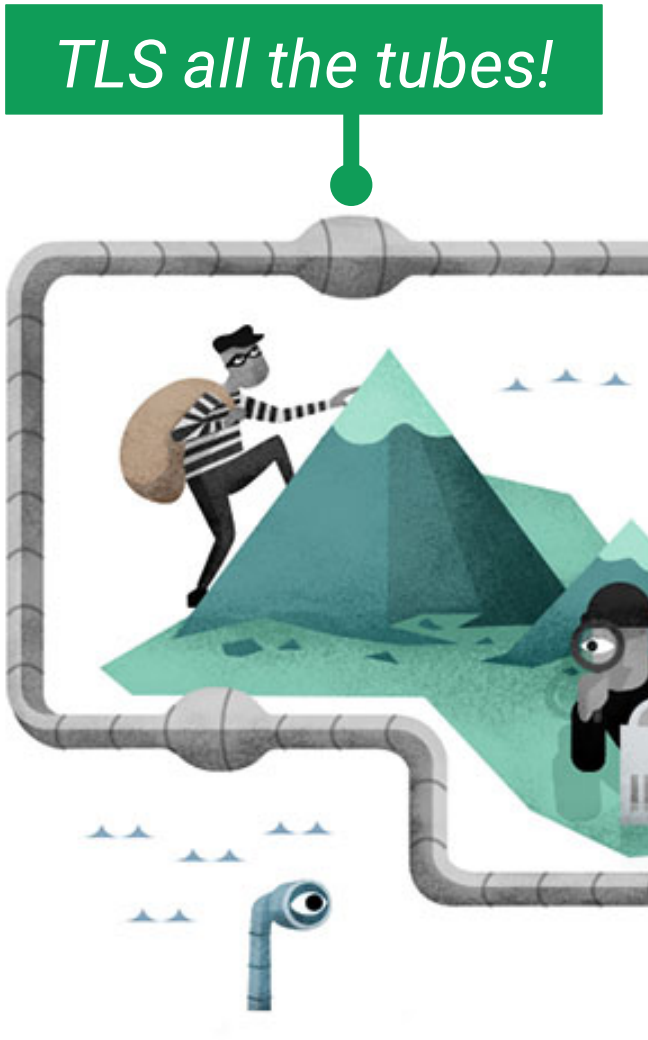
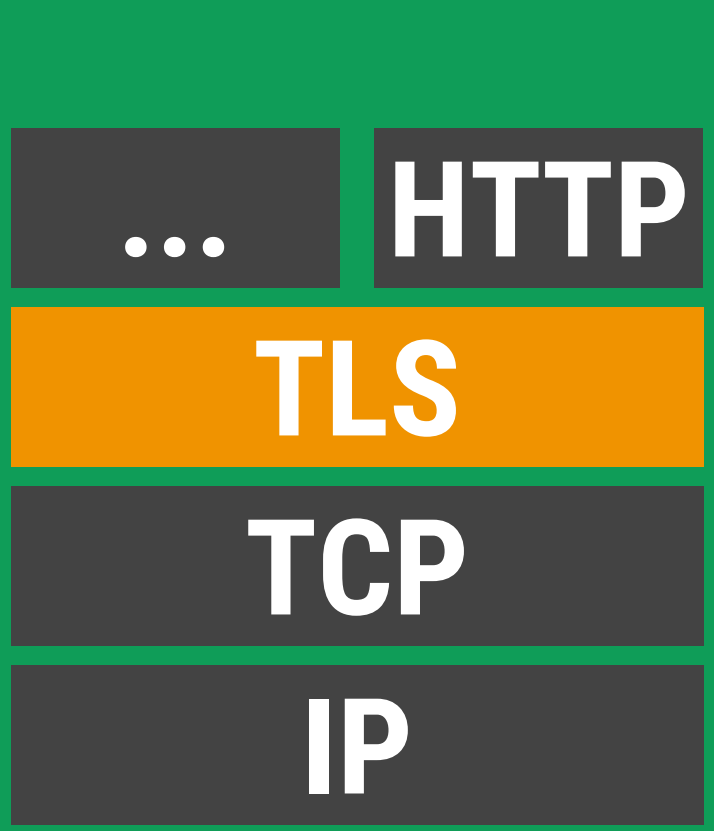
Is TLS Fast Yet?

Yes, it can be. *Let's take a peek under the hood...*

+Ilya Grigorik
@igrigorik





Authentication
identity verification of server + client

Data integrity
protection against malicious middlemen

Encryption
privacy of exchanged communication

Transport Layer Security

HTTPS-100 @ Google

- *Move all existing services to HTTPS only*
- *All new services deployed as HTTPS only*
- *All data encrypted in transit and at rest*

*We're not there yet, but we're making **rapid** progress.*

Google | Webmaster Central Blog

Official news on crawling and indexing sites for the Google index

HTTPS as a ranking signal

Posted: Wednesday, August 06, 2014

Webmaster level: all

Security is a top priority for Google. We invest a lot in making sure that our services use industry-leading security, like strong HTTPS encryption by default. That means that people using Search, Gmail and Google Drive, for example, automatically have a secure connection to Google.

Beyond our own stuff, we're also working to make the Internet safer more broadly. A big part of that is making sure that websites people access from Google are secure. For instance, we have created resources to help webmasters prevent and fix security breaches on their sites.

We want to go even further. At Google I/O a few months ago, we called for "HTTPS everywhere" on the web.

We've also seen more and more webmasters adopting HTTPS (also known as HTTP over TLS, or Transport Layer Security), on their website, which is encouraging.

For these reasons, over the past few months we've been running tests taking into account whether sites use secure, encrypted connections as a signal in our search ranking algorithms. We've seen positive results, so we're starting to use HTTPS as a ranking signal. For now it's only a very lightweight signal — affecting fewer than 1% of global queries, and carrying less weight than other signals such as high-quality content — while we give webmasters time to switch to HTTPS. But over time, we may decide to strengthen it, because we'd like to encourage all website owners to switch from HTTP to HTTPS to keep everyone safe on the web.

"Lightweight signal to start... but we may decide to strengthen it".

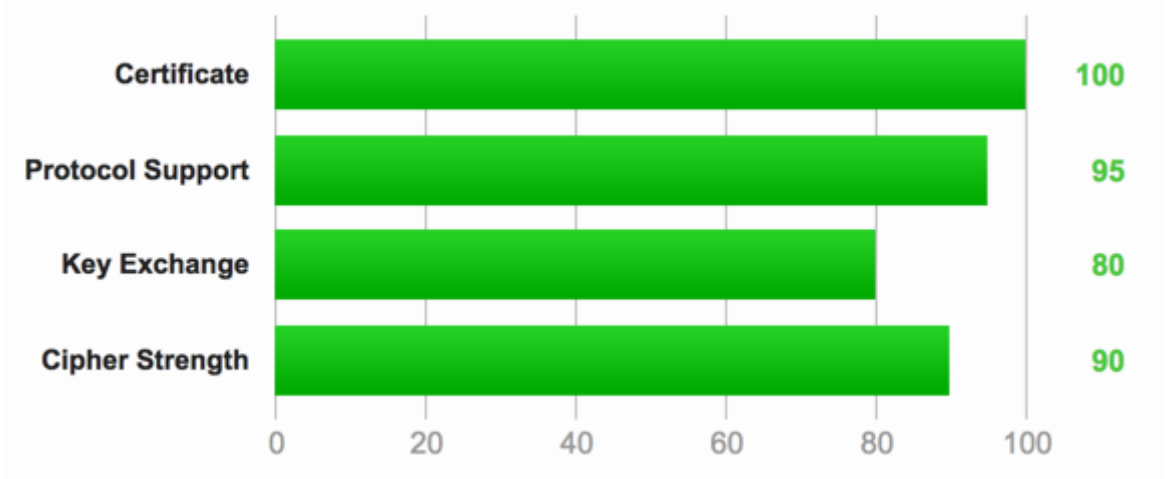
TLS has exactly one performance problem: it is not used widely enough.

