Optimise your web pages

Understanding how a browser renders a web page can help us speed it up, says **Julian Bucknall**

You'll need this...

▶ A WEBSITE AND A WEB BROWSER

We'll show you how to optimise your personal website so it loads as fast as possible, first time.

You probably surf the web every day. You no longer think about what's going on under the hood of your browser – you don't notice anything until your connection goes down, or some site serves up an auto-play Flash movie. But let's take a step back and explore how it works when you visit a site using your desktop browser. Once we have the general idea, we'll investigate how to improve the speed of loading your site and how the Amazon Kindle Fire promises to do it automatically.

We'll take the new PC Plus website for example, which, as I'm sure you know, is at pcplus.co.uk. You type that into your browser's address bar and hit [Enter]. What happens next?

First, the browser makes an assumption: that you're going to be using the HTTP protocol to retrieve whatever data happens to be at pcplus.co.uk. HTTP stands for hypertext transfer protocol, and is a definition of how the browser and web server communicate. HTTP is a text protocol, which defines the types and contents of data packets that are sent from the browser to the server and back again. It also describes the actions that a browser can ask of the server, whether a request is a GET ('I need to download this file'), POST ('here's the information from the filled-in form') or one of a dozen or so other verbs. Part of the protocol is a list of status numbers the server can reply with, of which the most famous is error 404, resource not found, and the most common is code 200, success.

HTTP, then, describes the questions and answers – the protocol – the two ends use in order for the browser to download the relevant files so it can render the web page.

But before the browser can start doing all this though, it needs to find pcplus.co.uk.

Finding the page

The Domain Name System (DNS) is a distributed system that translates human-readable domain names like pcplus.co.uk or my domain boyet.com into IP addresses like 89.167.143.56 or 184.168.192.115. DNS is implemented as a hierarchical system of servers whose only job is to store and translate name-address pairs. When you set up your connection to the internet in the first place, you define (or have defined for you) one or two upstream DNS servers that your PC will use to resolve human-readable domain names. Similarly, when a website admin sets up a new website, they have to define as part of the domain name registration a couple of DNS servers that are responsible for disseminating the domain name and its IP address to other DNS servers.

Back to our example: the browser resolves pcplus.co.uk to its IP address by querying DNS, then makes a connection

Browser plugins

Looking at the network traffic pane in Firebug or your preferred browser developer tool can be an eye-opening experience. Suddenly you see what a brake those social network buttons put on your website, but that data would be more informative if you could identify some tasks that will make your website load better.

Browser plugins like YSlow and Page Speed aim to do that. YSlow is now part of Firebug (there's a tab for it on the toolbar) and Page Speed can be downloaded from Google. They work in the background to analyse how a page is downloaded and rendered and grade the site accordingly. Page Speed even has features that will compress images for a page, minify JavaScript (if you aren't doing it already) and clean up CSS. ■

Spotlight on... Images

I know. You take a photo, you upload it to your blog and you don't think about it again. But in reality there are things you can do to optimise that image for displaying on the web, for optimising the download speed.

Firstly, photos are created with a whole set of metadata tacked onto the image file. This is usually known as the EXIF data and details things like the camera make, the aperture, the shutter speed, the date/time, and so on. None of this is needed for displaying images in a web page, although it can be handy in a photo network site like Flickr.

Secondly, many images in web pages are basically visual fluff. They're there to make the page look nice, but don't have any intrinsic additional information in themselves. You should therefore make sure that your images are sized accordingly for the page (it's pointless forcing the user to download a large photo that will be resized by the browser in order to fit on the page).

Thirdly, if you're using the JPG format, you might find that saving with a lower quality measurement will give you just as good an image as the maximum setting, but result in a much smaller file size.

Photoshop has a special 'Save for web' option that you can use which will compress the image ready for displaying in a web page. This not only removes the EXIF data and sets the image size, but also allows you to view the image at different quality levels so that you can make the decision between higher quality/larger file and lower quality/smaller file. ■

to that IP address, and sends an HTTP GET request asking for the root document (which is generally '/').

This is where it gets a little difficult with respect to pcplus.co.uk – the server is set up to redirect HTTP requests to pcplus.techradar.com. In essence, pcplus.co.uk replies the GET request with an error 301 (moved permanently) and the new address pcplus.techradar.com. Another DNS request and reply later, the browser asks this new address for the root document.

Most websites are set up to return a particular document when the root is requested. The standard names of this root document include default.html, index.html, or similar names with an .aspx or .php suffix. The text document that is returned to the browser is an HTML document, HTML standing for Hypertext Markup Language. I'm sure you're familiar with how HTML is structured: it's an ordinary text document with tags embedded that allow the writer to delimit paragraphs, titles, links, whether words are emphasised and so on. The tags themselves are embedded between angle brackets and usually have a starting and an ending version (for example, a paragraph starts with and ends with).

Manage resources

My intent is not to rehash how to write a valid HTML file, but to point out that there are tags in the HTML file that instruct the browser to download yet more resources. The simplest of these is the <img> tag, which defines an image to be displayed as part of the web page. The browser will parse the HTML file looking for tags like these, and fire off separate requests to the web server (and sometimes other web servers, necessitating more DNS requests) to retrieve these extra files.

