HTTP/2: the difference between HTTP/1.1, benefits and how to use it

Factory.hr
Jul 26, 2018 · 7 min read

Details on how to setup HTTP/2 for ubuntu and Server Push usage are also available on Factory Github.

What is HTTP?

If you are new to this theme, continue to read this block, but if you're familiar with this, skip this paragraph and go straight forward to the next one.

For those that are new to this theme, Hypertext Transfer Protocol (HTTP) is an application protocol that is, currently, **the foundation** of data communication for the World Wide Web.

HTTP is based on the Client/Server model. Client/Server model can be explained as two computers, Client (receiver of service) and Server (provider of service) that are communicating via requests and responses.

A simple and abstract example would be a **restaurant guest and a waiter**. The guest (**Client**) asks (**sends request**) waiter (**Server**) for a meal, then the waiter gets the meal from the restaurant chef (**your application logic**) and brings the meal to the guest.

This is a very simplistic example, but it is also the one that will help you understand the concept.

There are many more interesting HTTP concepts and utilities to discuss, but the star of this post is (not enough) famous **HTTP/2**.

What is HTTP/2?

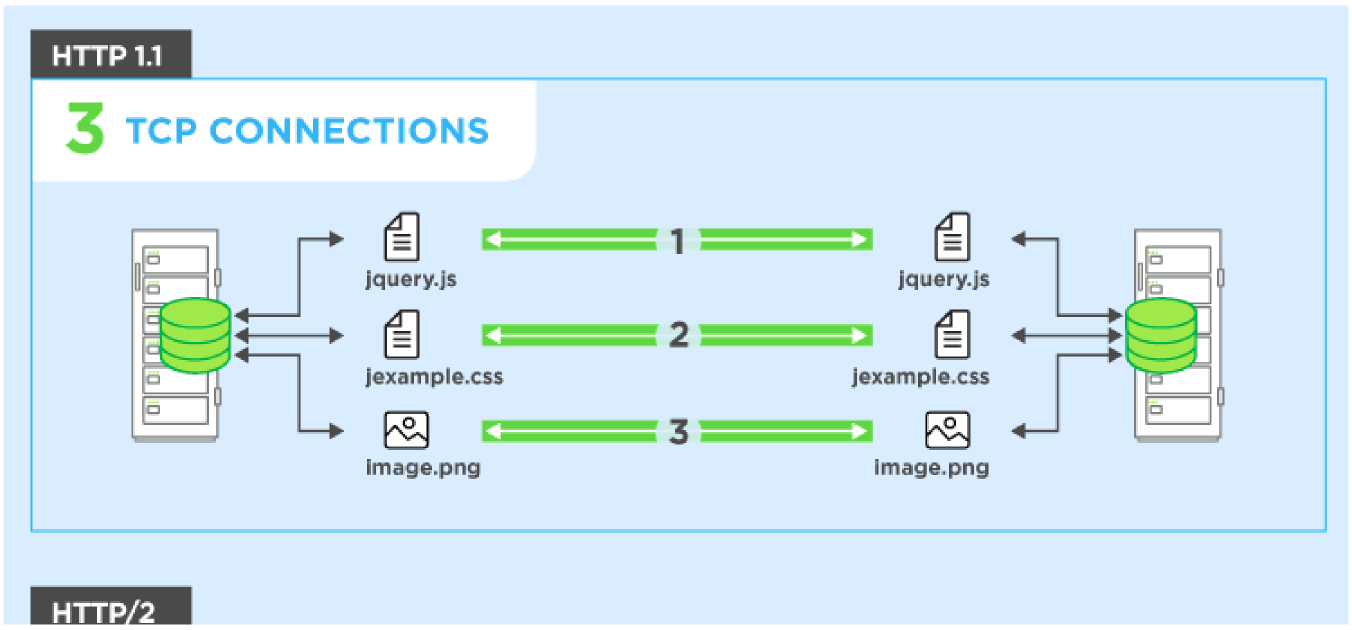
In 2015, Internet Engineering Task Force (IETF) release HTTP/2, the second major version of the most useful internet protocol, HTTP. It was derived from the earlier experimental SPDY protocol.