Everything else can be optimized.

Following along? Do this...

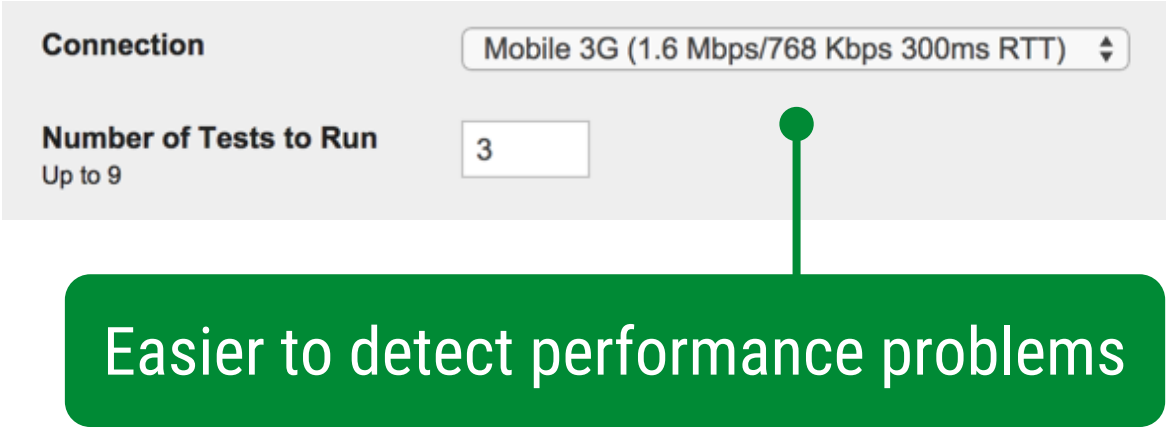
Verify your TLS configuration

- https://www.ssllabs.com/ssltest/



Optimize TLS performance

- http://www.webpagetest.org/
  - Use 300ms RTT profile

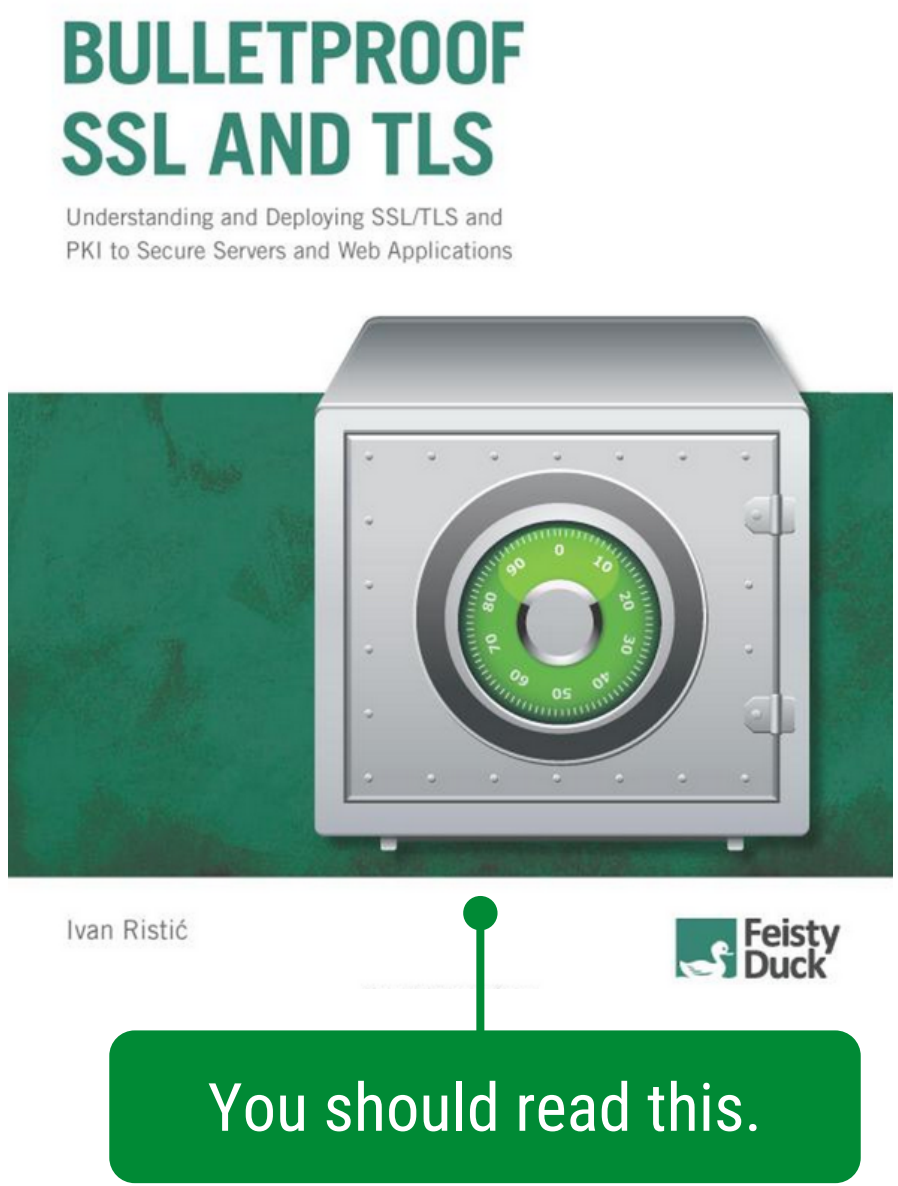


Verify your TLS configuration

- **You should have “A” rating or higher.**
  - https://www.ssllabs.com/ssltest/

- **Follow the recommendations for**
  - protocol support
  - ciphersuite list
  - key strength
  - etc.

Optimal configuration will vary based on your visitors - e.g. support for older clients, etc.



Optimize TLS performance

Let’s take a peek under the hood...

First, let's get the basics right…

Upgrade to latest kernel (3.7+)

- Lots of TCP performance improvements

Upgrade to latest OpenSSL (1.0.1j+)

- Security patches, performance improvements

Upgrade to latest server build

- Security, feature, performance improvements

* https://wiki.mozilla.org/Security/Server_Side_TLS

Computational costs

Asymmetric crypto (public key)

- O(1 ms) per handshake - expensive, relatively speaking.
- Used for the TLS handshake.

Tip: do fewer handshakes!

Symmetric crypto

- 150Mbps+ per core with sha256 and 1024 byte blocks (on my laptop).
- Used to encrypt application data.

```
# upgrade to latest
$> openssl version

# run benchmarks on own hardware
$> openssl speed sha ecdh
```

“We have deployed TLS at a large scale using both hardware and software load balancers. We have found that modern software-based TLS implementations running on ***commodity CPUs are fast enough to handle heavy HTTPS traffic load*** *without needing to resort to dedicated cryptographic hardware.”*

Doug Beaver, Facebook.

