Nuevos protocolos

Área de Ingeniería Telemática
Dpto. Automática y Computación
http://www.tlm.unavarra.es/

What's wrong with HTTP1 ? 1.0/1.1

Problemas de HTTP / 1.0 1.1

- El protocolo HTTP/1.1 se ha convertido en el protocolo para todo en Internet
  - No es tan simple como se pretendia
  - Incluso con extensiones para peticiones ser->cli y sockets
  - HTTP 1.1 especificación complicada con muchas opciones
- Uso ineficiente de TCP
  - Muchas conexiones pequeñas
- Objetos con cada vez mayores y muchos
  - Descargar unos 2MB para ver una pagina
  - En unos 90-100 objetos
  - 37 objetos por pagina en las 300000 sitios mas populares
- La web no es como en 1997

Problemas de HTTP / 1.0 1.1

- **Latencia**

- Head of Line Blocking

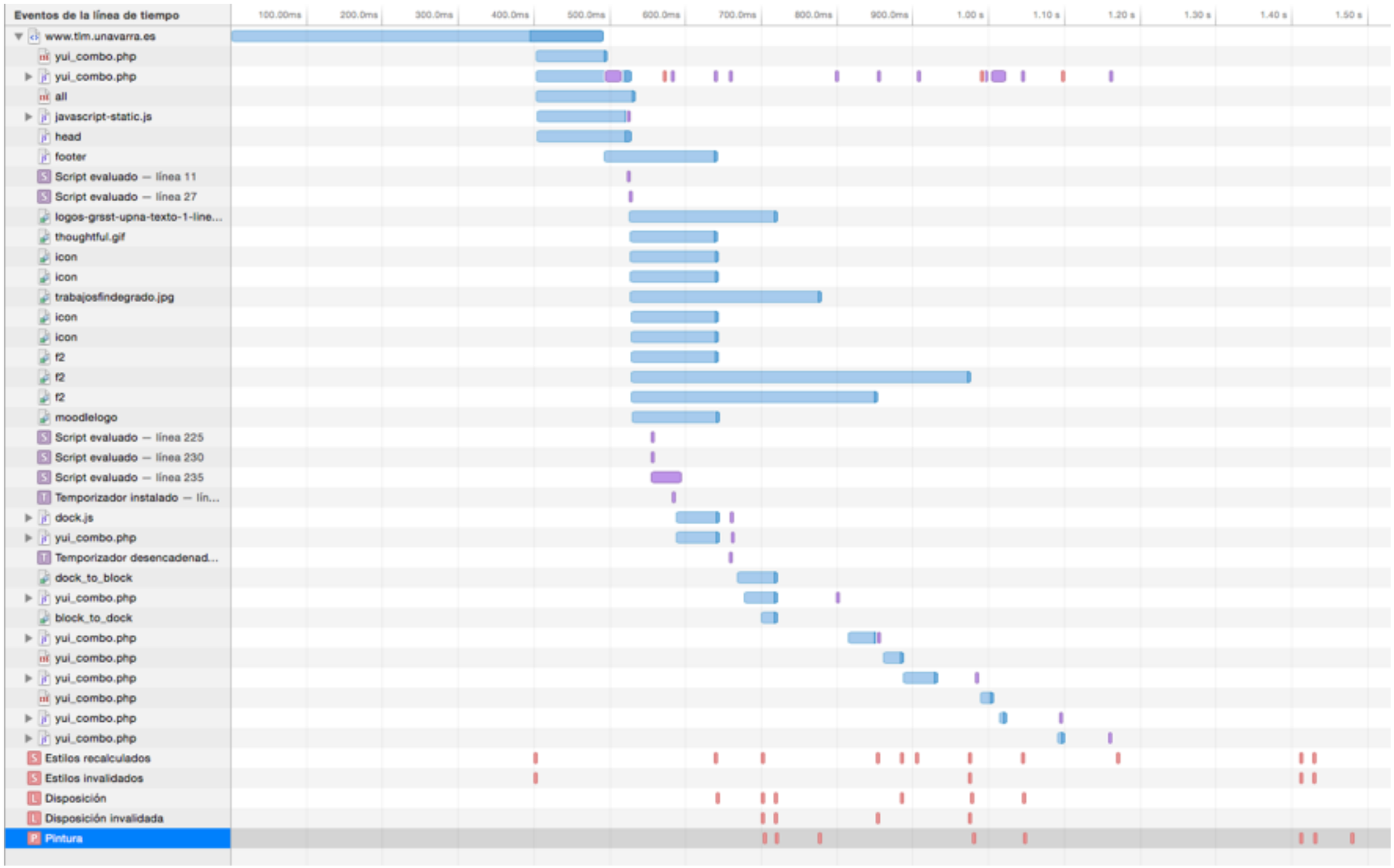
- El pipelining no lo arregla
- En 2014 la mayoría de los navegadores en 2014 no implementan pipelining

