# HTTP/2 – WHAT CAN BE GAINED?

HyperText Transfer Protocol (HTTP for short) is the means for data communication between clients and web servers across the World Wide Web. This protocol was initially proposed by Tim Berners-Lee in 1989 and released in 1991.

HTTP was a [stateless protocol](https://stackoverflow.com/questions/13200152/why-is-it-said-that-http-is-a-stateless-protocol) implemented to allow for faster content delivery or requests over a single HTTP connection without the need to re-establish connections. Each request was isolated and resolved independently; therefore, requests had to contain sufficient detail for the server to verify and deliver the right content back to the client. HTTP works on a client-server model where a client makes a **request** to view a website/ webpage and a server replies with a **response** message (over a HTTP connection). To elaborate, the client in this case could be anyone typing ‘www.ImportantWebsite.com’ (or any other non fictitious website name) into their favorite web browser and making a request to view this website. The server (the webpage hosting or storing the [HTML](https://html.com/) content) receives the request and if the status of the webpage is ‘OK’ the content is provided . Here ‘OK’ is represented by the 200 status code provided by the server in response header. For more information on response status codes, check out [Mozilla’s](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status) documentation on the topic . As a disclaimer, this article will assume HTTP with TCP (Transmission Control Protocol - data transmission over the web) connection over a UDP connection is used. UDP works similarly to TCP; however, is less popular due to its lack of error – checking and resulting large latency. Feel free to check out “[What’s the Difference Between TCP and UDP?”](https://www.howtogeek.com/190014/htg-explains-what-is-the-difference-between-tcp-and-udp/) for more information about these two protocols .

HTTP/1.x has served the web quite well in the past; however, with the increasing size of content rich Web pages and user client connections the technology has become outdated. Enter HTTP/2, developed in 2015 and largely based on [Google’s SPDY protocols](https://www.chromium.org/spdy/spdy-whitepaper) . This version was considered the largest, ground-breaking implementation since HTTP/1.1 (1999) and has grown in popularity ever since.

## Why is HTTP/2 a worthy replacement?

There are several reasons as to why making the shift from HTTP/1x to HTTP/2 can prove beneficial to users and developers. Improved performance and data transfer, easier implementation and lower latency leads to users receiving richer content a little bit faster, developers have more options available to them in terms of what and how they can send data as well as a relatively safer transfer of data.

### Performance

Content rich, resource hungry Web pages with dynamic page renders make it harder to work effectively with HTTP/1.x versions.

HTTP/1.x only allows for a single concurrent or outstanding request per TCP connection at one time. There were complex solutions implemented to deal with these issues; however large file size, poor connections, incompatible servers or declining processing capabilities (slower responses) still caused delays. These issues are more pronounced with today’s content driven Web pages where page content may be filled by making multiple requests to several servers at once. Figure 1 shows how a website with just two pictures on its homepage may make this request. For HTTP/1 in Figure 1, the request (REQ.) and response (RES.) to the dog server must be done first before resolving with the cat server. In the case of larger Web pages with exponentially larger content requirements it becomes more challenging to maintain. Content rendering solutions such as,

1. [Domain Sharding](https://www.keycdn.com/support/domain-sharding) – Causes more browse/ server load as it creates multiple sub domains to download content and therefore bypasses the server allocated active connections per domain
2. [Data URIs](https://blog.teamtreehouse.com/using-data-uris-speed-website) – Base64 encoded string representation for a file that can contain embedded HTML (webpage content) and/ or CSS code (styling) which is then decoded and converted to a webpage for the client. Some issues with this include added server complexity and URI’s are not cached therefore web page reloads take up more system resources (more evident in mobile browsers).
3. Resource Splitting – storing resources over multiple hosts
4. [CSS and JavaScript concatenation](https://stackoverflow.com/questions/28630108/does-minifying-and-concatenating-js-css-files-and-using-sprites-for-images-stil) – The combining of files reduces the overhead requirements of ending TCP/IP sessions per file download.

There were many more solutions (not mentioned in this article) being implemented for HTTP/1.x yet most of them still had issues that HTTP/2 eliminate or mitigate , , .

