IP doesn't know that Bobby, Susie, and Raquel are on the bus, just the total checksum. If somewhere on the route the checksum becomes invalid, the packet is terminated and the bus ride is over. Besides this header checksum, there is no error control and no acknowledgements between parties, so IP is not considered a robust communication facility. That is handled by TCP.

Transmission Control Protocol (TCP) provides for reliable data transmission over an unreliable network. It is optimized for accuracy and reliability, not for speed. The protocol ensures that data will arrive in the same order it was sent and that the data received is identical to the data originally transmitted. It's going to make sure that Bobby, Susie, and Raquel arrive at the destination, not just a headcount of three.

Characteristics of a TCP connection

Packet

Packet loss

TCP connections are designed for data packet loss. It recognizes that data will be lost in transmission and it is designed to handle that situation. When a packet is lost, the sender will re-transmit that packet. Both sender and receiver are equipped to put the data packets in the proper sequence.

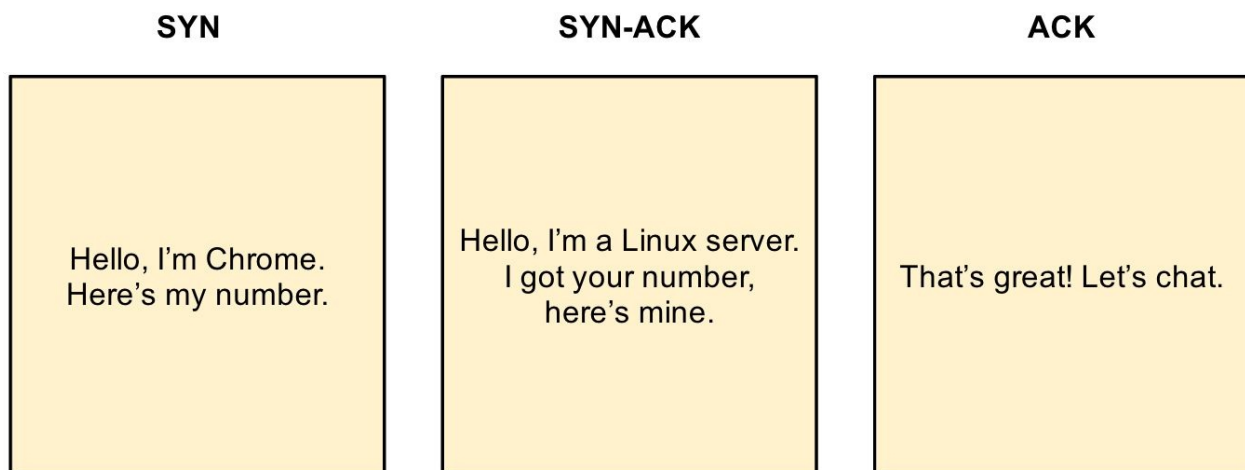
In fact, packet loss is a characteristic that is built in to how TCP functions.

What's your number?

When my son was a toddler, whenever he wanted to know your age, he would ask *What's your number?*. Adults found this entertaining after I translated the question, but most young children seemed to know the meaning of the question instinctively and respond accordingly.

When a TCP connection is established, there is a brief introduction known as a three-way-handshake.

SYN packet.



When a TCP connection is initialized, there is a simple exchange of information about the connection. A simple transmission and acknowledgement from the receiver. No application data transfer occurs here. After this initial connection, the data transfer begins.

Slow Start

The Slow Start algorithm is used as a tactic to gauge the health of the network connection between the sender and receiver. Suppose the sender sends four packets of data to the receiver. Then it will stop transmitting and wait for acknowledgment. Once the receiver acknowledges the first packet, then sender will transmit two more (so there are now 5 packets en route instead of 4).

This process continues until there is packet loss. This allows for maximum data transmission over the existing network because now the sender knows the maximum capacity of the connection. Once that occurs, then the sender begins to reduce the number of packets to a more moderate level that the network can handle.