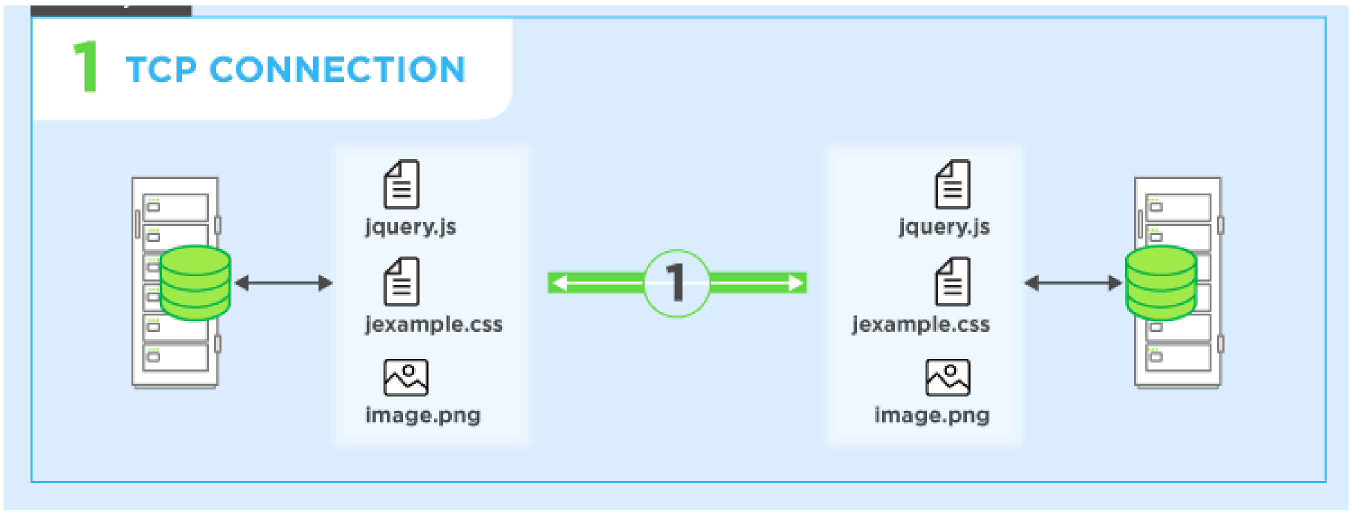
Main goals of developing HTTP/2 was:

- Protocol negotiation mechanism — protocol electing, eg. HTTP/1.1, HTTP/2 or other.
- High-level compatibility with HTTP/1.1 — methods, status codes, URIs and header fields.
- Page load speed improvements trough:
- Compression of request headers
- Binary protocol
- HTTP/2 Server Push
- Request multiplexing over a single TCP connection
- Request pipelining
- HOL blocking (Head-of-line) — Package blocking

Request multiplexing

HTTP/2 can send **multiple requests** for data in parallel over a **single** TCP connection. This is **the most advanced feature** of the HTTP/2 protocol because it **allows you to download web files asynchronously from one server**. Most modern browsers limit TCP connections to one server.



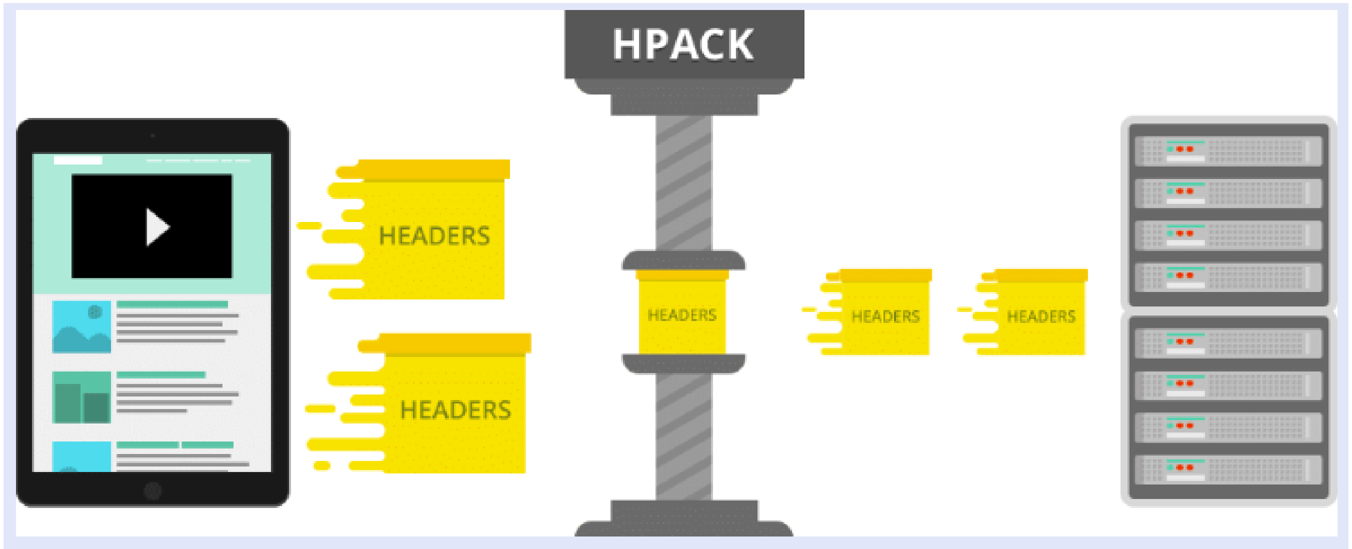


This reduces additional round trip time (RTT), **making your website load faster** without any optimization, and makes domain sharding unnecessary.

Header compression

HTTP/2 compress a large number of redundant header frames. It uses the HPACK specification as a simple and secure approach to header compression. Both client and server maintain a list of headers used in previous client-server requests.