“On our production frontend machines, ***SSL/TLS accounts for less than 1% of the CPU load, less than 10 KB of memory per connection and less than 2% of network overhead.*** *Many people believe that SSL/TLS takes a lot of CPU time and we hope the preceding numbers will help to dispel that.”*

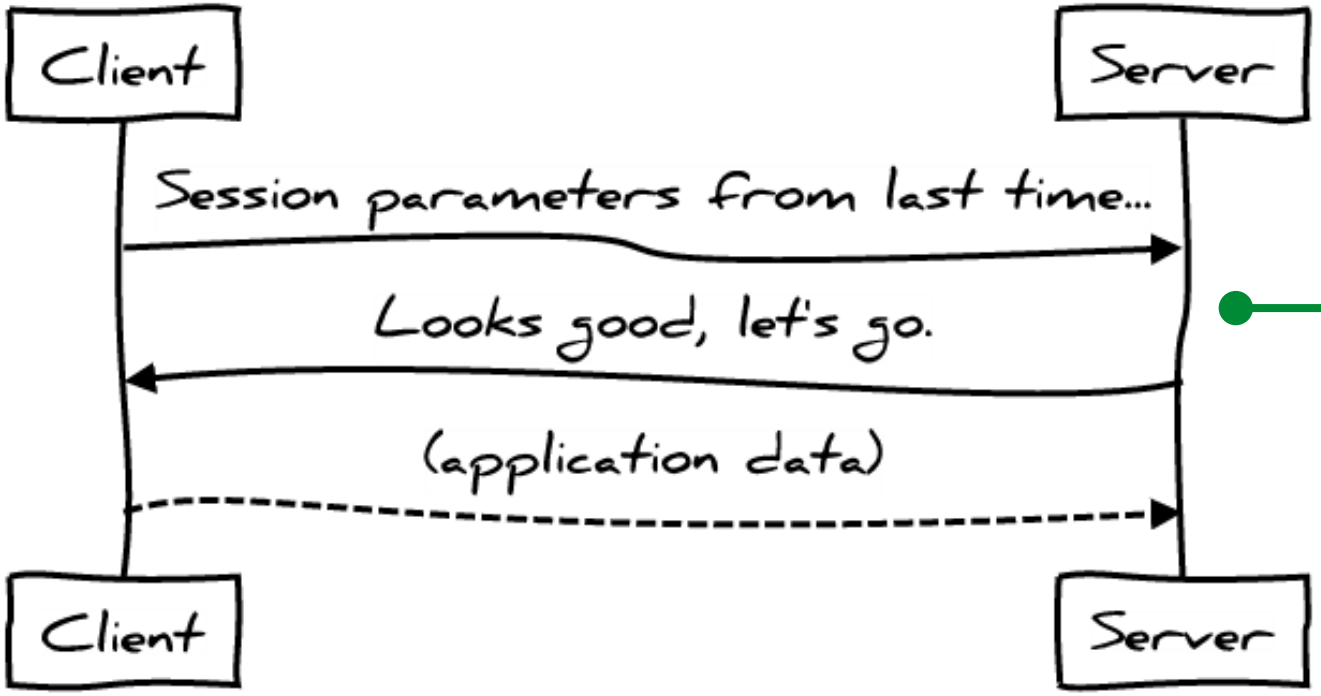
Adam Langley, Google.

Elliptic Curve Ephemeral Diffie-Hellman… enables Forward Secrecy.

*“In practical deployment, we found that enabling and prioritizing **ECDHE** cipher suites actually caused negligible increase in CPU usage. **HTTP keepalives and session resumption mean that most requests do not require a full handshake, so handshake operations do not dominate our CPU usage**.”*

Jacob Hoffman-Andrews, Twitter.

TLS resumption 101



Re-use negotiated parameters for the symmetric cipher

- Eliminates asymmetric crypto on the server via reuse of parameters
- Eliminates full roundtrip, allowing 1-RTT connection establishment

TLS Resumption

Session identifiers

Shared state is on the server

- Server assigns session ID
- Server caches parameters
- Client sends session ID
- Session is resumed

Session tickets

Shared state is on the client

- Server encrypts parameters
- Server sets opaque ticket to client
- Client sends opaque ticket on reconnect
- Server decrypts ticket and resumes session

TLS handshake with session resumption...

```
$> openssl s_client -connect example.com:443 -tls1 -tlsextdebug -status

SSL-Session:
  Protocol  : TLSv1
  Cipher    : RC4-SHA
  Session-ID: 8BE63F4825DDE238E0FE7574D7637080D1278537ECD783512872BFD6FDFB861E
  Session-ID-ctx:
  Master-Key: 2FA185F11A791EFB5BA24847FA448B7A0CE73F2D095191F949A35F68CE40FD4EC389E025CCD75
  Key-Arg   : None
  TLS session ticket lifetime hint: 600 (seconds)
  TLS session ticket:
  0000 - e4 34 51 9b 4c 13 9d ec-1f 1a 5a ea 89 c6 1f a7   .4Q.L.....Z.....
  0010 - b7 d5 25 4e 20 56 b6 00-c2 8d ce 6c 06 8b c9 ff   ..%N V.....l....
  (snip)
```

Session Identifier

Session Ticket

- You can enable both: older clients may not support session tickets
- Most servers support both, check the docs for configuration options

A few things to think about…

1. **Session identifiers**
   a. Require a shared cache between servers for best results
   b. Sessions must be expired and rotated in a secure manner

2. **Session tickets**
   a. Require a shared ticket encryption key
   b. Shared encryption key must be rotated in a secure manner

Disclaimer...

Shared (e.g. server cluster) session identifiers and ticket encryption keys require careful deployment best practices to provide Perfect Forward Secrecy.

https://www.imperialviolet.org/2011/11/22/forwardsecret.html

Do this at home...

a. Is your session cache large enough?
   i. *Apache provides stats via mod_status*
   ii. *Add logging for others, process logs.*

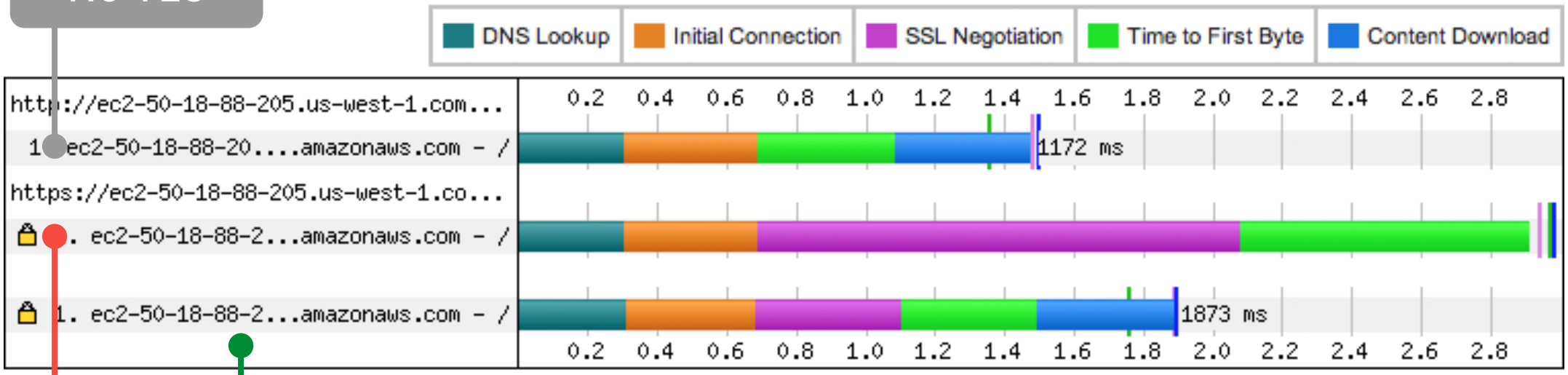
b. What is the ticket timeout?
   i. *Most server defaults are too low (~300s)*
   ii. *Most sites can use ~1 day*

Optimizing latency

*Your worst case should be **one** extra RTT!*

How many RTTs does your handshake incur?

