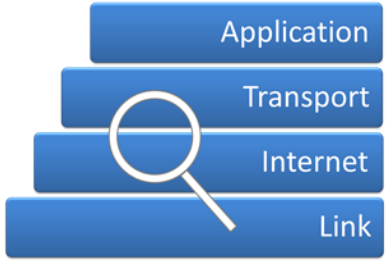
MindWorks UK

Technology Transfer Partners



New Protocols -- HTTP/2 and TLS v1.3

Joe Doupnik
Prof (ret.) Univ of Oxford
jrd@netlab1.net
jdoupnik@microfocus.com
Mindworksuk and Micro Focus

MICRO FOCUS

Bullet points about HTTP/2

- Formalized as RFC7540+7541 in 2015 and being widely deployed
- One continuous connecton between two machines, not one per fetch, thus fewer handshakes and similar overhead, more speed
- Multiplex data streams within that channel, no *head of line blocking*
- Headers are compressed binary for smaller size, but lots of them
- “Server Push”, sends without a request, we can set *H2Push off*
- Vendors really want proper TLS on the wire, to protect things and to satisfy the project’s mission creep

Some servers which support HTTP/2

- Apache v2.4.17, *pre-fork* Apache does not support HTTP/2 but its threaded versions (MPMs *worker* & *event*) do
- Nginx v1.95
- Tomcat v8.5, Jetty v9.3
- HAProxy v1.8
- MS IIS in Windows 10, Windows Server 2016, .NET core v2.2.x
- Node.js v5.0
- Many content delivery networks (CDNs) and major vendors
- Most browsers today support HTTP/2 (usually only over TLS)

HTTP/1.0 & 1.1 sequencing

- In original HTTP/1.0 each web request opens a new connection and closes it after the response. Requests form an orderly queue. The frequent open/close part is expensive and slow.
- HTTP/1.1 introduced header *keep-alive* to reduce the open/close frequency by letting successive requests reuse the same connection, yet request/responses still form an orderly queue.
- If a response were slow to arrive then the queue just waits and waits for it, twiddling its collective thumbs. *Head of line blocking*.
- Apache main body (top of /etc/apache2/default-server.conf)

```
KeepAlive on                 (default is on)
KeepAliveTimeout 180         (default is 5 seconds)
MaxKeepAliveRequests 50      (default is 100)
```

Multiplexing in HTTP/2

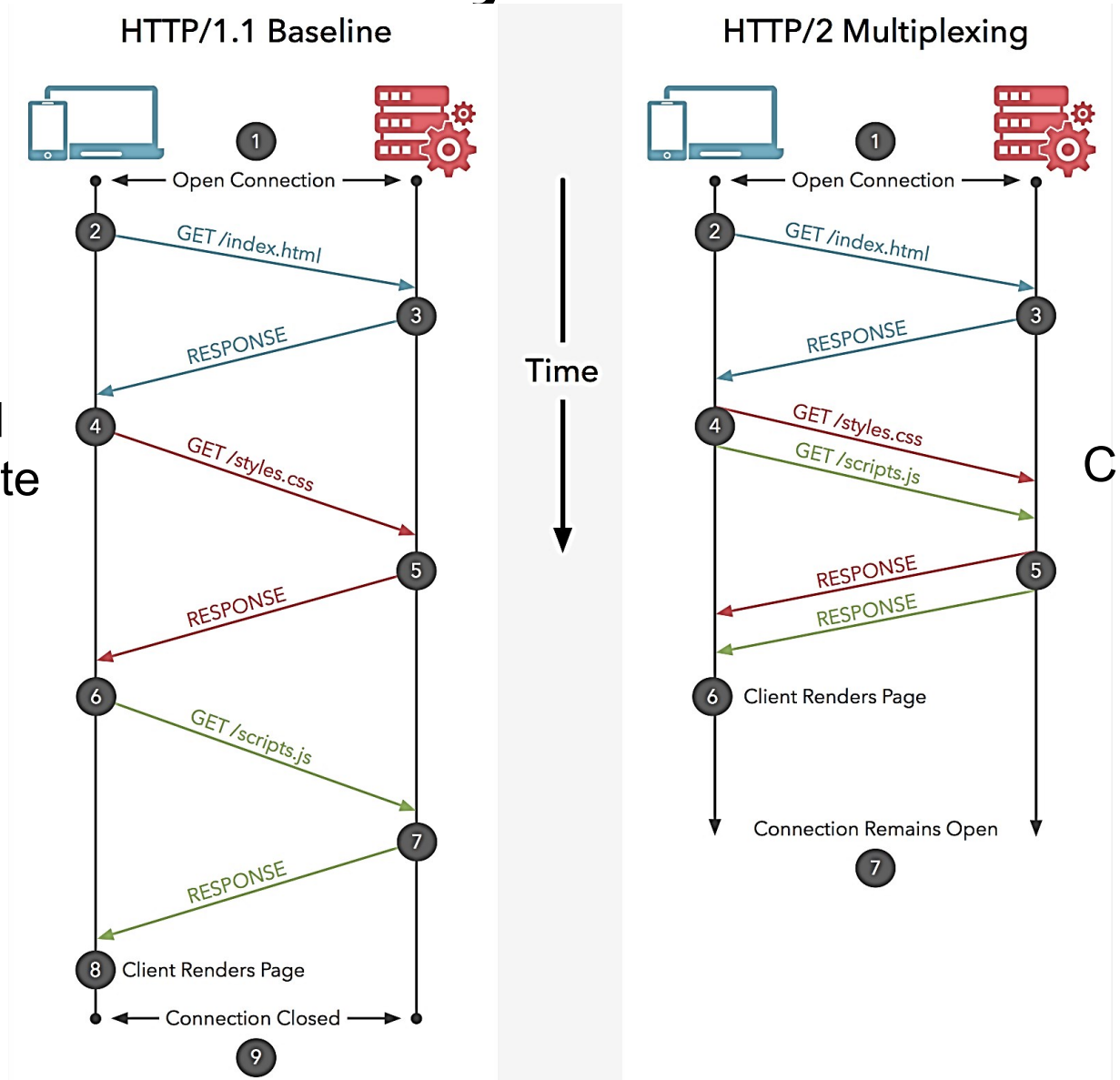
- HTTP/2 introduces a multiplexed channel
- Requests can be sent one after another over that single channel without waiting, and reponses arrive later, aka: *client greed*.
- The technique is number each request (and its response). Numbering reunites a response with its request.
- The numbers are called *streams* (invisible to applications)
- The queue is nearly a mob, bypassing slow request-response pairs. Applications may need to restore proper order. *H2Push* has many controls to impose some order on server sends.

Multiplexing also includes family members

- HTTP/2 permits a single TCP/IP connection to a server to carry traffic for more than one web virtual host, provided that --
  - The destination IP number remains the same as the current connection
  - The SSL certificate remains the same as for the current connection
- This means a client can talk to sundry web virtual severs at the same destination by using the current TCP/IP connection
- The facility is known as *connection coalescing*
- It can avoid extra connection overhead

A pictorial summary of HTTP 1.1 and 2

Polite and considerate



Charge!

From clip art

Wireshark: Firefox to Apache (HTTP2)