HPACK compresses the individual value of each header before it is transferred to the server, which then looks up the encoded information in a list of previously transferred header values to reconstruct the full header information.

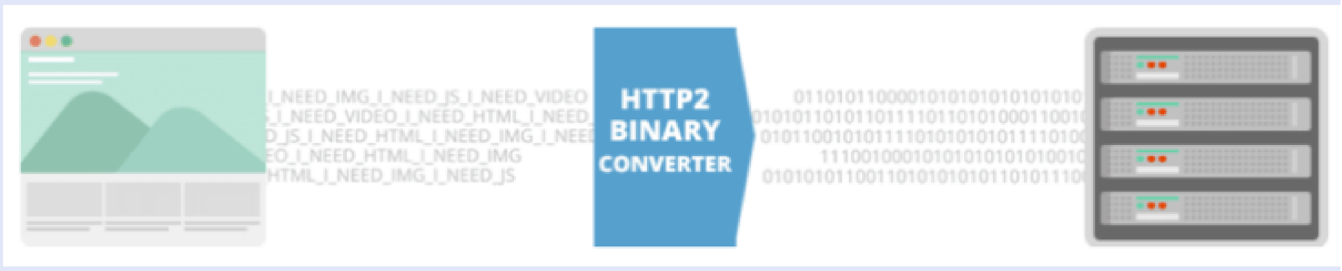


HTTP/2 HPACK compression

Binary protocol

The latest HTTP version has evolved significantly in terms of capabilities and attributes such as transforming from a text protocol to a binary protocol. HTTP1.x used to process text commands to complete request-response cycles. HTTP/2 will use binary commands (in 1s and 0s) to execute the same tasks. This attribute eases complications with framing and simplifies implementation of commands that were confusingly intermixed due to commands containing text and optional spaces.

Browsers using HTTP/2 implementation will convert the same text commands into binary before transmitting it over the network.



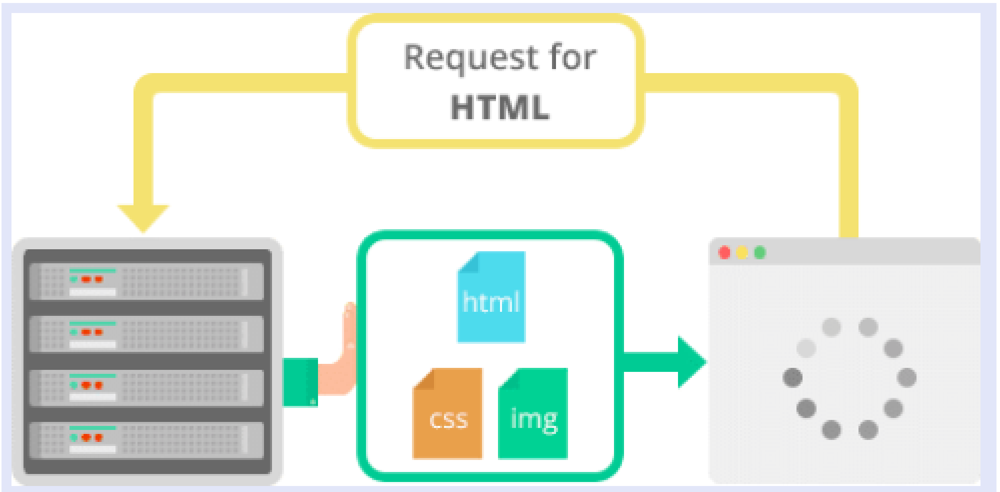
Binary protocols

Benefits:

- Low overhead in parsing data — a critical value proposition in HTTP/2 vs HTTP1.
- Less prone to errors.
- Lighter network footprint.
- Effective network resource utilization.
- Eliminating security concerns associated with the textual nature of HTTP1.x such as response splitting attacks.
- Enables other capabilities of the HTTP/2 including compression, multiplexing, prioritization, flow control and effective handling of TLS.
- Compact representation of commands for easier processing and implementation.
- Efficient and robust in terms of processing of data between client and server.
- Reduced network latency and improved throughput.

HTTP/2 Server Push

This capability allows the server to send additional cacheable information to the client that isn't requested but is anticipated in future requests. For example, if the client requests for the resource X and it is understood that the resource Y is referenced with the requested file, the server can choose to push Y along with X instead of waiting for an appropriate client request.



HTTP/2 push

Benefits:

- The client saves pushed resources in the cache.
- The client can reuse these cached resources across different pages.
- The server can multiplex pushed resources along with originally requested information within the same TCP connection.
- The server can prioritize pushed resources — a key performance differentiator in HTTP/2 vs HTTP1.
- The client can decline pushed resources to maintain an effective repository of cached resources or disable Server Push entirely.
- The client can also limit the number of pushed streams multiplexed concurrently.