No TLS

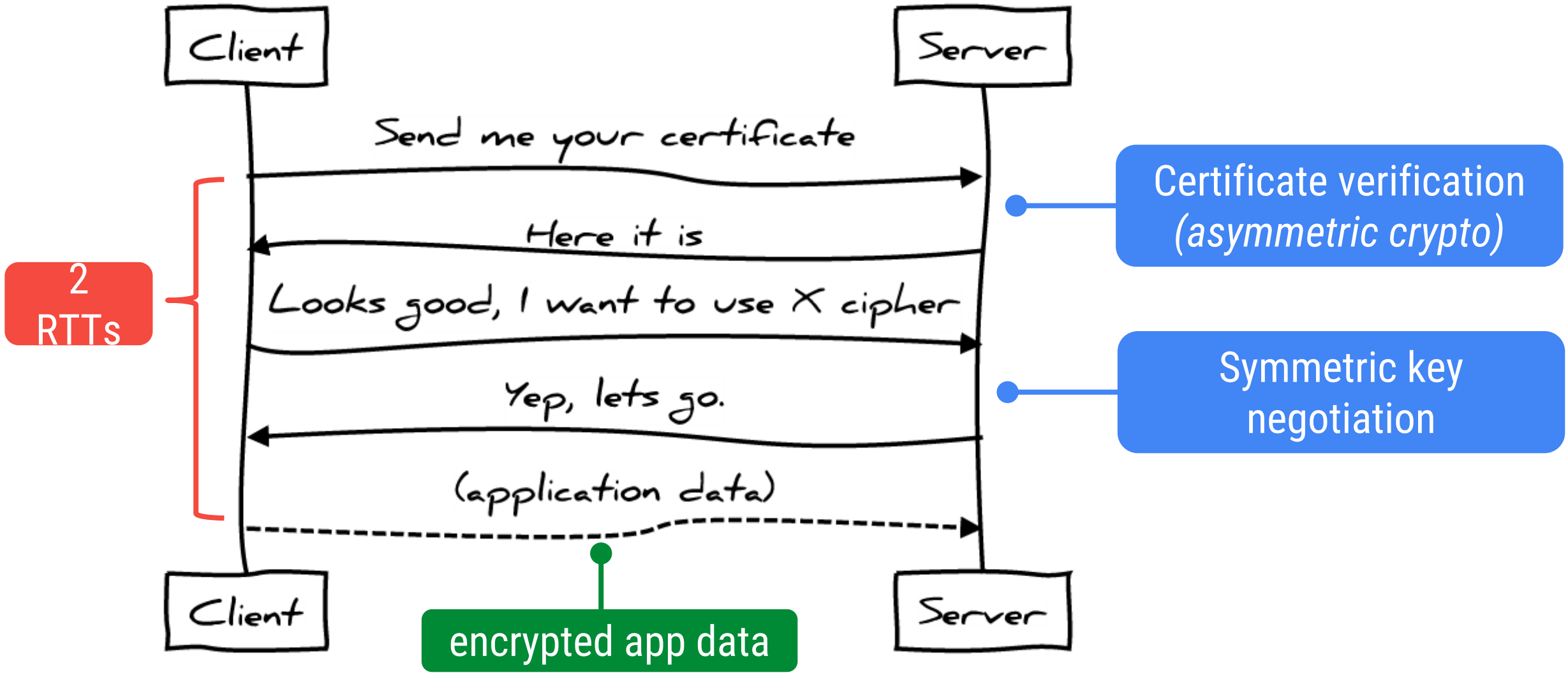


Optimized TLS handshake: **1-RTT** for new and resumed connections

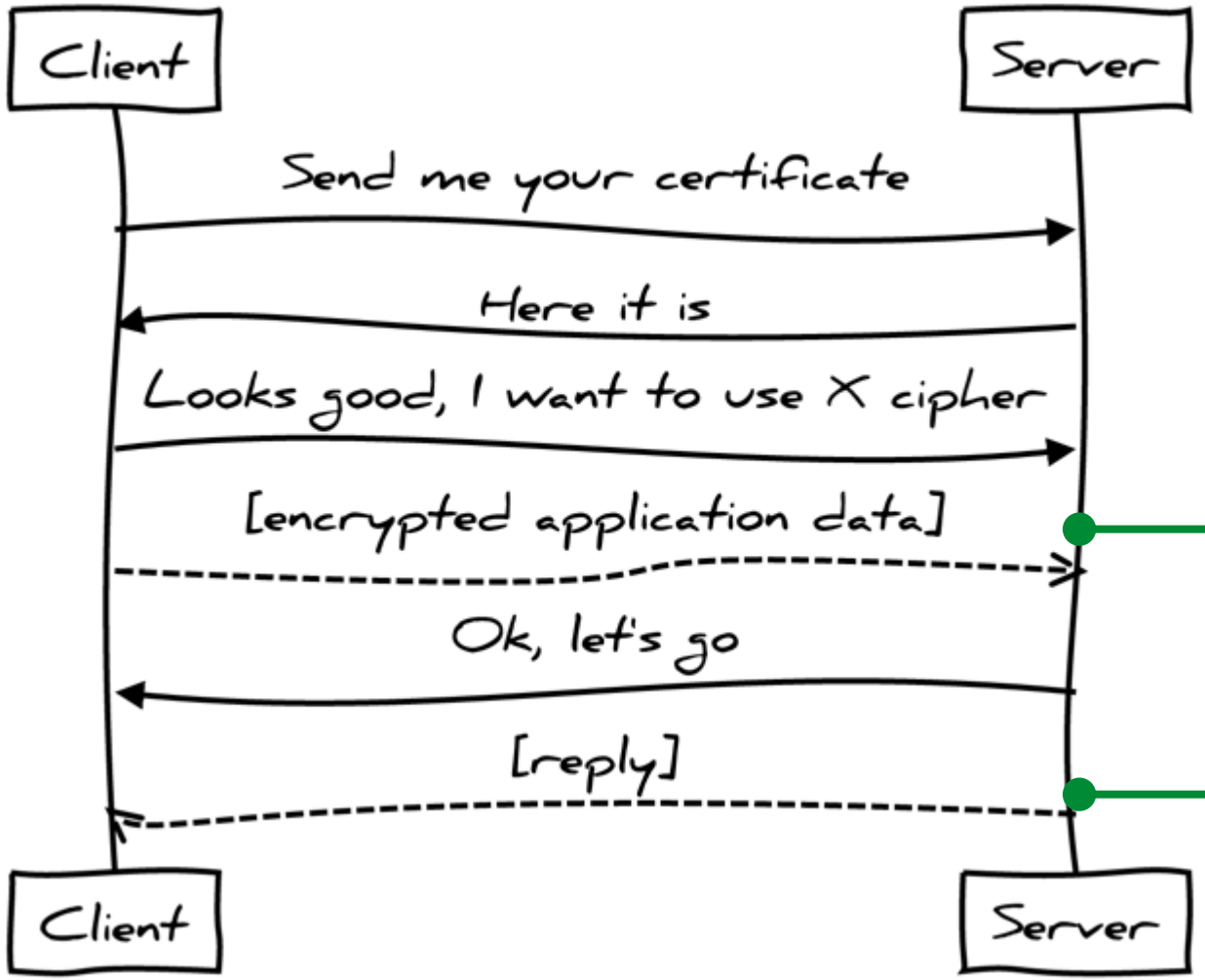
What you will probably see...

Tip: set WPT to use 300ms 3G profile, makes it much easier to detect handshake problems

Textbook TLS handshake



1-RTT non-resumed handshake with TLS False Start



*In practice… some servers break with False Start, hence it's done as an **opt-in behavior**.*

Client's ChangeCipherSpec followed by encrypted request

Server's ChangeCipherSpec followed by encrypted response

Deploying False Start...

