Performance

Useful links

https://developers.google.com/speed

https://hpbn.co/

https://developer.mozilla.org/en-US/docs/Learn/Performance

1 Userful Terms

1.1 Cumulative Layout Shift

CLS indicates how much layout shift is experienced by visitors as your page loads. Aim for a score of 0.1 or less.

1.2 First Contentful Paint

How quickly content like text or images are painted onto your page. Aim for 0.9s or less.

1.3 Largest Contentful Paint

LCP measures how long it takes for the **largest content element** (e.g. a hero image or heading text) on your page to become visible within your visitors' viewport. Aim for 1.2s or less

1.4 Speed Index

How quickly the contents of your page are visibly populated. Aim for 1.3s or less

1.5 Time to Interactive

How long it takes for your page to become fully interactive. Aim for 2.5s or less.

1.6 Total Blocking Time

TBT tells you how much time is blocked by scripts during your page loading process. Aim for 150ms or less.

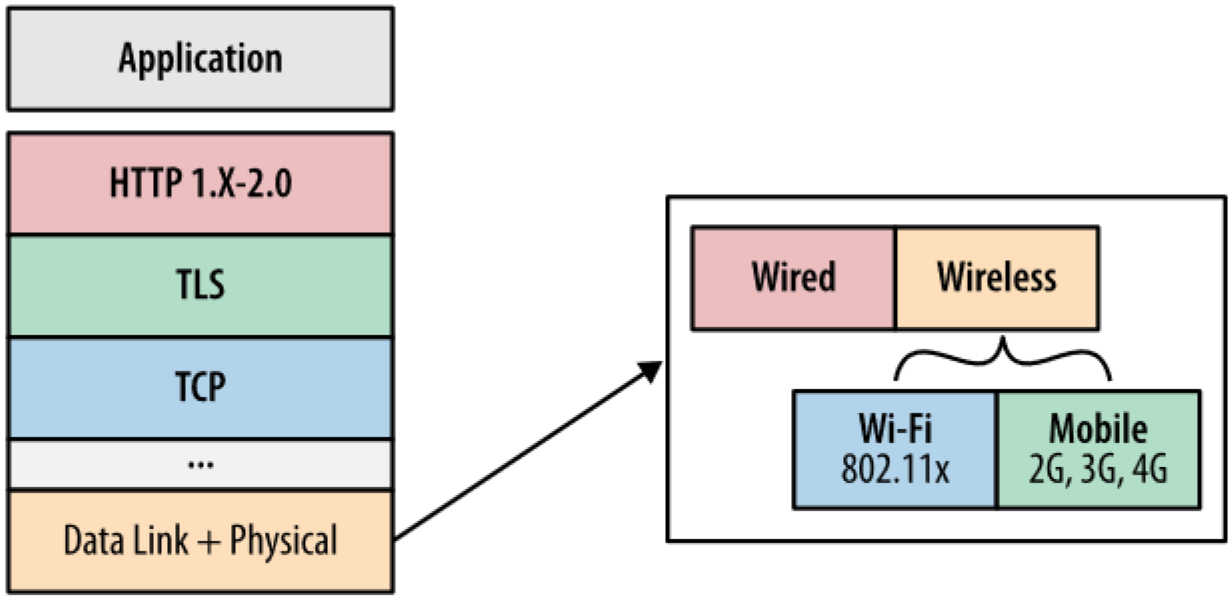
1.7 Apache PageSpeed

The PageSpeed Modules, mod_pagespeed and ngx_pagespeed, are open-source webserver modules that optimize your site automatically.

2 Best Practices

High-performance browser networking relies on a host of networking technologies and the overall performance of our applications is the sum total of each of their parts.

We cannot control the network weather between the client and server, nor the client hardware or the configuration of their device, but the rest is in our hands: TCP and TLS optimizations on the server, and dozens of application optimizations to account for the peculiarities of the different physical layers, versions of HTTP protocol in use, as well as general application best practices.



2.1 HTTP

https://hpbn.co/optimizing-application-delivery/

2.1.1 Reduce unnecessary network latency and minimize the number of transferred bytes

Regardless of the type of network or the type or version of the networking protocols in use, all applications should always seek to eliminate or reduce unnecessary network latency and minimize the number of transferred bytes. These two simple rules are the foundation for all of the evergreen performance best practices:

2.1.1.1 Reduce DNS lookups

Every hostname resolution requires a network roundtrip, imposing latency on the request and blocking the request while the lookup is in progress.

2.1.1.2 Reuse TCP connections

Leverage connection keepalive whenever possible to eliminate the TCP handshake and slow-start latency overhead; see Slow-Start.