If you remember the story about a guest in a restaurant and waiter, that would be an example

for HTTP/1.1 and HTTP/2 protocol with a slight difference. Imagine that waiters are TCP connections and you want to order your meal and a bottle of water. For HTTP/1.1 that would mean that you ask one waiter for your meal and another one for water, hence you would allocate two TCP connections. For HTTP/2 that would mean that you ask only one waiter for both, but he brings them separately. You only allocate one TCP connection and that will already result with lower server load, plus the server would have one extra free connection (waiter) for the next client (guest).

The real difference between HTTP/1.1 and HTTP/2 comes with server push example.

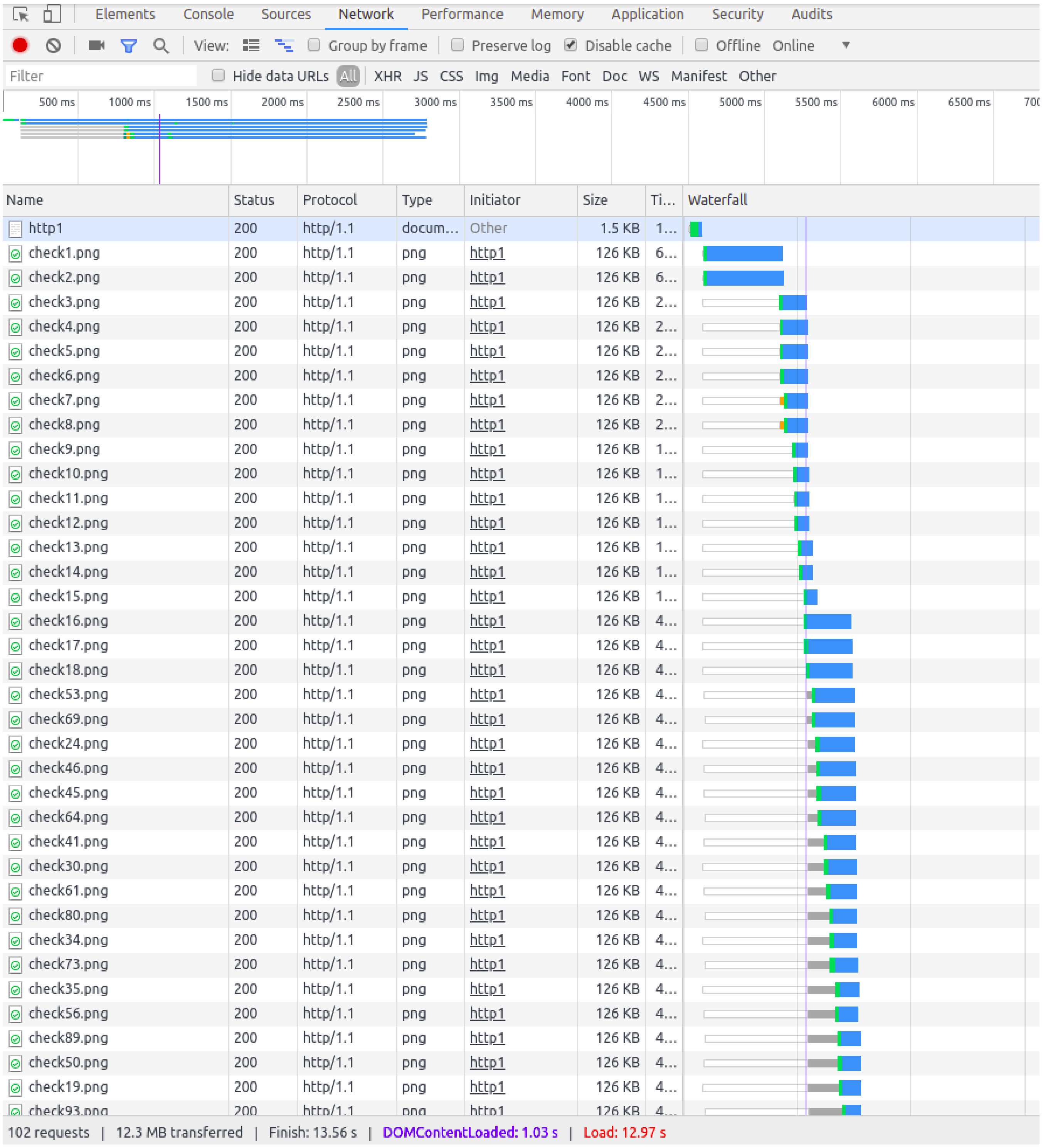
Imagine that the guest (Client) asks (sends request) waiter (Server) for a meal, then the waiter gets the meal from the restaurant chef (your application logic), but the waiter also thinks you would need a bottle of water so he brings that too with your meal. The end result of this would be only one TCP connection and only one request that will significantly lower the server load.

As a simple showcase of those mechanics, I made a simple page example.

What we have here is a simple page with 100 images of checks which I'll use to demonstrate HTTP/1.1, HTTP/2 AND HTTP/2 server push.

What is important to note in the picture above are number of requests, load time, protocol column, initiator column and waterfall diagram itself (we can see how requests are made through multiple batches, unfortunately, it is hard to see other data from it except TTFB and content download time; eg. resource scheduling and connection start time).