Intentando resolver los problemas

- Spriting (enviando una sola imagen para luego utilizar trozos)
- Inlining (incrustando datos de imágenes en css)
- Concatenation (uniendo todo js en un fichero)
- Sharding (manteniendo conexiones en paralelo)
- …

El estado actual…

- i.e: carga de www.tlm.unavarra.es (con RTT < 1ms !!)



Otros puntos de mejora…

- Las cabeceras son ASCII que podría comprimirse facilmente
- Los objetos son muy redundantes porque muchas cabeceras se repiten

- La seguridad extremo a extremo…
- HTTPS es opcional

HTTP2

The way to HTTP2

- HTTPbis working group, para actualizar HTTP1 (en 2007) y proponer HTTP2 (se empieza a hablar en 2012) (aun así HTTP1.1 no se completa hasta 2014
  - Call for proposals para HTTP2 (2012)
- Google propone SPDY de forma independiente (2009)
  - Hay implementaciones de SPDY en servidores de google y Chrome (2010) otros servidores y browsers (2011)
- SPDY se propone como candidato a HTTP2
- SPDY/3 basicamente se convierte en HTTP2
- HTTP2 es aprobado para RFC en 2015 (actualmente en espera para publicar)
- Con HTTP2 SPDY es deprecated y se retirara en 2016

Condiciones para HTTP2

- Tiene que mantener el paradigma de HTTP (c>s over TCP)
- Los URLs http:// y https:// no tienen que cambiar
- Los servidores HTTP1 seguirán por ahi durante décadas
- Los proxies tienen que poder convertir HTTP1 a HTTP2
  - La semantica GET/PUT… no puede cambiar solo la forma de transportarlo
- No más partes opcionales (evitar el caos de HTTP1.1)
- No mas versiones menores HTTP2 no 2.0

Iniciando HTTP2

- Los urls no son distinguibles… como se negocia HTTP2 o 1?

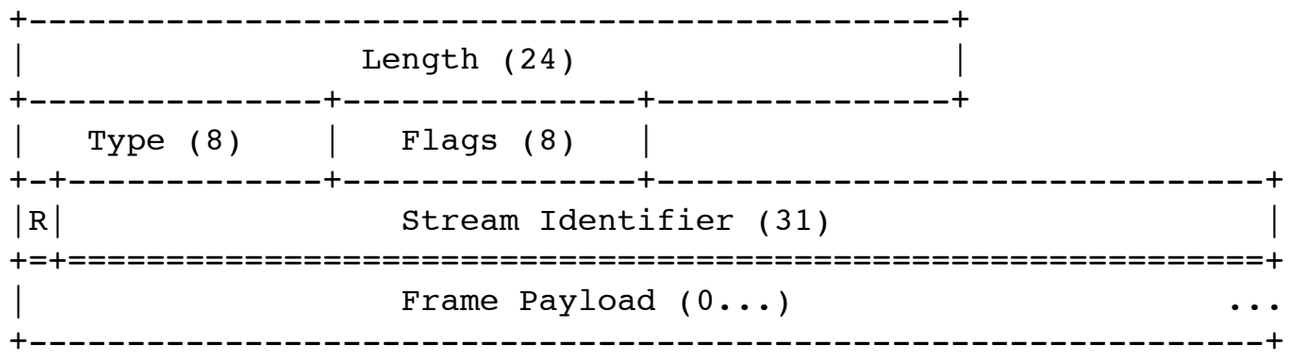
- Directo, petición HTTP1.1 y upgrade a HTTP2
  - Servidor contesta 101 switching protocolos (pero 1RTT)
  - Controvertido algunos navegadores no implementaran
- Sobre TLS dos metodos
  - NPN Next Protocol Negotiation (lo que usaba SPDY)
  - Estándar ALPN Aplication Level Protocol Negotiation
- Mientras se decida le standard la mayoría de las implementaciones soportan los dos NPN y ALPN

