HTTP/2

MAKING DRUPAL EVEN FASTER

THE NEW PROTOCOL FOR THE WEB

@mattiasgeniar

Drupal Camp Leuven 2015

WHAT'S THIS TALK ABOUT?

- History: what is HTTP/1.1
- How does HTTP work
- What does HTTP/2 do
- Benefits of HTTP/2 over HTTP/1.1
- Disadvantages of HTTP/2
- Performance comparisons
- Conclusion

WHO AM I?

- Mattias Geniar
- System Engineer / Support Lead @ Nucleus.be
- Former dev, mostly Ops now
- Strong advocate of #DevOps
- Blogger at https://ma.ttias.be/http2

CRON.WEEKLY

- Weekly newsletter with linux & open source content
- www.cronweekly.com

HISTORY: WHAT IS HTTP/1.1

- Client/server protocol
- Relies on requests & responses
- Defacto standard since 1997
- "Meta data" for requests hidden in HTTP headers
- Without HTTP, there is no web.
- Simple protocol, plain text. Easy to read, hard to parse.

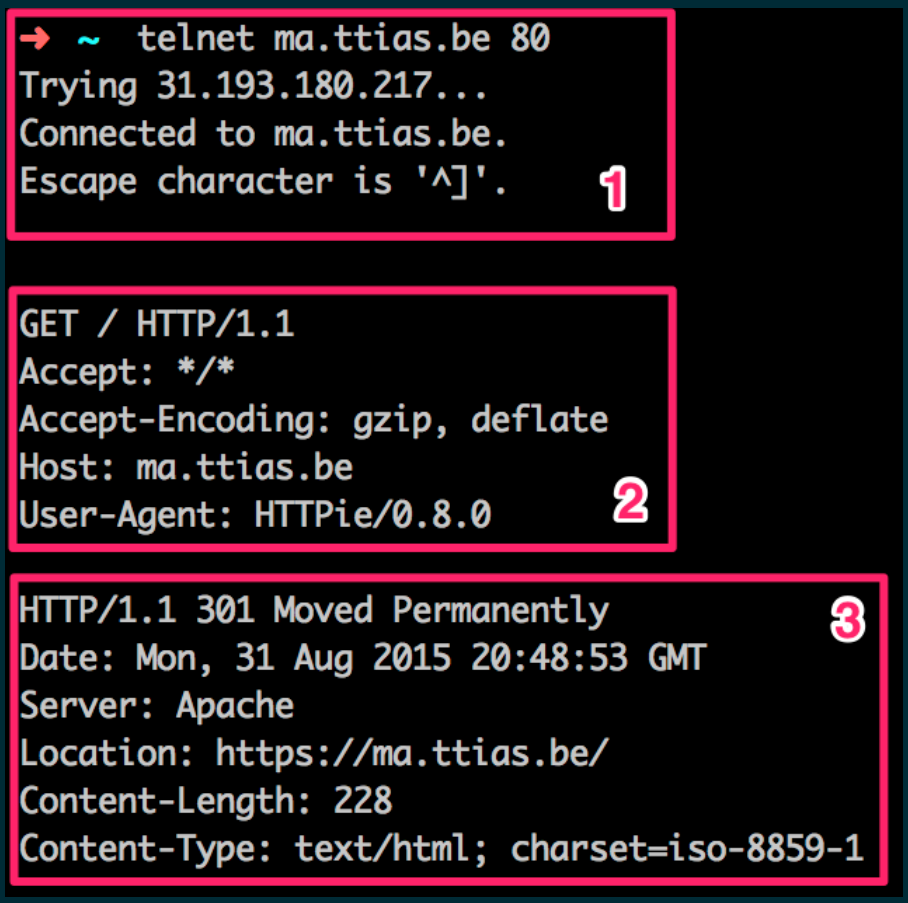
HISTORY: WHAT IS HTTP/1.1 (CONT)

- Request headers
  - Example: user requests http://ma.ttias.be/http2
  - TCP connection to 31.193.180.217 on port 80 is established
  - User Agent sends headers to describe the request

REQUEST HEADERS

```
GET /http2 HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Host: ma.ttias.be
User-Agent: IE, Chrome, Firefox, ...
```

Simple key/value pairs, new line separated. Double new line ends the headers.