For HTTP/1.1:



☐ Group by frame ☐ Preserve log ☑ Disable cache ☐ Offline Online

☐ Hide data URLs All

Name	Status	Protocol	Type	Initiator	Size	Ti...	Waterfall
http1	200	http/1.1	docum...	Other	1.5 KB	1...	
check1.png	200	http/1.1	png	http1	126 KB	6...	
check2.png	200	http/1.1	png	http1	126 KB	6...	
check3.png	200	http/1.1	png	http1	126 KB	2...	
check4.png	200	http/1.1	png	http1	126 KB	2...	
check5.png	200	http/1.1	png	http1	126 KB	2...	
check6.png	200	http/1.1	png	http1	126 KB	2...	
check7.png	200	http/1.1	png	http1	126 KB	2...	
check8.png	200	http/1.1	png	http1	126 KB	2...	
check9.png	200	http/1.1	png	http1	126 KB	1...	
check10.png	200	http/1.1	png	http1	126 KB	1...	
check11.png	200	http/1.1	png	http1	126 KB	1...	
check12.png	200	http/1.1	png	http1	126 KB	1...	
check13.png	200	http/1.1	png	http1	126 KB	1...	
check14.png	200	http/1.1	png	http1	126 KB	1...	
check15.png	200	http/1.1	png	http1	126 KB	1...	
check16.png	200	http/1.1	png	http1	126 KB	4...	
check17.png	200	http/1.1	png	http1	126 KB	4...	
check18.png	200	http/1.1	png	http1	126 KB	4...	
check53.png	200	http/1.1	png	http1	126 KB	4...	
check69.png	200	http/1.1	png	http1	126 KB	4...	
check24.png	200	http/1.1	png	http1	126 KB	4...	
check46.png	200	http/1.1	png	http1	126 KB	4...	
check45.png	200	http/1.1	png	http1	126 KB	4...	
check64.png	200	http/1.1	png	http1	126 KB	4...	
check41.png	200	http/1.1	png	http1	126 KB	4...	
check30.png	200	http/1.1	png	http1	126 KB	4...	
check61.png	200	http/1.1	png	http1	126 KB	4...	
check80.png	200	http/1.1	png	http1	126 KB	4...	
check34.png	200	http/1.1	png	http1	126 KB	4...	
check73.png	200	http/1.1	png	http1	126 KB	4...	
check35.png	200	http/1.1	png	http1	126 KB	4...	
check56.png	200	http/1.1	png	http1	126 KB	4...	
check89.png	200	http/1.1	png	http1	126 KB	4...	
check50.png	200	http/1.1	png	http1	126 KB	4...	
check19.png	200	http/1.1	png	http1	126 KB	4...	
check93.png	200	http/1.1	png	http1	126 KB	4...	

102 requests | 12.3 MB transferred | Finish: 13.56 s | DOMContentLoaded: 1.03 s | Load: 12.97 s

