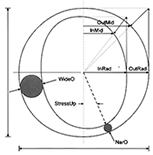
**Optimizing Web Font Rendering Performance**

*By*[*Ilya Grigorik*](https://www.igvita.com/)*on****January 31, 2014***

Web font adoption continues to accelerate across the web: according to HTTP Archive, [~37% of top 300K sites are using web fonts](http://httparchive.org/trends.php#perFonts) as of early 2014, which translates to a *2x+* increase over the past twelve months. Of course, this should not be all that surprising to most of us.**Typography has always been an important part of good design, branding, and readability and web fonts offer many additional benefits: the text is selectable, searchable, zoomable, and high-DPI friendly.** What's not to like?

Ah, but what about the rendering speed, don't web fonts come with a performance penalty? Fonts are an additional critical resource on the page, so yes, they can impact rendering speed of our pages. That said,**just because the page is using web fonts doesn't mean it will (or has to) render slower.**

There are four primary levers that determine the performance impact of web fonts on the page:

1. **The total number of fonts and font-weights used on the page.**
2. **The total byte size of fonts used on the page.**
3. **The transfer latency of the font resource.**
4. **The time when the font downloads are initiated.**

The first two levers are directly within the control of the designer of the page. The more fonts are used, the more requests will be made and more bytes will be incurred. The general UX best practice is to keep the number of used fonts at a minimum, which also aligns with our performance goals. Step one: use web fonts, but audit your font usage periodically and try to keep it lean.

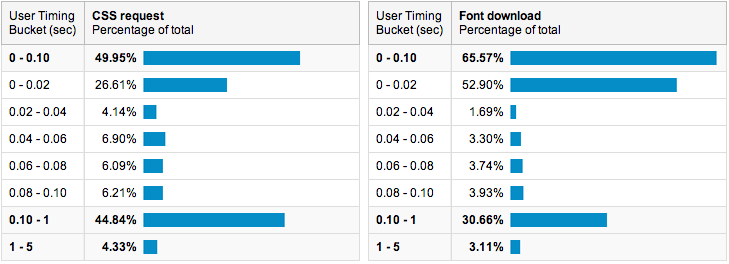
**Measuring web font latencies**

The transfer latency of each font file is dependent on its bytesize, which in turn is determined by the number of glyphs, font metadata (e.g [hinting](http://en.wikipedia.org/wiki/Font_hinting) for Windows platforms), and used compression method. Techniques such as font subsetting, [UA-specific optimization](http://www.igvita.com/2012/09/12/web-fonts-performance-making-pretty-fast/#optimizing), and more efficient compression (e.g. Google Fonts recently [switched to Zopfli](https://plus.google.com/+IlyaGrigorik/posts/1sxencNkbNS) for WOFF resources), are all key to optimizing the transfer size. Plus, since we're talking about latency, where the font is served from makes a difference also – i.e. a CDN, and ideally the user's cache!

That said, instead of talking in the abstract, **how long does it actually take the visitor to download the web font resource on your site?** The best way to answer this question is to instrument your site via the Resource Timing API, which allows us to get the DNS, TCP, and transfer time data for each font - as a bonus, Google Fonts [recently enabled Resource Timing](http://googledevelopers.blogspot.com/2013/12/measuring-network-performance-with.html) support! Here is an example snippet to report font latencies to Google Analytics:

// check if visitor's browser supports Resource Timing if (typeof window.performance == 'object') { if (typeof window.performance.getEntriesByName == 'function') { function logData(name, r) { var dns = Math.round(r.domainLookupEnd - r.domainLookupStart), tcp = Math.round(r.connectEnd - r.connectStart), total = Math.round(r.responseEnd - r.startTime); \_gaq.push( ['\_trackTiming', name, 'dns', dns], ['\_trackTiming', name, 'tcp', tcp], ['\_trackTiming', name, 'total', total] ); } var \_gaq = \_gaq || []; var resources = window.performance.getEntriesByType("resource"); for (var i in resources) { if (resources[i].name.indexOf("themes.googleusercontent.com") != -1) { logData("webfont-font", resources[i]) } if (resources[i].name.indexOf("fonts.googleapis.com") != -1) { logData("webfont-css", resources[i]) } } } }

The above example captures the key latency metrics both for the UA-optimized CSS file and the font files specified in that file: the CSS lives on fonts.googleapis.com and is cached for 24 hours, and font files live onthemes.googleusercontent.com and have a long-lived expiry. With that in place, let's take a look at the total(responseEnd - startTime) timing data in Google Analytics for my site:



For privacy reasons, the Resource Timing API intentionally does not provide a "fetched from cache” indicator, but we can nonetheless use a reasonable timing threshold - say, 20ms - to get an approximation. Why 20ms? Fetching a file from spinning rust, and even flash, is [not free](http://www.igvita.com/2009/06/23/measuring-optimizing-io-performance/). The actual cache-fetch timing will vary based on hardware, but for our purposes we'll go with a relatively aggressive 20ms threshold.