2.1.1.3 Minimize number of HTTP redirects

HTTP redirects impose high latency overhead—e.g., a single redirect to a different origin can result in DNS, TCP, TLS, and request-response roundtrips that can add hundreds to thousands of milliseconds of delay. The optimal number of redirects is zero.

2.1.1.4 Reduce roundtrip times

Locating servers closer to the user improves protocol performance by reducing roundtrip times (e.g., faster TCP and TLS handshakes), and improves the transfer throughput of static and dynamic content; see Uncached Origin Fetch.

2.1.1.5 Eliminate unnecessary resources

No request is faster than a request not made. Be vigilant about auditing and removing unnecessary resources.

2.1.2 Caching and Compression

By this point, all of these recommendations should require no explanation: latency is the bottleneck, and the fastest byte is a byte not sent. However, HTTP provides some additional mechanisms, such as caching and compression, as well as its set of version-specific performance quirks:

2.1.2.1 Cache resources on the client

Application resources should be cached to avoid re-requesting the same bytes each time the resources are required.

2.1.2.2 Compress assets during transfer

Application resources should be transferred with the minimum number of bytes: always apply the best compression method for each transferred asset.

2.1.2.3 Eliminate unnecessary request bytes

Reducing the transferred HTTP header data (e.g., HTTP cookies) can save entire roundtrips of network latency.

2.1.2.4 Parallelize request and response processing

Request and response queuing latency, both on the client and server, often goes unnoticed, but contributes significant and unnecessary latency delays.

2.1.2.5 Apply protocol-specific optimizations

HTTP/1.x offers limited parallelism, which requires that we bundle resources, split delivery across domains, and more. By contrast, HTTP/2 performs best when a single connection is used, and HTTP/1.x specific optimizations are removed.

2.2 Images

https://developer.mozilla.org/en-US/docs/Learn/Performance/Multimedia

Set an explicit width and height on image elements to reduce layout shifts.

2.2.1 CDNs

https://web.dev/image-cdns/

Image content delivery networks (CDNs) are excellent at optimizing images. Switching to an image CDN can yield a 40–80% savings in image file size.

For images loaded from an image CDN, an image URL indicates not only which image to load, but also parameters like size, format, and quality. This makes it easy to create variations of an image for different use cases.

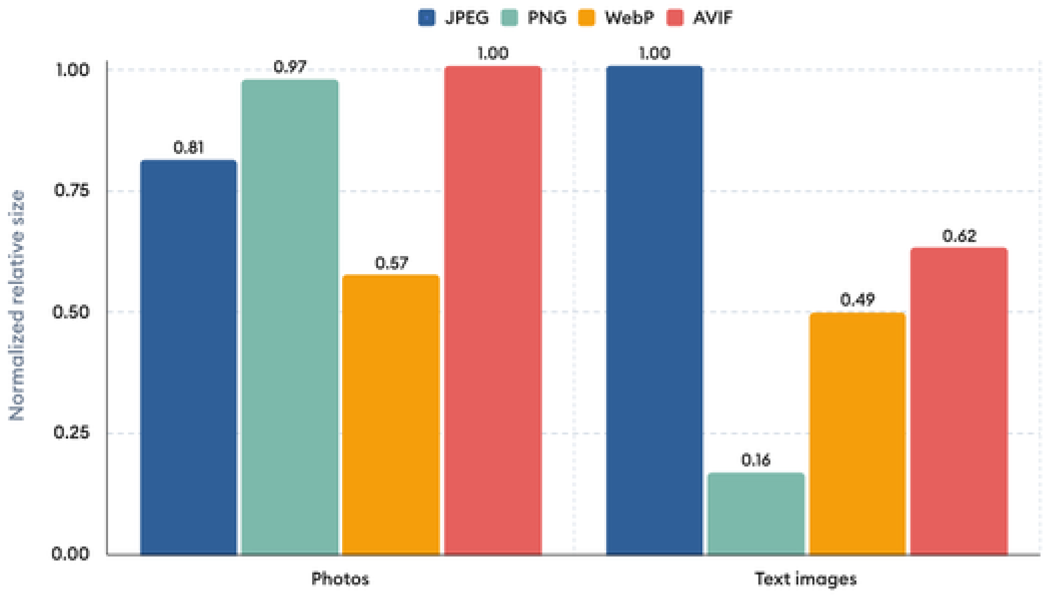
2.2.2 Compression

https://web.dev/use-imagemin-to-compress-images/

- AVIF is a solid first choice if lossy, low-fidelity compression is acceptable and saving bandwidth is the number one priority. Assuming encode /decode speeds meet your needs.
- WebP is more widely supported and may be used for rendering regular images where advanced features like wide color gamut or text overlays are not required.
- AVIF may not be able to compress non-photographic images as well as PNG or lossless WebP. Compression savings from WebP may be lower than JPEG for high-fidelity lossy compression.

https://web.dev/serve-images-webp/

WebP images are smaller than their JPEG and PNG counterparts—usually on the magnitude of a 25–35% reduction in filesize. This decreases page sizes and improves performance.



