Introducción a SPDY

¿Futuro HTTP/2.0?

Lic. Marcelo Fidel Fernández
http://www.marcelofernandez.info
marcelo.fidel.fernandez@gmail.com
@fidelfernandez

Agenda

- Características de la web antes y ahora
- HTTP y la Web actual, inconvenientes
- Introducción a SPDY, características
- Ejemplos
- Estado actual y futuro del protocolo
- HTTP/2.0
- Conclusiones Generales
- SPDY dentro del ecosistema Python

En los orígenes de la Web...

- **1991**: El servicio de WWW nace y **HTTP/0.9** fue “definido”. Sólo permitía un único método: GET.
- **1996**: **HTTP/1.0**. Se estandarizó la base mínima de lo que usamos a diario.
- **1997-1999**: **HTTP/1.1**. Se completó el protocolo. Escalabilidad, proxies, Keep-Alive y Pipelining.

¿Y cómo era la Web en ese entonces?

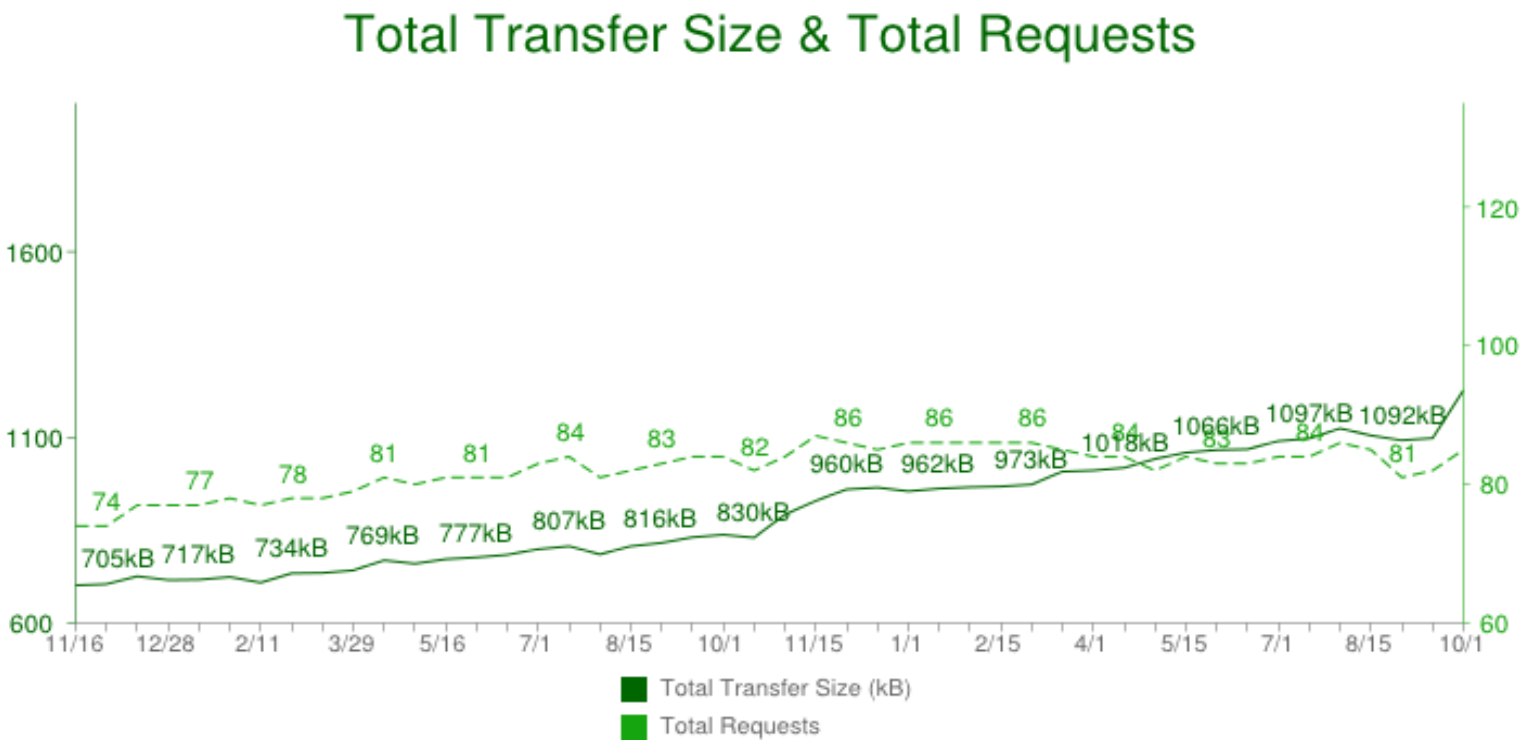
www.python.org @1997

- Prácticamente de texto, pocas imágenes, nada de interactividad.
- 60 KB de tamaño promedio [ref]

Fuente: www.archive.org

¿Cómo es la Web de Hoy?