With that in mind and based on above data for visitors coming to my site, the median time to get the CSS file is ~100ms, and ~26% of visitors get it from their local cache. Following that, we need to fetch the required font file(s), which take <20ms at the median – a significant portion of the visitors has them in their browser cache! This is great news, and a confirmation that the [Google Fonts strategy of long-lived and shared font resources](http://www.igvita.com/2012/09/12/web-fonts-performance-making-pretty-fast/#serving) is working.

Your results will vary based on the fonts used, amount and type of traffic, plus other variables. The point is that **we don't have to argue in the abstract about the latency and performance costs of web fonts: we have the tools and APIs to measure the incurred latencies precisely.** And what we can measure, we can optimize.

**Timing out slow font downloads**

Despite our best attempts to optimize delivery of font resources, sometimes the user may simply have a poor connection due to a congested link, poor reception, or a variety of other factors. In this instance, the critical resources – including font downloads – may block rendering of the page, which only makes the matter worse. To deal with this, and specifically for web fonts, different browsers have taken different routes:

* IE immediately renders text with the fallback font and re-renders it once the font download is complete.
* Firefox holds font rendering for up to 3 seconds, after which it uses a fallback font, and once the font download has finished it re-renders the text once more with the downloaded font.
* **Chrome and Safari hold font rendering until the font download is complete.**

There are many good arguments for and against each strategy and we won't go into that discussion here. That said, I think most will agree that the lack of *any* timeout in Chrome and Safari is not a great approach, and this is something that the Chrome team has been investigating for a while. What should the timeout value be? To answer this, we've instrumented Chrome to gather font-size and fetch times, which yielded the following results:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Webfont size range | Percent | 50th | 70th | 90th | 95th | 99th |
| 0KB - 10KB | 5.47% | 136 ms | 264 ms | 785 ms | 1.44 s | 5.05 s |
| 10KB - 50KB | 77.55% | 111 ms | 259 ms | 892 ms | 1.69 s | 6.43 s |
| 50KB - 100KB | 14.00% | 167 ms | 882 ms | 1.31 s | 2.54 s | 9.74 s |
| 100KB - 1MB | 2.96% | 198 ms | 534 ms | 2.06 s | 4.19 s | 10+ s |
| 1MB+ | 0.02% | 370 ms | 969 ms | 4.22 s | 9.21 s | 10+ s |

First, the good news is that the majority of web fonts are relatively small (<50KB). Second, most font downloads complete within several hundred milliseconds: picking a 10 second timeout would impact ~0.3% of font requests, and a 3 second timeout would raise that to ~1.1%. Based on this data, **the**[**conclusion**](https://code.google.com/p/chromium/issues/detail?id=235303#c17)**was to make Chrome mirror the Firefox behavior: timeout after 3 seconds and use a fallback font, and re-render text once the font download has completed.** This behavior will ship in Chrome M35, and I hope Safari will follow.

**Hands-on: initiating font resource requests**

We've covered how to measure the fetch latency of each resource, but there is one more variable that is often omitted and forgotten: **we also need optimize when the fetch is initiated.** This may seem obvious on the surface, except that it can be a tricky challenge for web fonts in particular. Let's take a look at a hands-on example:

@font-face { font-family: 'FontB'; src: local('FontB'), url('http://mysite.com/fonts/fontB.woff') format('woff'); } p { font-family: FontA }

<!DOCTYPE html> <html> <head> <link href='stylesheet.css' rel='stylesheet'> <!-- see content above --> <style> @font-face { font-family: 'FontA'; src: local('FontA'), url('http://mysite.com/fonts/fontA.woff') format('woff'); } </style> <script src='application.js' /> </head> <body> <p>Hello world!</p> </body> </html>

There is a lot going on above: we have an external CSS and JavaScript file, and inline CSS block, and two font declarations. **Question: when will the font requests be triggered by the browser?** Let's take it step by step:

1. Document parser discovers external stylesheet.css and a request is dispatched.
2. Document parser processes the inline CSS block which declares FontA - we're being clever here, we want the font request to go out as early as possible. Except, it doesn't. More on that in a second.
3. Document parser blocks on external script: we can't proceed until that's fetched and executed.
4. Once the script is fetched and executed we finish constructing the DOM, style calculation and layout is performed, and we finally dispatch request for fontA. At this point, we can also perform the first paint, but we can't render the text with our intended font since the font request is inflight... doh.

