TINVENTION

Introduction to HTTP2

Internal WorkShop - BootCamp

Stefano E. Campanini
Stefano.campanini@tinvention.net

Contents

- HTTP 1.1
  - The ownsides
  - Developers workarounds
- HTTP 2
  - The protocol and the advantages
  - Adoption and Implementations
- Questions
- Beer

HTTP 1.1 is huge:: Options

HTTP 1.1 is huge:

- The standard, is big and includes a myriad of details, subtleties and a lot of optional parts.

A world of options

- A lot of tiny details, few of them implemented e rarely used. Some of them produced interoperability problems (Es. HTTP Pipelining)

https://en.wikipedia.org/wiki/HTTP_pipelining

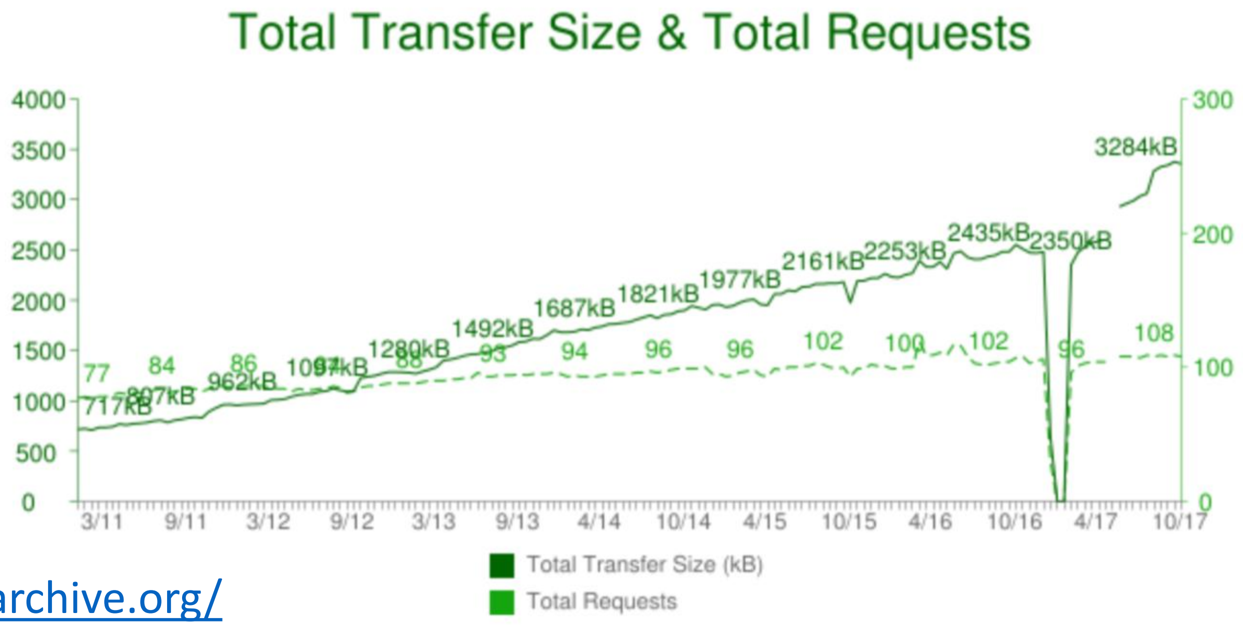
HTTP 1.1 :: TCP

Inadeguate use of TCP

TCP can be utilized better to avoid pauses or wasted intervals that could have been used to send or receive more data.

Transfer sizes and number of objects

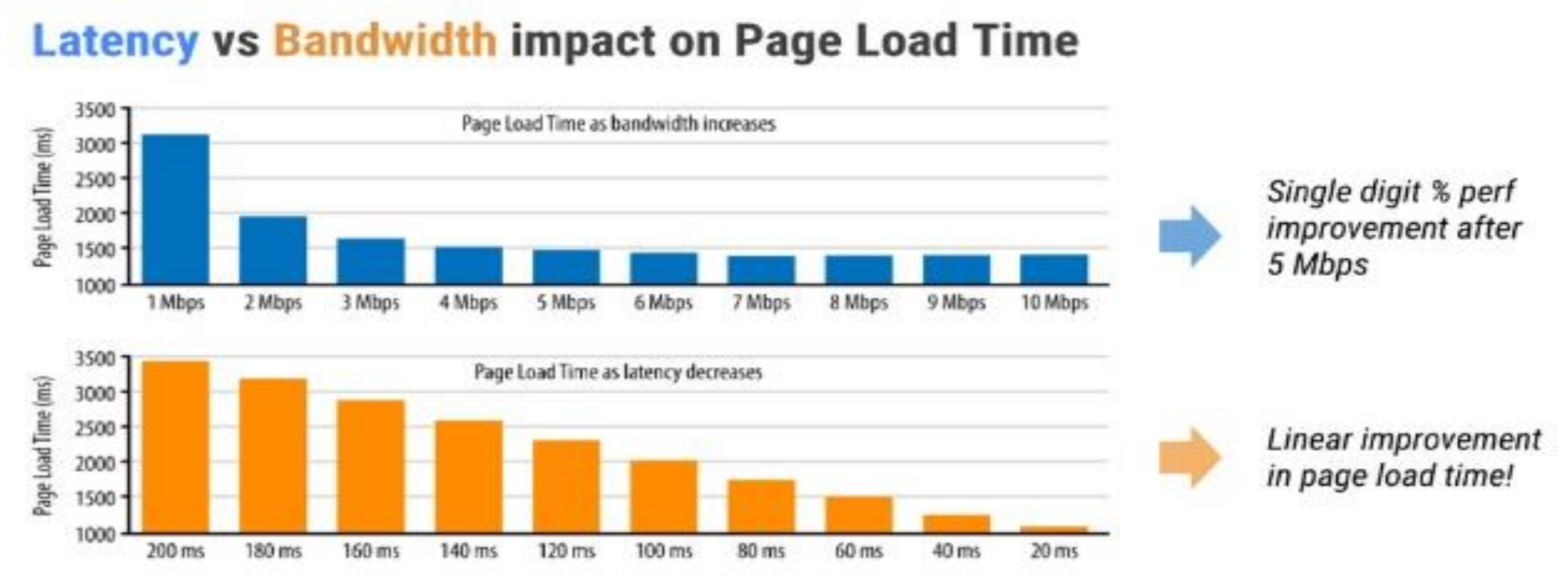
Average for web site:
* Higher Size
* High Num of requests



http://httparchive.org/

HTTP 1.1 :: Latency kills

Meaning Latency ≈ RTT => Time to answer , Interval input to rx output



https://hpbn.co/primer-on-web-performance/

HTTP 1.1 is very **latency sensitive**,

Demo: http://http2.golang.org/gophertiles

HTTP 1.1::persistent connections

The attempt to reduce latency: in 1.1, TCP conns. serve multiple requests.