If a connection becomes idle for a pre-set amount of time, some servers will implement **slow-start restart**. This means the *slow start* process will begin as if it were a new connection and ramp up from a low level. This setting can be disabled on the server to increase performance.

Congestion avoidance

As we saw with *slow start*, TCP uses packet loss as part of its feedback process to maximize reliable data transfer. TCP also uses packet loss as part of congestion avoidance.

When a data packet is lost, TCP uses that as a sign of congestion on the network. This normally occurs when there is a congested router along the network that dropped the data packet. TCP will continually adjust the amount of data being sent as packet losses occur.

Flow control

Flow control prevents the sending from overwhelming the receiver with more data than it can process. This can be dynamic as the server load varies.

Each side of the connection advertises the size of their *receive window*, or the amount of buffer space available for incoming data. If the receiver becomes overwhelmed, it may reduce the receive window to zero until it recovers. In that case, the sender would pause transmission until the receiver becomes ready.

Optimizing TCP

Use compression

Transfer less data

Reduce above-the-fold data to below 1,460 bytes, if possible.

Further Reading

[*A Protocol for Packet Network Intercommunication*](#) by Vinton Cerf and Robert Kahn (1974).

[*DARPA Internet Protocol Specification*](#) (1981)

[*The TCP Maximum Segment Size and Related Topics*](#) Network Working Group (1983)

[*W3C Web Performance Working Group*](#) (2010)

Never stop learning

Web Performance Today

Keep up with current practices and trends with [Web Performance Today](#).

Mobilewebbestpractices.com

[Mobile web best practices](#) by [Brad Frost](#) has a treasure trove of articles and other resources for mobile web strategy, development, design, and more.

For a wonderfully powerful three free slides on the future of the web, make sure to go to <http://bradfrostweb.com/blog/post/this-is-the-web/>.

Mobilexweb

[Breaking the Mobile Web](#) by [Max Firtman](#) is a great blog for staying current on mobile development. Max offers fantastic up-to-date articles on iOS, Android, HTML5, and Google Glass.

Zoompf web performance blog

The [blog](#) at Zoompf offers excellent articles on general web performance.

Mark Isham wrote the excellent [Step-by-Step Guide to Optimizing your Apache site with mod_pagespeed](#) for an Amazon AWS instance.

The Zoompf site also offers generous [Presentations and Whitepapers](#) on web performance topics.

Webmaster Tools

Speed

Google PageSpeed Insights

<https://developers.google.com/speed/pagespeed/insights>

Google's PageSpeed Insights will read your site's URL and recommend specific steps you can take to increase your site's responsiveness.

It measures both for mobile and for desktop.

There are also browser extensions for Chrome and Firefox available at https://developers.google.com/speed/pagespeed/insights_extensions.

Still some work to do

Avoid landing page redirects

Your page has no redirects. Learn more about [avoiding landing page redirects](#).

Minify JavaScript

Your JavaScript content is minified. Learn more about [minifying JavaScript](#).

Prioritize visible content

You have the above-the-fold content properly prioritized. Learn more about [prioritizing visible content](#).

Reduce server response time

Your server responded quickly. Learn more about [server response time optimization](#).

User Experience BETA

Not currently a part of the overall score

❗ Should Fix:

Size content to viewport

► [Show how to fix](#)

Size tap targets appropriately

► [Show how to fix](#)

✅ 3 Passed Rules

► [Hide details](#)

Avoid plugins

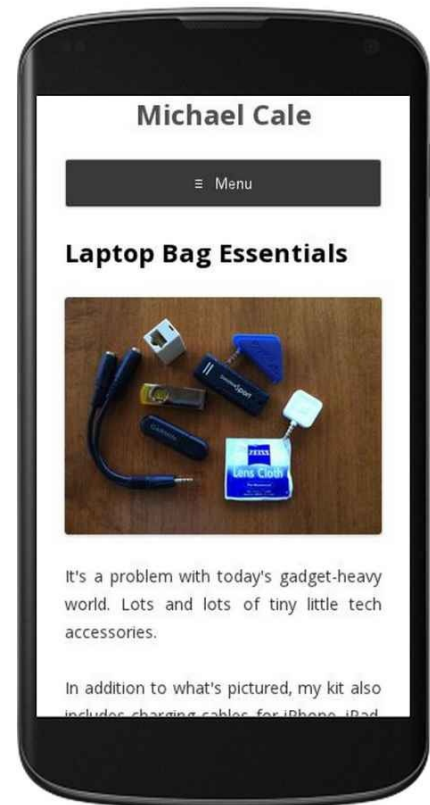
Your page does not appear to use plugins, which would prevent content from being usable on many platforms. Learn more about the importance of [avoiding plugins](#).

