Difference between HTTP1.1 vs HTTP2

Task From JavaScript - Day -1 : Introduction to Browser & web

Submitted By

Yeswanth R

Introduction:

HTTP/1.1

Developed by Timothy Berners-Lee in 1989 as a communication standard for the World Wide Web, HTTP is a top-level application protocol that exchanges information between a client computer and a local or remote web server. In this process, a client sends a text-based request to a server by calling a method like GET or POST. In response, the server sends a resource like an HTML page back to the client.

For example, let's say you are visiting a website at the domain www.example.com. When you navigate to this URL, the web browser on your computer sends an HTTP request in the form of a text-based message, similar to the one shown here:

GET /index.html HTTP/1.1

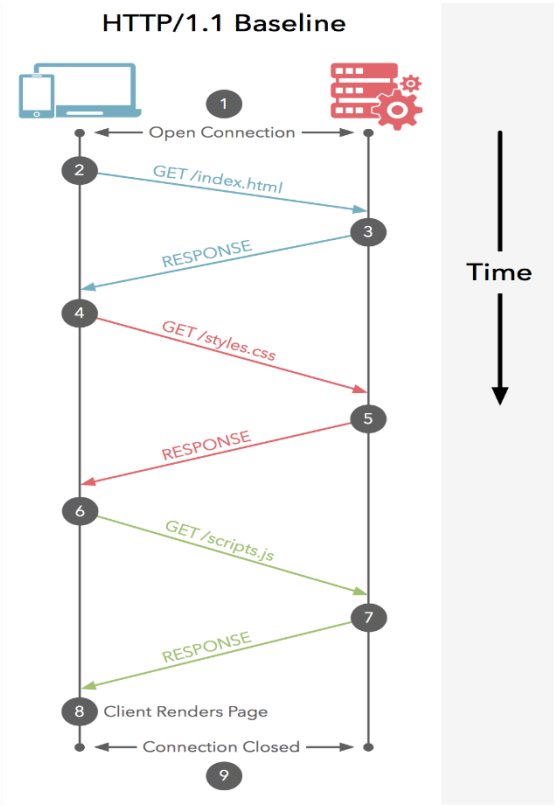
Host: www.example.com

This request uses the GET method, which asks for data from the host server listed after Host:. In response to this request, the example.com web server returns an HTML page to the requesting client, in addition to any

images, stylesheets, or other resources called for in the HTML.

Note that not all of the resources are returned to the client in the first call for data. The requests and responses will go back and forth between the server and client until the web browser has received all the resources necessary to render the contents of the HTML page on your screen.

You can think of this exchange of requests and responses as a single application layer of the internet protocol stack, sitting on top of the transfer layer (usually using the Transmission Control Protocol, or TCP) and networking layers (using the Internet Protocol, or IP)

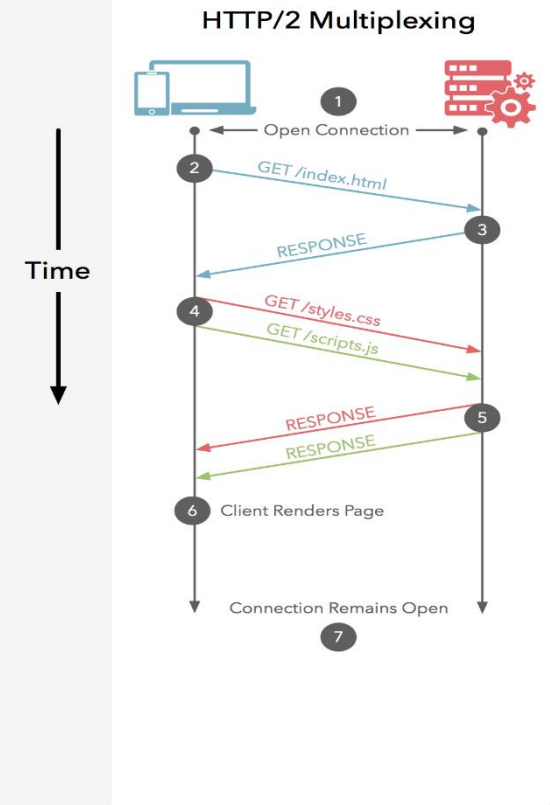


HTTP/2

HTTP/2 began as the SPDY protocol, developed primarily at Google with the intention of reducing web page load latency by using techniques such as compression, multiplexing, and prioritization. This protocol served as a template for HTTP/2 when the Hypertext Transfer Protocol working group http bis of the IETF (Internet Engineering Task Force) put the standard together, culminating in the publication of HTTP/2 in May 2015. From the beginning, many browsers supported this standardization effort, including Chrome, Opera, Internet Explorer, and Safari. Due in part to this browser support, there has been a significant adoption rate of the protocol since 2015, with especially high rates among new sites.