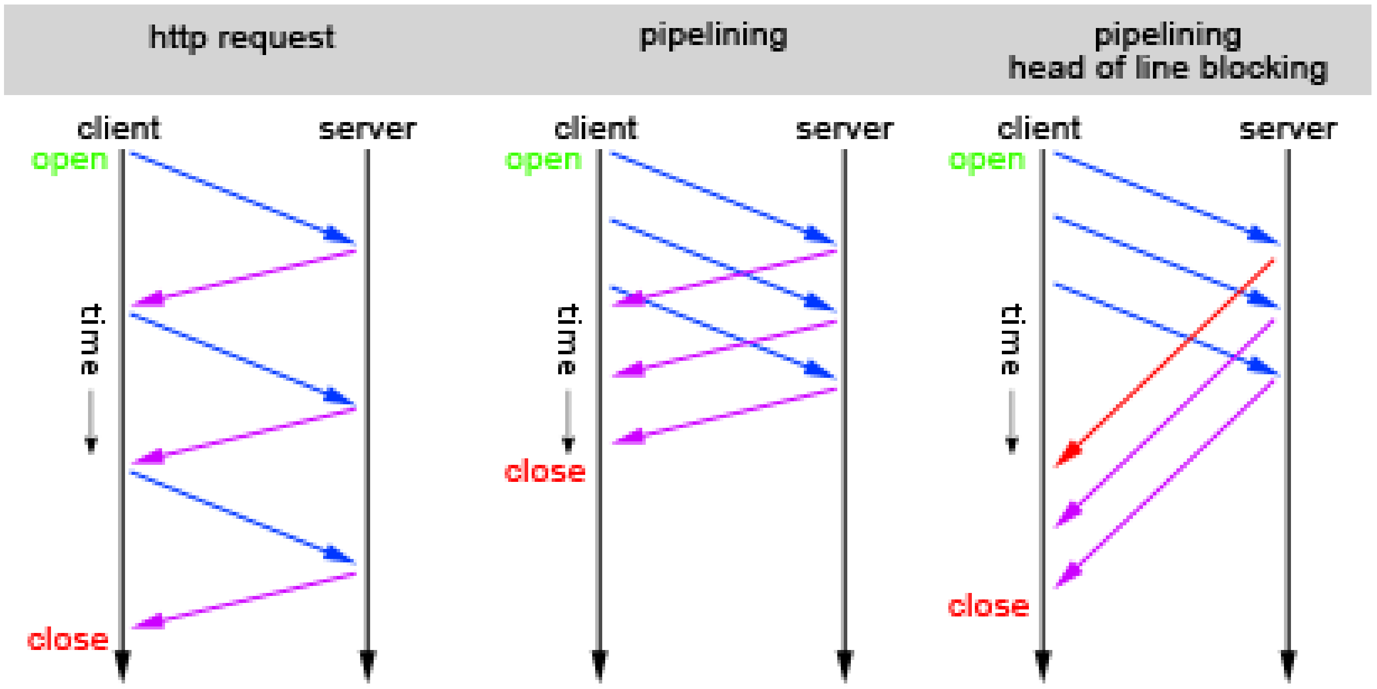
- However, the default connection timeout of Apache httpd is just 5 seconds.
- The browsers have a connection pool and reuse it per hostname.
    - Modern browsers mostly default to 6
    - Firefox, number can be customized (per-server, per-proxy, total). Persistent connections time out after 1.92 minutes of inactivity.

HTTP 1.1 :: Head of line blocking

The browser cannot know if some requests need a lot of time or not: head of line blocking.

Creating a new line is also associated with a performance and resource penalty so that's not scalable beyond a smaller number of lines. There's just no perfect solution to this.

Even today, most web browsers ship with HTTP pipelining disabled by default.



HTTP 1.1 :: high latency remedies

- Spriting,

  a lot of small images together into a single large image, then using javascript or CSS to "cut out" pieces to show smaller individual ones.

- Inlining, sending individual images as data URLs CSS

  es. `.icon1 {background: url(data:image/png;base64,<data>);}`

- Concatenation,

  a lot of different javascript file merged into a single huge lump

- Sharding,
  - serving aspects of your service on as many different hosts as possible
  - Recent stats, more than 40 sources on top 300K sites

HTTP / 2

The protocol

HTTP/2 Intro

- Primary goals are:
  - to reduce latency by multiplexing,
  - minimize protocol overhead via efficient compression of header
  - request prioritization
  - server push.
- It maintains HTTP semantics (URI, ..), it modifies how the data is formatted (framed) and transported, hides all the complexity from our applications within the new framing layer.

All existing applications can be delivered without modification.

History of SPDY, the father

SPDY was an experimental protocol, by Google, mid-2009. The goal was to reduce the load latency of web pages.

In 2012 SPDY was supported in Chrome, Firefox, and Opera, and a rapidly growing number of sites, both large (e.g., Google, Twitter, Facebook).

SPDY was on track to become a **de facto standard** through growing industry adoption.

SPDY and HTTP/2

Observing the SPDY trend, the HTTP Working Group (HTTP-WG IETF) kicked off a new effort to take the lessons learned from SPDY, build and improve on them, and deliver an official "HTTP/2" standard.

In early 2015 the IETF-IESG reviewed and approved the new HTTP/2 std. for publication.