2.2.3 Lazy loading

https://web.dev/use-lazysizes-to-lazyload-images/

You can use the “loading” attribute to completely defer the loading of offscreen images that can be reached by scrolling:

2.2.4 Responsive

https://web.dev/serve-responsive-images/

Serving desktop-sized images to mobile devices can use 2–4x more data than needed. Instead of a "one-size-fits-all" approach to images, serve different image sizes to different devices.

Specify multiple image versions and the browser will choose the best one to use:

https://web.dev/serve-images-with-correct-dimensions/

2.2.5 Preload

https://web.dev/preload-responsive-images/

2.3 JavaScript

By default, references to external JavaScript files will block the page from rendering while they are fetched and executed. Often, these files can be loaded in a different manner, freeing up the page to visually render sooner.

In modern websites, scripts are often “heavier” than HTML: their download size is larger, and processing time is also longer.

When the browser loads HTML and comes across a <script>...</script> tag, it can’t continue building the DOM. It must execute the script right now. The same happens for external scripts <script src="..."></script>: the browser must wait for the script to download, execute the downloaded script, and only then can it process the rest of the page.

That leads to two important issues:

- Scripts can’t see DOM elements below them, so they can’t add handlers etc.

- If there's a bulky script at the top of the page, it "blocks the page". Users can't see the page content till it downloads and runs:

2.3.1 Minify

2.3.2 Defer

A script that will be downloaded in parallel to parsing the page, and executed after the page has finished parsing. Deferred scripts still execute in the order they are defined in source.

2.3.3 Async

Similar to "defer" but Async scripts are not guaranteed to execute in the order they are defined in source.

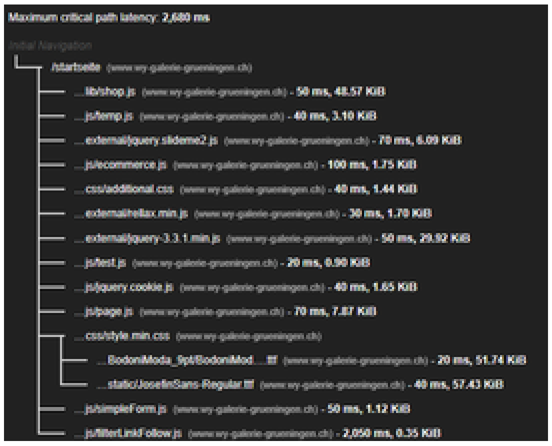
2.3.4 Split

2.3.5 Inline

This experiment embeds the contents of specified external scripts directly into the HTML within a script element. This increases the size of the HTML, but can often allow page page to display sooner by avoiding server round trips.Example implementation:

2.3.6 Avoid chaining critical requests

Critical request chains are series of dependent network requests important for page rendering. The greater the length of the chains and the larger the download sizes, the more significant the impact on page load performance. These Critical Request Chains show you what resources are loaded with a high priority. Consider reducing the length of chains, reducing the download size of resources, or deferring the download of unnecessary resources to improve page load.



```
Maximum critical path latency: 2,680 ms
Initial Navigation
└─ /startseite (www.wy-galerie-grueningen.ch)
   ├─ …lib/shop.js (www.wy-galerie-grueningen.ch) - 50 ms, 48.57 KiB
   ├─ …js/temp.js (www.wy-galerie-grueningen.ch) - 40 ms, 3.10 KiB
   ├─ …external/jquery.slideme2.js (www.wy-galerie-grueningen.ch) - 70 ms, 6.09 KiB
   ├─ …js/ecommerce.js (www.wy-galerie-grueningen.ch) - 100 ms, 1.75 KiB
   ├─ …css/additional.css (www.wy-galerie-grueningen.ch) - 40 ms, 1.44 KiB
   ├─ …external/rellax.min.js (www.wy-galerie-grueningen.ch) - 30 ms, 1.70 KiB
   ├─ …external/jquery-3.3.1.min.js (www.wy-galerie-grueningen.ch) - 50 ms, 29.92 KiB
   ├─ …js/test.js (www.wy-galerie-grueningen.ch) - 20 ms, 0.90 KiB
   ├─ …js/jquery.cookie.js (www.wy-galerie-grueningen.ch) - 40 ms, 1.65 KiB
   ├─ …js/page.js (www.wy-galerie-grueningen.ch) - 70 ms, 7.87 KiB
   ├─ …css/style.min.css (www.wy-galerie-grueningen.ch)
   │  ├─ …BodoniModa_9pt/BodoniMod….ttf (www.wy-galerie-grueningen.ch) - 20 ms, 51.74 KiB
   │  └─ …static/JosefinSans-Regular.ttf (www.wy-galerie-grueningen.ch) - 40 ms, 57.43 KiB
   ├─ …js/simpleForm.js (www.wy-galerie-grueningen.ch) - 50 ms, 1.12 KiB
   └─ …js/filterLinkFollow.js (www.wy-galerie-grueningen.ch) - 2,050 ms, 0.35 KiB
```