Chrome and Firefox

- NPN/ALPN advertisement - e.g. “http/1.1”
- Forward secrecy ciphersuite - e.g. ECDHE

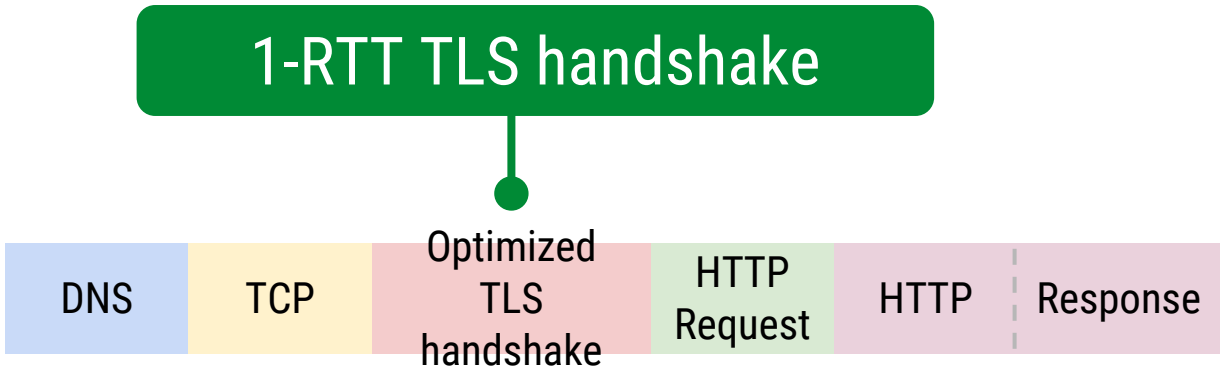
Safari

- Forward secrecy ciphersuite

Internet Explorer

- Blacklist + timeout
- If handshake fails, retry without False Start

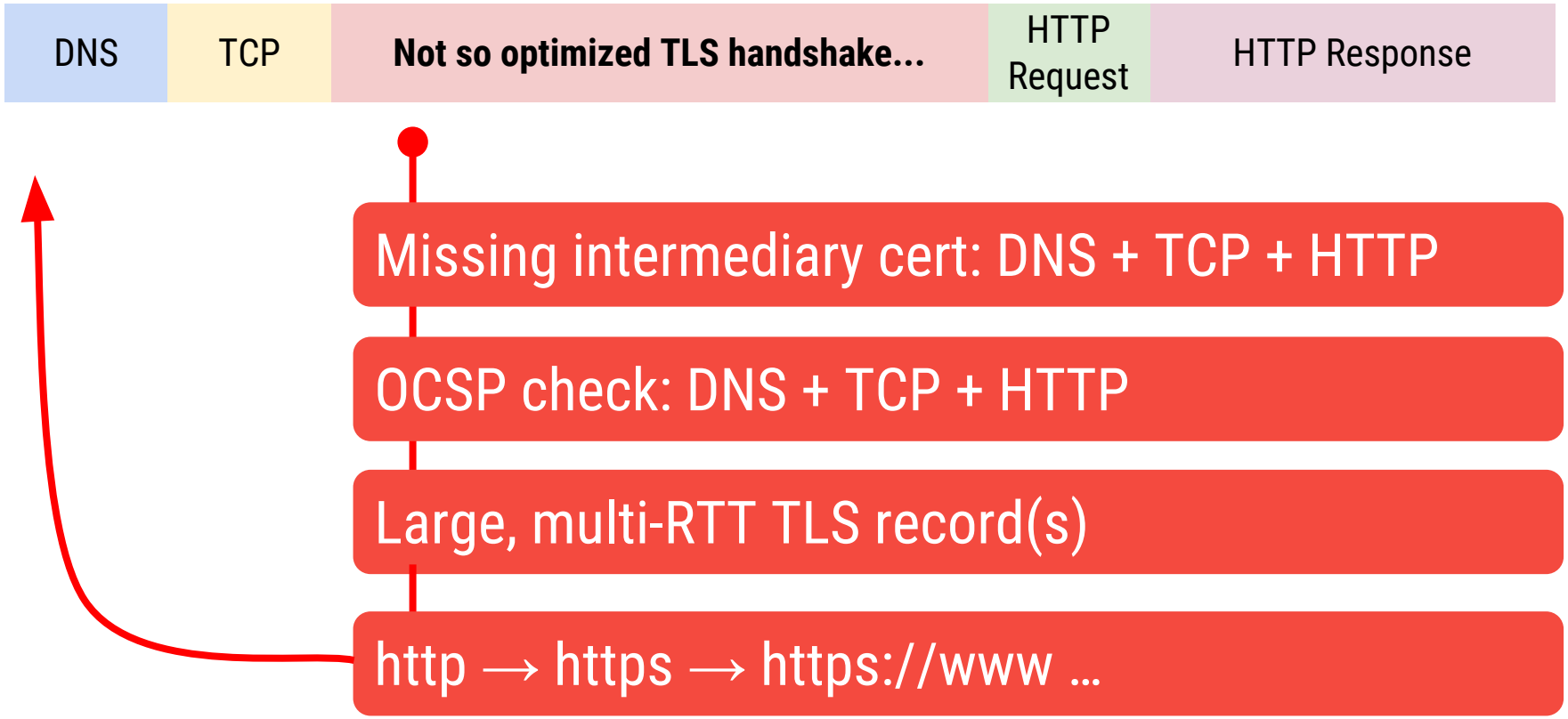
TL;DR: *enable NPN advertisement and forward secrecy to get 1RTT handshakes.*



You server should be delivering 1-RTT TLS handshakes

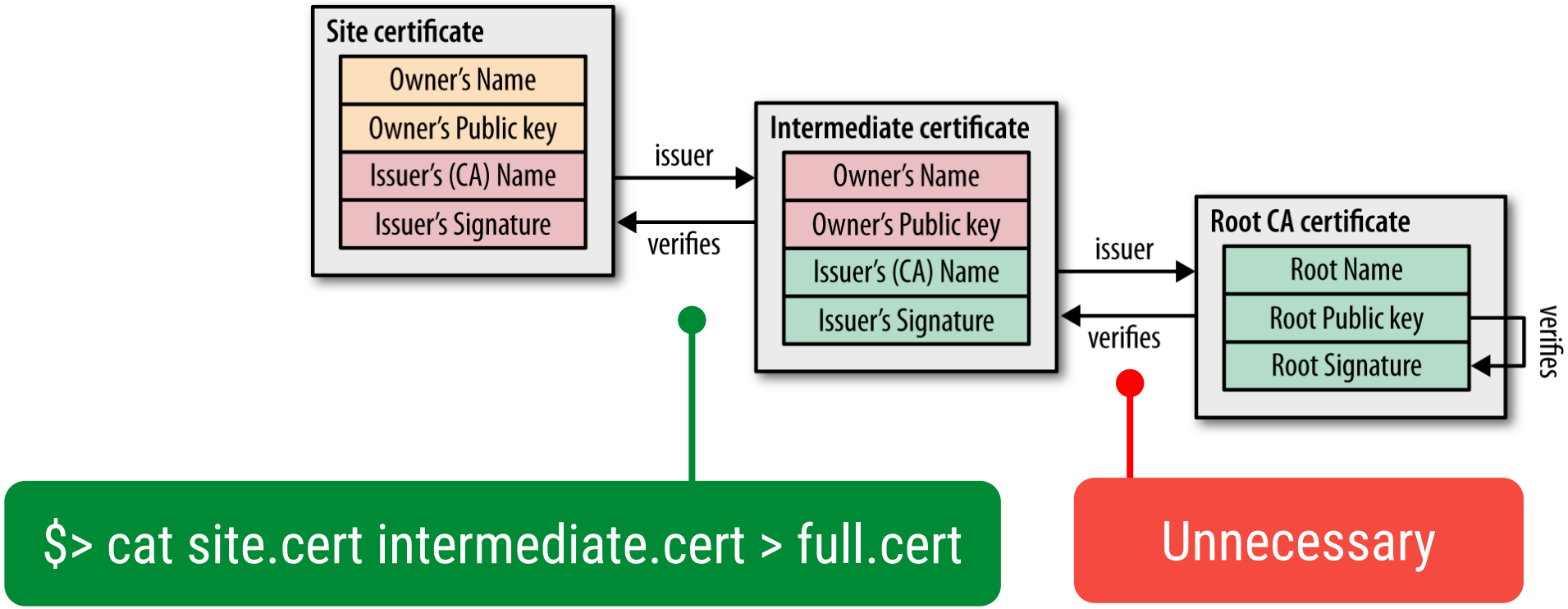
- **False Start:** 1-RTT handshake for new visitors
- **Resumption:** 1-RTT handshake for returning visitors
  - *Plus, we skip (expensive) asymmetric crypto!*

Common perf pitfals and misconfigurations...