In the 2016, Google removed SPDY support and NPM, in favor of HTTP/2 with ALPN

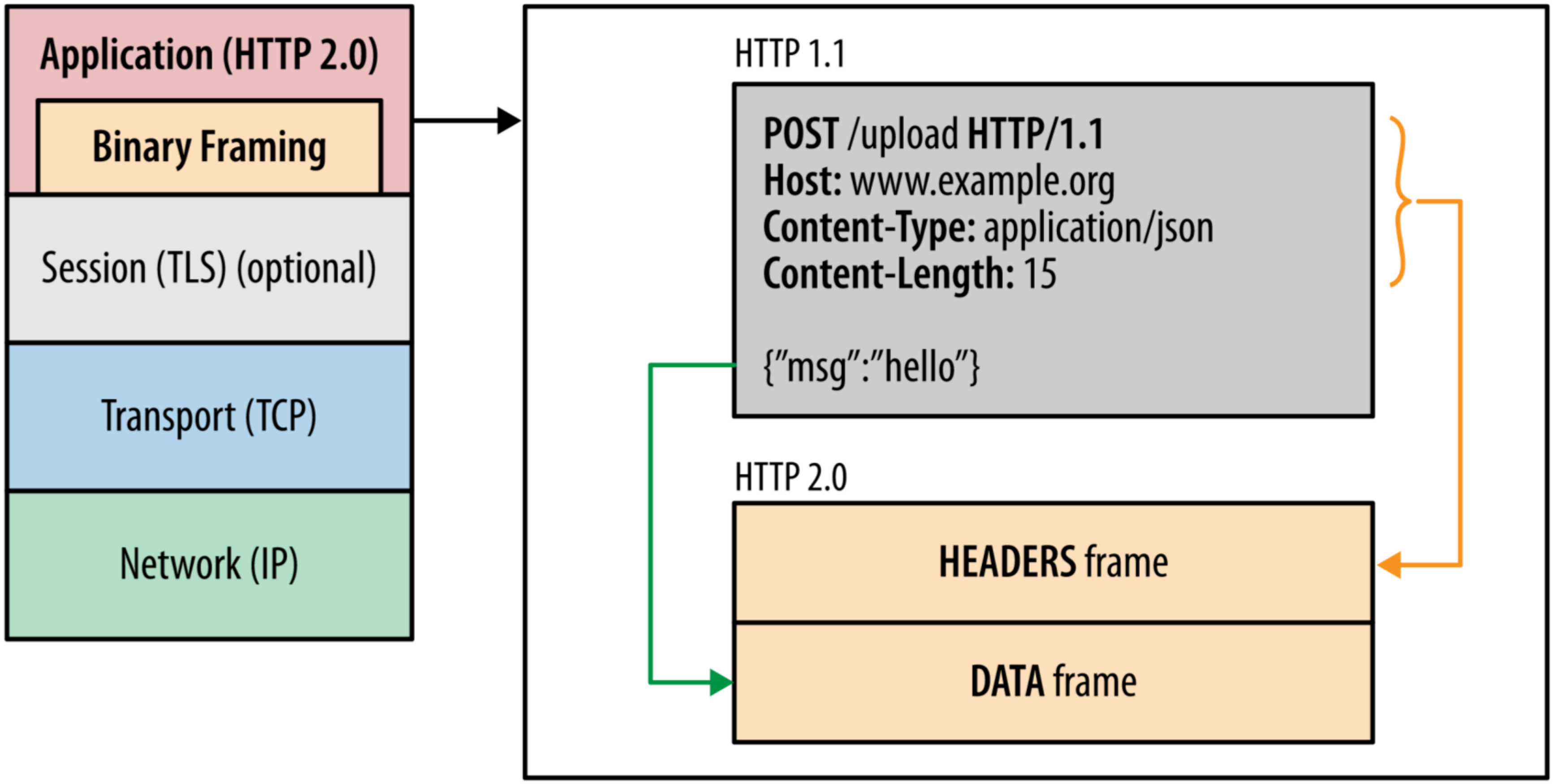
HTTP/2 – HTTP / HTTPS

- TLS is optional, but the almost all implementatios are based on TLS (ALPN).

- Today, no major browser supports HTTP2 without TLS.

Binary Framing Layer

http2 is a binary protocol.



Streams, Messages, and Frames

Stream, bidirectional flow of bytes within an established connection, which may carry one or more messages.

Message, complete sequence of frames that map to a logical request or response message. A message consists of one or more frames.

Frame, smallest unit of communication in HTTP/2, with a frame header, which at a minimum identifies the stream to which the frame belongs.

Frames

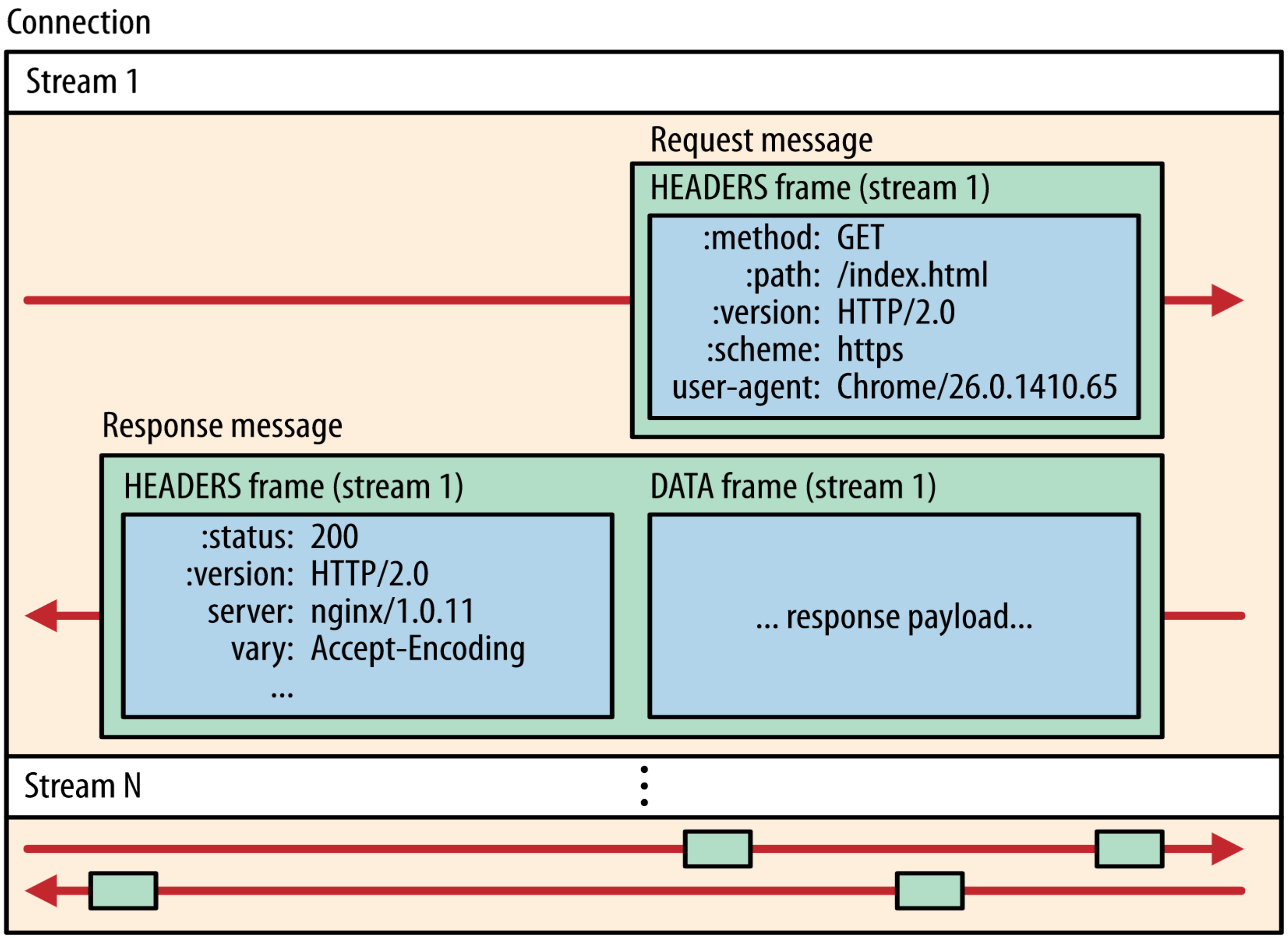
All *communication* is over a **single** TCP connection that can carry **any number of bidirectional streams**.