No.	Time	Source	Destination	Protocol	Length	Info
34	09:37:13.596277	leap.netlab1.net	desktop.netlab1.net	TCP	66	443 → 1805 [SYN, ACK] Seq=0 Ack=1 Win=2920…
35	09:37:13.596347	desktop.netlab1.net	leap.netlab1.net	TCP	54	1805 → 443 [ACK] Seq=1 Ack=1 Win=65700 Len…
36	09:37:13.601076	desktop.netlab1.net	leap.netlab1.net	TLSv1.2	571	Client Hello
37	09:37:13.601324	leap.netlab1.net	desktop.netlab1.net	TCP	60	443 → 1805 [ACK] Seq=1 Ack=518 Win=30336 L…
38	09:37:13.601995	leap.netlab1.net	desktop.netlab1.net	TLSv1.2	1514	Server Hello
39	09:37:13.602216	leap.netlab1.net	desktop.netlab1.net	TCP	1514	443 → 1805 [ACK] Seq=1461 Ack=518 Win=3033…
40	09:37:13.602217	leap.netlab1.net	desktop.netlab1.net	TLSv1.2	1230	Certificate [TCP segment of a reassembled …
41	09:37:13.602259	desktop.netlab1.net	leap.netlab1.net	TCP	54	1805 → 443 [ACK] Seq=518 Ack=4097 Win=6570…
42	09:37:13.604139	leap.netlab1.net	desktop.netlab1.net	TCP	1514	443 → 1805 [ACK] Seq=4097 Ack=518 Win=3033…
43	09:37:13.604140	leap.netlab1.net	desktop.netlab1.net	TLSv1.2	643	Certificate Status, Server Key Exchange, S…
44	09:37:13.604187	desktop.netlab1.net	leap.netlab1.net	TCP	54	1805 → 443 [ACK] Seq=518 Ack=6146 Win=6570…
45	09:37:13.612414	desktop.netlab1.net	leap.netlab1.net	TLSv1.2	147	Client Key Exchange, Change Cipher Spec, F…
46	09:37:13.612888	leap.netlab1.net	desktop.netlab1.net	TLSv1.2	105	Change Cipher Spec, Finished
47	09:37:13.613061	desktop.netlab1.net	leap.netlab1.net	HTTP2	231	Magic, SETTINGS[0], WINDOW_UPDATE[0], PRIO…
48	09:37:13.613088	leap.netlab1.net	desktop.netlab1.net	HTTP2	111	SETTINGS[0], WINDOW_UPDATE[0]
49	09:37:13.613109	desktop.netlab1.net	leap.netlab1.net	HTTP2	304	HEADERS[15]: GET /, WINDOW_UPDATE[15]
50	09:37:13.613290	leap.netlab1.net	desktop.netlab1.net	TCP	60	443 → 1805 [ACK] Seq=6254 Ack=1038 Win=325…
51	09:37:13.613455	desktop.netlab1.net	leap.netlab1.net	HTTP2	92	SETTINGS[0]
52	09:37:13.613663	leap.netlab1.net	desktop.netlab1.net	HTTP2	92	SETTINGS[0]
53	09:37:13.613851	leap.netlab1.net	desktop.netlab1.net	HTTP2	1514	HEADERS[15]: 200 OK
54	09:37:13.613853	leap.netlab1.net	desktop.netlab1.net	HTTP2	1514	DATA[15][SSL segment of a reassembled PDU]…
55	09:37:13.613867	desktop.netlab1.net	leap.netlab1.net	TCP	54	1805 → 443 [ACK] Seq=1076 Ack=9212 Win=657…
56	09:37:13.614057	leap.netlab1.net	desktop.netlab1.net	HTTP2	1514	DATA[15][SSL segment of a reassembled PDU]…
57	09:37:13.614058	leap.netlab1.net	desktop.netlab1.net	HTTP2	990	DATA[15][SSL segment of a reassembled PDU]
58	09:37:13.614059	leap.netlab1.net	desktop.netlab1.net	HTTP2	1514	DATA[15][SSL segment of a reassembled PDU]
59	09:37:13.614061	leap.netlab1.net	desktop.netlab1.net	HTTP2	498	DATA[15], DATA[15] (text/html)
60	09:37:13.614075	desktop.netlab1.net	leap.netlab1.net	TCP	54	1805 → 443 [ACK] Seq=1076 Ack=13512 Win=65…
61	09:37:13.653319	desktop.netlab1.net	leap.netlab1.net	HTTP2	165	HEADERS[17]: GET /ttp_logo_horizontal-02-m…
62	09:37:13.653521	desktop.netlab1.net	leap.netlab1.net	HTTP2	143	HEADERS[19]: GET /oxford-skyline-smallest.…
63	09:37:13.653702	desktop.netlab1.net	leap.netlab1.net	HTTP2	144	HEADERS[21]: GET /meetings.css, WINDOW_UPD…
64	09:37:13.653909	leap.netlab1.net	desktop.netlab1.net	TCP	60	443 → 1805 [ACK] Seq=13512 Ack=1276 Win=32…
65	09:37:13.654253	leap.netlab1.net	desktop.netlab1.net	HTTP2	1514	HEADERS[17]: 200 OK, HEADERS[19]: 200 OK

TLS handshake

HTTP2 “the menu please”

The menu arrives

Charge!
Client greed

Notes on HTTP/2 and Server Push

Can I just enable HTTP/2 and immediately start seeing performance benefits?

Some performance benefits will apply immediately, e.g: multiplexing more efficient binary protocol

Some benefits (e.g: effective use of push headers) will require change/optimization to take advantage of them

The community is still working out best practices

Upgrading is frictionless:

- Clients that can use HTTP/2 will do so
- Clients that don't support HTTP/2 can fall back to HTTP/1.1
- If you're using apache, enable mod_http2

How does server push work?

The server can send a frame called a *push promise*

The client can choose to accept or decline the asset

If the client accepts, the asset will be delivered as if it had been requested by the client and cached for future use

This could be used to help browsers to start pre-fetching/pre-caching assets such as javascript, css, images and web fonts before starting to parse the HTML of a web page, for example.

From https://www.phproundtable.com/episode/all-about-http2

See also https://w3c.github.io/preload/#link-element-interface-extensions

HTTP/2 Server Push (want fries with that?)

“8.2. Server Push

HTTP/2 allows a server to pre-emptively send (or "push") responses (along with corresponding "promised" requests) to a client in association with a previous client-initiated request. This can be useful when the server knows the client will need to have those responses available in order to fully process the response to the original request.”

“A client can request that server push be disabled, though this is negotiated for each hop independently. The

SETTINGS_ENABLE_PUSH

setting can be set to 0 to indicate that server push is disabled.”

From: RFC 7540

HTTP/2 Server Push, cont'd

"An intermediary can receive pushes from the server and choose not to forward them on to the client. In other words, how to make use of the pushed information is up to that intermediary. Equally, the intermediary might choose to make additional pushes to the client, without any action taken by the server.

A client cannot push."

From: RFC 7540

Hmmm. We might think about smart proxy interference and unwanted spam. This facility could be a problem, but the client can turn it off.

Server Push is one side of the aggresive coin; *client greed* is the other side.

Server Push in Apache, Link Header

If the connection supports PUSH, these two resources will be sent to the client. As a web developer, you may set these headers either directly in your application response or you configure the server via

```
<Location /xxx.html>
    Header add Link "</xxx.css>;rel=preload"
    Header add Link "</xxx.js>;rel=preload"
</Location>
```

If you want to use `preload` links without triggering a PUSH, you can use the `nopush` parameter, as in

```
Link </xxx.css>;rel=preload;nopush
```

or you may disable PUSHes for your server entirely with the directive

```
H2Push Off
```

And there is more:

https://httpd.apache.org/docs/2.4/howto/http2.html

See RFC 8288 about Link in HTML docs

Server Push in Apache, Early Hints

An alternative to PUSHing resources is to send `Link` headers to the client before the response is even ready. This uses the HTTP feature called "Early Hints" and is described in RFC 8297.

In order to use this, you need to explicitly enable it on the server via

```
H2EarlyHints on
```

(It is not enabled by default since some older browser tripped on such responses.)

If this feature is on, you can use the directive `H2PushResource` to trigger early hints and resource PUSHes:

```
<Location /xxx.html>
    H2PushResource /xxx.css
    H2PushResource /xxx.js
</Location>
```

This will send out a `"103 Early Hints"` response to a client as soon as the server *starts* processing the request. This may be much early than the time the first response headers have been determined, depending on your web application.

https://httpd.apache.org/docs/2.4/howto/http2.html

Server Push in Apache, alternatives

In Apache, H2 Push is on by default, and you can specify resources that should be pushed by either adding a link header to the response, or using the slightly better h2PushResource directive. Here's how that index.html example would look inside an Apache virtualhost or htaccess file:

```
<If "%{DOCUMENT_URI} == '/index.html'">
        H2PushResource add css/site.css
        H2PushResource add js/site.js
</If>
```

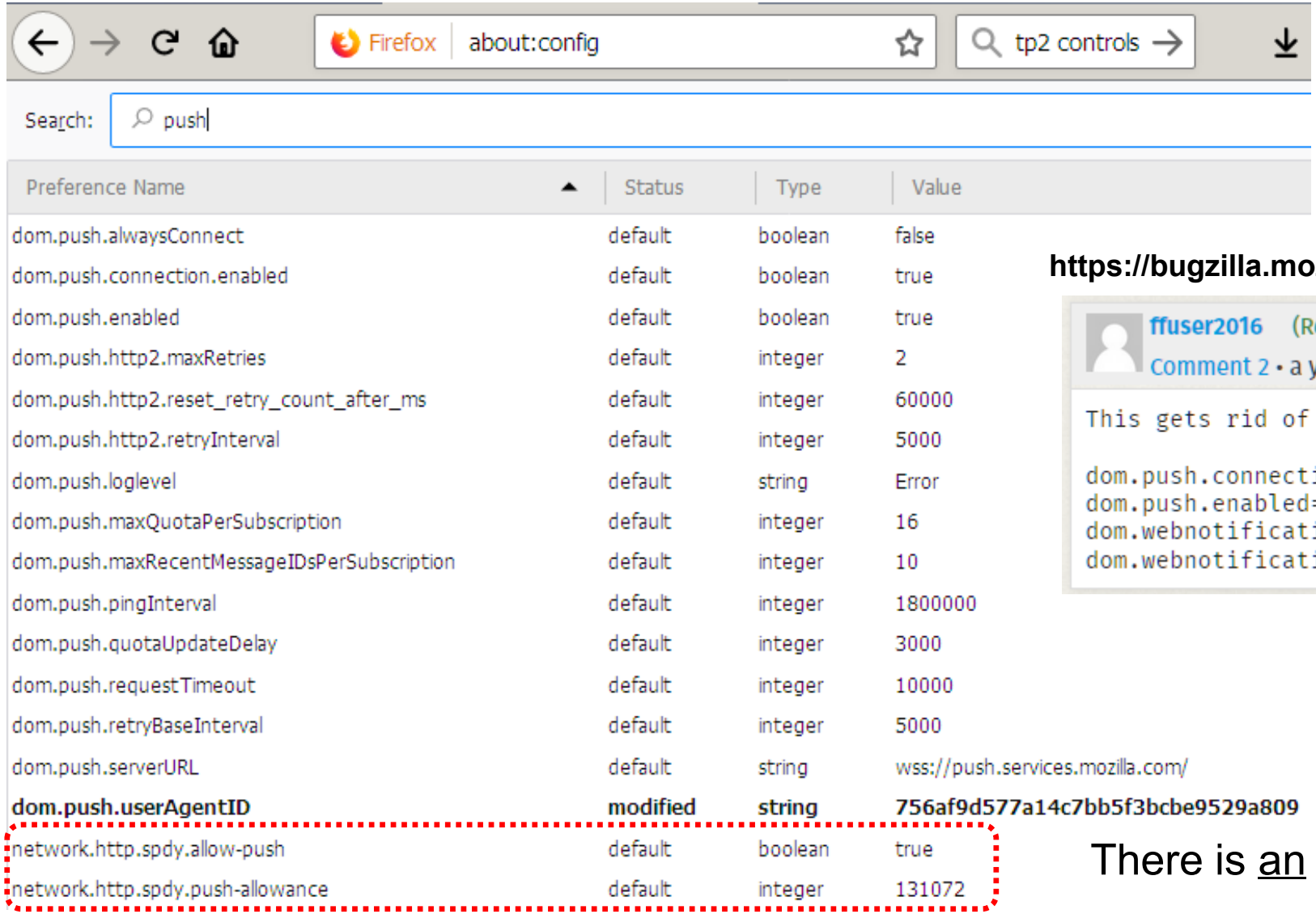
If ordered “index” burger:
add fries
add sauce

In English that says, "if the path to the responding document is /index.html, *push* these additional css and javascript files."