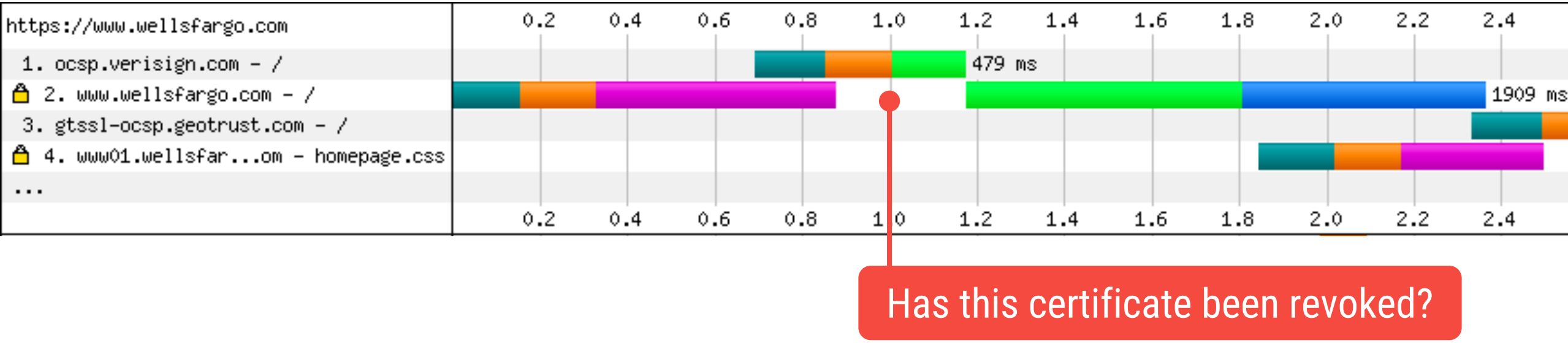
If your TLS handshake is not 1-RTT, chances are… it is due to one, or more, of above issues.

1. Missing intermediary certificates



- Provide site certificate + CA intermediary certificate!
  - *Otherwise, client must pause and fetch the intermediate cert.*

2. Online Certificate Status Protocol (OCSP)



Browser pauses navigation and queries the OCSP server

- DNS lookup, TCP handshake, HTTP request, ...

OCSP stapling improves security and performance. Use it.

- Enables revocation checks for non-EV certificates.
- Does not require client lookups, does not pause navigation.

TLS handshake with stapled OCSP response...

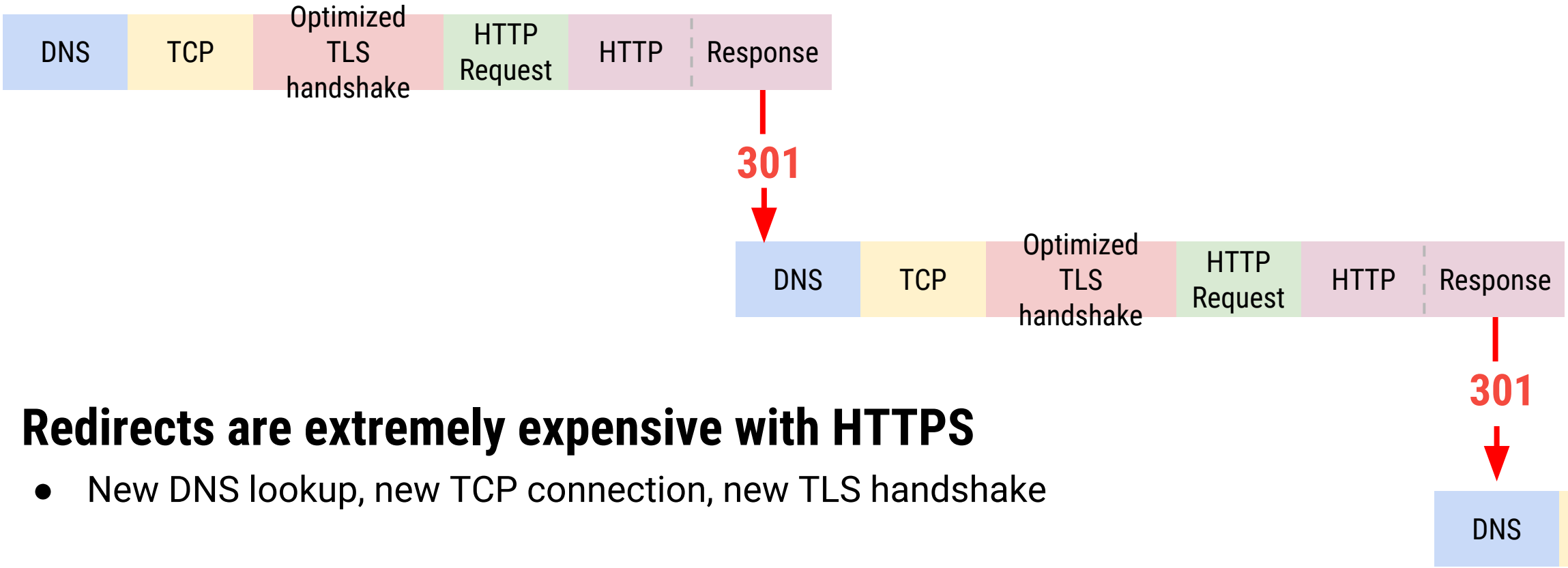
```
$> openssl s_client -connect example.com:443 -tls1 -tlsextdebug -status

OCSP Response Data:
  OCSP Response Status: successful (0x0)
  Response Type: Basic OCSP Response
  Version: 1 (0x0)
  Responder Id: C = IL, O = StartCom Ltd., CN = StartCom Class 1 Server OCSP Signer
  Produced At: Feb 18 17:53:53 2014 GMT
  Responses:
  Certificate ID:
    Hash Algorithm: sha1
    Issuer Name Hash: 6568874F40750F016A3475625E1F5C93E5A26D58
    Issuer Key Hash: EB4234D098B0AB9FF41B6B08F7CC642EEF0E2C45
    Serial Number: 0B60D5
  Cert Status: good
```

Stapled OCSP means no blocking!

Server performs the OCSP check and “staples” the signed response to the certificate.

3. Audit your redirect chains



Redirects are extremely expensive with HTTPS

- New DNS lookup, new TCP connection, new TLS handshake

I'm willing to bet **your** site has at least one unnecessary chain:

- http://… → http://www
- http://… → **https://…** → https://www… *(hint: http://… → https://www…)*

P.S. Set HSTS policy on the "naked" origin by requesting a special resource from www, or some such...