Each stream has a **unique identifier** and optional **priority information** that is used to carry bidirectional messages.

The frame carries a specific type of data—e.g., HTTP **headers, message payload, ...**

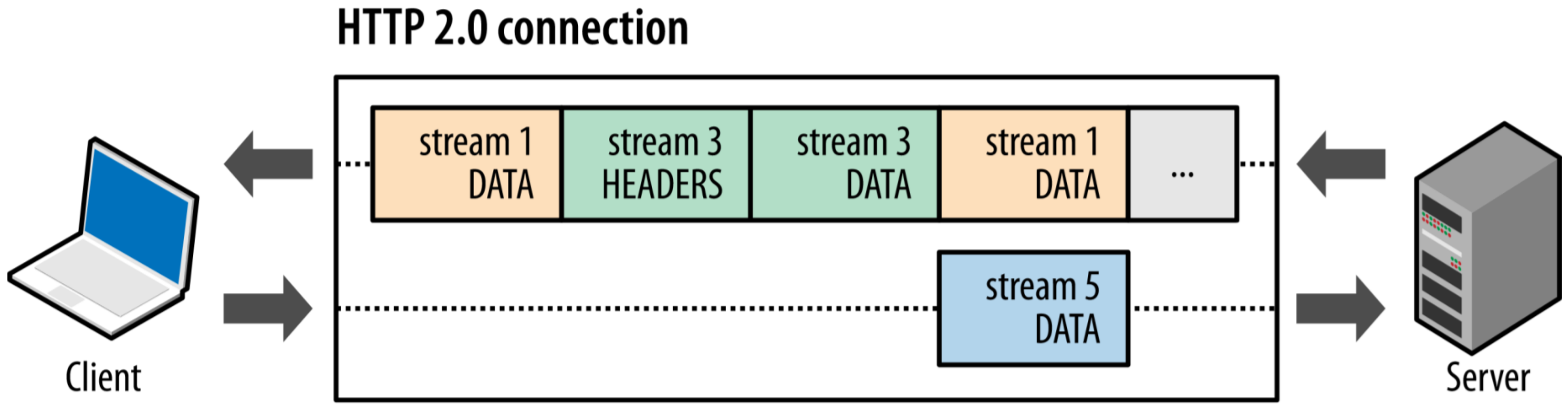
Frames from different streams may be **interleaved** and then **reassembled** via the embedded stream identifier in the header of each frame.

HTTP/2 streams, messages, and frames

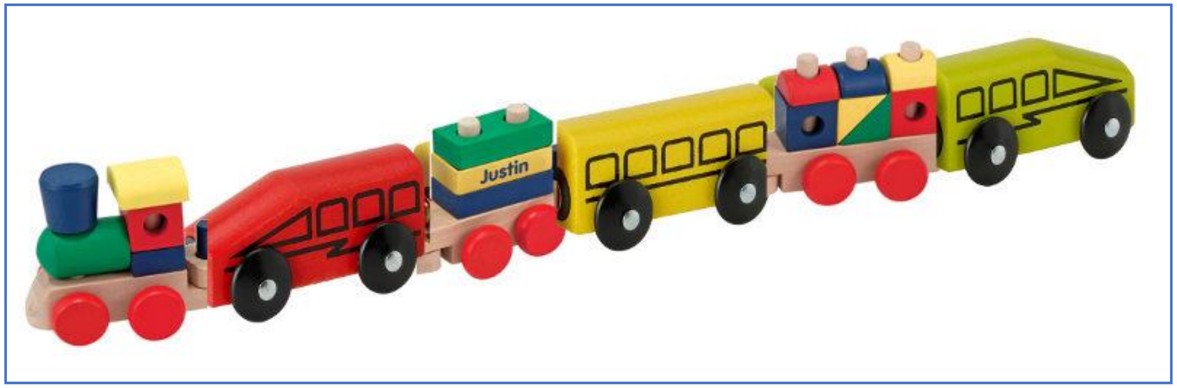


Multiplexing

Multiple streams in flight within the same connection: the client is transmitting a DATA frame (stream 5) to the server, while the server is transmitting an interleaved sequence of frames to the client for streams 1 and 3. As a result, there are **3 parallel streams in flight**!



Multiplexing :: by Trains



Src: https://bagder.gitbooks.io/http2-explained/en/part6.html

Multiplexing - Benefits

Multiplexing is the single most important enhancement of HTTP/2. In fact, it introduces a ripple effect of numerous performance benefits across the entire stack of all web technologies, enabling us to:

- Interleave multiple requests/ responses in parallel without blocking on any one
- Use a single connection to deliver multiple requests and responses in parallel
- Remove unnecessary HTTP/1.x workarounds, such as concatenated files, image sprites, and domain sharding
- Deliver lower page load times by eliminating unnecessary latency and improving utilization of available network capacity

The new binary framing layer in HTTP/2 resolves the **head-of-line blocking problem found in HTTP/1.x** and eliminates the need for multiple connections to enable parallel processing and delivery of requests and responses.