Tamaño de página y de peticiones promedio (2010-2012)

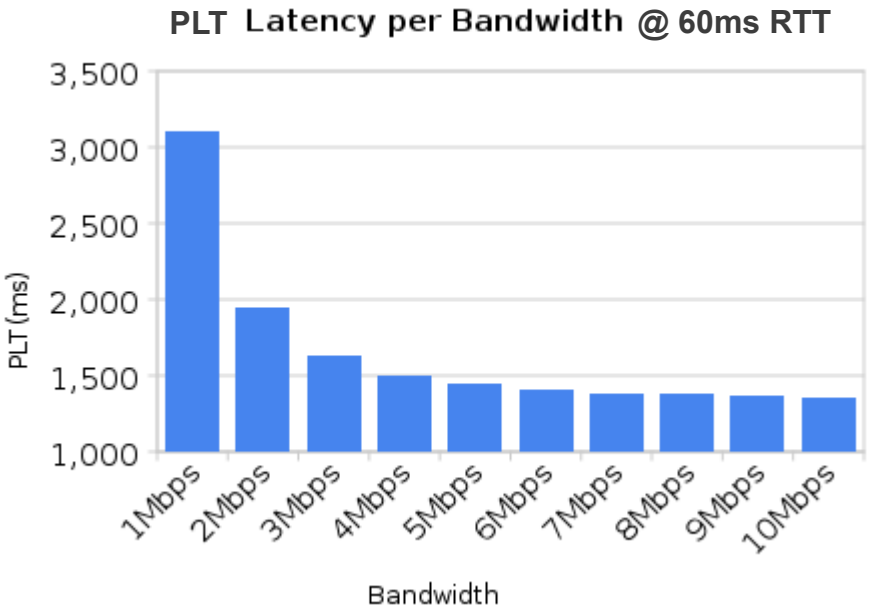


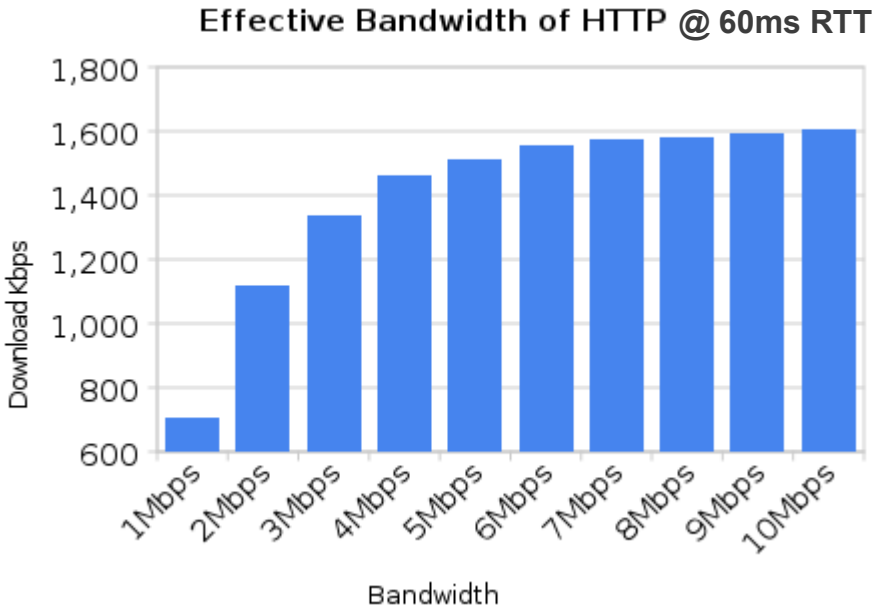
2010: **74** peticiones HTTP → más de **80** en 2012