HSTS eliminates HTTPS redirects

```
Strict-Transport-Security: max-age=10886400; includeSubDomains
```

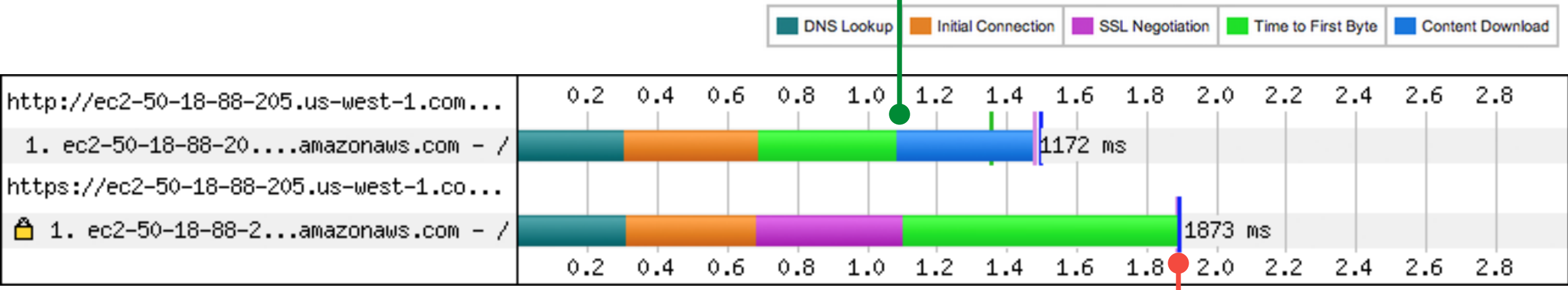
Browser remembers (for specified max-age) that **it should automatically request HTTPS resources** for this origin and its subdomains.

P.S. for bonus points, submit your site to HSTS preload list.

https://hstspreload.appspot.com/

4. Optimize your TLS record size

1-RTT Time to First Byte (TTFB)
+1-RTT for the rest of response



Why do we have a TTFB delay?

2-RTT Time to First Byte
… delays HTML processing

TLS record size + latency gotchas...

```
▽ [8 Reassembled TCP Segments (11221 bytes): #169(1460), #170(1460), #172(1460), #174(1460),
                                             #175(1460), #177(1460), #179(1460), #180(1001)]
    [Frame: 169, payload: 0-1459 (1460 bytes)]
    [Frame: 170, payload: 1460-2919 (1460 bytes)]
    [Frame: 172, payload: 2920-4379 (1460 bytes)]
    [Frame: 174, payload: 4380-5839 (1460 bytes)]
    [Frame: 175, payload: 5840-7299 (1460 bytes)]
    [Frame: 177, payload: 7300-8759 (1460 bytes)]
    [Frame: 179, payload: 8760-10219 (1460 bytes)]
    [Frame: 180, payload: 10220-11220 (1001 bytes)]
    [Segment count: 8]
    [Reassembled TCP length: 11221]
▽ Secure Sockets Layer
  ▽ TLSv1 Record Layer: Application Data Protocol: http
      Content Type: Application Data (23)
      Version: TLS 1.0 (0x0301)
      Length: 11216
```

This record is split across 8 TCP packets

TLS allows up to 16KB of application per record

- New connection + 16KB record = CWND overflow and an extra RTT
- Lost or delayed packet delays processing of entire record

Optimizing record size…

1. **Implement dynamic record sizing (Google servers)**
   a. New connections start with 1400 byte records (aka, single MTU)
   b. After ~1MB is sent, switch to 16K records
   c. After ~1s of inactivity, reset to 1400 byte records

2. **Check your server configuration...**
   a. Apache doesn't allow custom configuration.
   b. Nginx supports static override (via ssl_buffer_size)
      i. *Not optimal, but still worth setting to 4k.*
   c. HAProxy & ATS supports dynamic record sizing.

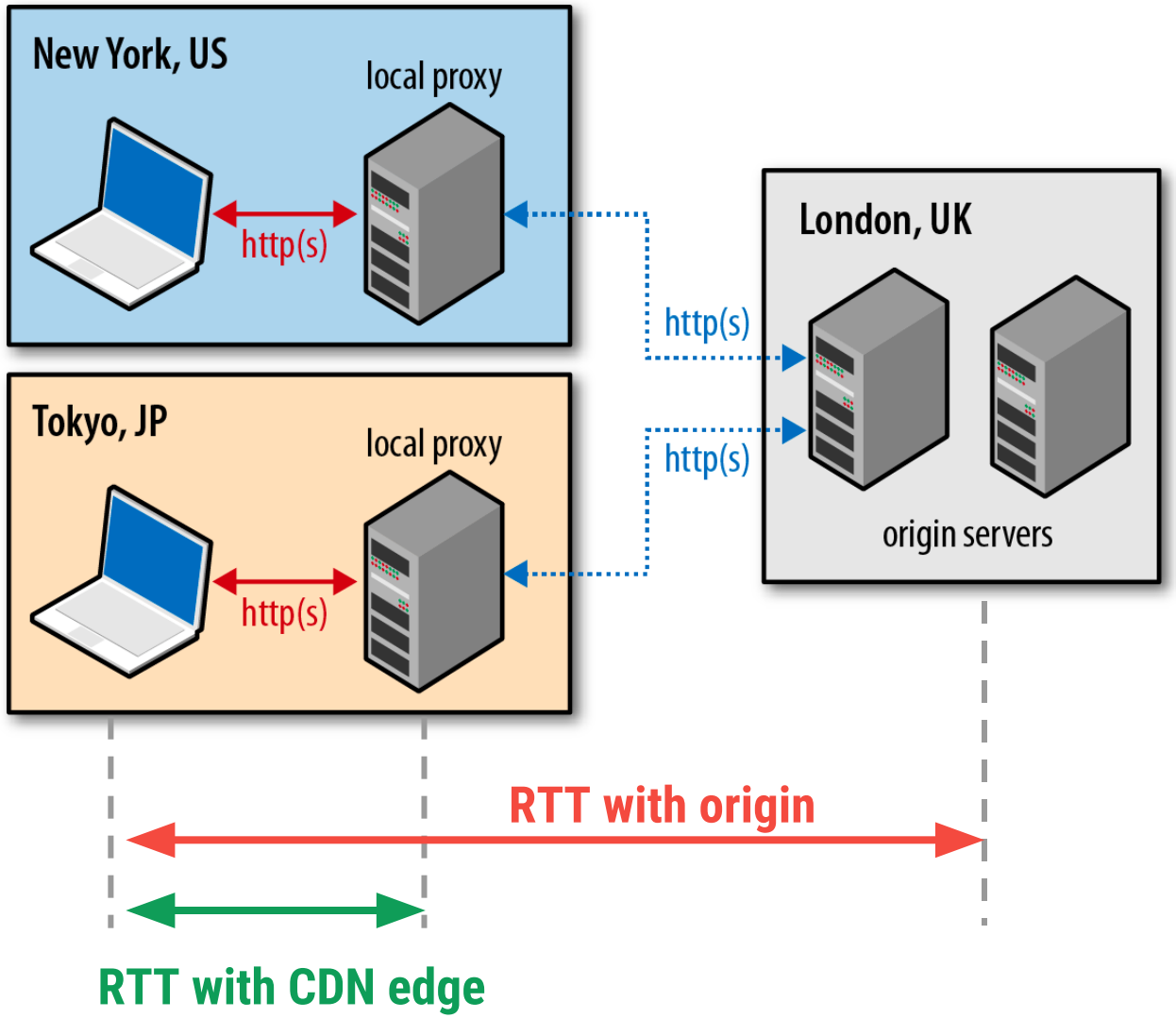
TL;DR: there is no "perfect record size". Adjust dynamically.