Inicio http://xxx

- Conexión TCP
- Abrir con HTTP/1.1 y upgrade

El protocolo

- Binario !!
- Frames



- Una vez establecida la conexión (Upgraded o TLS) los extremos intercambian frames

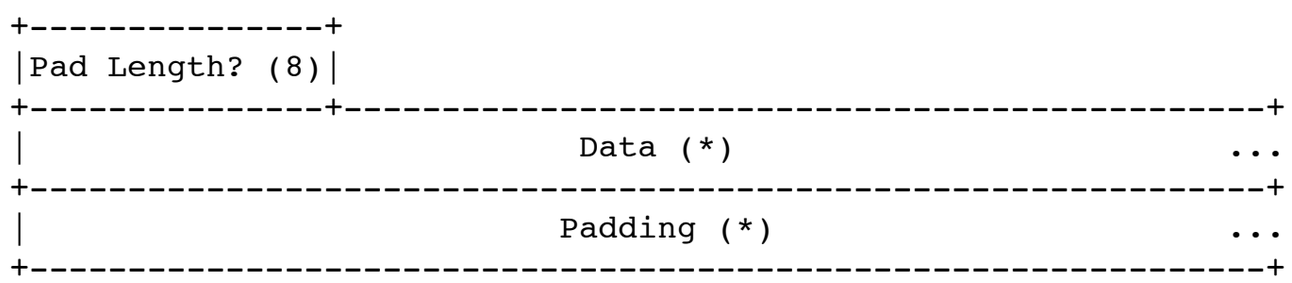
- length - payload size (max 16MB pero limitada por negociacion obligatorio soportar 16384)
- type - diferentes tipos de frames con funciones diferentes
- flags - dependiendo del tipo
- streamid - definir streams / 0 es relativo a la conexión

Frames

- DATA transportar datos request o response
- HEADERS inician una stream
- PRIORITY
- RST_STREAM cierra una stream
- SETTINGS negocian opciones
- PUSH_PROMISE
- PING
- GOAWAY
- WINDOW_UPDATE
- CONTINUATION continua con HEADERS

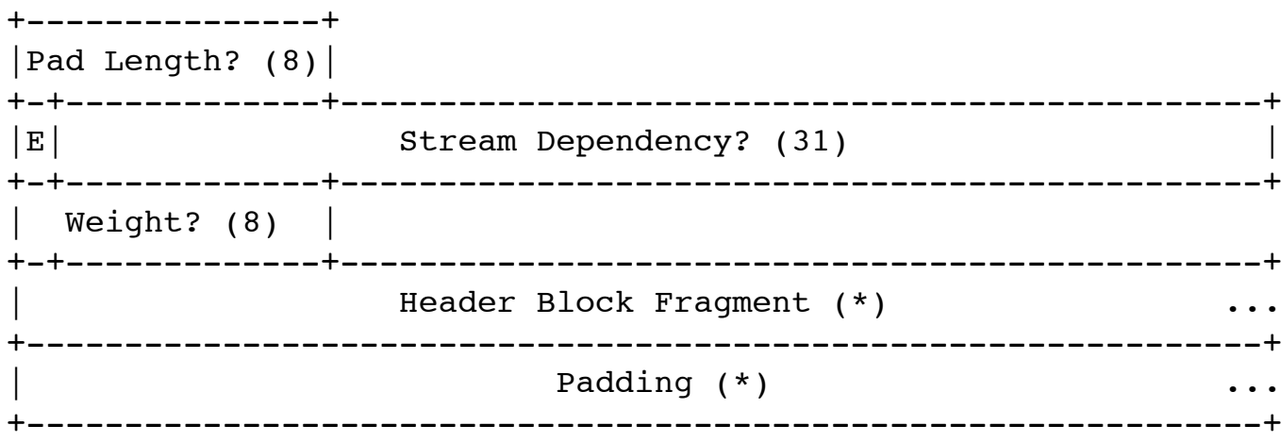
Frame DATA

- Flags: END_STREAM(0x1) PADDED(0x8)
- Padlength y padding solo si lo indica el flag
- Tiene que estar en una stream



Frame HEADER

- Flags: END_STREAM(0x1) END_HEADERS(0x4) PADDED(0x8) PRIORITY(0x20)
- Inicia una stream