➜ ~ telnet ma.ttias.be 80
Trying 31.193.180.217...
Connected to ma.ttias.be.
Escape character is '^]'.
1
GET / HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Host: ma.ttias.be
User-Agent: HTTPie/0.8.0
2
HTTP/1.1 301 Moved Permanently
Date: Mon, 31 Aug 2015 20:48:53 GMT
Server: Apache
Location: https://ma.ttias.be/
Content-Length: 228
Content-Type: text/html; charset=iso-8859-1
3

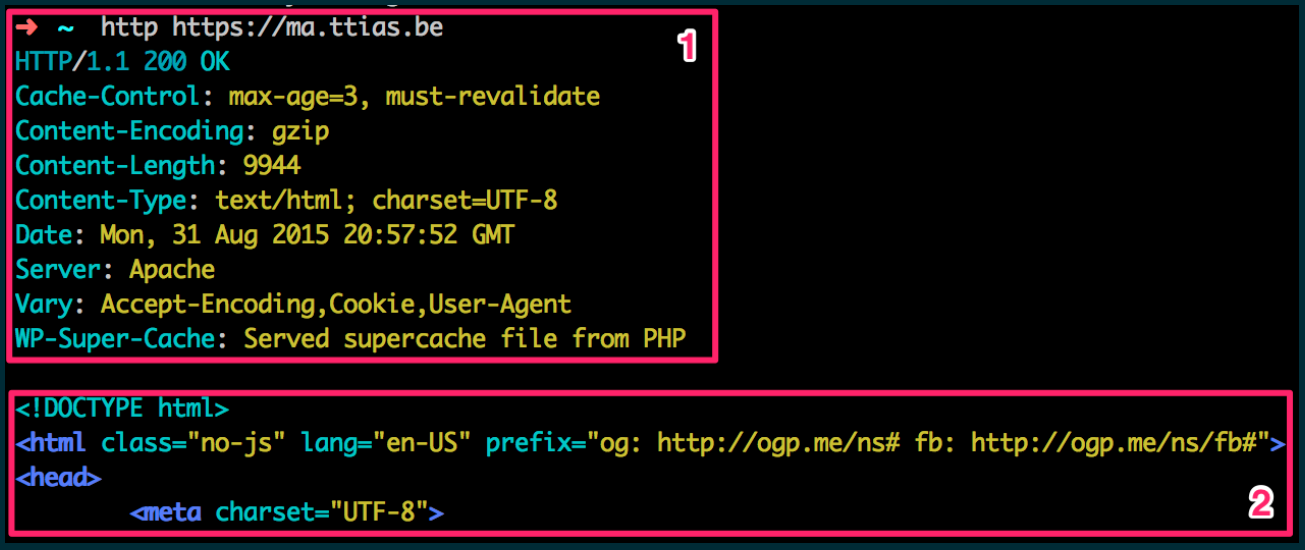
HISTORY: WHAT IS HTTP/1.1 (CONT)

- Response headers
  - Example: user requests http://ma.ttias.be/http2
  - Client sent all HTTP headers
  - Server generates response,
    sends HTTP headers + data

RESPONSE HEADERS

```
HTTP/1.1 200 OK
Cache-Control: max-age=3, must-revalidate
Content-Encoding: gzip
Content-Length: 9944
Content-Type: text/html; charset=UTF-8
Server: Apache
Date: Mon, 31 Aug 2015 20:55:50 GMT
```

Same kind of key/value pairs, new line separated. Double new line ends the headers.



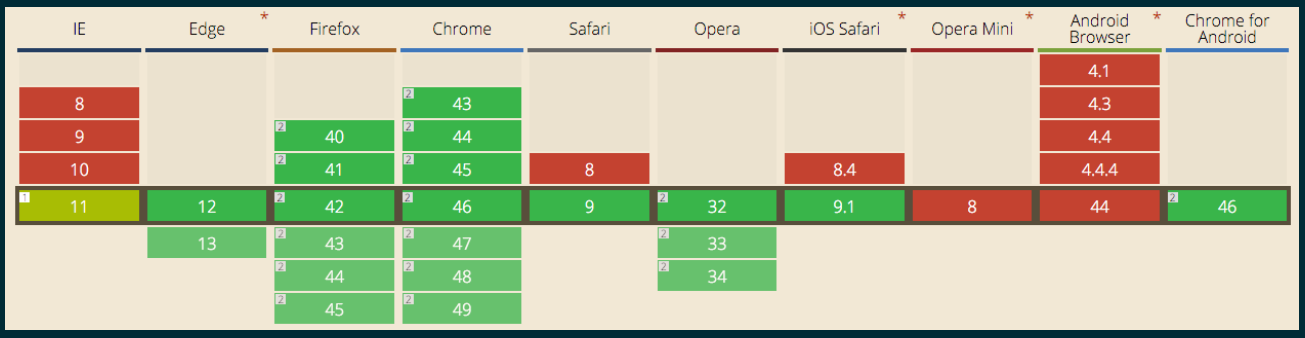
Uses the colorful httpie CLI client.