Configure the viewport

Your page specifies a viewport matching the device's size, which allows it to render properly on all devices. Learn more about [configuring viewports](#).

Use legible font sizes

The text on your page is legible. Learn more about [using legible font sizes](#).



My personal blog at PageSpeed Insights

Google Speed Tracer

Speed Tracer is an open source Chrome extension from Google that shows the duration of each event as your application loads.

<https://developers.google.com/web-toolkit/speedtracer/>

Google Webmaster Tools

<http://www.google.com/webmasters/tools/>

In Webmaster Tools, select Labs -> Site Performance for an overview of your site's performance.

Webmaster Tools will also point if there are any problems that their bot found while crawling your site.

Quicksprout

Quicksprout was created by Neil Patel, the found of KISSMetrics. Quicksprout offers a wealth of information about your site including page load data. It's quite the wonderful tool.

www.quicksprout.com

YSlow

<http://developer.yahoo.com/yslow/>

Yahoo's speed measuring tool - similar to PageSpeed Insights except requires browser extension.

WebPagetest

<http://www.webpagetest.org>

Similar to PageSpeed Insights, but WebPagetest will allow you to select a location and a browser type for the test.

Pingdom

Pingdom.com offers a tool that will measure your site's speed and provides:

- a performance grade e.g. 82/100
- the number of server requests
- page load time
- page size

SpeedCurve

SpeedCurve.com by Mark Zeman allows you to monitor your own web site performance as well as your competitors. SpeedCurve has an incredible user interface.

Sitespeed.io

Sitespeed.io by [Peter Hedenskog](#) is an open source tool to measure your website performance.

<http://www.sitespeed.io>

Zoompf

[Zoompf](#) offers a free web site scan in return for your email address. They also offer comprehensive web performance consultation. They also have an excellent [Web Performance blog](#).

File Compression

Zopfli compression

Google has developed a compression algorithm called Zopfli that is similar to gzip, but offers improved compression ratios.

<http://googledevelopers.blogspot.com/2013/02/compress-data-more-densely-with-zopfli.html>

Gzip Compression Test

Verify that your gzip compression is working properly.

<http://nontropo.org/tools/gziptest/>

CSS

CSS Lint

Use CSS Lint to validate your CSS to find errors and performance impediments.

<https://github.com/stubbornella/csslint>

CSSess

[CSSess](#) by [Dan DeFelippi](#) is a bookmarklet that identifies unused CSS styles. It works by loading each URL into a hidden iframe and then checks the CSS against the DOM to find inactive styles. It checks both stylesheets and inline CSS.

Identify CSS elements

[wtcss](#) by [Ben Foxall](#) allows you to see how your CSS is interacting with your page.

It offers a unique split screen for the URL you specify with the rendered site on the left side and the CSS elements on the right with a line from the CSS to the DOM element(s).

The screenshot displays the **wtcss** tool interface. On the left, a portion of the W3C website is visible, showing the 'STANDARDS' menu and a list of recent publications. On the right, a list of CSS selectors is shown, each with a line from the website's DOM element(s) it applies to. The selectors include **html**, **body**, **table**, **fieldset**, **img**, **caption**, **li**, **h1**, **h2**, **h3**, **h4**, **h5**, **h6**, **q::before**, **q::after**, **abbr**, **acronym**, **sup**, **sub**, **input**, **textarea**, **select**, **legend**, **pre**, **code**, **kbd**, **samp**, **tt**, **em**, and **strong**. The tool identifies 411 rules and 202 active elements.