As a result, this makes our applications faster, simpler, and cheaper to deploy.

Stream Prioritization

Multiplexing of frames, implies that the **order** in which the frames are interleaved and delivered becomes a **critical performance aspect**.

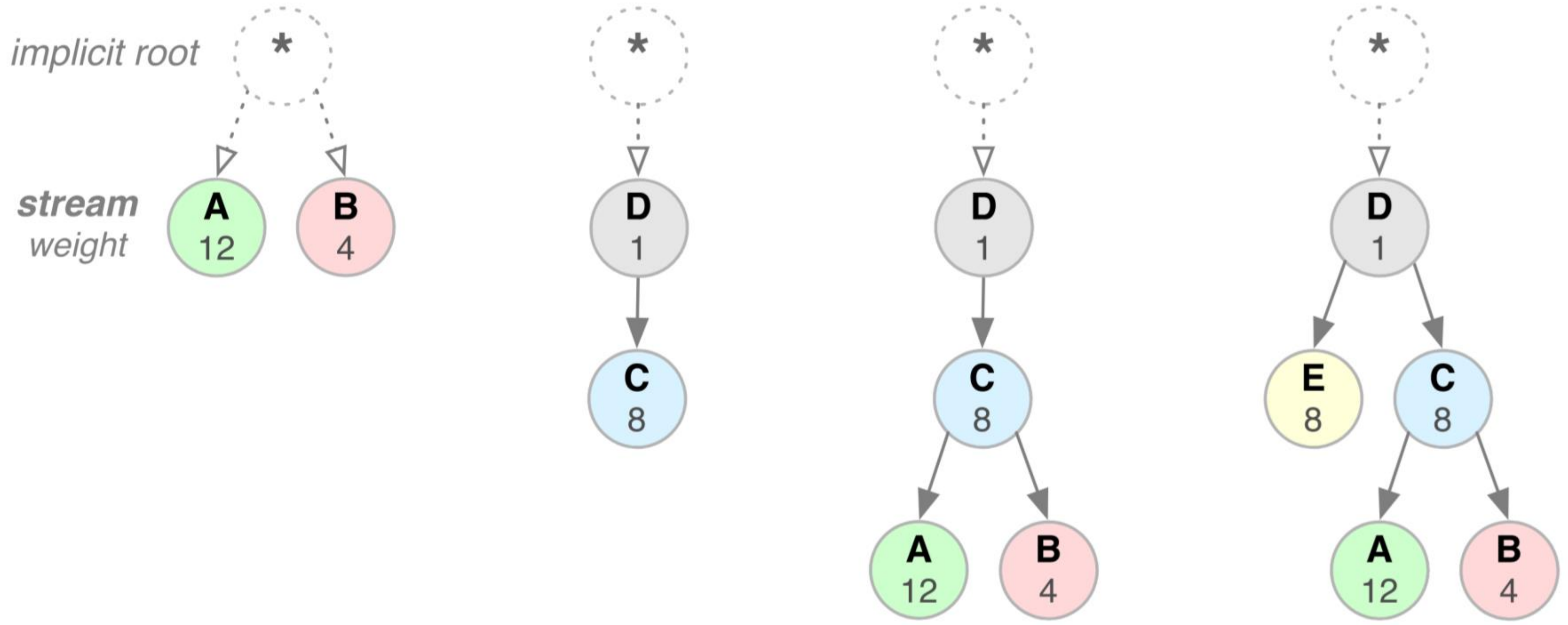
HTTP/2 standard allows each stream to have an associated weight and dependency:

- each stream may be assigned an integer **weight**
- each stream may be given an **explicit dependency** on another stream

The combination of stream dependencies and weights allows the client to construct and communicate a "prioritization tree“

The client can change dependencies and reallocate weights in response to user interaction and other signals.

Prioritization Examples



Declaring a stream dependency indicates that, if possible, the parent stream should be allocated resources ahead of its dependencies—e.g., please process and deliver response D before response C.

Browser Request Prioritization and HTTP/2

Resources need different priorities when rendering a page in the browser:

- the HTML document itself is critical to construct the DOM;
- the CSS is required to construct the CSSOM;
- both DOM and CSSOM construction can be blocked on JavaScript resources; and remaining resources, such as images, are often fetched with lower priority.

To accelerate the load time, modern browsers prioritize requests based on type of asset, their location on the page, and even learned priority from previous visits:

- —e.g., if the rendering was blocked on a certain asset in a previous visit, then the same asset may be prioritized higher in the future.

One Connection Per Origin

Multiplexing brings that all HTTP/2 connections are persistent, and only one connection per origin is required, which offers numerous performance benefits.

- TCP is optimized for long-lived,
- Fewer expensive TLS handshakes,
- …

=> Reduction of latency, throughput improvement and reduction of operational costs.

Packet Loss - HTTP/2 Performance

One TCP connection per origin downsides:

- There is still head-of-line blocking at the TCP level.
- When packet loss occurs, the TCP congestion window size is reduced, which reduces the maximum throughput of the entire connection.
- …

"In tests so far, the negative effects of head-of-line blocking (especially in the presence of packet loss) is outweighed by the benefits of compression and prioritization."