From: https://www.filamentgroup.com/lab/modernizing-delivery.html

Firefox HTTP/2 Push control in *about:config*

Firefox v62.0



Preference Name	Status	Type	Value
dom.push.alwaysConnect	default	boolean	false
dom.push.connection.enabled	default	boolean	true
dom.push.enabled	default	boolean	true
dom.push.http2.maxRetries	default	integer	2
dom.push.http2.reset_retry_count_after_ms	default	integer	60000
dom.push.http2.retryInterval	default	integer	5000
dom.push.loglevel	default	string	Error
dom.push.maxQuotaPerSubscription	default	integer	16
dom.push.maxRecentMessageIDsPerSubscription	default	integer	10
dom.push.pingInterval	default	integer	1800000
dom.push.quotaUpdateDelay	default	integer	3000
dom.push.requestTimeout	default	integer	10000
dom.push.retryBaseInterval	default	integer	5000
dom.push.serverURL	default	string	wss://push.services.mozilla.com/
dom.push.userAgentID	**modified**	**string**	**756af9d577a14c7bb5f3bcbe9529a809**
network.http.spdy.allow-push	default	boolean	true
network.http.spdy.push-allowance	default	integer	131072

https://bugzilla.mozilla.org/show_bug.cgi?id=1394407

ffuser2016 (Reporter)
Comment 2 • a year ago

```
This gets rid of push notification POPUPS:

dom.push.connection.enabled=false
dom.push.enabled=false
dom.webnotifications.serviceworkers.enabled=false
dom.webnotifications.enabled=false
```

There is an Off switch

Did that work? Picky clients want to know

8.1.4. Request Reliability Mechanisms in HTTP/2

In HTTP/1.1, an HTTP client is unable to retry a non-idempotent request when an error occurs because there is no means to determine the nature of the error. It is possible that some server processing occurred prior to the error, which could result in undesirable effects if the request were reattempted.

HTTP/2 provides two mechanisms for providing a guarantee to a client that a request has not been processed:

- The GOAWAY frame indicates the highest stream number that might have been processed. Requests on streams with higher numbers are therefore guaranteed to be safe to retry.

- The REFUSED_STREAM error code can be included in a RST_STREAM frame to indicate that the stream is being closed prior to any processing having occurred. Any request that was sent on the reset stream can be safely retried.

Requests that have not been processed have not failed; clients MAY automatically retry them, even those with non-idempotent methods.

From RFC7540. This could be complicated!

HTTP/2 + TLS fine print (an excerpt)

From RFC7540

9.2. Use of TLS Features

> Implementations of HTTP/2 MUST use TLS version 1.2 [TLS12] or higher for HTTP/2 over TLS. The general TLS usage guidance in [TLSBCP] SHOULD be followed, with some additional restrictions that are specific to HTTP/2.
>
> The TLS implementation MUST support the Server Name Indication (SNI) [TLS-EXT] extension to TLS. HTTP/2 clients MUST indicate the target domain name when negotiating TLS.
>
> Deployments of HTTP/2 that negotiate TLS 1.3 or higher need only support and use the SNI extension; deployments of TLS 1.2 are subject to the requirements in the following sections. Implementations are encouraged to provide defaults that comply, but it is recognized that deployments are ultimately responsible for compliance.

HTTP/2 with TLS must use TLS v1.2 or higher, not v1.1 nor 1.0

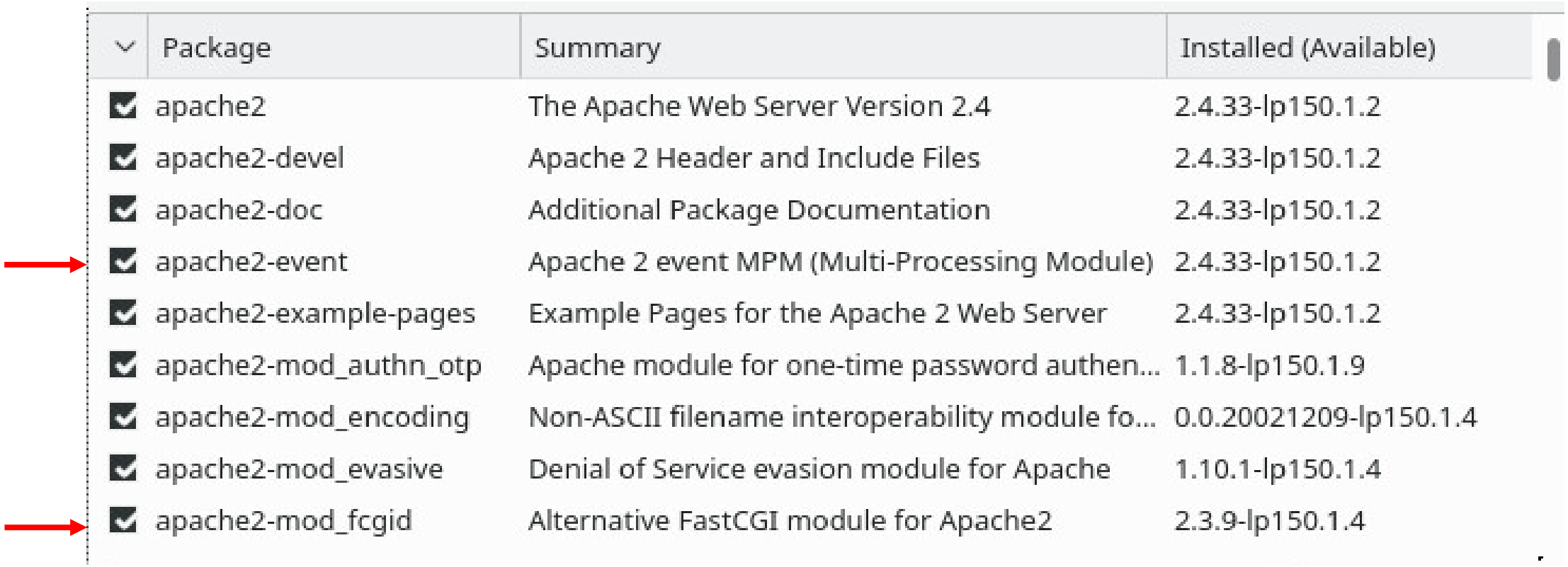
Note the SNI (Server Name Indication) item, shown in later slides

HTTP/2 with Apache, a how-to

- Five easy steps, with some comments in between
- Tweak a few configuration files
- Retains normal HTTP/1.1 behaviour
- Requires recent Apache (v2.4.17 or later)
- SLES15 & Leap15 support it, as shown in following tests

Step 1, add Apache modules

Leap15: YaST | Software Management | apache2



Package	Summary	Installed (Available)
apache2	The Apache Web Server Version 2.4	2.4.33-lp150.1.2
apache2-devel	Apache 2 Header and Include Files	2.4.33-lp150.1.2
apache2-doc	Additional Package Documentation	2.4.33-lp150.1.2
apache2-event	Apache 2 event MPM (Multi-Processing Module)	2.4.33-lp150.1.2
apache2-example-pages	Example Pages for the Apache 2 Web Server	2.4.33-lp150.1.2
apache2-mod_authn_otp	Apache module for one-time password authen...	1.1.8-lp150.1.9
apache2-mod_encoding	Non-ASCII filename interoperability module fo...	0.0.20021209-lp150.1.4
apache2-mod_evasive	Denial of Service evasion module for Apache	1.10.1-lp150.1.4
apache2-mod_fcgid	Alternative FastCGI module for Apache2	2.3.9-lp150.1.4

Install both *event* and *mod_fcgid* items.
mod_fcgid is to support PHP

Apache module nuances

- The *event* and *worker* MPMs run as **threaded** processors, not as a new process per request (such as with standard *pre-fork*)
- Embedded language interpreters (such as for PHP) are usually not thread-safe. Thus we use a *helper module* which separates Apache threads from language interpreters
- Mod_fcgid is such a helper which works well, shown here
- Mod_cgi is a much older helper
- Service php-fpm is a new standalone systemd helper daemon
- *Event* plus *fcgid* also works with older Apaches (say v2.2.x)

Apache module mod_fcgid note

“mod_fcgid is a high performance alternative to mod_cgi or mod_cgid, which starts a sufficient number instances of the CGI program to handle concurrent requests, and these programs remain running to handle further incoming requests.