“wtcss shows where CSS elements are used on the W3.org site”

Image Compression

Smush.it <http://developer.yahoo.com/yslow/smushit>

PunyPNG <http://www.punypng.com>

ImageOptim <http://www.imageoptim.com>

Optimizilla <http://optimizilla.com>

Base64 Encoding

Freeencoder.com <http://www.freeformatter.com/base64-encoder.html>

Wutils <http://base64.wutils.com/encoding-online/image-to-base64/>

Base64 Image <http://base64image.org>

base64-image.de <http://www.base64-image.de>

Validation

W3C Markup Validation Service <http://validator.w3.org/>

Freeformatter HTML <http://www.freeformatter.com/html-validator.html>

W3C CSS Validator <http://jigsaw.w3.org/css-validator/>.

Minification

Freeformatter CSS <http://www.freeformatter.com/css-minifier.html>

Freeformatter JS <http://www.freeformatter.com/javascript-minifier.html>

YUI <http://refresh-sf.com/yui/>

SVG Optimization

SVG Editor by Peter Collingridge <http://petercollingridge.appspot.com/svg-editor>

SVG Optimiser by Peter Collingridge <http://petercollingridge.appspot.com/svg-optimiser>

Sprites

SpriteMe

[SpriteMe](#) is a bookmarklet to combine background images into a single CSS Sprite and computes the proper CSS background-positions.

Sprite Box

Sprite Box by Gilbert Sinnott is an excellent tool to help you create CSS IDs from a single sprite image. It also has samples of sprite images from YouTube, Apple, and Google.

<http://www.spritebox.net/>

Stitches

Stitches is an HTML5 sprite sheet generator. It was developed by [Matthew Cobbs](#). It allows you to load up several images and it will create the sprite image and the appropriate CSS.

<http://draeton.github.io/stitches/>

Browser developer tools

Chrome and Firefox have built-in developer tools that provide great information.

For example, the Network tab in Chrome developer tools shows the load time of each file. Use it to identify your largest files.

User-Agent Switcher for Chrome

[User-Agent Switcher for Chrome](#) by Glen Wilson allows you to quickly switch between browser types to see how your site performs. This Chrome extension is rated at 4 out of 5 stars on 543 user reviews.

DOM Monster

[DOM Monster](#) by [Amy Hoy](#) and [Thomas Fuchs](#) is a bookmarklet that will analyze your site, identify any problems, and offer suggestions for improvement.

Browserscope

[Browserscope.org](#) allows you to test several key functions of your browser. For example, it will tell you whether your browser is capable of asynchronously downloading scripts with images, frames, etc.

Validation

Validate HTML

<http://validator.w3.org>

Validate CSS

<http://jigsaw.w3.org/css-validator/>

Server Tools

Server Pilot

[Serverpilot.io](#) is a great tool for managing your cloud servers.

Apache Bench

Apache Bench is a benchmarking tool for your server.

<http://httpd.apache.org/docs/2.4/programs/ab.html>

Apache JMeter

Apache JMeter is a desktop application for server performance measurement and load testing.

<http://jmeter.apache.org>

Analytics

Google Analytics

Google Analytics provides some useful data on page load times under Content -> Site Speed.

Keep up with Google's Speed page for new tools.

<https://developers.google.com/speed/>

Piwik

If you're not a fan of sharing so much of your data with Google, use the free open source alternative to Google Analytics.

The community is very concerned about privacy, so you keep your data on your own host. Plus, they have mobile apps for iPhone and Android so you can keep up with your site while on the go without letting strangers in on your data party.

www.piwik.org

Free Gift

As a special ***Thank You*** for investing your time in this book, I've created a Checklist for Speed with many of the techniques outlined in this book.

Send an email to **checklist@michaelcale.com** to receive the Checklist for Speed for free.

Blog

If you're interested in keeping current with web development techniques, sign up to my blog at

www.michaelcale.com/signup/