From a technical point of view, one of the most significant features that distinguishes HTTP/1.1 and HTTP/2 is the binary framing layer, which can be thought of as a part of the application layer in the internet protocol stack. As opposed to HTTP/1.1, which keeps all requests and responses in plain text format, HTTP/2 uses the binary framing layer to encapsulate all messages in binary format, while still maintaining HTTP semantics, such as verbs, methods, and headers. An application level API would still create messages in the conventional HTTP formats, but the underlying layer would then convert these messages into binary. This ensures that web applications created before HTTP/2 can continue functioning as normal when interacting with the new protocol.

The conversion of messages into binary allows HTTP/2 to try new approaches to data delivery not available in HTTP/1.1, a contrast that is at the root of the practical differences between the two protocols. The next section will take a look at the delivery model of HTTP/1.1, followed by what new models are made possible by HTTP/2.



What is prioritization?

In the context of web performance, prioritization refers to the order in which pieces of content are loaded. Suppose a user visits a news website and navigates to an article. Should the photo at the top of the article load first? Should the text of the article load first? Should the banner ads load first?

Prioritization affects a webpage's load time. For example, certain resources, like large JavaScript files, may block the rest of the page from loading if they have to load first. More of the page can load at once if these render-blocking resources load last.

In addition, the order in which these page resources load affects how the user perceives page load time. If only behind-the-scenes content (like a CSS file) or content the user can't see immediately (like banner ads at the bottom of the page) loads first, the user will think the page is not loading at all. If the content that's most important to the user loads first, such as the image at the top of the page, then the user will perceive the page as loading faster.

How does prioritization in HTTP/2 affect performance?

In HTTP/2, developers have hands-on, detailed control over prioritization. This allows them to maximize perceived and actual page load speed to a degree that was not possible in HTTP/1.1.

HTTP/2 offers a feature called weighted prioritization. This allows developers to decide which page resources will load first, every time. In HTTP/2, when a client makes a request for a webpage, the server sends several streams of data to the client at once, instead of sending one thing after another. This method of data delivery is known as multiplexing. Developers can assign each of these data streams a different weighted value, and the value tells the client which data stream to render first.

Imagine that Alice wants to read a novel that her friend Bob wrote, but both Alice and Bob only communicate through the regular mail. Alice sends a letter to Bob and asks Bob to send her his novel. Bob decides to send the novel HTTP/1.1-style: He mails one chapter at a time, and he only mails the next chapter after receiving a reply letter from Alice confirming that she received the previous chapter. Using this method of content delivery, it takes Alice many weeks to read Bob's novel.

Now imagine that Bob decides to send Alice his novel HTTP/2-style: In this case, he sends each chapter of the novel separately (to stay within the postal service's size limits) but all at the same time. He also numbers each chapter: Chapter 1, Chapter 2, etc. Now, Alice receives the novel all at once and can assemble it in the correct order on her own time. If a chapter is missing, she may send a quick reply asking for that specific chapter, but otherwise the process is complete, and Alice can read the novel in just a few days.

In HTTP/2, data is sent all at once, much like Bob when he sends Alice multiple chapters at once. And just like Bob, developers get to number the chapters in HTTP/2. They can decide if the text of a webpage loads first, or the CSS files, or the JavaScript, or whatever they feel is most important for the user experience.

What are the other differences between HTTP/2 and HTTP/1.1 that impact performance?

Multiplexing: HTTP/1.1 loads resources one after the other, so if one resource cannot be loaded, it blocks all the other resources behind it. In contrast, HTTP/2 is able to use a single TCP connection to send multiple streams of data at once so that no one resource blocks any other resource. HTTP/2 does this by splitting data into binary-code messages and numbering these messages so that the client knows which stream each binary message belongs to.