Number of requests: 102

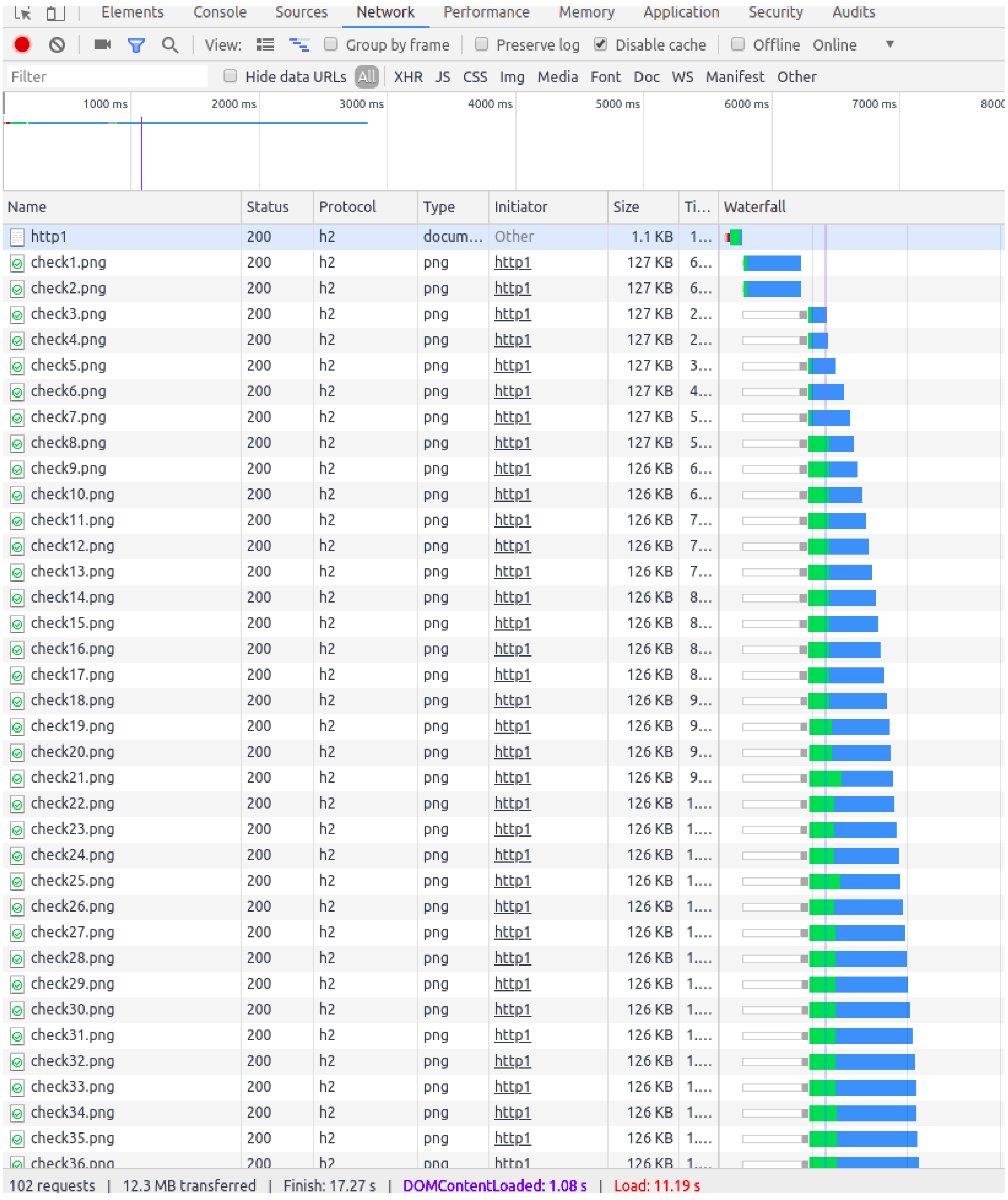
Load time: 12.97s

Protocol: “http/1.1”

Initiator column: Initiator of the first one is user/client and the rest of the requests are initiated by the response to client who realizes he needs some other resources (in this case, images).

Waterfall diagram: We can see how requests are made through multiple batches (TCP connections).

For HTTP/2:



[✓ Disable cache]

Name	Status	Protocol	Type	Initiator	Size	Ti...	Waterfall
http1	200	h2	docum...	Other	1.1 KB	1...	
check1.png	200	h2	png	http1	127 KB	6...	
check2.png	200	h2	png	http1	127 KB	6...	
check3.png	200	h2	png	http1	127 KB	2...	
check4.png	200	h2	png	http1	127 KB	2...	
check5.png	200	h2	png	http1	127 KB	3...	
check6.png	200	h2	png	http1	127 KB	4...	
check7.png	200	h2	png	http1	127 KB	5...	
check8.png	200	h2	png	http1	127 KB	5...	
check9.png	200	h2	png	http1	126 KB	6...	
check10.png	200	h2	png	http1	126 KB	6...	
check11.png	200	h2	png	http1	126 KB	7...	
check12.png	200	h2	png	http1	126 KB	7...	
check13.png	200	h2	png	http1	126 KB	7...	
check14.png	200	h2	png	http1	126 KB	8...	
check15.png	200	h2	png	http1	126 KB	8...	
check16.png	200	h2	png	http1	126 KB	8...	
check17.png	200	h2	png	http1	126 KB	8...	
check18.png	200	h2	png	http1	126 KB	9...	
check19.png	200	h2	png	http1	126 KB	9...	
check20.png	200	h2	png	http1	126 KB	9...	
check21.png	200	h2	png	http1	126 KB	9...	
check22.png	200	h2	png	http1	126 KB	1....	
check23.png	200	h2	png	http1	126 KB	1....	
check24.png	200	h2	png	http1	126 KB	1....	
check25.png	200	h2	png	http1	126 KB	1....	
check26.png	200	h2	png	http1	126 KB	1....	
check27.png	200	h2	png	http1	126 KB	1....	
check28.png	200	h2	png	http1	126 KB	1....	
check29.png	200	h2	png	http1	126 KB	1....	
check30.png	200	h2	png	http1	126 KB	1....	
check31.png	200	h2	png	http1	126 KB	1....	
check32.png	200	h2	png	http1	126 KB	1....	
check33.png	200	h2	png	http1	126 KB	1....	
check34.png	200	h2	png	http1	126 KB	1....	
check35.png	200	h2	png	http1	126 KB	1....	
check36.png	200	h2	png	http1	126 KB	1....	

