How GDS improved GOV.UK’s frontend performance with HTTP/2

Summary

Objectives

The aim was to improve the performance of GOV.UK for users, regardless of the type of device they use or their connection by enabling HTTP/2. During testing, the team identified a number of issues they needed to fix before they could fully enable HTTP/2.

The organisation

Government Digital Service (GDS) leads the digital transformation of the UK government. It runs GOV.UK, which is the government’s online brand, letting citizens interact with government for personal and professional needs.

The project

HTTP/2 (H2) is the latest version of the HTTP protocol. The HTTP Working Group, part of the Internet Engineering Task Force, published H2 in May 2015. H2 is based on the work Google carried out with the SPDY protocol to improve the performance of websites and other internet applications.

The primary goals for HTTP/2 are to:

- minimise protocol overhead via the compression of headers
- reduce network latency with the use of request and response multiplexing
- add support for request prioritisation

Minimise protocol overhead

H2 comes with a technology called HPACK header compression, which compresses any repeated headers. Headers contain metadata and are included in every request and response sent between the browser and the server. The size of all these headers can slow down the loading while a user is browsing a site.

HPACK can result in header compression of over 70% on each request sent to the server during a session.

Reduce network latency

With HTTP/1.1 (H1), the server could only send a single file over each Transmission Control Protocol (TCP) connection at one time. H2 includes a technology called multiplex streams, which send multiple files over the same established TCP connection at the same time.

This results in both parties using fewer resources, and more efficient usage of a user’s bandwidth.

Request prioritisation

Under H1, the browser would request the highest priority assets first. These are Cascading Style Sheets (CSS) and fonts, and the webpage cannot render until the browser receives them.

With H2, the browser looks for resources to download, and once it’s found them, the browser sends a request to the server for all assets at the same time. The server decides the order in which the assets are sent. This is where request prioritisation is important, as it can have a dramatic effect on the page rendering performance for the user.

This prioritisation is complex and easy to get wrong, and many content delivery network (CDN) providers have broken prioritisation implementations. GOV.UK uses a CDN provider which prioritises correctly.

The challenges GDS faced

Enabling H2 on GOV.UK was as simple as asking the CDN provider to switch it on. But before the frontend team at GDS was in a position to do that, it had to test H2 to make sure it would provide the performance improvements the team was looking for.

In October 2018, the team enabled H2 for a trial period of 6 weeks. In that time the team monitored performance across a range of devices. The team decided to only leave it on if they were sure enabling it would actually improve performance for its users.

Unfortunately, synthetic testing found that in many cases H2 actually made performance worse for GOV.UK users. Synthetic testing is a method of testing that simulates the journey a user would take on a service and monitors the performance.

The team compared metrics like:

- first visual change – video analysis to look when page rendering was first visible
- fully loaded – the point at which network activity goes quiet, indicating the page has completely loaded
- SpeedIndex – how quickly the contents of a page are visibly populated (lower is better)
- last visual change – a visual measurement to see when the page last changed

In most cases, the overall timings actually increased, which is not what the team was expecting.

In some instances the difference was minor, maybe a few milliseconds slower for each metric. But in other cases, for example on a slow connection on an older mobile device, there was a big difference in performance.

The GDS frontend team found that H2 actually added seconds to the page load time, which was an unacceptable drop in performance. The team has to make sure GOV.UK works well for every person who needs to use it, regardless of what connection they've got or what device they're using.

The team spent a number of weeks trying to identify why this time increase happened, investigating resource hints such as:

The team felt the issue was related to the assets domain used to serve its static assets. However the team had other work priorities, so they decided to disable H2 until they had time to investigate further.

Re-examining results

While researching another H2 technology called connection coalescing, the team decided to revisit some of the old tests they had conducted when trialing H2, to see if there was any evidence of connection coalescing happening.

The team had previously looked at connection coalescing in case it was causing the slowdown. The team could not confirm it at the time, but the new investigation showed H2 connection coalescing was happening. For example, the team found the browser downloading an image from the assets domain before the domain connection was negotiated, which should not be possible.

The team discovered that connection coalescing was happening between the origin and the assets domain. They found the image downloading on the pre-established TCP connection. So while the browser downloaded HTML and images through the same connection, CSS and JavaScript needed to download through a second connection.

This was because the frontend team had a browser security feature called Subresource Integrity (SRI) enabled for its CSS and JavaScript assets. A requirement of SRI is that you must include the crossorigin attribute.

This attribute is a Cross-Origin Resource Sharing (CORS) security feature that controls exposure of error information to scripts from another domain. Changing the setting to anonymous tells the browser that it must use a TCP connection that does not share user credentials, like cookies, client-side SSL certificates, or HTTP authentication.