WHAT DOES HTTP/2 DO?

OR: WHAT PROBLEM IS HTTP/2 TRYING TO SOLVE?

- Binary stream, no more plain text.
- Based on Google's SPDY Protocol
- Multiplexed connections: multiple requests, one TCP/IP connection.
- Server side push
- Request priorities

WHO SUPPORTS HTTP/2: CLIENTS



IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
								4.1	
8			43					4.3	
9		40	44					4.4	
10		41	45	8		8.4		4.4.4	
11	12	42	46	9	32	9.1	8	44	46
	13	43	47		33				
		44	48		34				
		45	49						

Image source: caniuse.com
(updated 11/2015)

WHO SUPPORTS HTTP/2: SERVERS

- Apache: module, mod_http2 (based on mod_h2)
  (Apache 2.4, it's being backported to Apache 2.2)
- Nginx 1.9.5+: beta, but stable, running on nucleus.be
  howto: enable HTTP/2 in nginx
- Microsoft IIS 10, only in Windows 10 and Server 2016
- Alternative servers: H2O, nghttp2

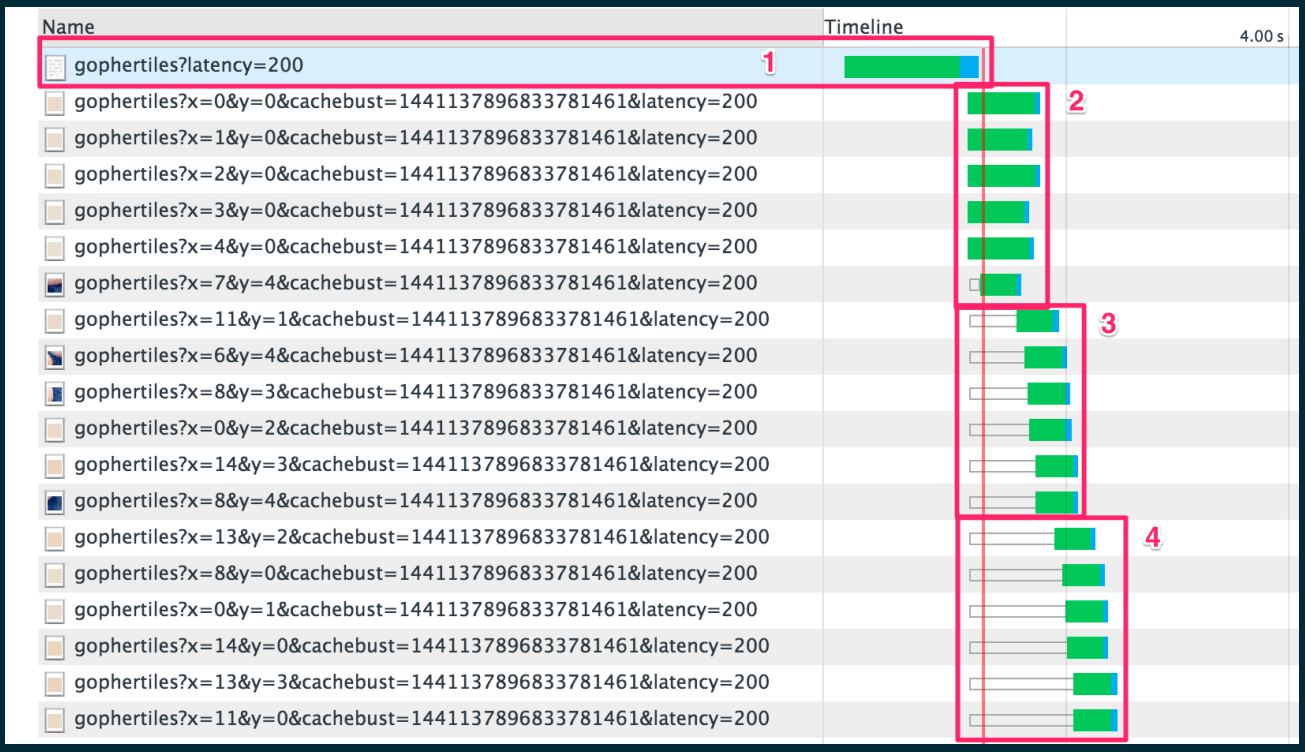
Bottom line: it's getting easier to run HTTP/2 in production today on your servers. But we're not there yet.