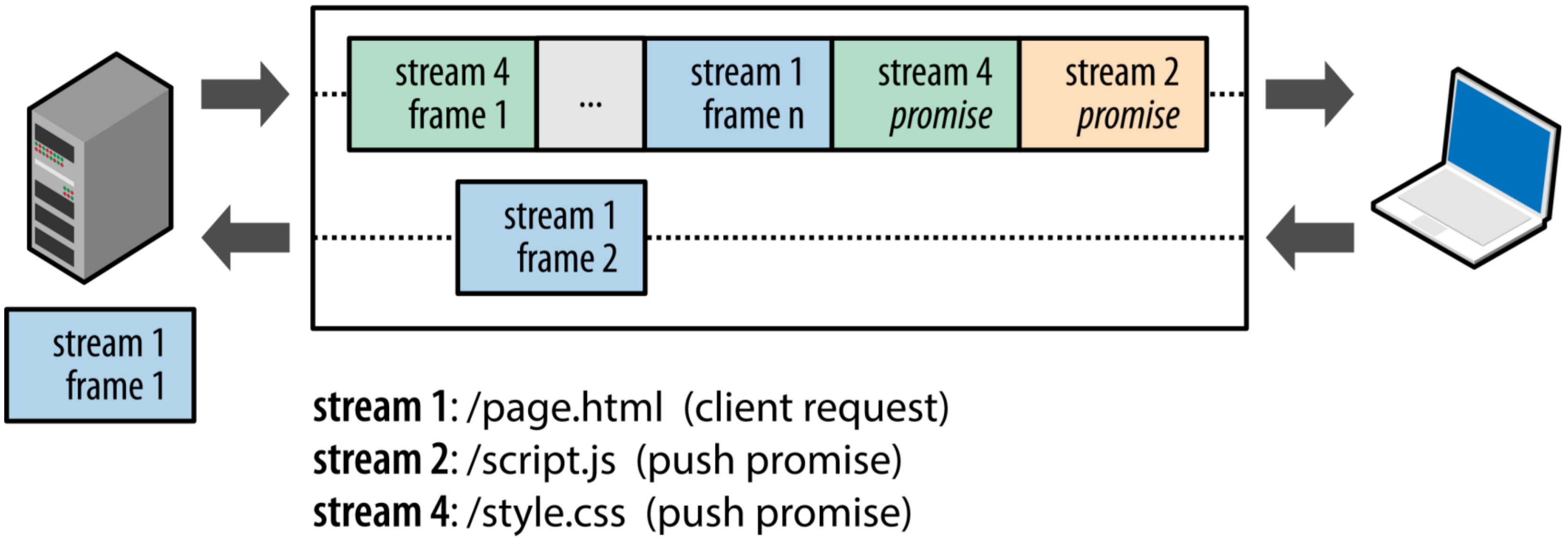
Hypertext Transfer Protocol version 2, Draft 2

Flow Control

- It is a mechanism to prevent the sender from overwhelming the receiver with data it may not want or be able to process. Example:
  - The client have requested a large video stream with high priority, but the user has paused the video and the client now wants to pause or throttle its delivery from the server to avoid fetching and buffering unnecessary data.
- HTTP/2 provides a set of simple building blocks that allow the client and server to implement their own stream- and connection-level flow control.

Server Push

The server can send multiple responses for a single client request. So, in addition to the response to the original request, the server can push additional resources to the client.

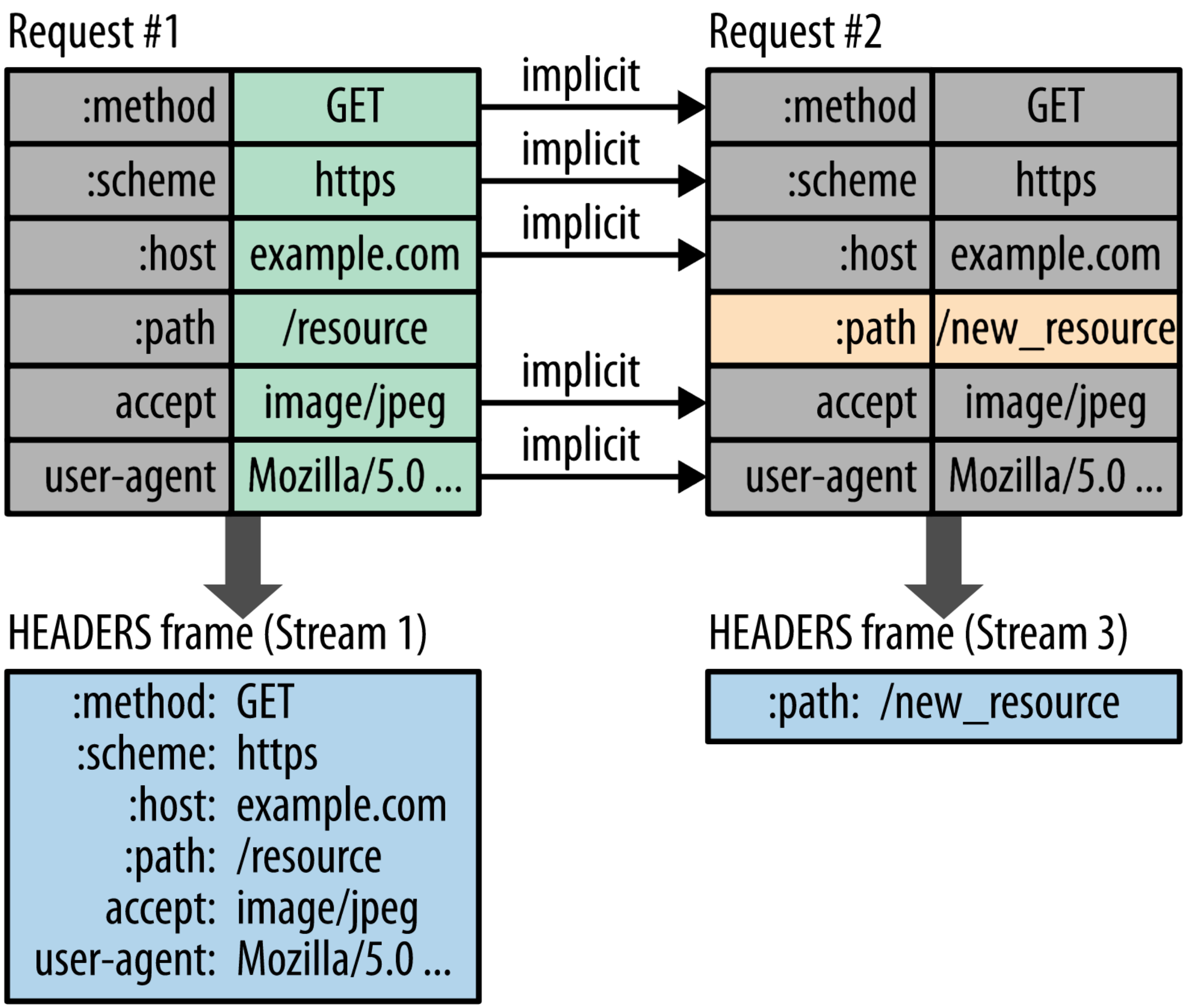

stream 4
frame 1
...
stream 1
frame n
stream 4
promise
stream 2
promise
stream 1
frame 2
stream 1
frame 1
stream 1: /page.html (client request)
stream 2: /script.js (push promise)
stream 4: /style.css (push promise)

Header Compression

It is used HPACK compression format, that uses two simple but powerful techniques:

- encode via a static Huffman code, reduces their individual transfer size.
- requires that both the client and server maintain and update an indexed list of previously seen header fields (i.e., establishes a shared compression context), to efficiently encode previously transmitted values.

