APPLIED COMPUTER SCIENCES

FH JOANNEUM
University of Applied Sciences

HTTP/2

A new way to speak with your servers

John Feiner

Why not HTTP 1.1?

= how to optimise data transfer

HTTP 1.x is

- simple: stateless and text based
- allows upgrade to web socket protocol

BUT:

- polling web services
- not efficient: rather large headers, need to reconnect, no compression, no security, multiple connection...

History: HTTP/2 (was SPDY)

performance (including latency, round-trip time):

- binary, compressed headers (HPACK), ...

features:

- multiplexed (prioritised) streams, ...
- server push, ...

Improve HTTP 1.x Performance

HTTP 1.x allows

- 4,8, 16 open connections per domain<= check your current browser
- upgrade to web socket protocol

Web site optimisation (toolchains: concat JS/CSS, minify)

- reduce number of requests: combine resources
- gzip, Content Delivery Networks (CDN), reduce DNS lookup, cache
- rendering (put styles/scripts at the top/bottom)

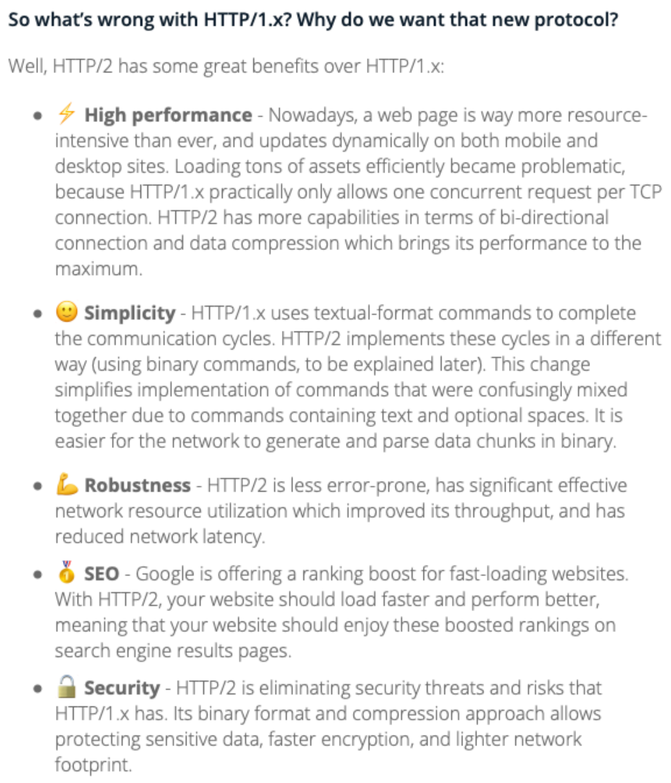
http://yslow.org

Why HTTP/2?

Optimised performance, security, startup time, loading time, responsiveness

HTTP/2 supports:

a binary protocol, binary headers, compressed headers, multiplexed connections, server push, encryption, caching, preloading.



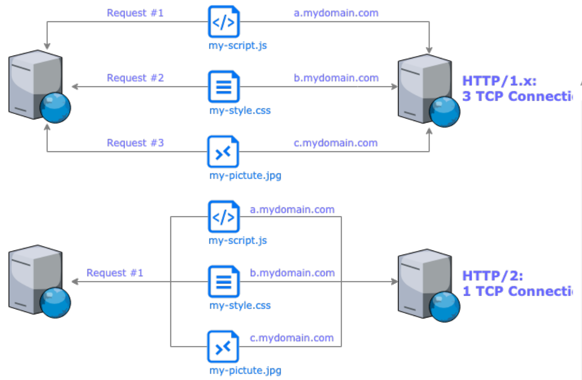
So what's wrong with HTTP/1.x? Why do we want that new protocol?