It is favored by the PHP developers, for example, as a preferred alternative to running mod_php in-process, delivering very similar performance.”

From http://httpd.apache.org/mod_fcgid/

Step 2, sysconfig updates

/etc/sysconfig/apache2 snippets

```
APACHE_MODULES="actions alias ssl auth_basic authn_core authn_file authz_host authz_groupfile authz_core authz_
user autoindex cgi dir env expires include rewrite header ldap authnz_ldap log_config mime negotiation setenvif
 socache_shmcb userdir reqtimeout proxy proxy_html proxy_ajp proxy_connect xml2enc status version http2 fcgid "

APACHE_SERVER_FLAGS="SSL HTTP2 STATUS"

APACHE_MPM="event"
```

Remove *php* or *php5* or *php7* from list APACHE_MODULES=
because they conflict with threaded Apache when using *http2*.
fcgid substitutes for them instead (and does it well).
Note early appearance of *ssl* to avoid later dependency problems.

Apache support for HTTP/2 needs to use MPM *worker* or *event*,
not default *pre-forked*. *Event* is the better choice.

Step 3, fcgid configuration

/etc/apache2/conf.d/mod_fcgid.conf

```
## PHP via FastCGI
##
## uncomment the following line if you want to handle php via mod_fcgid
## see http://httpd.apache.org/mod_fcgid/mod/mod_fcgid.html#examples
##
<FilesMatch "\.php$">
    AddHandler fcgid-script .php
    Options +ExecCGI
##JRD WAS  FcgidWrapper /srv/www/cgi-bin/php5 .php
    FcgidWrapper /srv/www/cgi-bin/php .php
</FilesMatch>
## Added by JRD. Do not buffer output, allow long running sessions
    FcgidOutputBufferSize 0
    FcgidMaxRequestLen 25600000
    FcgidMaxRequestsPerProcess 0
    FcgidBusyTimeout 86400

</IfModule>
# End of <IfModule fcgid_module>
```

Edit as shown

Also see https://httpd.apache.org/mod_fcgid/mod/mod_fcgid.html about parameters FcgidOutputBufferSize, FcgidBusyTimeout etc and startup /etc/php<n>/fastcgi/php.ini
For Apache v2.2.x, omit the Fcgid prefix on names of parameters

PHP programs run fine as-is. Perl programs may need to use CGI:: et al.

PHP with standalone systemd service php-fpm

1. File /etc/sysconfig/apache2 -- add proxy and proxy_fcgi, omit php5/7:

```
APACHE_MODULES="actions alias ssl auth_basic authn_core authn_file authz_host authz_groupfile
 authz_core authz_user autoindex cgi dir env expires include rewrite header ldap authnz_ldap
log_config mime negotiation setenvif socache_shmcb userdir reqtimeout proxy proxy_html proxy_
ajp proxy_connect xml2enc status version proxy_fcgi http2"
```

2. /etc/apache2/conf.d/example.conf -- use proxying with php_fpm
(There are many variations on the proxy theme, see Apache docs for details)

```
Alias /config /home/search/config.php
<Directory /home/search>
        Options +ExecCGI
        Require ip 82.70.37.210/24 10.0.0.1/24 127.0.0.1
</Directory>
<Location "/config">
        ProxyPass "fcgi://127.0.0.1:9000/home/search/config.php" flushpackets=on keepalive=on
        ProxyPassReverse "fcgi://127.0.0.1:9000/home/search/config.php"
</Location>
```

3. /etc/php<digit>/fpm -- copy *.default files to non-.default file names, modify file …/php-fpm.d/www.conf to specify UID/GID of web server:

```
; Unix user/group of processes
; Note: The user is mandatory. If the group is not set, the default user's group
;       will be used.
;;WAS user = nobody
;;WAS group = nobody
user = wwwrun
group = www
```

Enable systemd service php-fpm

/etc/apache2/default-server.conf

```
# Global configuration that will be applicable for all virtual hosts, unless
# deleted here, or overriden elswhere.
#
# OCSP Stapling, only in httpd 2.3.3 and later
SSLUseStapling          on
SSLStaplingResponderTimeout 5
SSLStaplingReturnResponderErrors off
SSLStaplingCache  shmcb:/var/run/ocsp(128000)
SSLStrictSNIVHOSTCheck  on
<IfModule mod_http2.c>
        Protocols h2 h2c http/1.1
        H2Upgrade on
</IfModule>

DocumentRoot "/srv/www/htdocs"
```

Edit as shown

Certificate stapling

SNI, require clients to state server DNS name

Define offered HTTP protocols, in order.
SUSE: defined in /etc/apache2/protocols.conf

Certificate **stapling** is optional but beneficial: server not client verifies signing CA.
SNI = Server Name Indication, server name requested by client, good

h2 = HTTP/2 over TLS, **h2c** = HTTP/2 over clear text, **http/1.1** = normal way
Most browsers insist upon TLS when using HTTP/2

Protocols statement

HTTP/2 in a VirtualHost context (TLS only)

```
Protocols h2 http/1.1
```

Allows HTTP/2 negotiation (h2) via TLS ALPN in a secure <VirtualHost>. HTTP/2 preamble checking (Direct mode, see H2Direct) is disabled by default for h2.

HTTP/2 in a Server context (TLS and cleartext)

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

Allows HTTP/2 negotiation (h2) via TLS ALPN for secure <VirtualHost>. Allows HTTP/2 cleartext negotiation (h2c) upgrading from an initial HTTP/1.1 connection or via HTTP/2 preamble checking (Direct mode, see H2Direct).

This is **ALPN**: Application Layer Protocol Negotiation, meaning kinds of HTTP protocol ALPN data is conveyed within TLS handshake Options frames

From https://httpd.apache.org/docs/2.4/mod/mod_http2.html

Note on h2c upgrade (HTTP/2 without TLS)

Upgrade on request body

The h2c Upgrade dance will not work on requests that have a body. Those are PUT and POST requests (form submits and uploads). If you write a client, you may precede those requests with a simple GET or an OPTIONS * to trigger the upgrade.

The reason is quite technical in nature, but in case you want to know: during Upgrade, the connection is in a half insane state. The request is coming in HTTP/1.1 format and the response is being written in HTTP/2 frames. If the request carries a body, the server needs to read the whole body before it sends a response back. Because the response might need answers from the client for flow control among other things. But if the HTTP/1.1 request is still being sent, the client is unable to talk HTTP/2 yet.

In order to make behaviour predictable, several server implementors decided to not do an Upgrade in the presence of any request bodies, even small ones.

From https://icing.github.io/mod_h2/howto.html#debugging
Interpretation: h2c is unlikely to work in today's environment

MindWorks UK

A virtual host (/etc/apache2/vhosts.d/vhost-ssl.conf)

```
<VirtualHost _default_:443>

        #  General setup for the virtual host
        DocumentRoot "/srv/www/htdocs"
        ServerName leap.netlab1.net:443
        #ServerName www.example.com:443
        #ServerAdmin webmaster@example.com
        ErrorLog /var/log/apache2/error.log
        TransferLog /var/log/apache2/access.log

        #   SSL Engine Switch:
        #   Enable/Disable SSL for this virtual host.
        SSLEngine on
        SSLCompression          off
        SSLSessionTickets       off

        #   You can use per vhost certificates if SNI is supported
        SSLCertificateFile /etc/ssl/private/123-certs/new/servercert.pem
        SSLCertificateKeyFile /etc/ssl/private/123-certs/private-key.pem
        SSLCertificateChainFile /etc/ssl/private/123-certs/new/sfig2_bundle.crt

        #   Per-Server Logging:
        #   The home of a custom SSL log file. Use this when you want a
        #   compact non-error SSL logfile on a virtual host basis.
        CustomLog /var/log/apache2/ssl_request.log   ssl_combined

</VirtualHost>
```

ServerName for SNI testing

Let file */etc/apache2/ssl-global.conf* define cipher suites and protocols

Certificates for <u>this</u> vhost

Step 0, all done!

/etc/apache2/ssl-global.conf used as-is

The cipher suites part, not adjusted. Not shown here is defining certs globally. Note default selection of TLS v1.2 only, no TLS v1.1 nor v1.0. Add if wanted.

```
        #  SSL protocols
        #  Supporting TLS only is adequate nowadays
        SSLProtocol TLSv1.2

        #   SSL Cipher Suite:
        #   List the ciphers that the client is permitted to negotiate.
        #   See the mod_ssl documentation for a complete list.
        SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:EC
DHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-AES128-GCM-SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SH
A256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-
AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-DSS-AES1
28-SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-AES256-SHA:DHE-RSA-AES256-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES12
8-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:AES:CAMELLIA:DES-CBC3-SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PS
K:!aECDH:!EDH-DSS-DES-CBC3-SHA:!EDH-RSA-DES-CBC3-SHA:!KRB5-DES-CBC3-SHA

        #   SSLHonorCipherOrder
        #   If SSLHonorCipherOrder is disabled, then the client's preferences
        #   for chosing the cipher during the TLS handshake are used.
        #   If set to on, then the above SSLCipherSuite is used, in the order
        #   given, with the first supported match on both ends.
        SSLHonorCipherOrder on
```

RFC7540: “Cipher suite must list TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 [TLS-ECHDE] with P-256 elliptic curve [FIPS186].” The above list meets these requirements for HTTP/2.