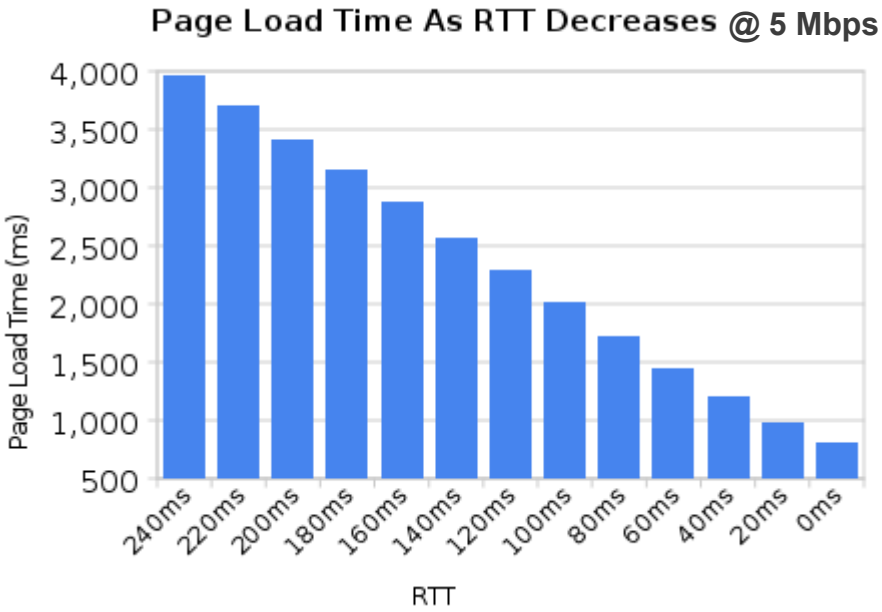
2010: **705 KB → 1092 KB** en 2012

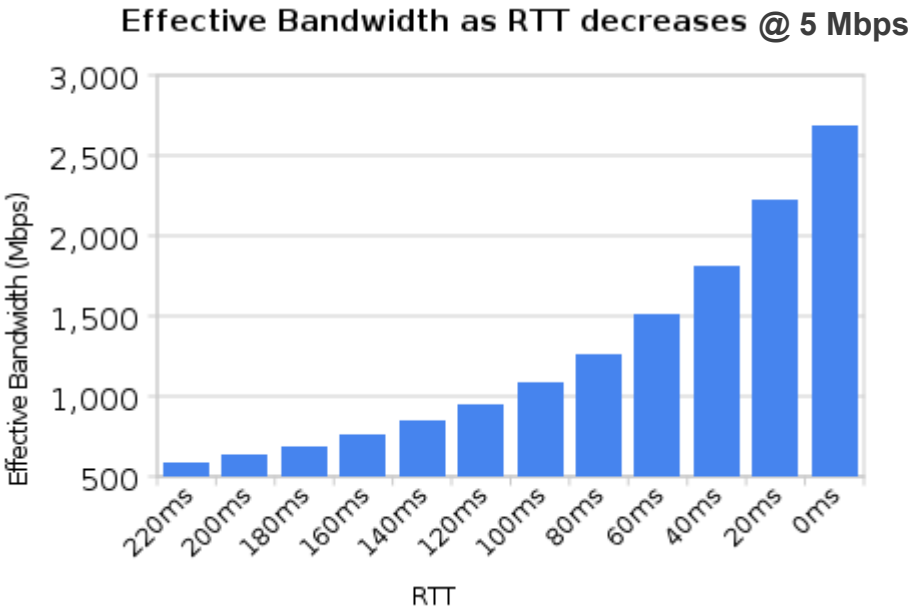
Fuente: http://httparchive.org

¿Cómo es la Web de Hoy?
Ancho de Banda y Latencia (RTT)