102 requests | 12.3 MB transferred | Finish: 17.27 s | DOMContentLoaded: 1.08 s | Load: 11.19 s

Number of requests: 102

Load time: 11.19s

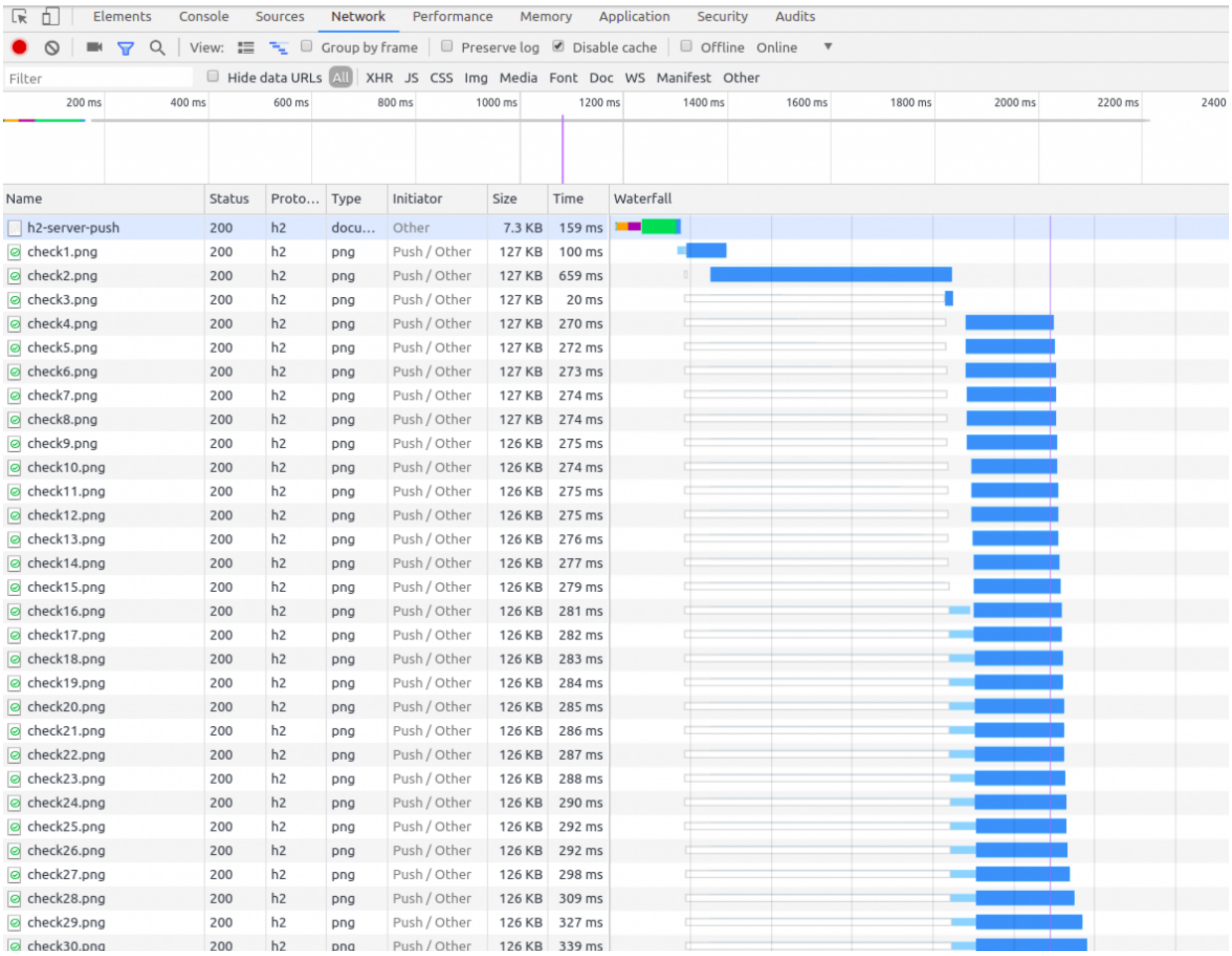
Protocol: “h2” (HTTP/2)

Initiator column: Initiator of the first one is user/client and the rest of the requests are initiated by response to client who realizes he needs some other resources (in this case, images).

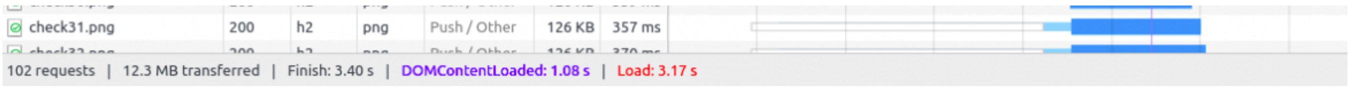
Waterfall diagram: We can see how requests are made through 2 batches (TCP connections).

Take note of the load time. In this case, it is a bit lower than the load time of HTTP/1.1 example but it doesn’t have to be always. This example shows the multiplexing of client requests.