BENEFITS OF HTTP/2

- Faster?
- Less resource intensive?
- Better bandwidth usage?
- More control on the server?

BENEFIT #1: DOMAIN SHARDING

Most browsers only allow 6 connections per hostname. This is why people shard.



BENEFIT #1: DOMAIN SHARDING

- Browsers limit connections per hostname
- Devs are smart: cdn1.mydomain.tld, cdn2.mydomain.tld, ...
- Browser now starts multiple simultaneous per domain, yay!
- Downsides
  - multiple DNS lookup
  - new TCP connections (3-way handshake)
  - TCP slow start (congestion window)
- Despites downsides, still a performance win (in most cases) in HTTP/1.1

BENEFIT #1: DOMAIN SHARDING - THE HTTP/2 FIX

- Multiplexed TCP connection: one connection to rule them all
- Sharding now hurts performance, because with HTTP/2
  - ... only 1 DNS lookup
  - ... only one TCP/IP connection
  - ... only one TCP slow start
- Additional benefit: request priorities (later)
- Less concatenated large CSS/JavaScript files (*)

(*) Depends: no point in sending > 150KB CSS files if current page only needs 5KB of that CSS. Could make sense in HTTP/1.1, to have it cached in the browser during initial page load.

BENEFIT #2: HTTPS / TLS EVERYWHERE

- In the HTTP/2 protocol, HTTPS is not required.
- All major browsers *do* require HTTPS for HTTP/2
- H2C: HTTP/2 over plain text (used: nowhere, yet)
- More fun managing SSL certificates (*)

(*) Letsencrypt.org (EFF) to offer free certificates, just don't screw up.

BENEFIT #3: HEADER COMPRESSION

- In HTTP/1.1, headers are never compressed or encrypted.
- Some sites send > 100KB worth of cookies (*)
- Could easily have > 75% compression ratio
- HPACK: HTTP Header Compression
- For example, random website:
  - HTTP/1.1 header size: 235 Bytes
  - SPDY 3.1 header size: 59 Bytes
  - HTTP/2 header size: 28 Bytes
  - 8x reduction in size

(*) Research: 1MB of data for cookies

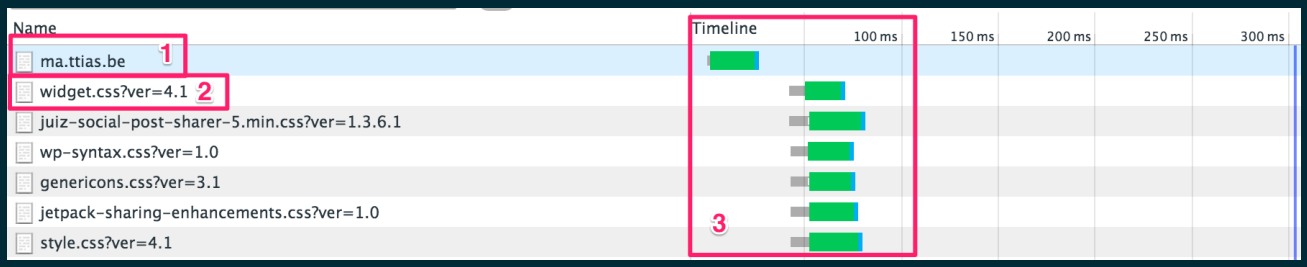
BENEFIT #4: SERVER SIDE PUSH

- In HTTP/1.1, client (UA) decides priority
- HTTP/2 can send additional responses that weren't requested yet

- ie: CSS or javascript the client would request anyhow
- Can be denied by the client
- Does not replace websockets, no Javascript API for server side push

BENEFIT #4: SERVER SIDE PUSH

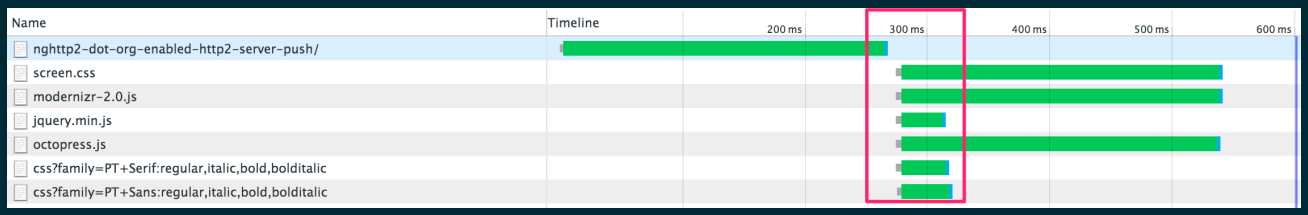
Normal HTTP/1.1



Client downloads page, parses it, finds additional resources & requests them. ~50ms delay for parsing.