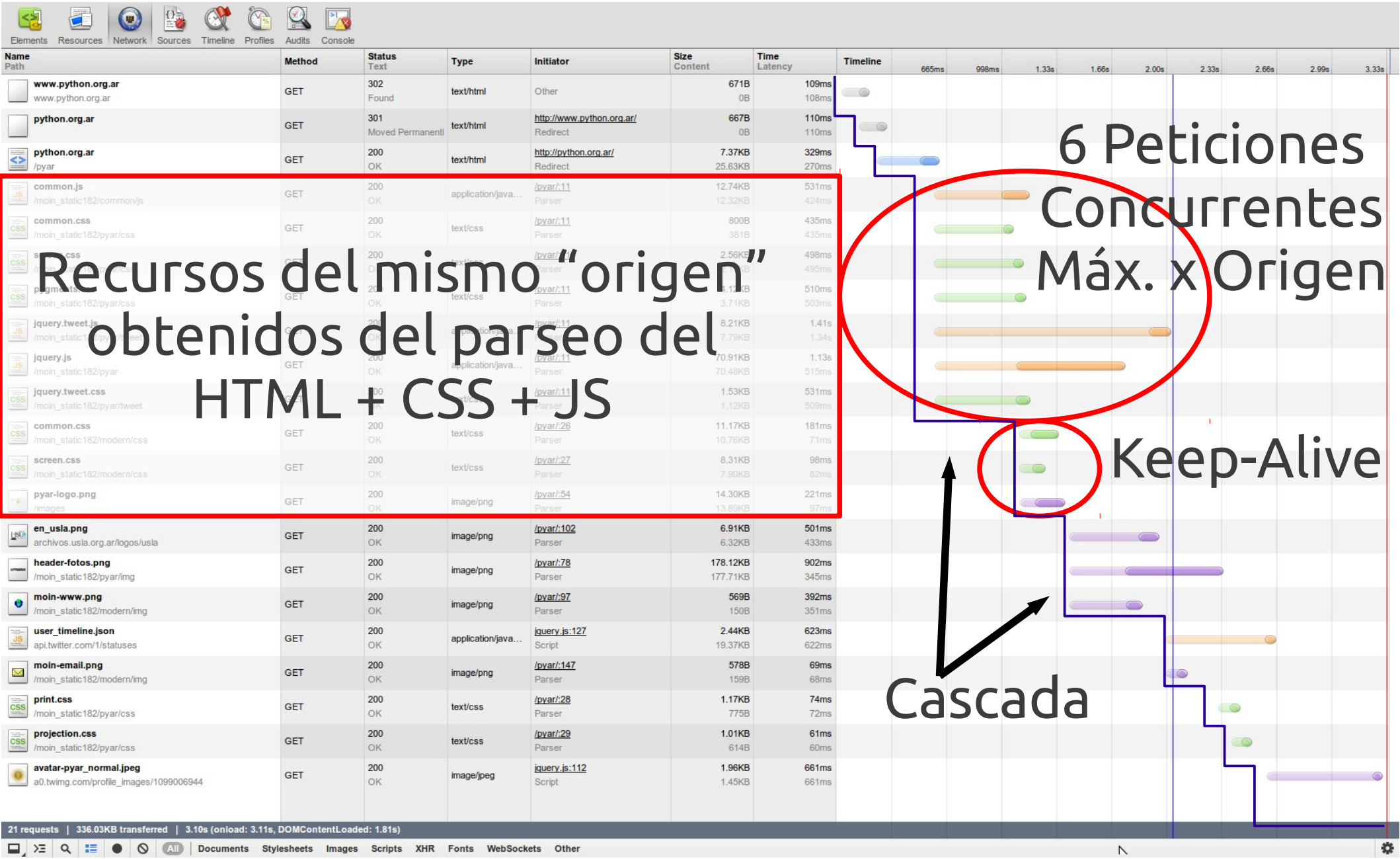




Fuente: http://www.belshe.com/2010/05/24/more-bandwidth-doesnt-matter-much/

“Cascada” HTTP - ww.python.org.ar

(18 + 3 requests, ~336 KB, 3.1 seg = ~100 KB/seg)



HTTP y la Web actual

- El **RTT es determinante** en el tiempo de carga de la página en HTTP.
- Keep-Alive + Pipelining + Múltiples Conexiones != Paralelización
- HTTP es un protocolo que obliga a serializar las peticiones.
- Mucha heurística de optimización de tráfico y recursos en el browser.

HTTP y la Web actual (cont.)

- *Hacks* para evitar limitaciones de HTTP
  - Domain Sharding
  - Recursos inline, minificados, image maps, CSS sprites
  - Ordenamiento, dependencias...
- Headers cada vez más grandes
- TCP fue hecho para conexiones con un tiempo de vida largo.
- Los browsers usan HTTP sobre TCP con ráfagas de conexiones.
- El Handshake TCP, Slow Start y Bufferbloat se multiplican por cada conexión.