Curl test: curl -v https://host

top part

To HTTP/2 adapted Apache web server

```
# curl -v https://leap.netlab1.net
* Rebuilt URL to: https://leap.netlab1.net/
*   Trying 82.70.37.213...
* TCP_NODELAY set
* Connected to leap.netlab1.net (82.70.37.213) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Client hello (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
* ALPN, server accepted to use h2
* Server certificate:
*  subject: OU=Domain Control Validated; CN=*.netlab1.net
*  start date: Apr  7 07:00:08 2018 GMT
*  expire date: Apr  7 07:00:08 2020 GMT
*  subjectAltName: host "leap.netlab1.net" matched cert's "*.netlab1.net"
*  issuer: C=US; ST=Arizona; L=Scottsdale; O=Starfield Technologies, Inc.; (
Starfield Secure Certificate Authority - G2
*  SSL certificate verify ok.
```

Curl test: curl -v https://host

cont'd

```
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade: len=0
* Using Stream ID: 1 (easy handle 0x564f4106f170)
> GET / HTTP/2
> Host: leap.netlab1.net
> User-Agent: curl/7.60.0
> Accept: */*
>
* Connection state changed (MAX_CONCURRENT_STREAMS == 100)!
< HTTP/2 200
< date: Mon, 13 Aug 2018 13:00:39 GMT
< server: Apache
< last-modified: Sat, 11 Aug 2018 17:57:17 GMT
< etag: "15c-5732c974b86a9"
< accept-ranges: bytes
< content-length: 348
< content-type: text/html
<
<!DOCTYPE html>
<html lang="en-GB">
<head>
        <meta name="robots" content="noarchive">
```

> from Curl
< from server

The web page appears

Curl to http://host, curl did not try HTTP/2

```
# curl -v http://leap.netlab1.net/
*   Trying 82.70.37.213...
* TCP_NODELAY set
* Connected to leap.netlab1.net (82.70.37.213) port 80 (#0)
> GET / HTTP/1.1
> Host: leap.netlab1.net
> User-Agent: curl/7.60.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 16 Aug 2018 09:07:35 GMT
< Server: Apache
< Upgrade: h2,h2c
< Connection: Upgrade
< Last-Modified: Sat, 11 Aug 2018 17:57:17 GMT
< ETag: "15c-5732c974b86a9"
< Accept-Ranges: bytes
< Content-Length: 348
< Content-Type: text/html
<
<!DOCTYPE html>
```

Server (<) made these offers
Curl (>) simply ignored them

The web page appears

Curl, to http:// with “force start as HTTP/2”

```
# curl -v --http2-prior-knowledge http://leap.netlab1.net/
*   Trying 82.70.37.213...
* TCP_NODELAY set
* Connected to leap.netlab1.net (82.70.37.213) port 80 (#0)
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade: len=0
* Using Stream ID: 1 (easy handle 0x55c809865160)
> GET / HTTP/2
> Host: leap.netlab1.net
> User-Agent: curl/7.60.0
> Accept: */*
>
* Connection state changed (MAX_CONCURRENT_STREAMS == 100)!
< HTTP/2 200
< date: Thu, 16 Aug 2018 07:13:04 GMT
< server: Apache
< last-modified: Sat, 11 Aug 2018 17:57:17 GMT
< etag: "15c-5732c974b86a9"
< accept-ranges: bytes
< content-length: 348
< content-type: text/html
<
<!DOCTYPE html>
<html lang="en-GB">
<head>
        <meta name="robots" content="noarchive">
```

The web page appears

Curl, to http:// with “try upgrading to HTTP/2”

```
# curl -v --http2 http://leap.netlab1.net/
*   Trying 82.70.37.213...
* TCP_NODELAY set
* Connected to leap.netlab1.net (82.70.37.213) port 80 (#0)
> GET / HTTP/1.1
> Host: leap.netlab1.net
> User-Agent: curl/7.60.0
> Accept: */*
> Connection: Upgrade, HTTP2-Settings
> Upgrade: h2c
> HTTP2-Settings: AAMAAABkAARAAAAAAAIAAAAA
>
< HTTP/1.1 101 Switching Protocols
< Upgrade: h2c
< Connection: Upgrade
* Received 101
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade: len=28
* Connection state changed (MAX_CONCURRENT_STREAMS == 100)!
< HTTP/2 200
< date: Thu, 01 Jan 1970 00:00:00 GMT
< server: Apache
< last-modified: Sat, 11 Aug 2018 17:57:17 GMT
< etag: W/"15c-5732c974b86a9"
< accept-ranges: bytes
< content-length: 348
< content-type: text/html
<
<!DOCTYPE html>
```

These are client actions; shy clients will not use them

Curl (>) suggests upgrade

Server (<) agrees

The web page appears

Curl, https://host to a non-HTTP/2 server

```
# curl  -v https://netlab1.net/
*   Trying 82.70.37.210...
* TCP_NODELAY set
* Connected to netlab1.net (82.70.37.210) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Client hello (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
* ALPN, server did not agree to a protocol
* Server certificate:
*  subject: OU=Domain Control Validated; CN=*.netlab1.net
*  start date: Apr  7 07:00:08 2018 GMT
*  expire date: Apr  7 07:00:08 2020 GMT
*  subjectAltName: host "netlab1.net" matched cert's "netlab1.net"
*  issuer: C=US; ST=Arizona; L=Scottsdale; O=Starfield Technologies
d Secure Certificate Authority - G2
*  SSL certificate verify ok.
> GET / HTTP/1.1
> Host: netlab1.net
> User-Agent: curl/7.60.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 16 Aug 2018 09:31:09 GMT
```