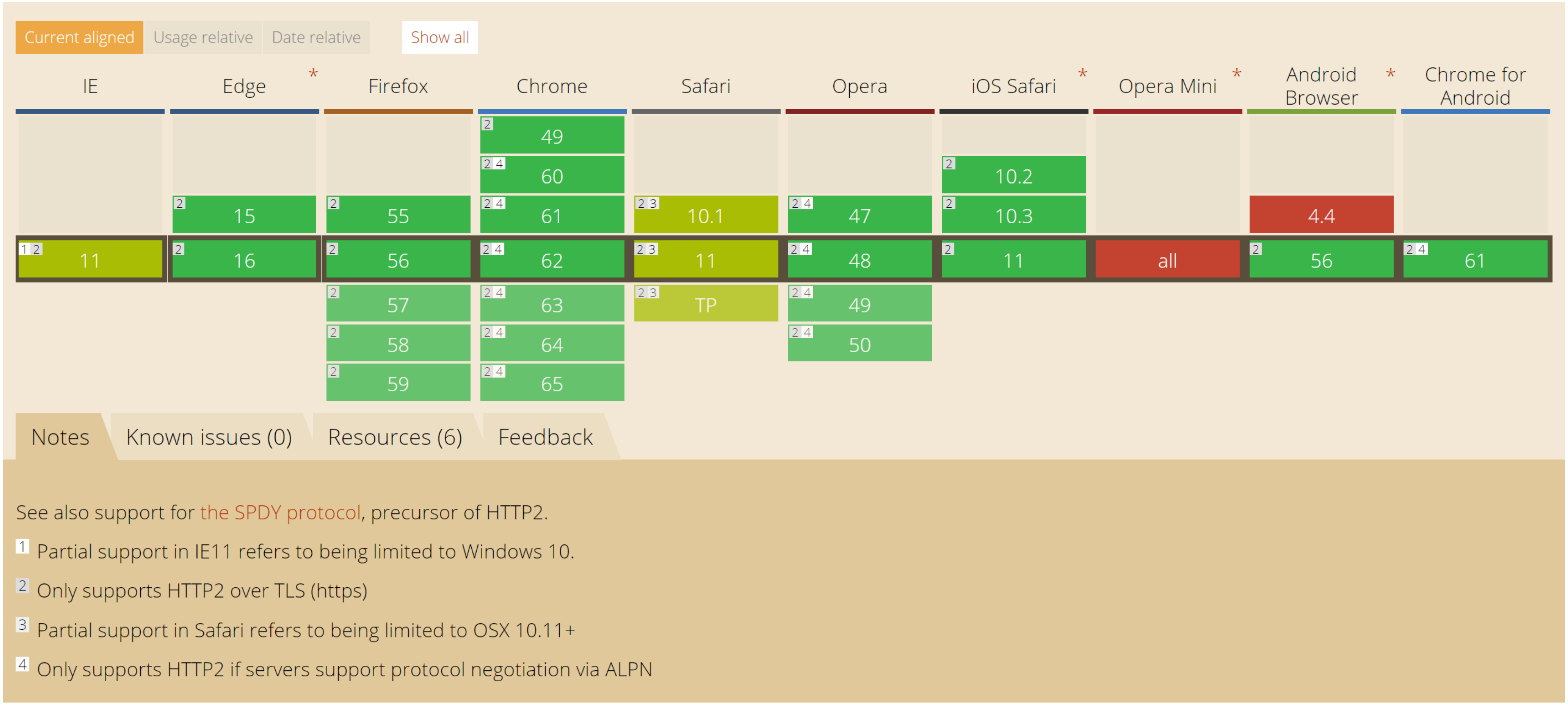
Header Compression



HTTP / 2

Support and Implementations

HTTP/2 – Browser support



Current aligned | Usage relative | Date relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			[2] 49						
			[2 4] 60			[2] 10.2			
	[2] 15	[2] 55	[2 4] 61	[2 3] 10.1	[2 4] 47	[2] 10.3		4.4	
[1 2] 11	[2] 16	[2] 56	[2 4] 62	[2 3] 11	[2 4] 48	[2] 11	all	[2] 56	[2 4] 61
		[2] 57	[2 4] 63	[2 3] TP	[2 4] 49				
		[2] 58	[2 4] 64		[2 4] 50				
		[2] 59	[2 4] 65						

Notes | Known issues (0) | Resources (6) | Feedback

See also support for the SPDY protocol, precursor of HTTP2.

[1] Partial support in IE11 refers to being limited to Windows 10.

[2] Only supports HTTP2 over TLS (https)

[3] Partial support in Safari refers to being limited to OSX 10.11+

[4] Only supports HTTP2 if servers support protocol negotiation via ALPN

Src: http://caniuse.com/#search=http2

HTTP/2 -Stats

Adoption

- http://isthewebhttp2yet.com/measurements/adoption.html

Server / Libs / Clients - Implementations / Support

  - ***h2*** is HTTP/2 over TLS (negotiation via ALPN).
  - ***h2c*** is HTTP/2 over TCP.

- https://github.com/http2/http2-spec/wiki/Implementations

Server Conf Tips

- ***h2*** is HTTP/2 over TLS (negotiation via ALPN).
- ***h2c*** is HTTP/2 over TCP.