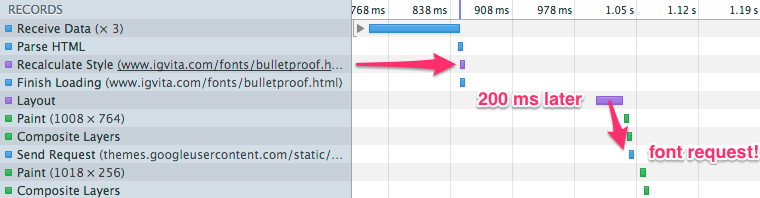
The key observation in the above sequence is that font requests are not initiated until the browser knows that the font is actually required to render some content on the page - e.g. we never request FontB since there is no content that uses it in above example! On one hand, this is great since it minimizes the number of downloads. On the other, it also means that **the browser can't initiate the font request until it has both the DOM and the CSSOM and is able to resolve which fonts are required for the current page.**

In the above example, our external JavaScript blocks DOM construction until it is fetched and executed, which also delays the font download. To fix this, we have a few options at our disposal: (a) eliminate the JavaScript, (b) add an async attribute (if possible), or (c) move it to the bottom of the page. However, the more general takeaway is that font downloads won't start until the browser can compute the the render tree. **To make fonts render faster we need to**[**optimize the critical rendering path**](http://www.youtube.com/watch?v=I4vX-twze9I)**of the page.**

*Tip: in addition to measuring the relative request latencies for each resource, we can also measure and analyze the request start time with Resource Timing! Tracking this timestamp will allow us to determine when the font request is initiated.*

**Optimizing font fetching in Chrome M33**

Chrome M33 landed an important optimization that will significantly improve font rendering performance. The easiest way to explain the optimization is to look at a pre-M33 example timeline that illustrates the problem:



1. Style calculation completed at ~840ms into the lifecycle of the page.
2. Layout is triggered at ~1040ms, and font request is dispatched immediately after.

Except, why did we wait for layout if we already resolved the styles two hundred milliseconds earlier? Once we know the styles we can figure out which fonts we'll need and immediately initiate the appropriate requests – that's the new behavior in Chrome M33! On the surface, this optimization may not seem like much, but based on our Chrome instrumentation the gap between style and layout is actually much larger than one would think:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Percentile | 50th | 60th | 70th | 80th | 90th |
| Time from Style → Layout | 132 ms | 182 ms | 259 ms | 410 ms | 820 ms |

**By dispatching the font requests immediately after first style calculation the font download will be initiated ~130ms earlier at the median and ~800ms earlier at 90th percentile!** Cross-referencing these savings with the font fetch latencies we saw earlier shows that in many cases this will allow us to fetch the font *before* the layout is done, which means that we won't have to block text rendering at all – this is a huge performance win.

*Of course, one also should ask the obvious question... Why is the gap between style calculation and layout so large? The first place to start is in Chrome DevTools: capture a timeline trace and check for slow operations (e.g. long-running JavaScript, etc). Then, if you're feeling adventurous, head to chrome://tracing to take a peek under the hood – it may well be that the browser is simply busy processing and laying out the page.*

**Optimizing web fonts with Font Load Events API**

Finally, we come to the most exciting part of this entire story: [Font Load Events API](http://dev.w3.org/csswg/css-font-loading/). In a nutshell, this API will allow us to manage and define how and when the fonts are loaded – **we can schedule font downloads at will, we can specify how and when the font will be rendered, and more.** If you're familiar with the [Web Font Loader](https://github.com/typekit/webfontloader) JS library, then think of this API as that and more but implemented natively in the browser:

var font = new FontFace("FontA", "url(http://mysite.com/fonts/fontA.woff)", {}); font.ready().then(function() { // font loaded.. swap in the text / define own behavior. }); font.load(); // initiate immediate fetch / don't block on render tree!

Font Load Events API gives us complete control over which fonts are used, when they are swapped in (i.e. should they block rendering), and when they're downloaded. In the example above we construct a FontFace object directly in JavaScript and trigger an immediate fetch – we can inline this snippet at the top of our page and avoid blocking on CSSOM and DOM entirely! **Best of all, you can already play with this API in Canary builds of Chrome, and if all goes well it should find its way into stable release by M35.**

**Web font performance checklist**

Web fonts offer a lot of benefits: improved readability, accessibility (searchable, selectable, zoomable), branding, and when done well, [beautiful results](http://hellohappy.org/beautiful-web-type/). **It's not a question of if web fonts should be used, but how to optimize their use.** To that end, a quick performance checklist:

1. Audit your font usage and keep it lean.
2. Make sure font resources are optimized - see [Google Web Fonts tricks](http://www.igvita.com/2012/09/12/web-fonts-performance-making-pretty-fast/#optimizing).
3. Instrument your font resources with Resource Timing: measure → optimize.
4. Optimize the transfer latency and time of initial fetch for each font.
5. Optimize your critical rendering path, eliminate unnecessary JS, etc.
6. Spend some time playing with the Font Load Events API.

Just because the page is using a web font, or several, doesn't mean it will (or has to) render slower. A well optimized site can deliver a better and faster experience by using web fonts.