This is a real problem for the browser as it cannot establish an anonymous connection quick enough for it not to block page rendering (even when the team tried using the Preconnect resource hint). In turn this new connection negotiation blocks all page rendering, as without the CSS, the browser cannot render anything to the page.

Proposing changes

To fix this issue the team proposed changes through the Request for Comments

(RFC) process used for making changes on GOV.UK.

Their initial proposal in RFC-114 was a relatively simple update to change the crossorigin attribute's value from anonymous to use-credentials.

The theory behind this change was that by letting the browser know it could use a credentialed connection for these assets, it could then use the already established connection to the origin and take advantage of H2 connection coalescing. This would remove the render-blocking wait for the CSS.

Unfortunately, this change was not possible because the team currently serves its assets with the Access-Control-Allow-Origin: * header. The Fetch specification forbids the use of a use-credentials value where Access-Control-Allow-Origin is set to *. This change causes a CORS error in the browser console, and the browser does not fetch the resources.

The team then wrote RFC-115. This RFC proposed to refactor the Access-Control-Allow-Origin header so it was only served with GOV.UK's font assets, and removed SRI completely when requesting the CSS and JavaScript code.

This meant the team could remove the crossorigin attribute, allowing these assets to use the same connection that was established to the origin domain via H2 connection coalescing.

Making changes

After some feedback and adjustments to the RFC, it was finalised and merged as an accepted proposal.

Removing SRI from the CSS and JS assets was an easy change to make, and it was the last blocker to enabling H2 on GOV.UK. This change was made via 9 pull requests to separate GOV.UK applications, and once merged and released the team could test the change on its integration server to verify it had worked.

The tests showed CSS, JavaScript and images were transferred over the same TCP connection, and only fonts were downloaded over an anonymous connection. This meant the browser was downloading CSS and JavaScript through a credentialed connection.

Making HTTP/2 live

The last step of the process involved enabling H2 on both the origin and assets domains. Once done, the difference in the connection views from H1 to H2 was striking. GOV.UK went from 13 TCP connections under H1 with SRI enabled down to 2 TCP connections under H2.

The team tracked the results in a spreadsheet, and found that most device and connection combinations showed performance improvements.

The team still has some work left to do, mainly involving removing the 'assets' domain for its static assets (CSS, JavaScript, fonts, images), and serving them from the origin. Doing so removes the need for all browsers to coalesce these connections. As some browsers either do not support or have broken implementations of H2 connection coalescing, this change will allow all browsers to fully benefit from the H2 optimisations.

User impact

The team used synthetic web performance testing to monitor the change moving from H1 to H2 has had on GOV.UK users. The team tested 3 of the most common device and connection combinations used to access GOV.UK. They looked at the data from an aggregate of results from 18 pages from different areas of GOV.UK.

The team monitored web performance metrics to test the impact the change had on users, including:

Low specification mobile device on a slow connection

The team used SpeedCurve to simulate what connecting to GOV.UK would be like for a user with a slow mobile device on a 2G connection. The tests showed the following improvements after enabling H2:

- page load time dropped by 5.5 seconds
- visually complete dropped by 10 seconds
- time to Interactive dropped by 2.5 seconds
- first contentful paint dropped by 4 seconds

So for users on legacy devices with a slow connection, H2 has improved the total page load time and improved perceived performance by 4 seconds, meaning a user can start reading and interacting with pages much sooner.

Medium specification mobile device on a medium connection

The team also tested a medium specification mobile device on a 3G connection. The tests showed the following improvements after enabling H2:

- page load time dropped by 1100 milliseconds
- visually complete dropped by 500 milliseconds
- time to interactive dropped by 600 milliseconds
- first contentful paint dropped by 900 milliseconds

Although not such a dramatic improvement as seen on low specification devices, it’s still quite a considerable improvement. This is especially important considering Google research found that 53% of mobile website visitors will leave a webpage if it does not load within 3 seconds. This change allows the first content to be rendered to the page just under this cut-off point.

High specification desktop device on a fast wired connection

The team also tested what connecting to GOV.UK would be like with a cable connection (5Mbps down and 1Mbps up) on a desktop computer using the Chrome browser. The tests showed the following improvements after enabling H2:

- page load time dropped by 100 milliseconds
- visually complete dropped by 200 milliseconds
- time to interactive dropped by 100 milliseconds
- first contentful paint dropped by 130 milliseconds

An improvement of 100 milliseconds does not sound like a huge amount, but when you consider the pages are loading in less than a second, that’s a 10% improvement.

What’s next for web performance on GOV.UK

The frontend team at GDS has more changes in the pipeline for web performance on GOV.UK.

The team is gradually rolling out updates to move the frontend from its legacy codebase to the GOV.UK Design System. This offers a new optimised web font, as well as more optimised CSS and JavaScript. This should reduce the amount of data sent over the network and speed up load times.