- Minimize the number of critical resources: eliminate them, defer their download, mark them as async, and so on.
- Optimize the number of critical bytes to reduce the download time (number of round trips).
- Optimize the order in which the remaining critical resources are loaded: download all critical assets as early as possible to shorten the critical path length.

2.3.7 External Scripts

If a third-party script is slowing down your page load, you have several options to improve performance:

- Load the script using the async or defer attribute to avoid blocking document parsing.
- Consider self-hosting the script if the third-party server is slow.
- Consider removing the script if it doesn't add clear value to your site.
- Consider Resource Hints like <link rel=preconnect> or <link rel=dns-prefetch> to perform a DNS lookup for domains hosting third-party scripts.
- "Sandbox" scripts with an iframe

2.4 CSS

https://developer.mozilla.org/en-US/docs/Learn/Performance/CSS

By default, references to external CSS files will block the page from rendering while they are fetched and executed. Sometimes these files should block rendering, but can be inlined to avoid additional round-trips while the page is waiting to render. Sometimes, such as with stylesheets that are only used for loading custom fonts, inline or async CSS can greatly improve perceived performance.

The contain CSS property allows an author to indicate that an element and its contents are, as much as possible, *independent* of the rest of the document tree. This allows the browser to recalculate layout, style, paint, size, or any combination of them for a limited area of the DOM and not the entire page.

2.4.1 Minify

2.4.2 Inline external CSS

This experiment embeds the contents of specified external stylesheets directly into the HTML within a <style> element. This increases the size of the HTML, but can often allow page page to display sooner by avoiding server round trips

2.4.3 Async

To load CSS Files asynchronously in both Chrome and Firefox, we can use “preload” browser hint and “media=’print’” attribute along with onload event feature in a ordered way.

Move the not-immediately used styles into separate file

If you're going to load CSS asynchronously, it's generally recommended that you inline critical CSS, since CSS is a render-blocking resource for a reason.

Credit to filament group for their many async CSS solutions.

This approach may not work with content security policy enabled.

2.4.4 Preload

https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/rel/preload

The preload value of the <link> element's rel attribute lets you declare fetch requests in the HTML's <head>, specifying resources that your page will need very soon, which you want to start loading early in the page lifecycle, before browsers' main rendering machinery kicks in. This ensures they are available earlier and are less likely to block the page's render, improving performance. Even though the name contains the term *load*, it doesn't load and execute the script but only schedules it to be downloaded and cached with a higher priority.

2.4.5 Fonts

When fonts are loaded with default display settings, like font-display="block", browsers will hide text entirely for several seconds instead of showing text with a fallback font.

Applied to the @font-face rule, the font-display property defines how font files are loaded and displayed by the browser, allowing text to appear with a fallback font while a font loads, or fails to load. This improves performance by making the text visible instead of having a blank screen, with a trade-off being a flash of unstyled text.

EOT and TTF formats are not compressed by default. Apply compression such as GZIP or Brotli for these file types. Use WOFF and WOFF2. These formats have compression built in.

2.4.6 Animations

https://developer.mozilla.org/en-US/docs/Web/Performance/CSS_JavaScript_animation_performance

To improve performance, the node being animated can be moved off the main thread and onto the GPU. When an element is promoted as a layer, also known as composited, animating transform properties is done in the GPU, resulting in improved performance, especially on mobile.

The CSS will-change property hints to browsers how an element is expected to change. Browsers may set up optimizations before an element is actually changed. These kinds of optimizations can increase the responsiveness of a page by doing potentially expensive work before it is actually required.