Server push: Typically, a server only serves content to a client device if the client asks for it. However, this approach is not always practical for modern webpages, which often involve several dozen separate resources that the client must request. HTTP/2 solves this problem by allowing a server to "push" content to a client before the client asks for it. The server also sends a message letting the client know what pushed content to expect – like if Bob had sent Alice a Table of Contents of his novel before sending the whole thing.

Header compression: Small files load more quickly than large ones. To speed up web performance, both HTTP/1.1 and HTTP/2 compress HTTP messages to make them smaller. However, HTTP/2 uses a more advanced compression method called HPACK that eliminates redundant information in HTTP header packets. This eliminates a few bytes from every HTTP packet. Given the volume of HTTP packets involved in loading even a single webpage, those bytes add up quickly, resulting in faster loading.

The HTTP/1.1 To HTTP/2 Transition

The breaking point in HTTP/1.1 was reached well before the 2015 introduction of HTTP/2. In fact, Google was working on it's own replacement for HTTP/1.1 since the early 2010s, called SPDY (pronounced "speedy"). This protocol used the existing infrastructure built for HTTP/1.1, but modified how the requests worked over the infrastructure. SPDY used multiplexing to download multiple resources efficiently over a single connection, and could be "back-ported" to existing applications with a translation layer.

It makes sense that Google would take the lead on this, as they had been developing increasingly complex web-based applications that operated more like desktop applications than websites, like GMail and Google Apps. In fact, the SPDY protocol was so well-designed, w3 used it as the basis for HTTP/2.

So, in 2015, w3 officially adopted the HTTP/2 specification based on SPDY, and all major browsers began supporting the protocol.

Why HTTP/2 Is Faster Than HTTP/1.1

Today, the concept of "webpages" is anachronistic, in much the same way as "videotaping" something with your smartphone. Modern websites function much more like applications, with a constant two-way stream of data being an essential part of their functionality.

For example, when you're typing something in a Google Doc, as I am typing this article right now, every keystroke sends data to Google's servers. Google's servers process that data, and then send updates back to your browser with the text you've typed, along with other helpful information like suggestions, the last edit status of the document, and much more. Over HTTP/1.1, each of your keypresses would initiate a new connection to the server, to send each character you typed over the wire. Then, your browser would have to constantly "ping" Google's server to see if the status of the document changed, to add the character to the screen you're looking at. That's a boatload of connections, and each one takes precious time.

With HTTP/2 though, it's essentially a constant two-way stream over a single connection. Google's server is always listening for data coming from your browser, and your browser is always listening for data to come back from Google. There's no more send data, wait for response, update the screen, send more data, wait for a response, etc. Instead, everything happens in real-time. In this way, a

web “page” like a Google Doc can update itself so frequently as to feel like a native application on your computer.

These are the high-level differences between HTTP1 and HTTP2:

- HTTP2 is binary, instead of textual
- HTTP2 is fully multiplexed, instead of ordered and blocking
- HTTP2 can, therefore, use one connection for parallelism
- HTP2 uses header compression to reduce overhead
- HTTP2 allows servers to “push” responses proactively into client caches

Why You Should Switch To HTTP/2

When Greenlane performs technical audits of websites, switching from HTTP/1.1 to HTTP/2 is one of the most common, and impactful recommendations that we make. We can often dramatically improve the performance of a website, with very little cost or time expenditure.

Perhaps your business does not have any dynamic components. Perhaps your pages are primarily static compositions of text and images, just like the old days of the web. This is great – in your case, HTTP/2 is extremely low-hanging fruit! Simply upgrading your hosting to use HTTP/2 will dramatically speed up the delivery of your content to users, as all of your page’s content will come down the pipe on one connection instead of over multiple connections.

It’s so important, that this is likely a pass/fail test in Google’s algorithm – if we use Google’s Lighthouse to test a page, the tool doesn’t even offer recommendations to improve your HTTP/1.1 delivery – it simply says to upgrade to HTTP/2.

Conclusion

In conclusion we can see that HTTP/2 is better in every imaginable way possible. As it doesn’t even take that many steps to implement is a cherry on top for future sites and it is much easier to update from the previous version makes it a no brainer.

Reference

Dzone

CheapSSLSecurity

Cloudflare

Greenlane

DigitalOcean