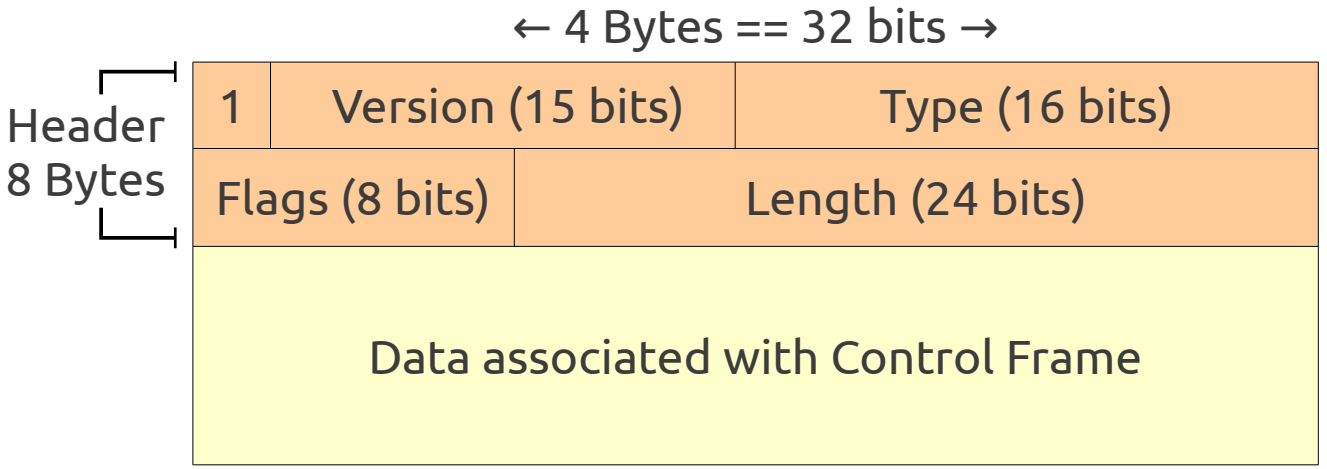
SPDY

- Desarrollado abiertamente por Google desde 2009, RFC desde Febrero 2012.
- Capa 5 OSI. Modifica cómo se lee/escribe el tráfico HTTP en el socket.
- Toda la semántica de HTTP se mantiene.
- El objetivo es reducir el tiempo de carga de las páginas web en forma global.
- Lo que hace no es nada novedoso.

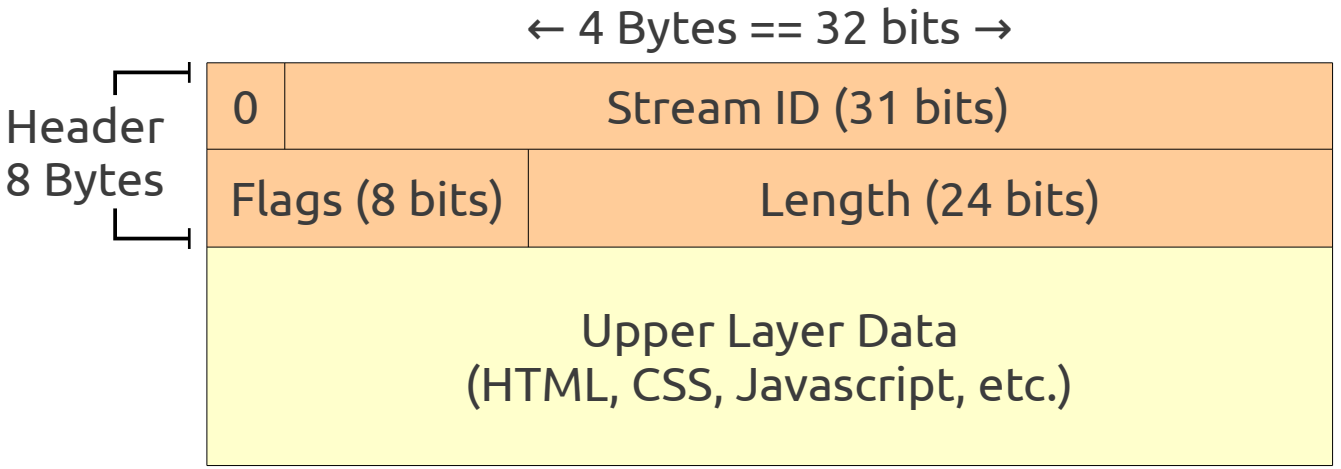
SPDY

- Multiplexación del tráfico por una única conexión TCP persistente.
- Priorización de Streams (“Peticiones”).
- Binario.
- Compresión obligatoria, incluye encabezados.
- Server-Pushed Streams
- En la práctica, se utiliza sobre TLS: Cifrado.
- TLS NPN: Next-Protocol Negotiation Extension.