For images, the browser can fire off these download requests in parallel. It does not have to wait for a download request to be replied to by the server; it can continue rendering and flowing the page and slot in the images (for example) when they arrive.

Page 74 shows the Network Traffic pane of Firebug. I'd cleared Firefox's cache before loading a simple web page (HTML, CSS and some JavaScript). You can see that the first item is the HTML file itself, and after a small amount of time parsing that, the browser kicks off three file downloads at once: the two CSS files and an image from LinkedIn. The CSS files were being parsed in parallel (one is a print stylesheet and so isn't used for the screen), and while that was happening the browser kicked off some more image downloads. Note that the first JavaScript file is only downloaded once the HTML file is completely parsed since I added the <script> tag right at the end of the HTML.

If the HTML states the height and width of the image, the browser can reserve some space on the page for it as it renders the page, and the image will be inserted once the GET request is complete. This is highly efficient. If not, the browser will have to reflow the page, incurring extra processing time, once the image files are received.

Speed up rendering

However, some files will affect the display of the page when downloaded, often drastically. It makes sense for the browser to essentially stop rendering until these files are completely downloaded and parsed and whatever actions are defined in the file are completed.

The first type of file is a cascading style sheet (CSS). This file contains styling information for the various content tags in the main HTML file: the colour of the text and background, the fonts to be used, the text attributes like bold, underline and so on, whether a particular tag floats within the main flow of the document text, and whether it is visible or not.

CSS files should pretty much always be defined in the <head> section of the HTML file, although it's possible for JavaScript to load them on the fly. Since CSS files can be downloaded in parallel, it makes sense to group them together and have as few as possible. Once the browser has parsed the CSS files completely it will have built up a data structure containing the styles, and will be able to apply them as the rest of the HTML file is parsed, flowed, and rendered.

The second file type that causes the browser to stop what it's doing until the entire file is downloaded is JavaScript. These are even worse than CSS files in some respects: the browser *must* download a JavaScript file in its entirety, parse it and then run whatever code needs to be executed before moving on with the page rendering.

The reason for this is that JavaScript code can (and usually will) modify the Document Object Model (DOM) of the page. The DOM is a complicated data structure that holds the compiled version of the HTML text. For a long time now, JavaScript code has been allowed to modify this structure, adding, changing or deleting elements. Every change to the DOM is then reflected in the rendered page by the browser.

Another of JavaScript's abilities is the so-called document.write technique. This is frowned upon these days for production websites because of the deleterious effect it can have on the rendering speed of the page. In essence, document.write will insert text into the HTML file at the exact point where the JavaScript code was added and executed. The browser has to stop parsing the HTML, insert the new HTML code into the stream, and then start over.

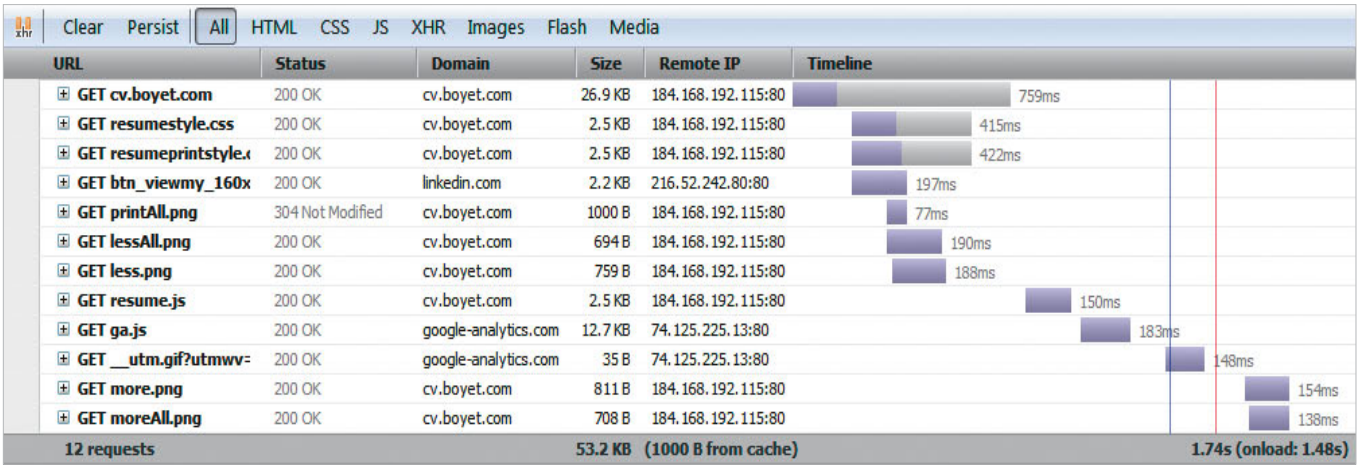
It gets even worse – if you download other JavaScript files dynamically in your JavaScript code, Internet ▶

Opera Turbo

For mobile phones, Opera Software offers a replacement browser called Opera Mini. One of this browser's most interesting features is very Kindle Fire-like in its behaviour: it routes all HTTP requests through a proxy server to Opera's servers. These servers request the original page, compress it and send it to the phone. Some work is also done to process and reformat the page for display on the smaller screen.