HTTP/2 server push:



Name	Status	Proto...	Type	Initiator	Size	Time	Waterfall
h2-server-push	200	h2	docu...	Other	7.3 KB	159 ms	
check1.png	200	h2	png	Push / Other	127 KB	100 ms	
check2.png	200	h2	png	Push / Other	127 KB	659 ms	
check3.png	200	h2	png	Push / Other	127 KB	20 ms	
check4.png	200	h2	png	Push / Other	127 KB	270 ms	
check5.png	200	h2	png	Push / Other	127 KB	272 ms	
check6.png	200	h2	png	Push / Other	127 KB	273 ms	
check7.png	200	h2	png	Push / Other	127 KB	274 ms	
check8.png	200	h2	png	Push / Other	127 KB	274 ms	
check9.png	200	h2	png	Push / Other	126 KB	275 ms	
check10.png	200	h2	png	Push / Other	126 KB	274 ms	
check11.png	200	h2	png	Push / Other	126 KB	275 ms	
check12.png	200	h2	png	Push / Other	126 KB	275 ms	
check13.png	200	h2	png	Push / Other	126 KB	276 ms	
check14.png	200	h2	png	Push / Other	126 KB	277 ms	
check15.png	200	h2	png	Push / Other	126 KB	279 ms	
check16.png	200	h2	png	Push / Other	126 KB	281 ms	
check17.png	200	h2	png	Push / Other	126 KB	282 ms	
check18.png	200	h2	png	Push / Other	126 KB	283 ms	
check19.png	200	h2	png	Push / Other	126 KB	284 ms	
check20.png	200	h2	png	Push / Other	126 KB	285 ms	
check21.png	200	h2	png	Push / Other	126 KB	286 ms	
check22.png	200	h2	png	Push / Other	126 KB	287 ms	
check23.png	200	h2	png	Push / Other	126 KB	288 ms	
check24.png	200	h2	png	Push / Other	126 KB	290 ms	
check25.png	200	h2	png	Push / Other	126 KB	292 ms	
check26.png	200	h2	png	Push / Other	126 KB	292 ms	
check27.png	200	h2	png	Push / Other	126 KB	298 ms	
check28.png	200	h2	png	Push / Other	126 KB	309 ms	
check29.png	200	h2	png	Push / Other	126 KB	327 ms	
check30.png	200	h2	png	Push / Other	126 KB	339 ms	



check31.png 200 h2 png Push / Other 126 KB 357 ms

102 requests | 12.3 MB transferred | Finish: 3.40 s | DOMContentLoaded: 1.08 s | Load: 3.17 s

Number of requests: 102

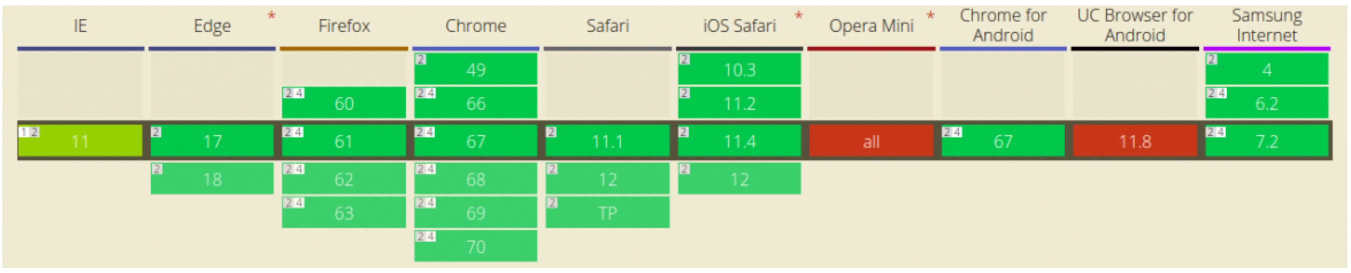
Load time: 3.17 s

Protocol: “h2” (HTTP/2)

Initiator column: Initiator of the first one is user/client and the rest of requests are initiated by the push of the server (virtually one request/response cycle).

Waterfall diagram: We can see how requests are made through 1 batch (1 TCP connection).

Browser Compatibility



IE	Edge *	Firefox	Chrome	Safari	iOS Safari *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
			[2] 49		[2] 10.3				[2] 4
		[2 4] 60	[2 4] 66		[2] 11.2				[2 4] 6.2
[1 2] 11	[2] 17	[2 4] 61	[2 4] 67	[2] 11.1	[2] 11.4	all	[2 4] 67	11.8	[2 4] 7.2
	[2] 18	[2 4] 62	[2 4] 68	[2] 12	[2] 12				
		[2 4] 63	[2 4] 69	[2] TP					
			[2 4] 70						

Most of the modern browser fully support HTTP/2 protocol with an exception (red) of Opera mini (all versions) and UC Browser for Android. There are also the ones that have partial support (light green) like IE11.

You can find more details on browser support on this link https://caniuse.com/#feat=http2

Use HTTP/2 and speed up your site

HTTP/2 provides us with **many new mechanics** that will mitigate HTTP/1.1 issues and ones that **will boost your web page performance**. Currently, it is widely supported by web clients so its implementation is painless. Although implementation of HTTP/2 protocol is easy **you should have in mind** that with it you will probably have to change the mechanics (like serving assets to the client) to use the full potential of this protocol.

If you have some other information and experience about this theme feel free to share it and also share this article if you think it would help out your fellow colleagues or friends.

Read more from our team @ Factory blog

Web Development   Http2   Speed Up Your Site   How To   Backend Development