SPDY – Frame Types

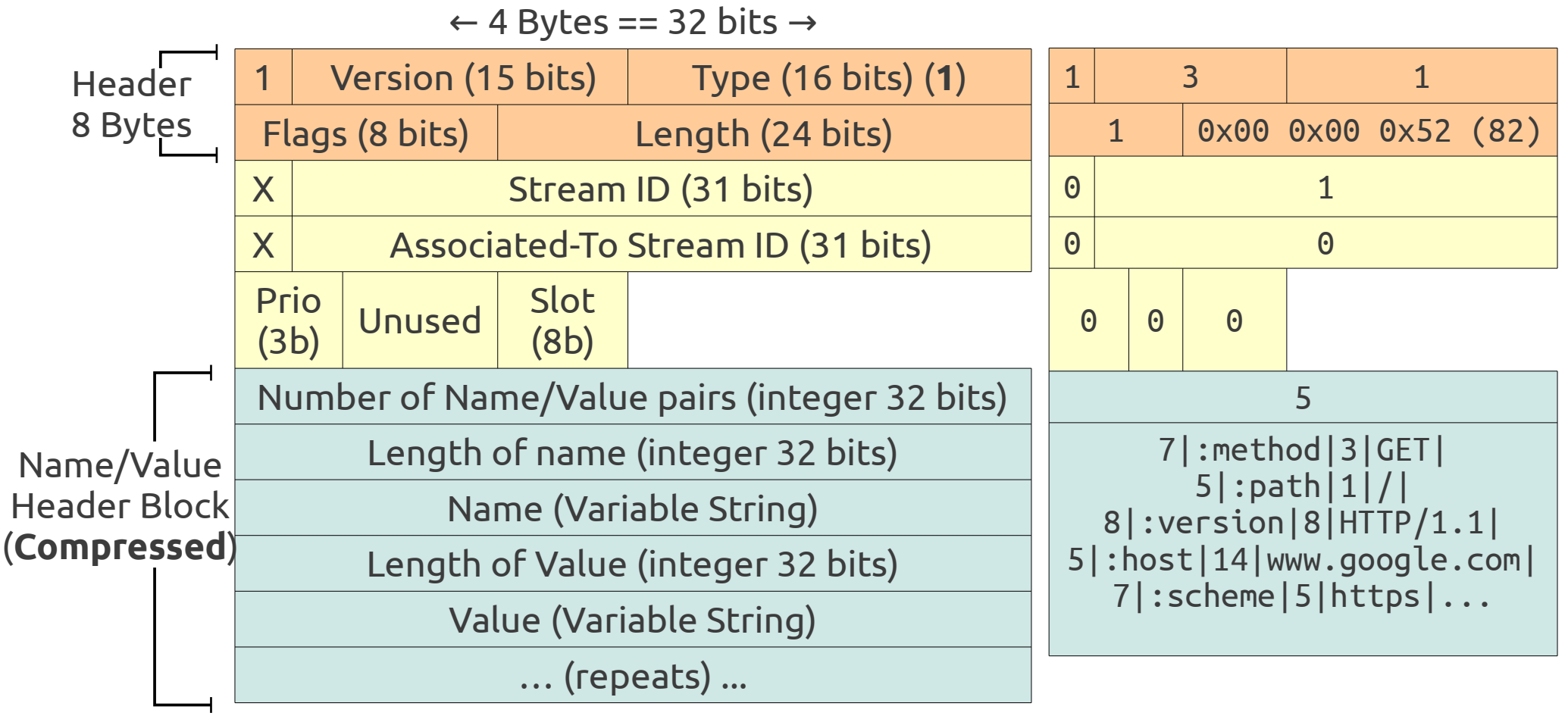


Control Frame



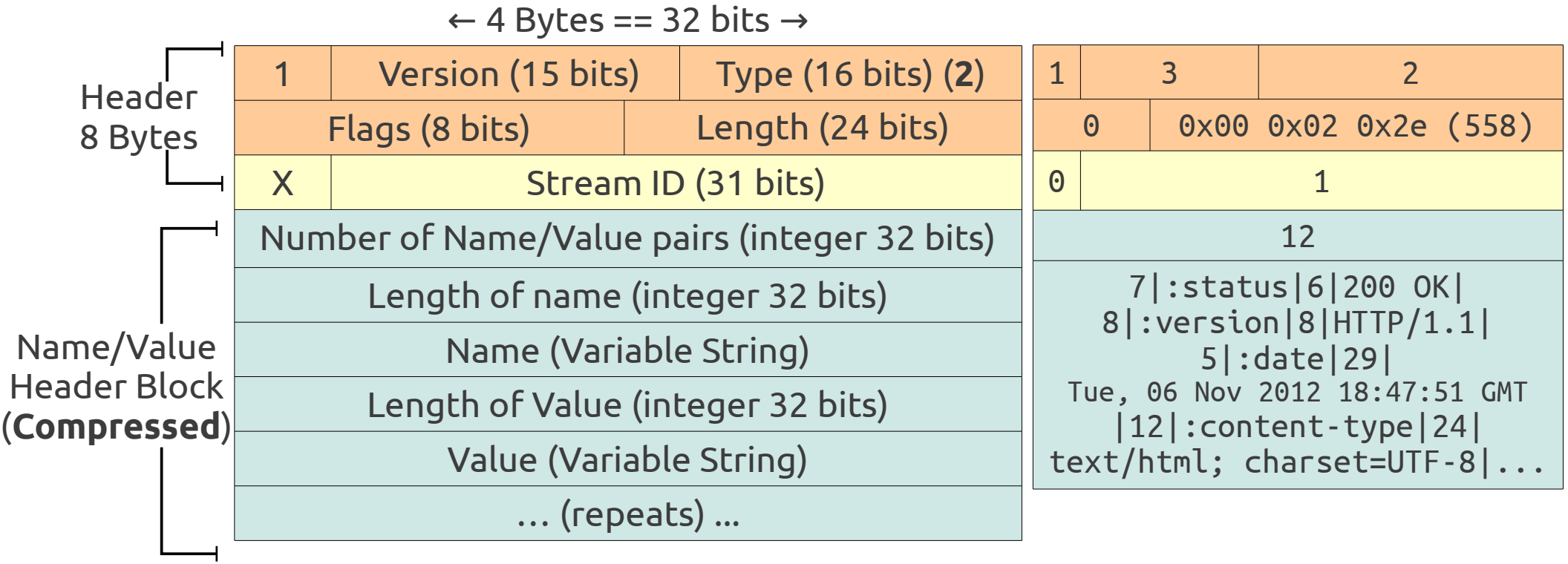
Data Frame

SPDY – Control Frames
1 - Syn_Stream



Representación Teórica y “en el cable”

SPDY – Control Frames
2 - Syn_Reply



Representación Teórica y “en el cable”

SPDY – Control Frames
3 - Rst_Stream

← 4 Bytes == 32 bits →

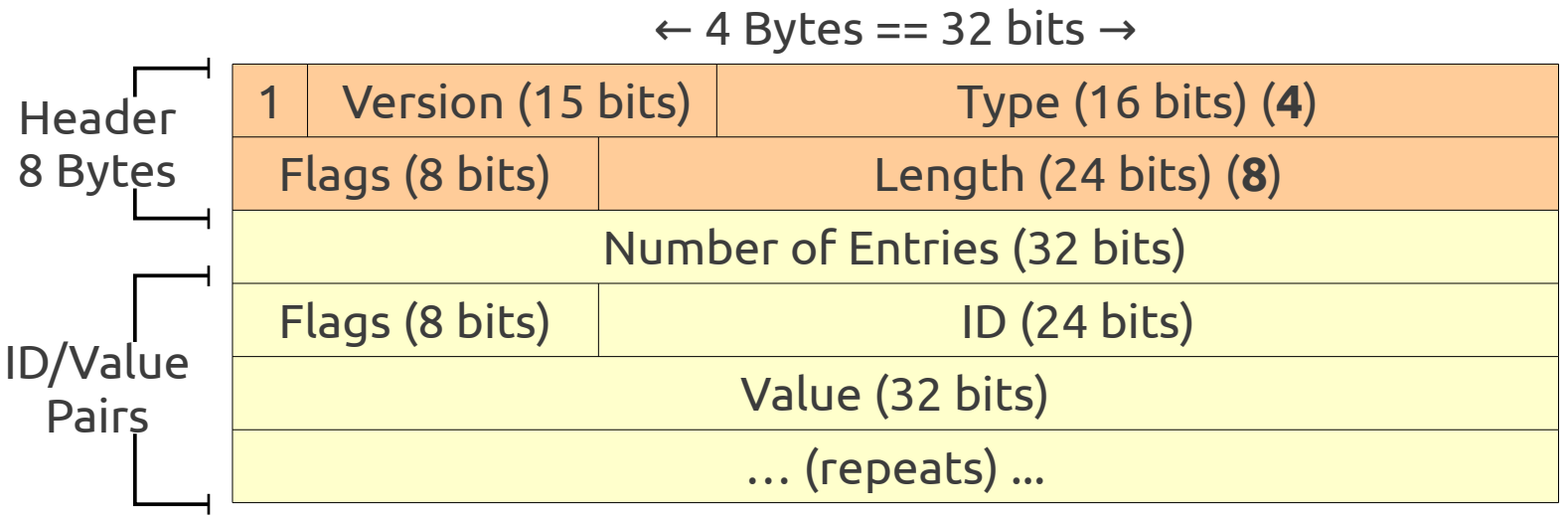
Header
8 Bytes

1	Version (15 bits)	Type (16 bits) (**3**)
Flags (8 bits)		Length (24 bits) (**8**)
X	Stream ID (31 bits)	
Status Code (32 bits)		

Status Codes:

- 1: Protocol Error
- 2: Invalid Stream
- 3: Refused Stream
- 4: Unsupported Version
- 5: Cancel
- 6: Internal Error
- 7: Flow Control Error
- 8: Stream In Use
- 9: Stream Already Closed
- 10: Invalid Credentials
- 11: Frame Too Large