The compression process actually converts textual HTML into a binary markup language called OBML to increase the compression and to reduce the bandwidth needed to download the page to the device.

Other processing includes the ability to pre-execute JavaScript on the server and send the resulting page to the device. For heavily JavaScript-driven sites, this can produce a sub-optimal view of the site. ■



▲ **Network traffic for displaying a simple web page.**

Explorer will execute them in the order they are received, while the other major browsers will execute them in the order they are added to the DOM.

Eventually, the full HTML file is downloaded, the DOM is created, and the page is rendered inside the browser.

Now that we know how a web page gets rendered in a desktop or laptop browser, we can see various techniques to ensure that the first-time rendering is as fast as possible. The first is that all referenced files in an HTML file must be downloaded. This might need a DNS query if it's from a new domain. It certainly requires another HTTP request to be sent to the server and the file to be received. Therefore, the fewer referenced files, the better. With regard to images, there's not a lot we can do – they are usually all separate images from different places.

The number of CSS and JavaScript files can certainly be reduced. Instead of having several CSS and JavaScript files as part of your web page, concatenate them together to make one CSS file and one JavaScript file. Your web page will instantly load faster, especially as there is just one JavaScript file to evaluate and execute. Another trick is to minify your CSS and JavaScript files prior to uploading them to your site, and turn on gzip compression if you aren't already using it.

Amazon Silk

Although the description of how a browser renders a page is valid for a desktop or a laptop, another browser was announced recently. This browser, known as Silk, splits the traditional work of the browser between the rendering app on the Kindle and Amazon's EC2 cloud.

The first thing Silk does is use Amazon's cloud as a cache for the various files for web pages. When you use the browser on the Kindle Fire to display, say, the home page for PC Plus, it will connect to Amazon's servers and not pcplus.co.uk. If you happen to be the first person to request the page, Amazon's EC2 will fetch all the required files store them in the cloud, then send them down. The next person to access the page through their Kindle Fire will, in theory, benefit from the cache.

Of course, in order for that to happen, the Kindle Fire, by default, uses a proxy server to Amazon. This means that all HTTP requests are routed through Amazon, and its EC2 servers can process all data fetched via the proxy server.

This particular feature doesn't mean that the device's browser won't cache files as well. All browsers cache files from the web: reading a file from the local drive is always going to be faster than going out to the internet to fetch it. The Fire's onboard memory is very limited (8GB of flash storage), so the Silk browser is more aggressive about clearing out seldom-used files than a PC browser like Firefox.

Not only do Amazon's cloud servers cache web files like HTML, CSS, JS, JPG, and PNG, they also optimise them for the device automatically. This could mean something as simple as minifying and compressing the various text files before sending them to the Fire, but it also means that image files are automatically optimised for viewing on the device. So instead of downloading a large image for a web page, Amazon's servers will compress the image's resolution for the device, making a smaller file. I could find no information to indicate that they optimise the image's colour palette for web viewing.

A question of privacy

The final feature is perhaps the most controversial of all. The cloud part of the Silk browser will monitor the sites and pages you visit. It will aggregate that data over all Kindle Fire users and use the analysis to guess which pages you will be visiting next and make sure they are preloaded in the cloud's cache.

All of these Silk features, no matter how efficient they are at improving the Fire user's surfing experience, raise significant privacy concerns. After all, you're surfing the internet through Amazon's servers. It could aggregate data on your surfing habits quite easily, especially data that helps predict what site you might visit next. But then again, your ISP already has some knowledge about what you look at and where you surf – it just doesn't aggregate that data particularly well.

How much faster does that make Silk? Amazon lets you turn off the cloud caching aspects of the browser (touch the menu icon for the browser, touch 'Settings', then touch 'Accelerate page loading'), so you can experiment with using the browser with the cloud and traditionally. For sites like Facebook, I found that using the cloud made the whole experience appallingly slow; everyone's differing view of Facebook doesn't allow for many benefits from caching. News sites were marginally faster using the cloud (but they have a lot of images, so maybe the pre-compression of these helped). Sites like PC Plus or my site are about the same.

I've noticed that you become extremely dependent on the speed of Amazon's servers. If the connection to them is slow then you'll really notice it. I have, in fact, turned off Accelerated Page Loading on my Kindle Fire. It's not worth it in my experience. PCP

Julian M Bucknall has worked for companies ranging from TurboPower to Microsoft and is now CTO for Developer Express..
feedback@pcplus.co.uk

When to load JS

If you take another look at the image above, you can see that the first JavaScript file (resume.js) is loaded after the HTML for the page has been fully processed, and in fact the page has been displayed. This is another technique to speed up the display of web pages. Although the JavaScript code still has to be parsed and executed, the user is already looking at the web page in the browser. To the end user it seems as if the page is already fully rendered, even though there is still some page processing going on in the background.

If, on the other hand, I'd placed the <script> tag at the top of the HTML page, the user would have to wait until the JS file were downloaded, parsed, and executed before the content of the page became visible.