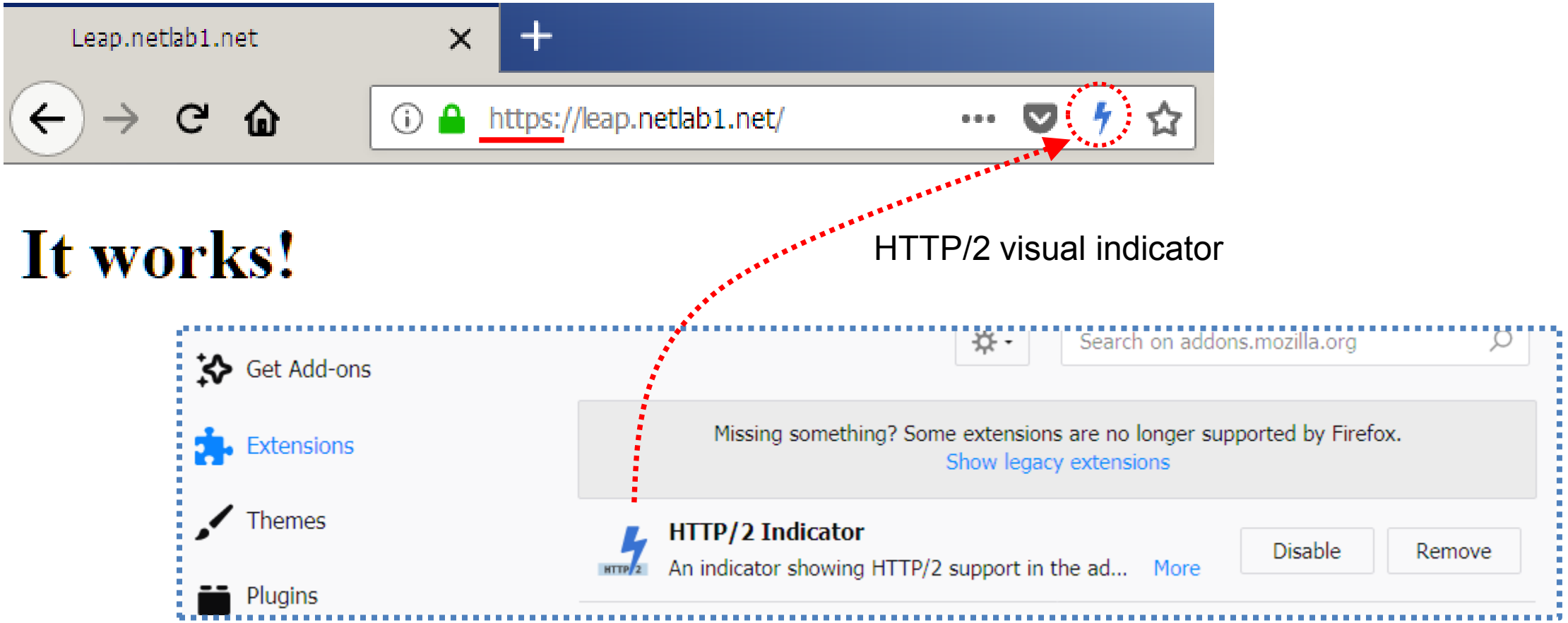
Curl client ALPN offerings

Server supports https://
but not HTTP/2 nor ALPN

Faithful HTTP/1.1 works

The web page appears

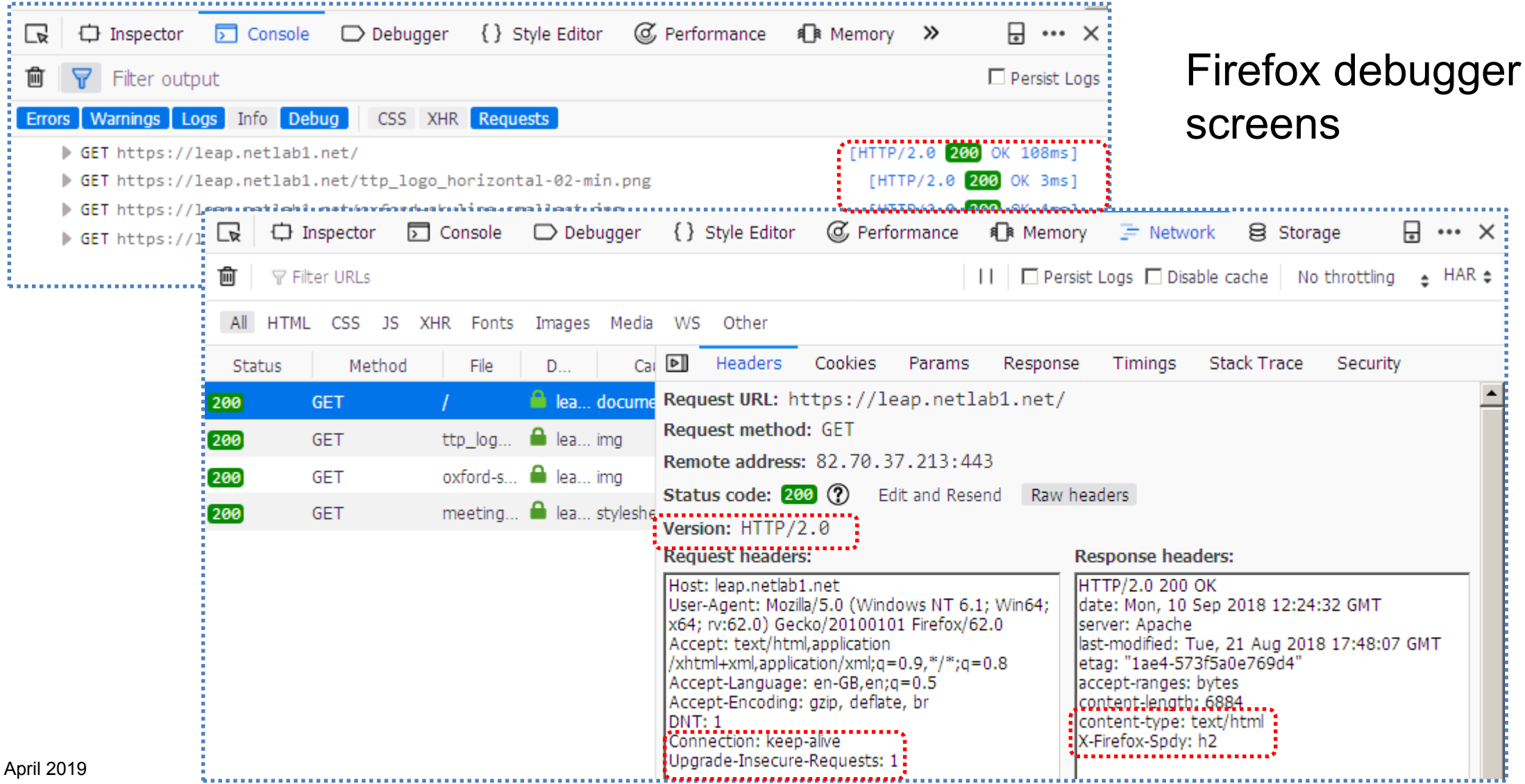
Firefox test to Apache (HTTP/2 adjusted)



Apache v2.4.33 server on Leap15, not built for TLS v1.3

MinnWorks UK

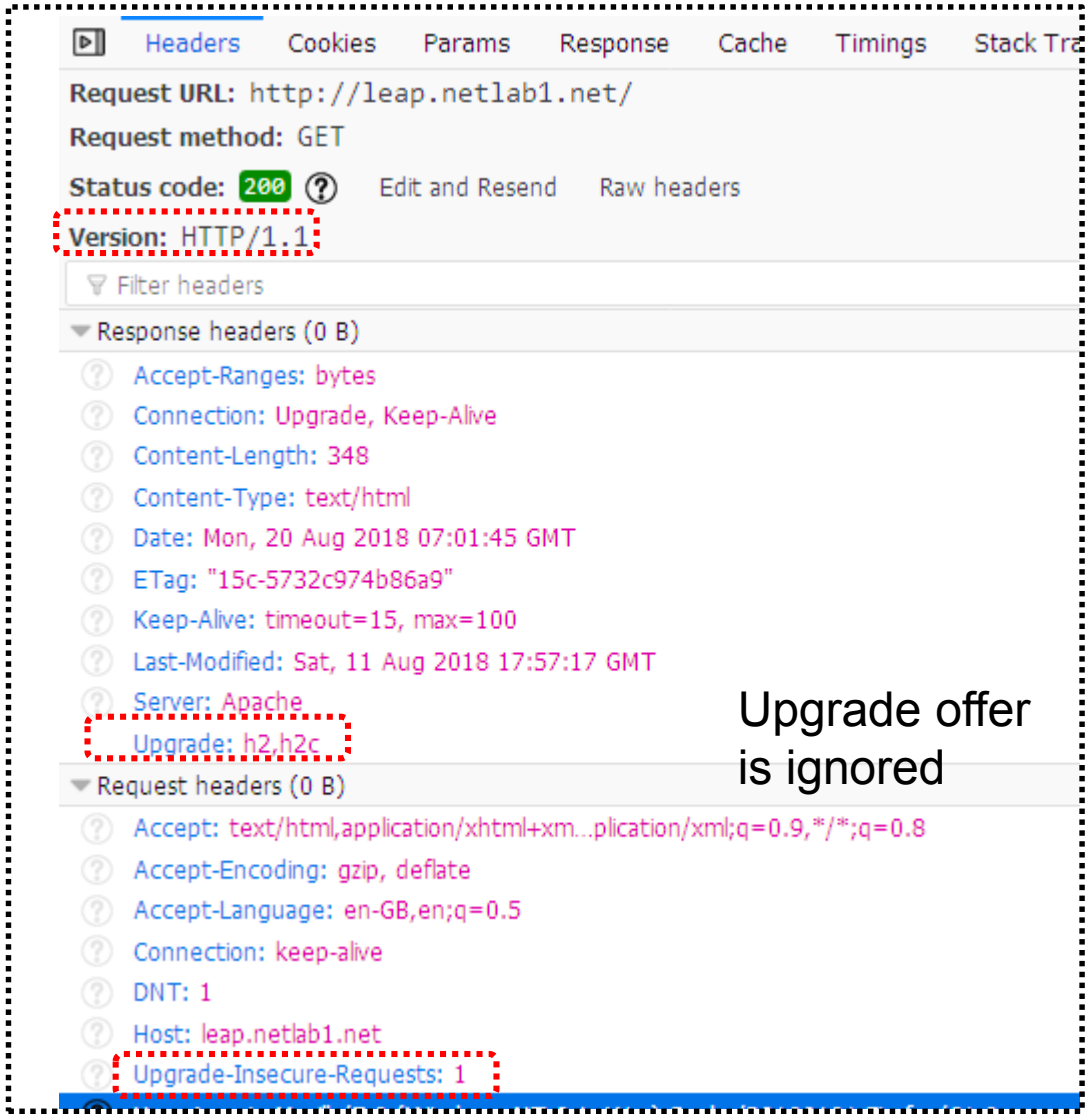
Firefox https:// to Apache (HTTP/2 capable)



Firefox debugger screens

Firefox to Apache http://host, uses HTTP/1.1

No HTTP/2 indicator



Upgrade offer is ignored

Some vendors are refusing to support HTTP/2 unless it is run over TLS.
Firefox people are amongst them.
Mission creep can be costly.

TLS v1.3 "Free at last" ™ML King

RFC8446, published August 2018, over 4 years in the making

In openssl v1.1.1 (or equivalents) or later, is v1.1.0 ABI compatible

Allowed cipher suites (not TLS v1.2 compatible names), mandatory

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256

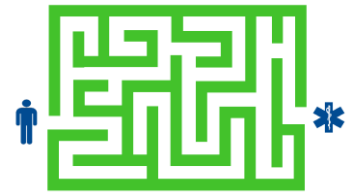
plus optional

- TLS_AES_128_CCM_8_SHA256
- TLS_AES_128_CCM_SHA256

(Hint: out with old weak algorithms, in with newer <weak?> ones)

Fields formatted to "look like" TLS v1.2 to help middle boxes

TLS v1.3 aspects



- TLS v1.3 “sessions” do not start until the handshake completes successfully, thus a possible time gap between them
- Handshake is now better protected against intruders, and complex
- Perfect forward secrecy (PFS, change secrets often) designed in
- No renegotiation of crypto properties, no SSL compression
- No DSA certificates but RSA certificate is better protected in transit
- Allow fallback to TLS v1.2 if cannot negotiate v1.3
- TLS version & HTTP ALPN happen during the handshake phase
- Programming with openssl for TLS v1.3 is complicated
- Visit https://www.feistyduck.com and obtain OpenSSL Cookbook