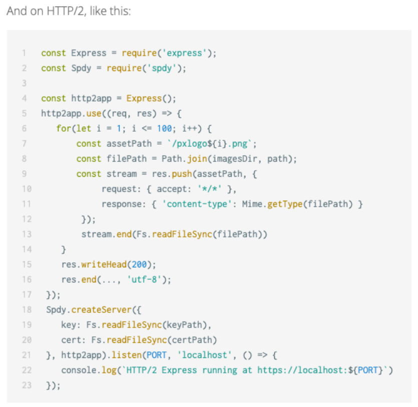
Well, HTTP/2 has some great benefits over HTTP/1.x:

- ⚡ **High performance** - Nowadays, a web page is way more resource-intensive than ever, and updates dynamically on both mobile and desktop sites. Loading tons of assets efficiently became problematic, because HTTP/1.x practically only allows one concurrent request per TCP connection. HTTP/2 has more capabilities in terms of bi-directional connection and data compression which brings its performance to the maximum.
- 🙂 **Simplicity** - HTTP/1.x uses textual-format commands to complete the communication cycles. HTTP/2 implements these cycles in a different way (using binary commands, to be explained later). This change simplifies implementation of commands that were confusingly mixed together due to commands containing text and optional spaces. It is easier for the network to generate and parse data chunks in binary.
- 💪 **Robustness** - HTTP/2 is less error-prone, has significant effective network resource utilization which improved its throughput, and has reduced network latency.
- 🥇 **SEO** - Google is offering a ranking boost for fast-loading websites. With HTTP/2, your website should load faster and perform better, meaning that your website should enjoy these boosted rankings on search engine results pages.
- 🔒 **Security** - HTTP/2 is eliminating security threats and risks that HTTP/1.x has. Its binary format and compression approach allows protecting sensitive data, faster encryption, and lighter network footprint.

https://www.perimeterx.com/blog/http2/

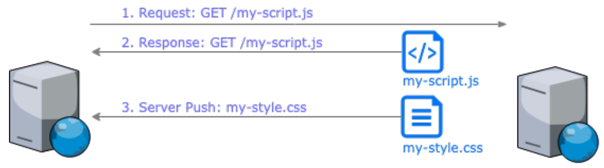
Check out features

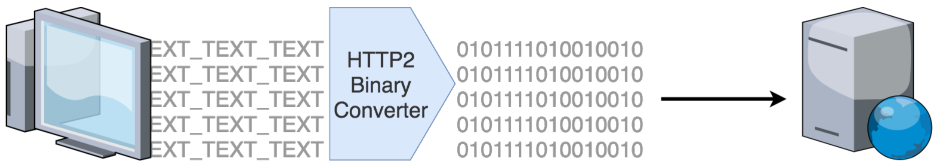


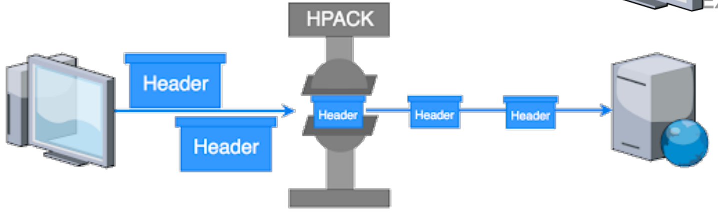

And on HTTP/2, like this:

```
const Express = require('express');
const Spdy = require('spdy');

const http2app = Express();
http2app.use((req, res) => {
   for(let i = 1; i <= 100; i++) {
       const assetPath = `/pxlogo${i}.png`;
       const filePath = Path.join(imagesDir, path);
       const stream = res.push(assetPath, {
            request: { accept: '*/*' },
            response: { 'content-type': Mime.getType(filePath) }
        });
        stream.end(Fs.readFileSync(filePath))
    }
    res.writeHead(200);
    res.end(..., 'utf-8');
 });
 Spdy.createServer({
    key: Fs.readFileSync(keyPath),
    cert: Fs.readFileSync(certPath)
 }, http2app).listen(PORT, 'localhost', () => {
    console.log(`HTTP/2 Express running at https://localhost:${PORT}`)
 });
```