Apache > 2.4 , it is possible to set preferred protocols

```
LoadModule http2_module modules/mod_http2.so
```

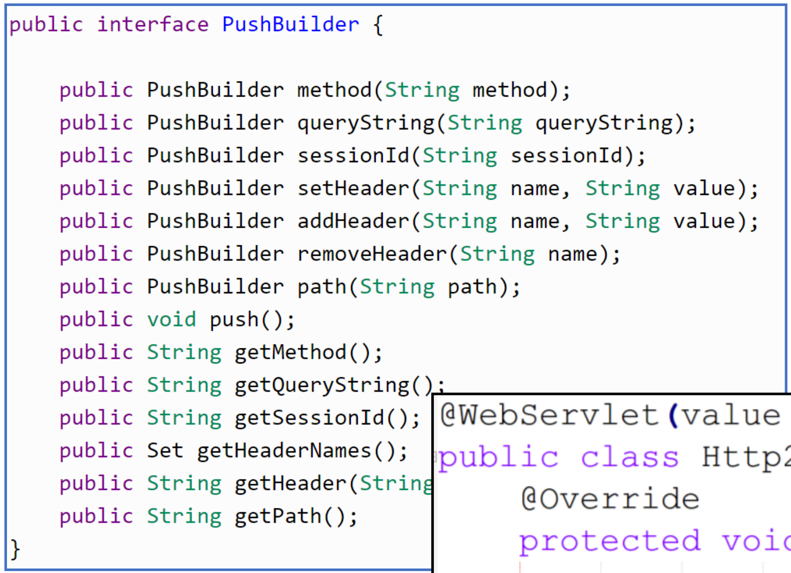
```
Protocols h2 h2c http/1.1
```

Tomcat > 8.5 , TLS support is needed

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11AprProtocol"
  maxThreads="150" SSLEnabled="true">
    <UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol"/>
    <SSLHostConfig honorCipherOrder="false">
        <Certificate certificateKeyFile="conf/ca.key" certificateFile="conf/ca.crt"/>
    </SSLHostConfig>
```

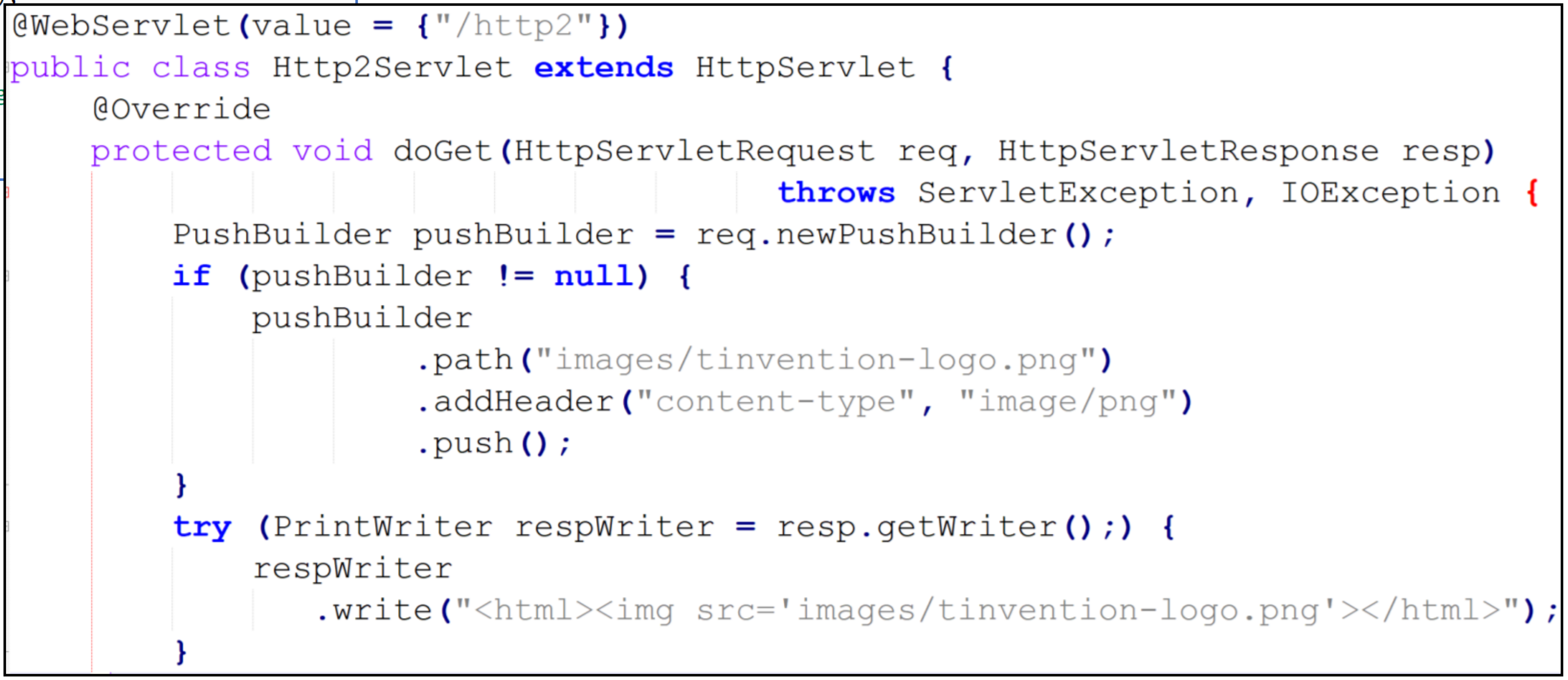
NB. each HTTP/2 stream requires a dedicated container thread for the duration of that stream.

Servlet 4.0 – HTTP/2



```java
public interface PushBuilder {

    public PushBuilder method(String method);
    public PushBuilder queryString(String queryString);
    public PushBuilder sessionId(String sessionId);
    public PushBuilder setHeader(String name, String value);
    public PushBuilder addHeader(String name, String value);
    public PushBuilder removeHeader(String name);
    public PushBuilder path(String path);
    public void push();
    public String getMethod();
    public String getQueryString();
    public String getSessionId();
    public Set getHeaderNames();
    public String getHeader(String
    public String getPath();
}
```



```java
@WebServlet(value = {"/http2"})
public class Http2Servlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                    throws ServletException, IOException {
        PushBuilder pushBuilder = req.newPushBuilder();
        if (pushBuilder != null) {
            pushBuilder
                    .path("images/tinvention-logo.png")
                    .addHeader("content-type", "image/png")
                    .push();
        }
        try (PrintWriter respWriter = resp.getWriter();) {
            respWriter
                .write("<html><img src='images/tinvention-logo.png'></html>");
        }
```

My Questions…

So, are you using TLS with HTTP 1.1 ?

Why ? :D

Will HTTP/2 replace websocket in future ?

Got Beer ?

Ref & links

- *High Performance Browser Networking*, Ilya Grigorik | O'Reilly
  - https://hpbn.co/http2/
- *HTT2 explained* , Daniel Stenberg | GitHub
  - https://daniel.haxx.se/http2/
- Connection management in HTTP/1.x
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Connection_management_in_HTTP_1.x
- Go + HTTP/2, by *bradfitz@golang.org*,
  - http://http2.golang.org/gophertiles