Testssl.sh to a test program


```
# ./testssl.sh --openssl /usr/bin/openssl leap.netlab1.net:9410

 Using "OpenSSL 1.1.1  11 Sep 2018" [~165 ciphers]
 on leap:/usr/bin/openssl
 (built: "Sep 14 12:26:25 2018", platform: "linux-x86_64")


 Testing protocols via sockets except NPN+ALPN

 SSLv2      not offered (OK)
 SSLv3      not offered (OK)
 TLS 1      offered
 TLS 1.1    offered
 TLS 1.2    offered (OK)
 TLS 1.3    offered (OK): final
 NPN/SPDY   not offered
 ALPN/HTTP2 h2, http/1.1 (offered)

 Testing cipher categories

 NULL ciphers (no encryption)                  not offered (OK)
 Anonymous NULL Ciphers (no authentication)    not offered (OK)
 Export ciphers (w/o ADH+NULL)                 not offered (OK)
 LOW: 64 Bit + DES encryption (w/o export)     not offered (OK)
 Weak 128 Bit ciphers (SEED, IDEA, RC[2,4])    not offered (OK)
 Triple DES Ciphers (Medium)                   not offered (OK)
 High encryption (AES+Camellia, no AEAD)       offered (OK)
 Strong encryption (AEAD ciphers)              offered (OK)
```

testssl.sh is from https://testssl.sh/

Being considerate, program offers all TLS versions: 1.0, 1.1, 1.2, 1.3

Testssl.sh to a test program


```
 Testing robust (perfect) forward secrecy, (P)FS -- omitting Null Authentication/Encryption, 3DES, RC4

 PFS is offered (OK)          TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 ECDHE-RSA-AES256-GCM-SHA384
                              ECDHE-RSA-AES256-SHA384 ECDHE-RSA-AES256-SHA ECDHE-RSA-CAMELLIA256-SHA384
                              TLS_AES_128_GCM_SHA256 TLS_AES_128_CCM_SHA256 TLS_AES_128_CCM_8_SHA256
                              ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-SHA
                              ECDHE-RSA-CAMELLIA128-SHA256
 Elliptic curves offered:     prime256v1 secp384r1 secp521r1 X25519 X448


 Testing server preferences

 Has server cipher order?     yes (OK)
 Negotiated protocol          default proto empty
 Negotiated cipher            default cipher empty, 253 bit ECDH (X25519)
 Cipher order
    TLSv1:     ECDHE-RSA-AES128-SHA ECDHE-RSA-AES256-SHA AES128-SHA AES256-SHA CAMELLIA256-SHA CAMELLIA128-SHA
    TLSv1.1:   ECDHE-RSA-AES128-SHA ECDHE-RSA-AES256-SHA AES128-SHA AES256-SHA CAMELLIA256-SHA CAMELLIA128-SHA
    TLSv1.2:   ECDHE-RSA-AES128-GCM-SHA256 ECDHE-RSA-AES256-GCM-SHA384 ECDHE-RSA-AES128-SHA256 ECDHE-RSA-AES128-SHA
               ECDHE-RSA-AES256-SHA384 ECDHE-RSA-AES256-SHA AES128-GCM-SHA256 AES256-GCM-SHA384 AES128-SHA256
               AES256-SHA256 AES128-SHA AES256-SHA AES256-CCM8 AES256-CCM AES128-CCM8 AES128-CCM
               ECDHE-RSA-CAMELLIA256-SHA384 ECDHE-RSA-CAMELLIA128-SHA256 CAMELLIA256-SHA256 CAMELLIA128-SHA256
               CAMELLIA256-SHA CAMELLIA128-SHA
    TLSv1.3:   TLS_AES_128_GCM_SHA256 TLS_AES_256_GCM_SHA384 TLS_CHACHA20_POLY1305_SHA256 TLS_AES_128_CCM_SHA256
               TLS_AES_128_CCM_8_SHA256
```

testssl.sh v2.9dev program known glitches

Crypto mumbo-jumbo, but all TLS versions are supported
TLS v1.3 was negotiated

Testssl.sh to a test program


```
 Testing vulnerabilities

 Heartbleed (CVE-2014-0160)                not vulnerable (OK), no heartbeat extension
 CCS (CVE-2014-0224)                       not vulnerable (OK)
 Ticketbleed (CVE-2016-9244), experiment.  --   (applicable only for HTTPS)
 ROBOT                                     not vulnerable (OK)
 Secure Renegotiation (CVE-2009-3555)      not vulnerable (OK)
 Secure Client-Initiated Renegotiation     not vulnerable (OK)
 CRIME, TLS (CVE-2012-4929)                not vulnerable (OK) (not using HTTP anyway)
 POODLE, SSL (CVE-2014-3566)               not vulnerable (OK)
 TLS_FALLBACK_SCSV (RFC 7507)              Downgrade attack prevention supported (OK)
 SWEET32 (CVE-2016-2183, CVE-2016-6329)    not vulnerable (OK)
 FREAK (CVE-2015-0204)                     not vulnerable (OK)
 DROWN (CVE-2016-0800, CVE-2016-0703)      not vulnerable on this host and port (OK)
                                           make sure you don't use this certificate elsewhere with SSLv2 enabled services
                                           https://censys.io/ipv4?q=2CBF6148D97A6DA5C1F267E10481F405C7426420DA6DE1909694C2

C8299A1C8D could help you to find out
 LOGJAM (CVE-2015-4000), experimental      not vulnerable (OK): no DH EXPORT ciphers, no DH key detected
 BEAST (CVE-2011-3389)                     TLS1: ECDHE-RSA-AES128-SHA ECDHE-RSA-AES256-SHA AES128-SHA AES256-SHA
                                                 CAMELLIA256-SHA CAMELLIA128-SHA
                                           VULNERABLE -- but also supports higher protocols  TLSv1.1 TLSv1.2 (likely mitig
ated)
 LUCKY13 (CVE-2013-0169), experimental     potentially VULNERABLE, uses cipher block chaining (CBC) ciphers with TLS. Chec
k patches
 RC4 (CVE-2013-2566, CVE-2015-2808)        no RC4 ciphers detected (OK)
```

Good results, but as expected TLS v1.0 uses weak ciphers (yellow)

Curl to a test program

```
# curl -v --CApath /etc/ssl/certs https://leap.netlab1.net:9410/
*   Trying 82.70.37.213...
* TCP_NODELAY set
* Connected to leap.netlab1.net (82.70.37.213) port 9410 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: none
  CApath: /etc/ssl/certs
* (304) (OUT), TLS handshake, Client hello (1):
* (304) (IN), TLS handshake, Server hello (2):
* (304) (IN), TLS Unknown, Certificate Status (22):
* (304) (IN), TLS handshake, Unknown (8):
* (304) (IN), TLS Unknown, Certificate Status (22):
* (304) (IN), TLS handshake, Certificate (11):
* (304) (IN), TLS Unknown, Certificate Status (22):
* (304) (IN), TLS handshake, CERT verify (15):
* (304) (IN), TLS Unknown, Certificate Status (22):
* (304) (IN), TLS handshake, Finished (20):
* (304) (OUT), TLS change cipher, Client hello (1):
* (304) (OUT), TLS Unknown, Certificate Status (22):
* (304) (OUT), TLS handshake, Finished (20):
* SSL connection using unknown / TLS_AES_256_GCM_SHA384
* ALPN, server accepted to use h2
* Server certificate:
*  subject: OU=Domain Control Validated; CN=*.netlab1.net
*  start date: Apr  7 07:00:08 2018 GMT
```

(304) == TLSv1.3

Built with openssl v1.1.1

Note: ALPN statement about HTTP versions is done within TLS options negotiation

```
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer a
* (304) (OUT), TLS Unknown, Unknown (23):
* (304) (OUT), TLS Unknown, Unknown (23):
* (304) (OUT), TLS Unknown, Unknown (23):
* Using Stream ID: 1 (easy handle 0x55de1689f240)
* (304) (OUT), TLS Unknown, Unknown (23):
> GET / HTTP/2
> Host: leap.netlab1.net:9410
> User-Agent: curl/7.60.0
> Accept: */*
>
```

Session setup

Uses both HTTP/2 and TLS v1.3

Openssl s_client to a test program

```
# openssl s_client --CApath /etc/ssl/certs -alpn h2,h2c,http/1.1 -connect leap.netlab1.net:9410
```

```
---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: RSA-PSS
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 4404 bytes and written 404 bytes
Verification: OK
---
New, TLSv1.3, Cipher is TLS_AES_128_GCM_SHA256
Server public key is 2048 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
ALPN protocol: h2
Early data was not sent
Verify return code: 0 (ok)
```