1. Request: GET /my-script.js
2. Response: GET /my-script.js
my-script.js
3. Server Push: my-style.css
my-style.css





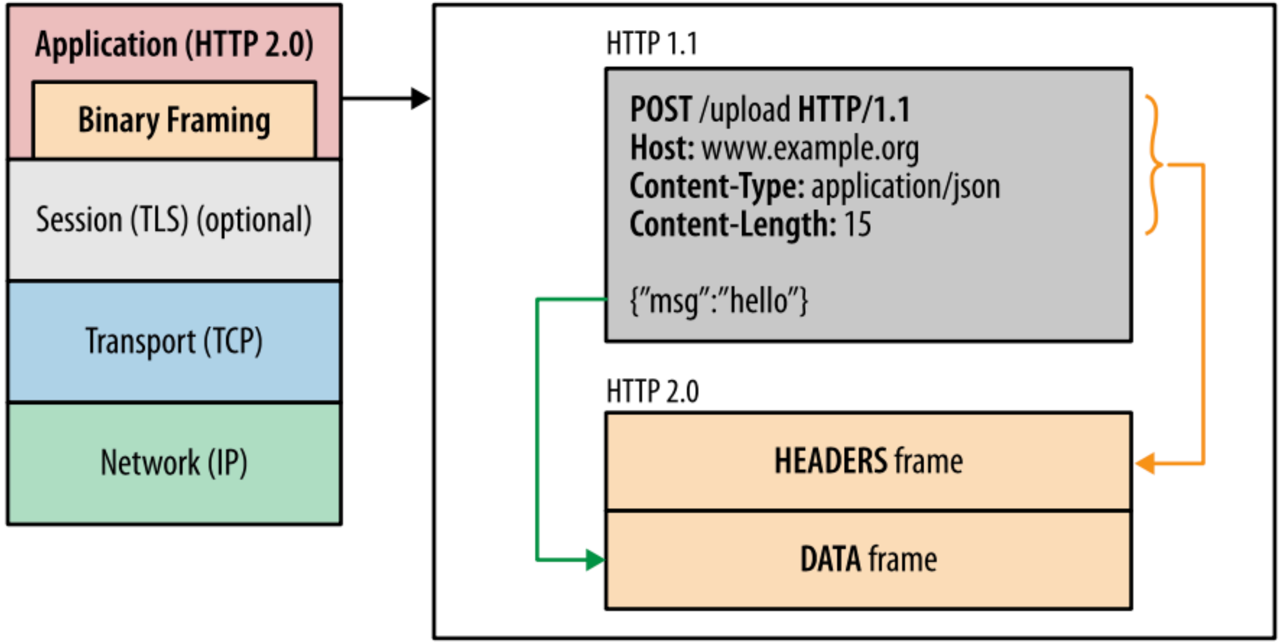
https://www.perimeterx.com/blog/http2/

HTTP/2

Binary framing layer

Binary framing layer

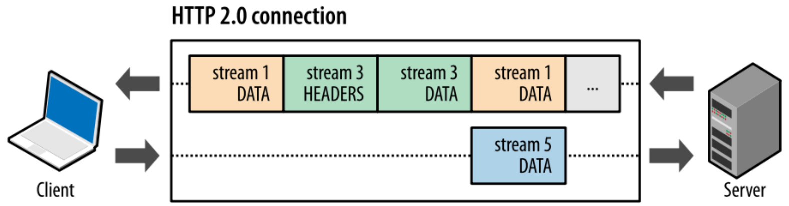
At the core of all performance enhancements of HTTP/2 is the new binary framing layer, which dictates how the HTTP messages are encapsulated and transferred between the client and server.