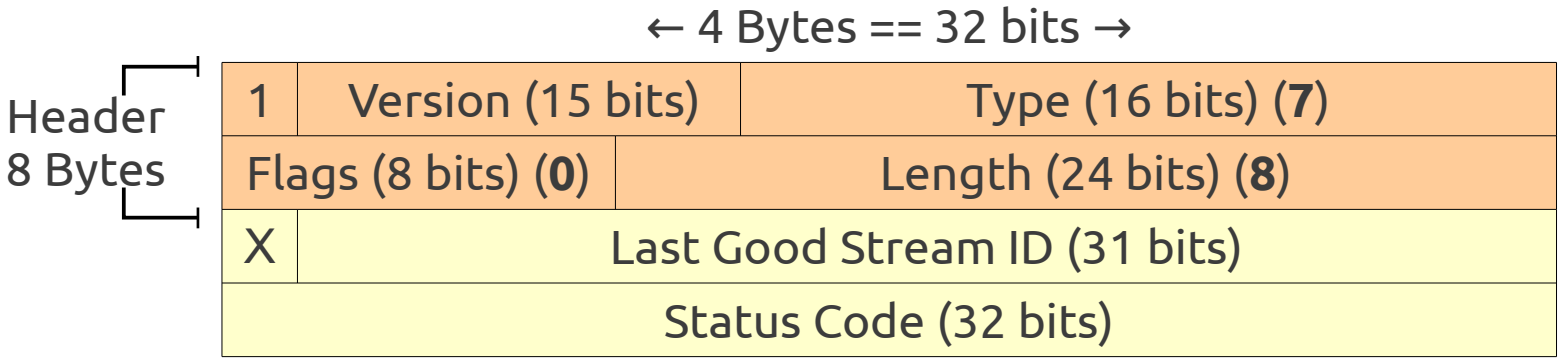
SPDY – Control Frames
4 - Settings



Settings IDs available:

- 1: Upload Bandwidth
- 2: Download Bandwidth
- 3: Round Trip Time
- 4: Max Concurrent Streams
- 5: Current Cwd
- 6: Download Retrans Rate
- 7: Initial Windows Size
- 8: Client Certificate Vector Size

SPDY – Control Frames
7 - Goaway y otros



Status Codes:

- 0: OK
- 1: Protocol Error
- 11: Internal Error

Otros Control Frames:

- **6 – Ping**: Medir RTT
- **8 – Headers**: Permite intercambiar headers adicionales sobre un stream
- **9 - Window Update**: Control de flujo por stream
- **10 – Credentials**: Envío de certificados SSL adicionales

Ejemplos

chrome://net-internals/

Python-SPDY Examples
Client SPDY <--> Server SPDY

Server Push example

Estado Actual y Futuro de SPDY

- SPDY/3 lanzado en Febrero 2012, SPDY/4 en 2013
- Implementaciones:
  - Clientes: Chrome/Chromium, Firefox, Opera y Android ya soportan SPDY/3.
  - Servidores: mod_spdy, nginx, F5, Jetty, HAProxy
  - Infraestructura: Google (GAE sobre HTTPS), Twitter, Wordpress, Akamai, Cloudflare, Strangeloop….

    Amazon Kindle browser and reverse proxy [ref]
- Interesados: Facebook, *Microsoft*, libcurl...

HTTP/2.0

- SPDY/3 fue tomado como base para el próximo HTTP/2.0 en el marco del HTTPbis WG (IETF).
- Mantener los conceptos básicos del protocolo.
- Resta tiempo para definirse (ETA fines de 2014)
- Quedan **muchas** cosas por definir [0]:
  - Headers binarios/compresión? [1][2][3]
  - *Upgrade* o algún otro mecanismo de negociación
  - Cifrado y/o TLS obligatorio/opcional/indefinido?
  - Server Push? [4]
  - Mecanismos de Autenticación [5]
  - Proxies, escalabilidad.

Conclusiones Generales

- HTTP/1.1 está mostrando sus años con las características de los sitios y conexiones actuales.
- Los *hacks* no escalan y aumentan la complejidad.
- Cerca del 70% de los usuarios ya soportan SPDY.
- SPDY mejora mucho el rendimiento, pero para implementarlo bien™ hay que *deshackear* lo hecho.
- La migración no es *painless* (aunque podría ser peor).
- Resta mucho software dentro de la arquitectura Web por construir y estabilizar (Proxys, Load Balancers, Servers, Firewalls...)
- SPDY todavía está en evolución.

SPDY dentro del ecosistema Python

- Python 3 incorporó muchas cosas necesarias para:
  - Hacer más simple el manejo de streams de bytes,
  - Conversión desde/hacia bytes desde tipos builtin (3.2)
  - SSL y Zlib en particular para SPDY (3.3)
- No hay ningún proyecto serio de infraestructura de red que planee seriamente soportarlo (todavía): Twisted, Requests, ¿otros?
- Otros lenguajes están siendo utilizados para experimentación e implementación y están más adelantados: Node.js, Ruby, Java, y obviamente C/C++

¡Muchas Gracias!

http://www.marcelofernandez.info
marcelo.fidel.fernandez@gmail.com
@fidelfernandez