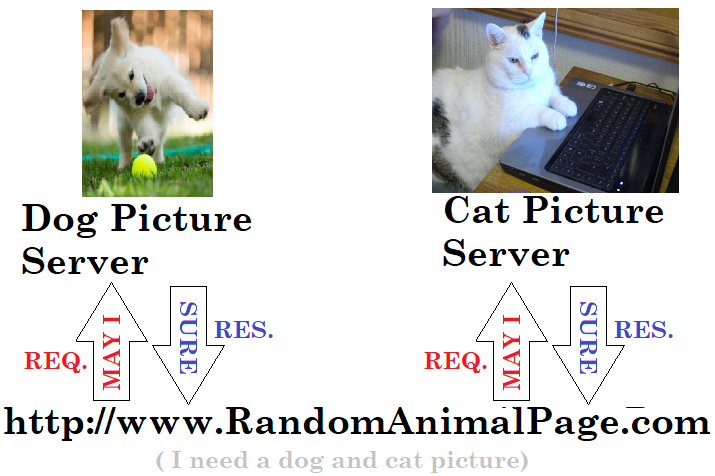


Figure : An example of a site making a request to get pictures from 2 different servers

HTTP/2 allows for [data compression and minification](https://stackoverflow.com/questions/28630108/does-minifying-and-concatenating-js-css-files-and-using-sprites-for-images-stil) as well as bi-directional connections allowing for more requests and efficient data transfers. It allows for multiple request 🡪 response (req 🡪 res) cycles simultaneously, ensuring that delays in one req 🡪 res cycle does not block other cycles. Referring to the previously mentioned example (Figure 1) requests to the dog and cat servers/ hosts can be made independently. Additionally, most of the content rendering solutions mentioned above can also be employed for HTTP/2 with greatly reduced drawbacks. For example,

1. [Bandwidth demand](https://www.digitalocean.com/community/tutorials/http-1-1-vs-http-2-what-s-the-difference) – is greatly reduced due to minification and data compression (compression algorithms used in content body and message headers respectively) with thus reducing the TCP/IP session cycle loads and related resource overheads.
2. [Header compression](https://www.digitalocean.com/community/tutorials/http-1-1-vs-http-2-what-s-the-difference) (HPACK) – In addition to compressing message headers (mentioned above), HPACK removes unnecessary or identical headers. The client (having HTTP/2 support) will have access to, and can reconstruct any of the previously resolved headers within a communication session thereby reducing bandwidth throughput and increasing content delivery speed. Due to HTTPs stateless nature HTTP/1.x would require multiple requests to be sent over web (a large contributor to the bottleneck in the webpage rendering process) for the same headers.
3. [Connection Coalescing](https://www.perimeterx.com/blog/http2/) – HTTP/2 can combine multiple requests (per IP address and hostname) for a domain into one connection. Figure 2 shows a simplified diagram of how requests are coalesced. A major benefit of this is a singular TCP/IP connection is required per multiplexed streams (req. 🡪 res. cycle) rather than HTTP/1.x’s multiple TCP connections. The importance of this feature is made more evident when dealing with HTTPS and secure data transfers. Before a response can be sent a [TLS handshake](https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/) must be established, the domain’s security certification and premaster secret must be authentified and decrypted. This adds additional steps before a response can be resolved with content.

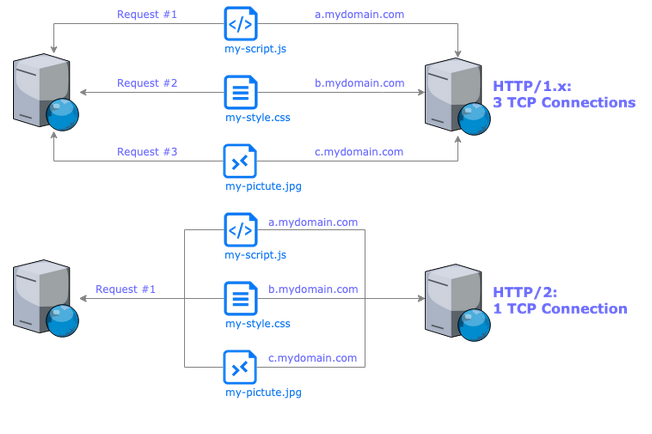


Figure : [Connection Coalescing](https://www.perimeterx.com/blog/http2/) in HTTP/2 vs. HTTP/1.x individual requests

1. [Server Push](https://medium.com/@factoryhr/http-2-the-difference-between-http-1-1-benefits-and-how-to-use-it-38094fa0e95b) – HTTP/2 allows for the optional sending of content without the client requesting for it specifically. Figure 3 shows a simplified version of how related content may be pushed. This content is cacheable and can be used across multiple pages that use aspects of the same downloaded content (without the need for another request). Server pushing requires client consent and can be customized as per client requirements thus making this feature optional. Server pushing can reduce network latency because it serves to reduce the total number of request and response cycles.

