

# HTTP

Sistemas Telemáticos para Medios Audiovisuales

GSyC

Departamento de Teoría de la Señal y Comunicaciones y  
Sistemas Telemáticos y Computación

Octubre de 2018



©2018 Grupo de Sistemas y Comunicaciones.  
Algunos derechos reservados.  
Este trabajo se distribuye bajo la licencia  
Creative Commons Attribution Share-Alike  
disponible en <http://creativecommons.org/licenses/by-sa/3.0/es>

# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias

# Contenidos

- 1 **Introducción**
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias

# Definiciones

## URL (*Universal Resource Locator*)

Interfaz común para acceder a diferentes tipos de servicios/documentos en Internet a través de un sistema de nombres.

## HTML (*HyperText Markup Language*)

Lenguaje de marcado para la elaboración de contenidos integrados por texto, gráficos, etc, que permite incluir en un documento referencias a otros recursos mediante URLs

## HTTP (*HyperText Transfer Protocol*)

Protocolo entre navegadores y servidores WWW para transferir recursos hipermedia (texto, gráficos, audio, vídeo).

# URL



- **Protocolo**: Protocolo por el que se accede al recurso. Por defecto el predeterminado para la aplicación que usa la URL.
- **Máquina**: Máquina en la que reside el recurso. Por defecto la máquina local.
- **Puerto**: Puerto de la máquina a través del que se pide el recurso. Por defecto el predeterminado para el protocolo (http=80)
- **Recurso**: Identificación del recurso dentro de la máquina, incluyendo (a veces) un *path*. Por defecto el recurso predeterminado para la máquina.
- **Identificador de fragmento**: Opcionalmente, se utiliza para identificar un fragmento del recurso.

# HTML

```
<!DOCTYPE html>
<html>
<head>
<title>Mi primera página HTML</title> </head>
<body>
<h1>Primer Título</h1>
<p>
<a href="http://www.w3schools.com">Esto es un enlace</a> </p>
<h2>Primer Subtítulo</h2>
<p>
<!-- Esto es un comentario -->
Una imagen:
 </p>
<p>
<!-- Imagen con URL completa por si viene de otro directorio y/o servidor -->
 </p>
<p>
Si en una URL no se especifica servidor, se entiende que es el mismo. Idem para el directorio</p>
</body>
</html>
```

# Primer Título

[Esto es un enlace](#)

## Primer Subtítulo

Una imagen:

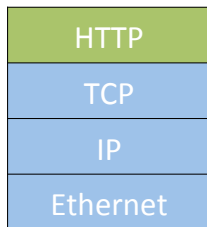


Si en una URL no se especifica servidor, se entiende que es el mismo. Idem para el directorio



# HTTP

- Protocolo de nivel de aplicación utilizado para transferir recursos hipermedia entre ordenadores.
- Sigue el **modelo Cliente-Servidor**:
  - **Cliente HTTP**: navegador web que pide páginas y, al recibirlas, las muestra al usuario. Ej: Firefox, Explorer, Chrome, Safari. . .
  - **Servidor HTTP**: servidor web en el que están alojadas páginas que piden los clientes. Ej: Apache, IIS. . .
- Funciona sobre TCP como protocolo de transporte
- Por defecto un servidor HTTP escucha en el puerto 80, pero puede usar cualquier otro puerto.
- HTTP puede servir tanto **contenido estático** (ficheros) como **contenido dinámico** (el resultado de ejecutar programas en el servidor).



# Versiones de HTTP

- **0.9**: Primera versión documentada, no tiene número de versión oficial, pero es referida como versión 0.9 (1991)
- **1.0**: Primera versión oficial (RFC 1945, año 1996)
- **1.1**: Versión “clásica” (RFC 2068, año 1997 y RFC 2616, año 1999)
- **2.0**: Versión “nueva” (RFC 7540, junio 2015)
  - Soportado en las versiones actuales de todos los navegadores
  - Soportado por el 26 % de los servidores HTTP actuales (según W<sup>3</sup>Tech)

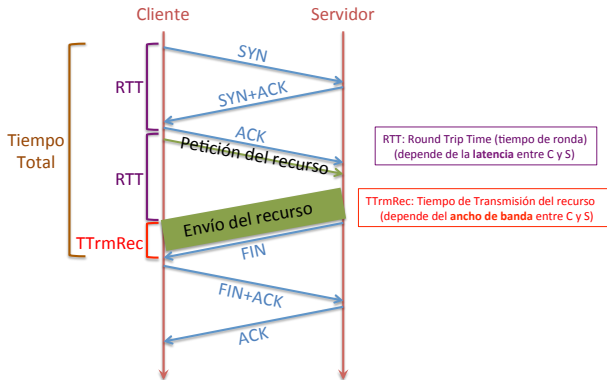
# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP**
- 3 Formato de mensajes HTTP
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias

# Páginas web

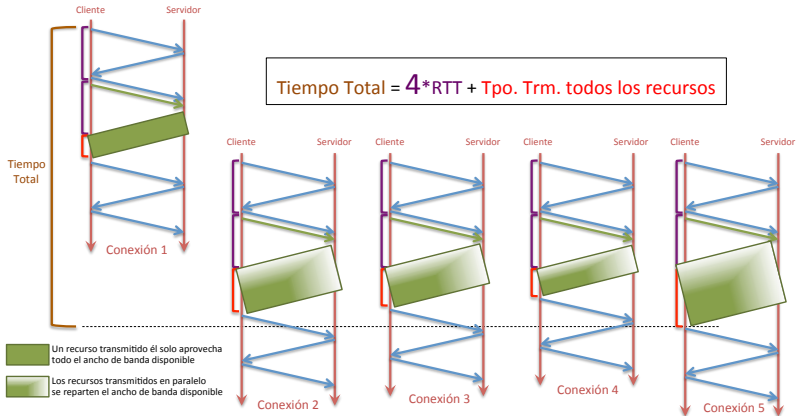
- Una **página web** se compone de uno o más **recursos**.
- Cada recurso suele ser un archivo, y hay recursos de distinto tipo (archivos HTML, imágenes PNG, vídeos AVI, applets Java, etc)
- A un recurso se hace referencia a través de su URL.
- La mayoría de las páginas web están formadas por **un archivo HTML base y varios recursos referenciados** dentro de ese archivo base como contenido adicional de la misma página.
  - Ej: Una página web puede estar compuesta por 6 recursos: 1 fichero HTML y 5 imágenes PNG.

# Petición de una página web de un solo recurso: el archivo base



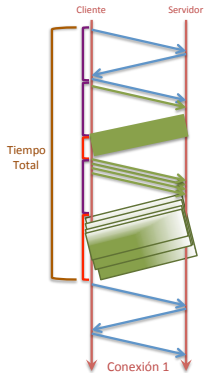
- El tiempo total mide lo que se tarda en tener el recurso en el cliente (para mostrarlo, p.ej), por lo que el tiempo de cerrar la conexión no cuenta.

# Una página con 5 recursos: Conexiones NO Persistentes





- Una vez que el cliente tiene el recurso principal, lo analiza y abre con el servidor conexiones en paralelo con la actual para pedir los 4 recursos adicionales.

# Una página con 5 recursos: Conexiones Persistentes

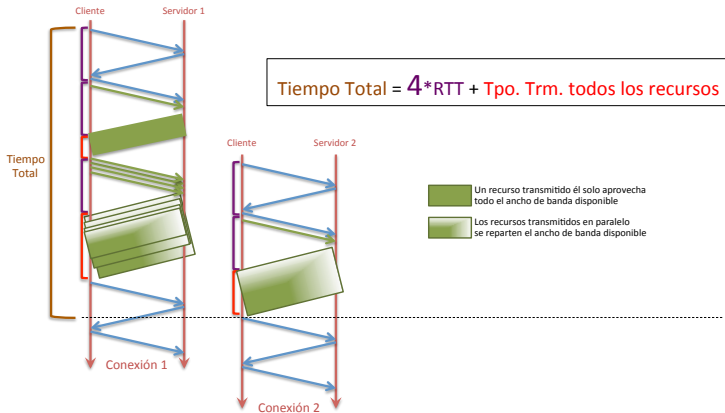


$$\text{Tiempo Total} = 3 * \text{RTT} + \text{Tpo. Trm. todos los recursos}$$

-  Un recurso transmitido él solo aprovecha todo el ancho de banda disponible
-  Los recursos transmitidos en paralelo se reparten el ancho de banda disponible

- Una vez que el cliente tiene el recurso principal, lo analiza y pide por la misma conexión los 4 recursos adicionales, que se envían en paralelo.
- Es 1 RTT más rápido que con conexiones persistentes.

# Conexiones Persistentes con recursos en servidores diferentes



- En cuanto al menos un recurso adicional esté en otro servidor será imprescindible abrir una nueva conexión, lo que introduce el RTT adicional.



# Detalles del tiempo de respuesta de un recurso HTTP



- **Búsqueda en el DNS:** Depende del RTT de acceso al servidor de DNS de la máquina, y del tiempo que tarda en encontrar la respuesta el servidor de DNS.
- **Establecimiento de la conexión TCP:** Depende del RTT de acceso al servidor web.
- **Solicitud de HTTP:** Depende del RTT de acceso al servidor web.
- **Respuesta de HTTP:** Depende del ancho de banda entre el cliente y el servidor web. Si el recurso es grande, este tiempo es el más grande, si el recurso es pequeño, este tiempo es el más pequeño.