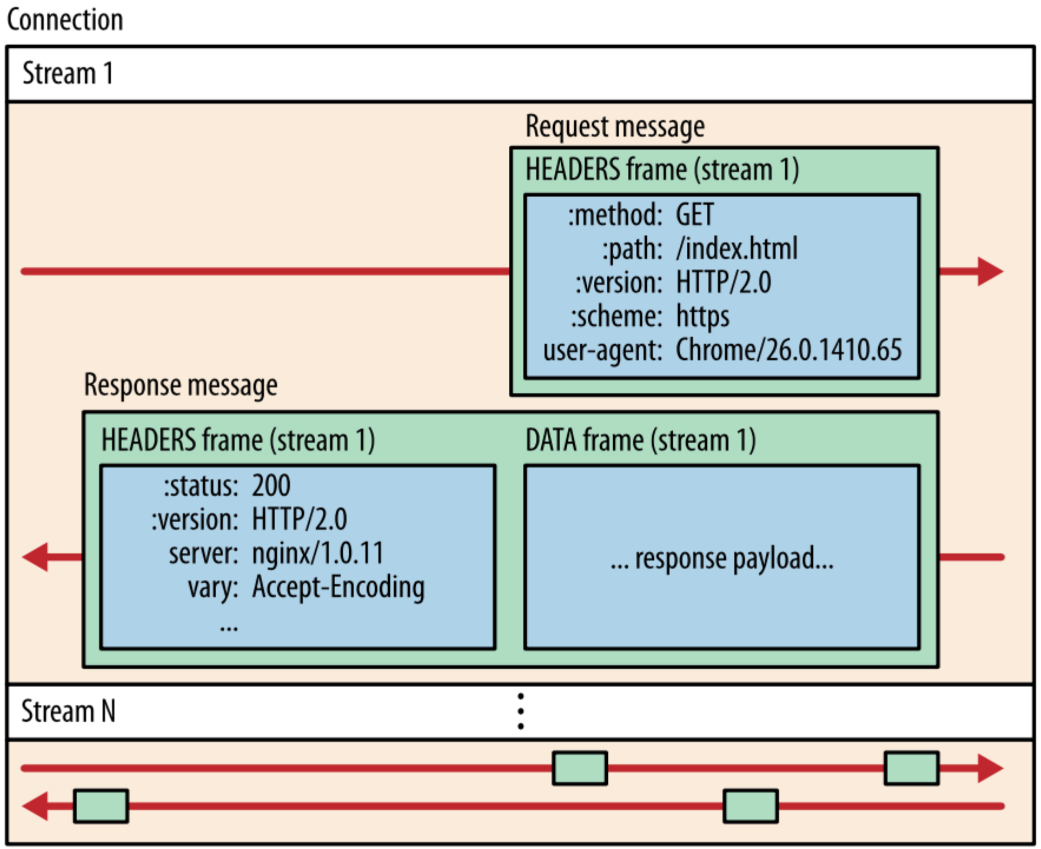


HTTP/2

Streams

Stream prioritisation

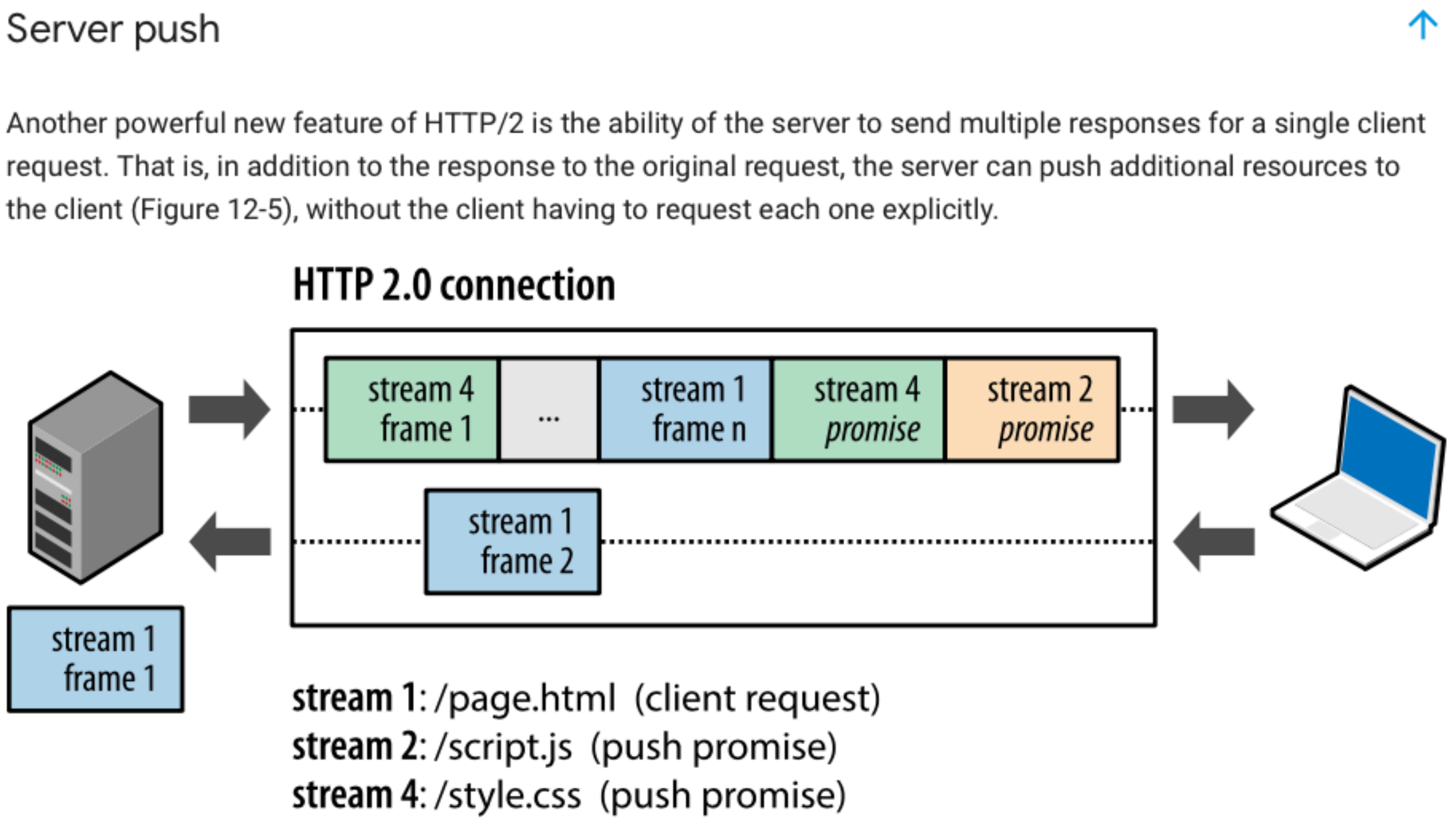




https://developers.google.com/web/fundamentals/performance/http2/

HTTP/2

Server Push


Server push

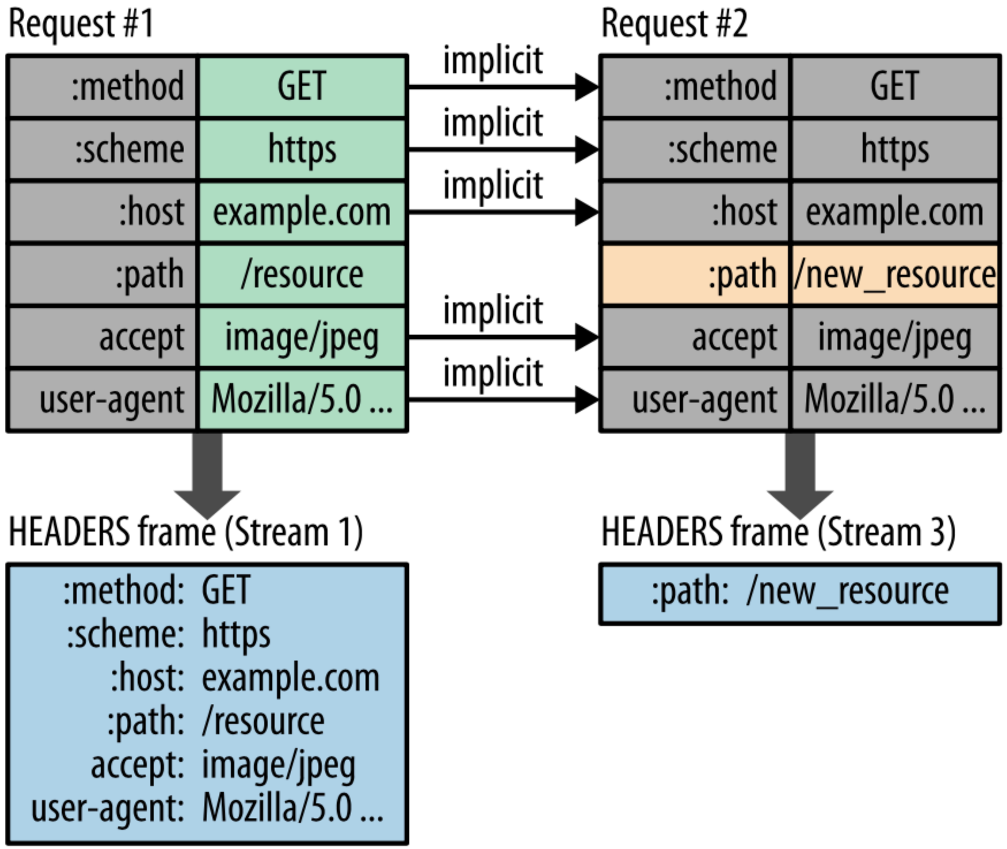
Another powerful new feature of HTTP/2 is the ability of the server to send multiple responses for a single client request. That is, in addition to the response to the original request, the server can push additional resources to the client (Figure 12-5), without the client having to request each one explicitly.