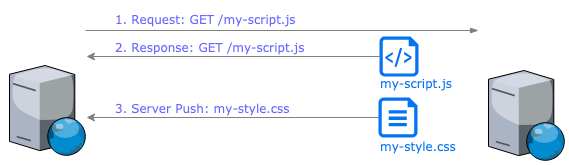


Figure : [Server Push](https://www.perimeterx.com/blog/http2/) feature

1. [Stream prioritization](https://www.perimeterx.com/blog/http2/) – Multiplexed content streams can be prioritized by a client. If the server accepts this prioritization then important content (deemed by the client in their prioritization request) can be loaded first and all other content can be loaded later.

### Security and Simplicity

HTTP/1.x utilizes text format commands during request, response cycles putting more computing load on the network and server to parse data into binary format. Browsers that utilize HTTP/2 (which should include all the major ones) convert text commands into binary before sending to a server endpoint, thus reducing net transfer payload, command execution complexity (as it is easier to transfer and compose packets of binary data rather than text) and overall latency. In addition, the use of [binary protocols](https://www.perimeterx.com/blog/http2/) decreases the security risks HTTP/2 faces compared to its predecessor and is easier to encrypt.

### Compatibility, Coverage and Support

Since its release in 2015 most major browsers released support packages in the same year or prior to release. So unless your computer runs on Windows ‘BronzeAge’ NT or Mac OS Y ‘SaberToothedTiger’ is still alive and is yet to learn of the internet, your computer and browser should be supported. HTTP/2 was also designed to be backward compatible with older version thus maintaining the preexisting clientele , , .

# So why would I care?

To summarize the benefits

* **If you are a user** - content is delivered to you quicker, safer and with more control over what is downloaded or requested.
* **If you are a developer** – your content is reaching your clientele faster, you have less security concerns to in terms of binary protocol resilience, lower complexity as content can be multiplexed streams to the user, content is still accessible to those without HTTP/2 support (due to its backwards compatibility), increased network utilization and decreased server loads (due to reduced request and response cycles).

### these are all throretical benefits, any practical ones

Checkout this [website](https://www.perimeterx.com/blog/http2/) where the author requests 100 small images from a server (with node.js) using a script for HTTP1 and HTTP2. The HTTP/2 script uses a server push method with a helper method to get all the necessary files and render the image. This script utilizes a single TCP connection with multiplexed streams. The [scripts](https://github.com/PerimeterX/node-http2-server-push/tree/master/native) utilize the ‘http1’ and ‘http2’ libraries in Native Node.js. The results show an 85% content render advantage over HTTP1 . There are many more websites dedicated to visually depicting the benefits of HTTP/2 vs. HTTP/1.x; however, this website provides access to the code used. One need only fork and clone the script to test the speed advantages of HTTP/2 .

## Conclusion

Implementing HTTP/2 will greatly improve website performance, user experience and reduce over network footprint/ latency. Although HTTP/2 is backwards compatible to fully utilize its potential, scripts must be written to account for the added capabilities.

HTTP/3 has also been announced recently; however, without sufficient implementation and evidence of its benefits (due to it still being in the experimental phase) it was not considered for this article.

# Works Cited

ABCOM. (2019, Jan 30). *HTTP/1.1 vs HTTP/2: What's the Difference?* Retrieved Dec 07, 2019, from digitalocean: https://www.digitalocean.com/community/tutorials/http-1-1-vs-http-2-what-s-the-difference

cHao. (2012, Nov 2). *Why is it said that “HTTP is a stateless protocol”*. Retrieved Dec 11, 2019, from stackoverflow.com: https://stackoverflow.com/questions/13200152/why-is-it-said-that-http-is-a-stateless-protocol

Factory.hr. (2018, Jul 26). *HTTP/2: the difference between HTTP/1.1, benefits and how to use it*. Retrieved Dec 07, 2019, from medium.com: https://medium.com/@factoryhr/http-2-the-difference-between-http-1-1-benefits-and-how-to-use-it-38094fa0e95b

Hoffman, C. (2017, Jul 03). *What’s the Difference Between TCP and UDP?* Retrieved Dec 06, 2019, from howtogeek: https://www.howtogeek.com/190014/htg-explains-what-is-the-difference-between-tcp-and-udp/

HTML. (2019, Nov 27). *HTML For Beginners & Experts Made Simple: Learn HTML & CSS Today*. Retrieved Dec 06, 2019, from HTML.com: https://html.com/

Mozilla. (2019, Oct 29). *HTTP response status codes*. Retrieved Dec 06, 2019, from Mozilla: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Shmuel, L. (2019, Feb 20). *HTTP/2 - The Reasons, The Features And The Node.js Support*. Retrieved Dec 07, 2019, from perimeterx: https://www.perimeterx.com/blog/http2/

T.C.P. (n.d.). *SPDY: An experimental protocol for a faster web*. Retrieved Dec 06, 2019, from The Chromium Projects: https://www.chromium.org/spdy/spdy-whitepaper