Using openssl v1.1.1
Must state the ALPN set for http kind checking

It does TLS v1.3

It speaks HTTP/2

No “pushy“ stuff from my server

Read more about HTTP/2

https://mozilla.github.io/server-side-tls/ssl-config-generator/

https://icing.github.io/mod_h2/howto.html

https://http2.github.io/faq/

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

https://legacy.gitbook.com/book/bagder/http2-explained/details

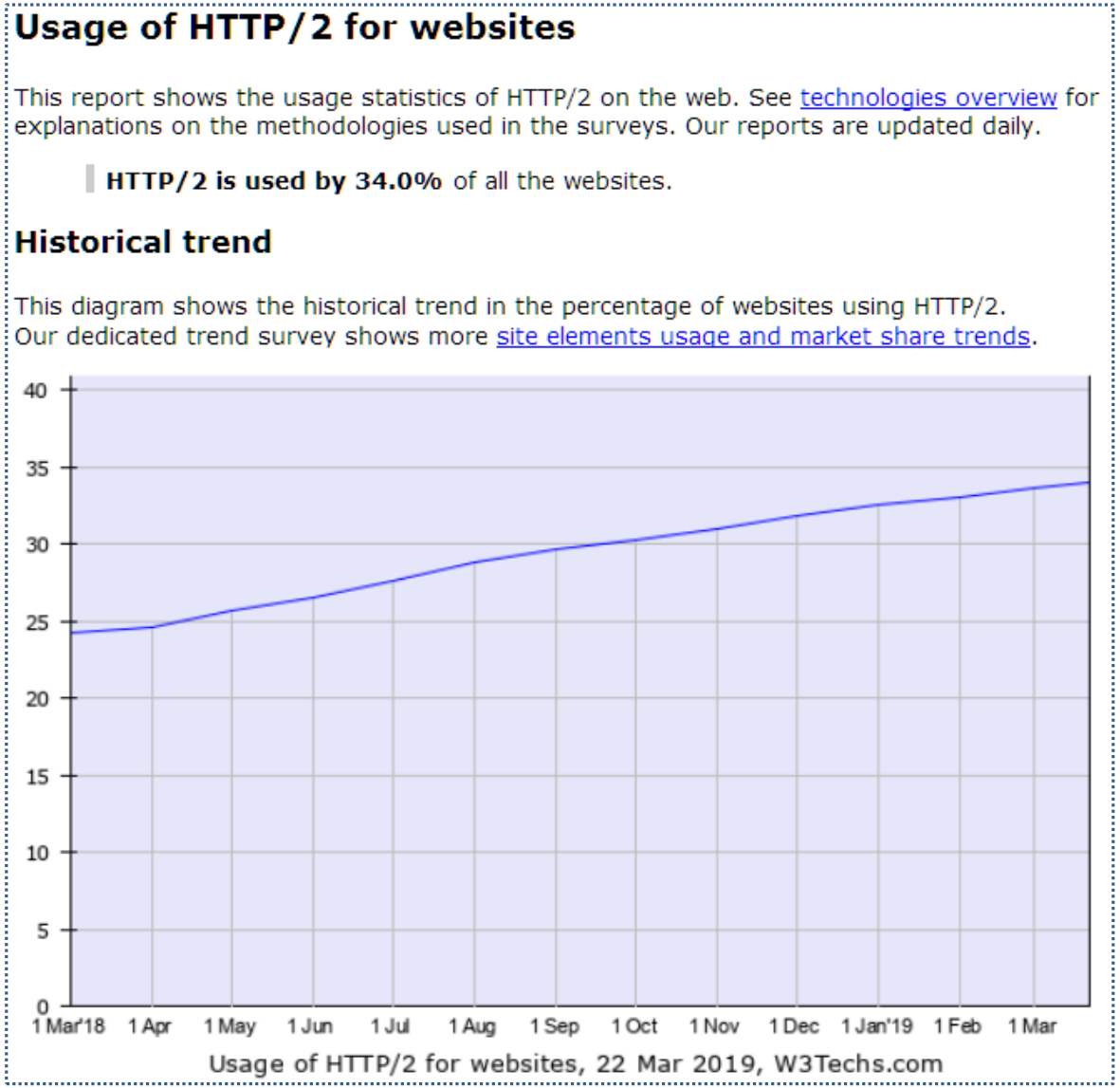
https://calendar.perfplanet.com/2016/http2-push-the-details/

https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/

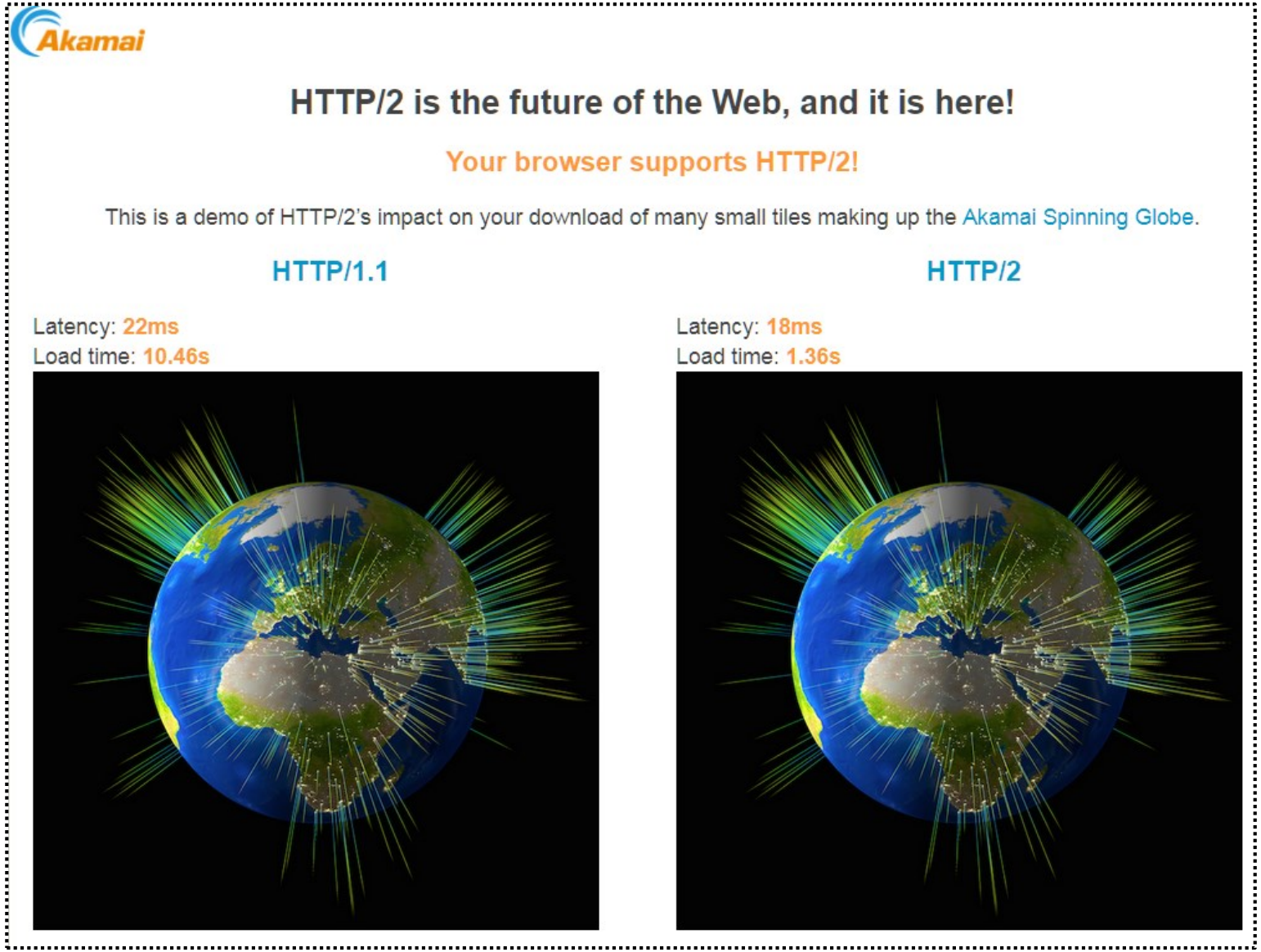
RFC7540 and 7541

https://nghttp2.org & https://github.com/nghttp2/nghttp2

https://w3techs.com/technologies/details/ce-http2/all/all (has lots of industry usage information, incl graph -->)



https://http2.akamai.com/demo to Firefox



Clear browser cache between runs

10.4 vs 1.36 seconds for many 3KB files. It is a demo corner case

A more realistic comparison, UK to …

Country	h1 time	h2 time	% reduction
Argentina	4.78	3.57	34.0%
New Zealand	5.19	4.05	28.2%
Japan	4.70	3.73	26.2%
China (Beijing)	5.40	4.64	16.5%
South Africa	4.19	3.80	10.2%
United Kingdom	0.94	0.92	2.7%

Time is in seconds

“The page tested (signout) is very typical of BBC web pages so I’d expect the results to be similar across other pages from BBC Online.”

From https://medium.com/bbc-design-engineering/http-2-is-easy-just-turn-it-on-34baad2d1fb1

Still awake? Vocabulary drill items

- h2, h2c, http/1.1
- Curl
- ALPN
- SNI
- Server Push
- TLS v1.3
- Certificate Stapling
- Perfect Forward Secrecy
- Multiplexing (no, not the cinema variety)

The near future

- HTTP/2 is deployable now, if a relevant web server is used (such as Apache 2.4.17 or later)
- TLS v1.3 is brand new. Openssl and similar vendors are adjusting to the release RFC. This will take time. Apache v2.4.36 supports it.
- TLS dependent applications will likely require rebuilding for a changed SSL library. Openssl v1.1.1 was released on 11 Sept 2018.
- Thus anticipate Apache 2.4.36 with TLS 1.3, and in due course for SSH, Postfix and other SSL applications
- Expect development of much more “streaming”/multiplexing of traffic
- Writing application code directly for HTTP2 & TLSv1.3 is complicated

MindWorks UK

MindWorks Inc. Ltd
210 Burnley Road
Weir
Bacup
OL13 8QE UK

Telephone: +44 (0) 170 687 1900
Fax: +44 (0) 170 687 8203
Web: www.mindworksuk.com
Email: training@mindworksuk.com