BENEFIT #4: SERVER SIDE PUSH

HTTP/2



Safe to assume client will want CSS, push it with initial HTTP request.

BENEFIT #4: SERVER SIDE PUSH

- How to manipulate from your PHP code?
- Each webserver may implement its own method
- Headers will be used to manipulate the request
- Example, via the nghttp2 server:

```
header('Link: </path/to/your/style.css>;');
```

BENEFIT #4: SERVER SIDE PUSH

- Webserver interprets response, sends Server Side Push to client

```
client --> webserver --> PHP code

PHP code --> webserver --> client
```

- Unknowns: Nginx, Apache, IIS, presumably Link-header as well?

BENEFIT #5: REQUEST PRIORITIES

- Pretty obscure feature
- Initiated by the client (browser) to the server
- It's a preference, not a requirement. Server can ignore this.
- Browser fires of all HTTP requests immediately (as they are discovered), assigns them a priority, processes the responses by the server.

BENEFIT #6: SAME HTTP STATUS CODES & METHODS

- Not really a benefit, but still convenient
- 404, 503, 401, ... all the same
- PSR7 still applies: POST, PUT, GET, ... methods are the same

BENEFITS, RECAPPED

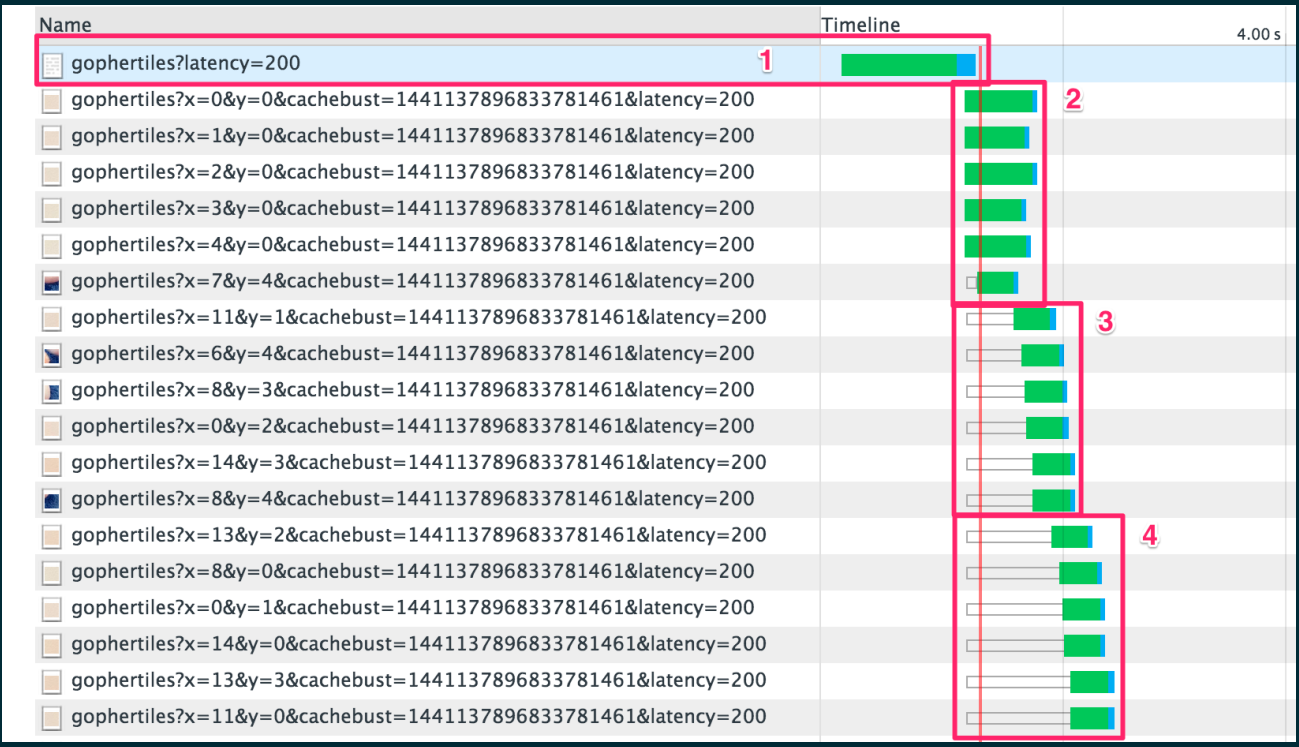
- Less domain sharding
- TLS everywhere
- Header compression in HPACK
- Server side push
- Request priorities