Do this at home...

a. Does your infrastructure support False Start?
   i. *Enable ALPN / NPN*
   ii. *Enable Forward Secrecy*

b. Does your WPT run show 1-RTT handshakes?
   i. *Provide full certificate chain*
   ii. *Configure OCSP stapling*
   iii. *Eliminate redirects*
   iv. *Optimize record size*

Terminate TLS at the CDN edge…



CDNs are not just for static content.

*Edge termination can significantly reduce TCP **and** TLS handshake costs!*

E.g. Server in London, client in NYC.

- *RTT to origin is ~50ms.*
- *RTT to edge server is ~10ms.*
- ***TCP + TLS handshake with origin:***
  - *2 x 50ms = 100ms*
- ***TCP + TLS handshake with CDN:***
  - *2 x 10ms = 20ms*

Pick your server (and CDN) wisely

We have a lot of optimization work to do across the industry...

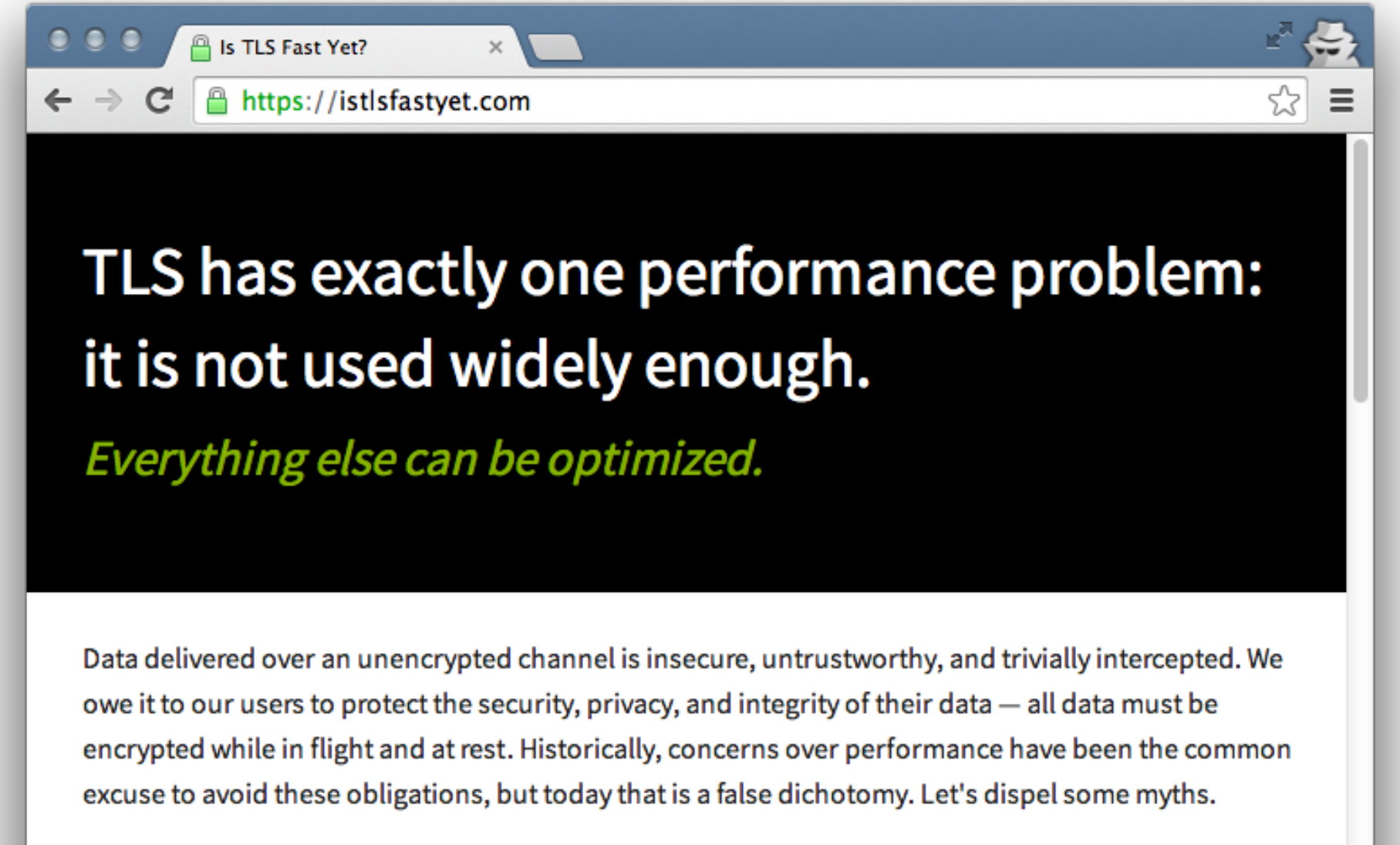
	Session identifiers	Session tickets	OCSP stapling	Dynamic record sizing	ALPN / NPN	Forward secrecy	SPDY & HTTP/2
Apache	yes	yes	yes	no	no*	yes	mod_spdy
ATS	yes	yes	yes	dynamic	yes	yes	spdy/3.1
bud	no	yes	yes	static	yes	yes	no
HAProxy	yes	yes	yes	dynamic	yes	yes	no
IIS	yes	yes	yes	no	yes	yes	no
NetScaler	yes	no	no	no	yes	yes	spdy/3.0
NGINX	yes	yes	yes	static	yes	yes	spdy/3.1
node.js	yes	no	no	no	yes	no	spdy/3.1

- Most servers have **a lot** of room for improvement.
- Apache Traffic Server is the only one that's all green!

	Session identifiers	Session tickets	OCSP stapling	Dynamic record sizing	ALPN / NPN	Forward secrecy	SPDY & HTTP/2
Akamai	yes	yes	yes	configurable (static)	yes	yes	spdy/3 spdy3.1 (opt-in)
CloudFlare	yes	yes	yes	4KB (static)	yes	yes	spdy/3.1
AWS ELB	yes	yes	no	no	no	yes	no
AWS CloudFront	no	yes	yes	no	yes	yes	no
EdgeCast	no	yes	yes	no	no	yes	no
Fastly	yes	yes	yes	no	no	yes	no
Google App Engine	yes	yes	no	dynamic	yes	yes	spdy/3.1
Heroku	yes	yes	no	no	no	yes	no
Instart Logic	yes	yes	no	configurable (static)	no	yes	no
Limelight	yes	yes	no	no	no	yes	no
MaxCDN	yes	yes	no	no	yes	no	spdy/3.1

There is way too much red here… Bug your CDN about fixing this!

isTLSfastyet.com



What if I told you that…

HTTPS is actually faster and more efficient?

Courtesy of SPDY and HTTP/2!

SPDY and HTTP/2 improve client latency

	Page load time improvement with SPDY enabled...			
	Google News	**Google Sites**	**Google Drive**	**Google Maps**
Median	**43%**	**27%**	**23%**	**24%**
95th percentile	**44%**	**33%**	**36%**	**28%**

Improvement over **HTTPS/1.1**

Some Google services are faster with SPDY than over plaintext HTTP, and more efficient too! Why? Multiplexing, prioritization, header compression.

http://blog.chromium.org/2013/11/making-web-faster-with-spdy-and-http2.html

SPDY and HTTP/2 improve server performance

Designed to use a single connection:

1. Fewer sockets → **fewer TLS handshakes**
2. Fewer buffers → **memory savings**
3. Connection re-use → **faster data delivery**

… TLS + SPDY → lower operational costs

Do this at home…

a. **If you enabled HTTPS… enable SPDY!**
   i. *Fewer connections, fewer handshakes, …*
   ii. *Better client and server performance*

b. **Start investigating HTTP/2**
   i. *Already available in Chrome and Firefox **stable**!*
   ii. *IE11 techinical preview supports it as well!*

Slides
bit.ly/1A2HVEX

Learn more
isTLSfastyet.com

Thanks! Questions?

+Ilya Grigorik
@igrigorik