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

FH | JOANNEUM

HTTP/2

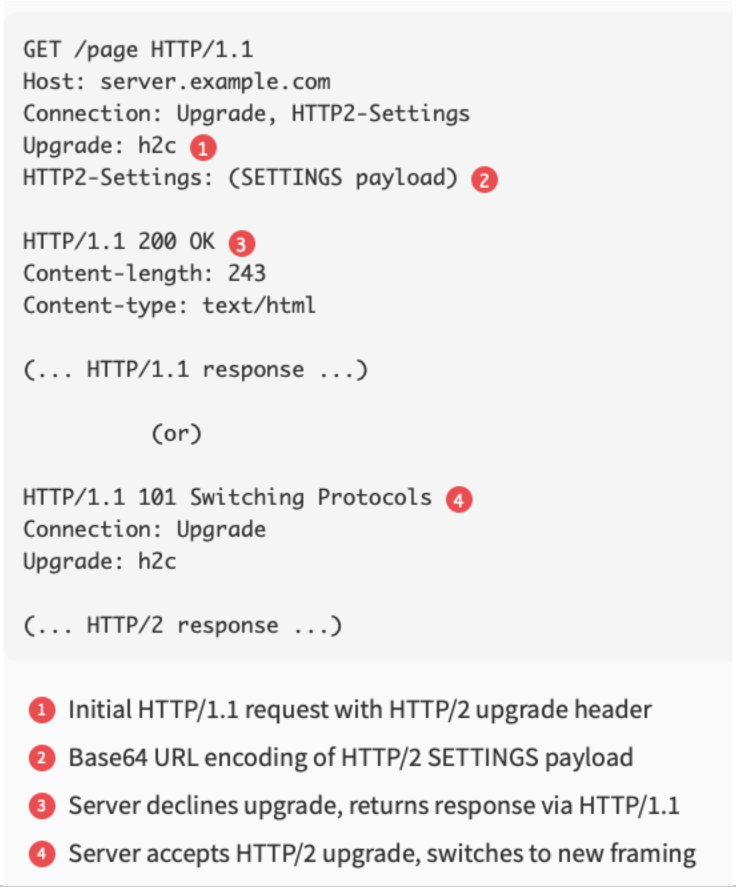
Header Compression



https://developers.google.com/web/fundamentals/performance/http2/

How HTTP/2?

Upgrade protocol



```
GET /page HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c ①
HTTP2-Settings: (SETTINGS payload) ②

HTTP/1.1 200 OK ③
Content-length: 243
Content-type: text/html

(... HTTP/1.1 response ...)

          (or)

HTTP/1.1 101 Switching Protocols ④
Connection: Upgrade
Upgrade: h2c

(... HTTP/2 response ...)
```

① Initial HTTP/1.1 request with HTTP/2 upgrade header

② Base64 URL encoding of HTTP/2 SETTINGS payload

③ Server declines upgrade, returns response via HTTP/1.1

④ Server accepts HTTP/2 upgrade, switches to new framing

https://hpbn.co/http2/

HTTP/2 Server

Apache 2.4.17+, NGINX 1.9.5+
http2 (Go), http-2 (Ruby),
Jetty (Java), Twisted (Python) Node.js (8.4.0+)

HTTP/2 for Web Developers

Undo HTTP 1.x optimisations

Server <= one domain is ok (no domain sharding), …
Connections <= many resources are ok, …
Browser <= no inline images, …
…

https://www.smashingmagazine.com/2016/02/getting-ready-for-http2/