DISADVANTAGES

- Still beta in most webservers (nginx/apache)
- "Babysteps", no protocol changes, critics argue "did not do enough"
- Supporting HTTP/1.1 and HTTP/2 at the same time is hard: what's good for HTTP/1.1 is bad for HTTP/2 and vica versa
- HTTP/2 is new, not enough real world usage?
  (Firefox in July 2015: 13% HTTP requests are HTTP/2)

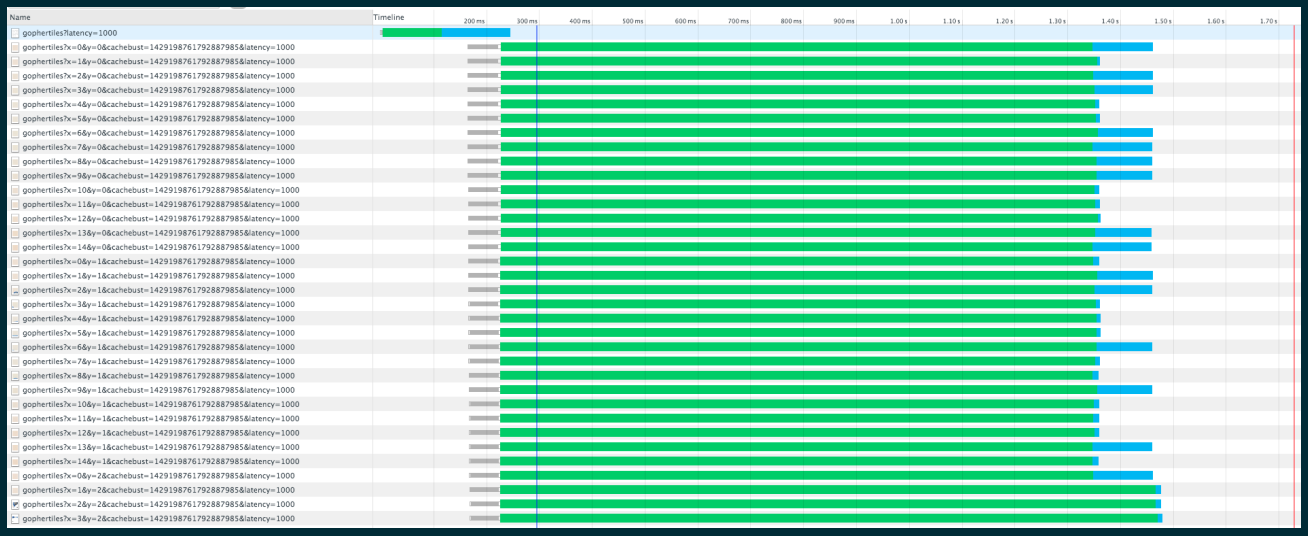
PERFORMANCE COMPARISON

ON HTTP/1.1: 6 CONCURRENT CONNECTIONS PER DOMAIN: 30S LOAD



PERFORMANCE COMPARISON

ON HTTP/2: MULTIPLE STREAMS OVER ONE TCP/IP CONNECTION: 1.5S LOAD



CONCLUSION #1

“If your application is slow on HTTP/1.1, it'll be slow on HTTP/2.

If your application is fast on HTTP/1.1, it'll only get faster on HTTP/2.”

CONCLUSION #2

“Supporting HTTP/2 on your site is relatively easy: enable server-side support.

All clients (that matter) already have HTTP/2 support.”

CONCLUSION #3

“Supporting both HTTP/1.1 and HTTP/2 at the same will be a challenge.”

THANK YOU

ANY QUESTIONS?

Contact via @mattiasgeniar or via m@ttias.be

www.nucleus.be || ma.ttias.be || cronweekly.com