- Priority indica dependencia con otra stream (E significa exclusiva)

Compresión

- Cabeceras HTTP repetitivas (HTTP stateless)
- Compresión de cabeceras
  - Especificación separada para compresión de cabeceras
- Compresión de datos

Multiplexacion de streams

- Varias streams independientes
  - Bidireccionales
  - Poco coste
  - Diferentes objetos
- Cada stream tiene su estado, se puede cerrar, tiene flow control
- Dependencias de unas streams con respecto a otras
- Prioridades de unas sobre otras, el cliente puede cambiar la prioridad conforme hacemos scroll por ejemplo

Cancelar streams

- Se pueden cancelar streams una vez que se han pedido
- Frames RST_STREAM
- Por ejemplo cancelar un video que el usuario ha parado

Server PUSH

- El servidor puede prever que si has pedido un recurso también querrás otro y enviártelo en una stream de tipo PUSH_PROMISE
- El cliente puede rechazarlo con RST_STREAM

Prioridades y dependencias

- Declarar dependencias entre streams
- Arbol de dependencias
- Dependencia exclusiva
- Las dependientes solo se envían si no hay trabajo en las superiores

- Prioridades entre las del mismo nivel
- Gobernadas por el cliente que asigna prioridades con frames

Errores

- Error global con frame GOAWAY y cierra la conexión
- Error de stream con RST_STREAM y cierra el stream

Headers y pseudo-headers

- Se añaden pseudo-headers con los campos que en HTTP1 van en la linea de petición
- Request :method :scheme :path :authority
- Response :status

Ejemplos

- Petición

```
GET /resource HTTP/1.1            HEADERS
Host: example.org          ==>      + END_STREAM
Accept: image/jpeg                  + END_HEADERS
                                      :method = GET
                                      :scheme = https
                                      :path = /resource
                                      host = example.org
                                      accept = image/jpeg
```

- Respuesta

```
HTTP/1.1 304 Not Modified         HEADERS
ETag: "xyzzy"              ==>      + END_STREAM
Expires: Thu, 23 Jan ...            + END_HEADERS
                                      :status = 304
                                      etag = "xyzzy"
                                      expires = Thu, 23 Jan ...
```

Ejemplos

- POST

```
POST /resource HTTP/1.1
Host: example.org
Content-Type: image/jpeg
Content-Length: 123

{binary data}
```

==>

```
HEADERS
  - END_STREAM
  - END_HEADERS
    :method = POST
    :path = /resource
    :scheme = https

CONTINUATION
  + END_HEADERS
    content-type = image/jpeg
    host = example.org
    content-length = 123

DATA
  + END_STREAM
{binary data}
```

- Envío objeto

```
HTTP/1.1 200 OK
Content-Type: image/jpeg
Content-Length: 123

{binary data}
```

==>

```
HEADERS
  - END_STREAM
  + END_HEADERS
    :status = 200
    content-type = image/jpeg
    content-length = 123

DATA
  + END_STREAM
{binary data}
```

CONNECT

- Soporta reenvío de túnel con CONNECT como HTTP1

Extensible

- Se puede extender el protocolo añadiendo nuevos tipos de frames en el futuro
- Si una implementación no conoce un tipo de frame debe ignorarla

- BLOCKED
- ALTSVC

Efectos

- Para el usuario
  - Descarga web mas rapida ¿Cuanto mas?
  - Seguridad obligatoria

- Para el desarrollador web
  - Cuidado con los trucos para optimizar HTTP1

Criticas

- Esta hecho para google
- Solo es util para los grandes servicios
- TLS lo hara mas lento
- No es ASCII será mas dificil de depurar
- No es green porque TLS hara que aumente el consumo de energía
- Los navegadores seguirán no aceptando los certificados autofirmados
- No ha eliminado las cookies y sigue haciendo fácil el user tracking
- No se extenderá su uso igual que con IPv6

Implementaciones

- Browsers: Firefox, Chrome, IE
- Servers: …
  - Apache, NGINX, GWS… implementan SPDY
- Proxies: Akamai Ghost, Apache Traffic Server
- Webs: Google, Twitter, Facebook…

- Tools: curl y libcurl, wireshark(decodificar)
- Librerias: en C,nodejs, python, erlang, ruby, java…

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

Nuevos protocolos sobre HTTP

Protocolos avanzados sobre HTTP

- Websockets
- ReverseHTTP