# Tiempo de carga de página: Ingeniería de “usabilidad”

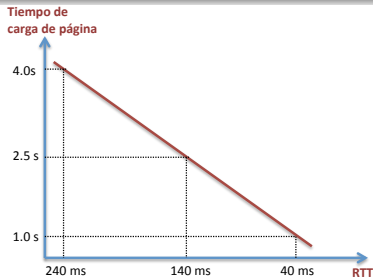
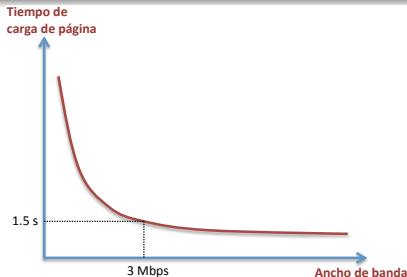
## Tiempo de carga de página (*Page Load Time*)

Tiempo desde que el usuario hace clic en un enlace hasta que ve la página pedida “razonablemente” completa.

Tiempo	Reacción del usuario
0 - 100 ms	carga instantánea
100 - 300 ms	sensación de lentitud
300 - 1000 ms	espera apreciable
1 - 10 s	cambio de contexto mental
> 10 s	abandono de la página

- Se considera importante mantenerse por debajo de los **250 ms** para que la navegación sea fluida.
- Valores experimentales medios a día de hoy:
  - RTT: 100 ms
  - Tiempo de carga de página: 7 s (mediana: 3 s)
- Valores aún peores usando dispositivos móviles.

# Tiempo de carga de página: cómo mejorarlo



- El tiempo de carga de página depende sobre todo del RTT, y no tanto del ancho de banda.
- El RTT depende de la latencia, no del ancho de banda (básicamente,  $RTT = 2 * \text{latencia}$ )
- A día de hoy, mejorar el ancho de banda ya no mejora el tiempo de carga de la página. Hay que trabajar en el RTT:
  - reduciendo la latencia
  - minimizando el número de RTTs necesarios para cargar la página

# Valores típicos en una aplicación web hoy

Carga de la página principal de una aplicación web:

- 90 solicitudes de HTTP, obtenidas de 15 servidores, 1300 KB transferidos, 3 segundos:
  - HTML: 10 solicitudes, 52 KB
  - Imágenes: 55 solicitudes, 812 KB
  - JavaScript: 15 solicitudes, 216 KB
  - CSS: 5 solicitudes, 36 KB
  - Otros: 5 solicitudes, 195 KB

# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP**
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias

- ## Línea inicial

## Líneas de cabecera

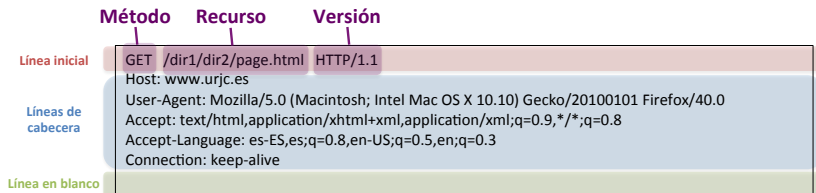
**Línea en blanco**

b  
bb  
bbb  
bbbb  
bbbbb  
bbbbb  
bbbbb  
bbbbb  
bbbbb  
bbbbb  
bbbbb  
bbbbb  
bbbbb

# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP**
  - Petición HTTP
  - Respuesta HTTP
  - Líneas de cabecera en petición/respuesta HTTP
  - GET y POST para envío de datos al servidor
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP
  - Cache-Control y ETag en HTTP/1.1
  - Expires, Last-Modified en HTTP/1.0
  - Uso de Cache-Control en una petición
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias

# Formato peticiones



- La especificación del recurso incluye la ruta (*path*), pero no el nombre de máquina.
- La versión del protocolo indica la versión de HTTP bajo la que se hace la petición.
- Las peticiones normalmente no llevan cuerpo (aunque, a veces, sí).



# Métodos de las peticiones

- **GET:**
  - Solicita un recurso, la petición no lleva cuerpo. El recurso solicitado se indica en la línea inicial.
  - Ejemplo: solicitar una página web.
- **POST:**
  - Envía datos a un recurso del servidor. El recurso se indica en la línea inicial y los datos van en el cuerpo de la petición.
  - Ejemplo: enviar los datos que rellena el usuario a través de un formulario web.
- **PUT:**
  - Actualiza un recurso existente en el servidor (“actualiza” una página web).
  - Hoy no se usa para este fin, pues las páginas WWW se colocan en los servidores por mecanismos externos a HTTP.
- **DELETE:**
  - Elimina en el servidor el recurso especificado.
  - También en desuso para este fin.
- ...

# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP**
  - Petición HTTP
  - **Respuesta HTTP**
  - Líneas de cabecera en petición/respuesta HTTP
  - GET y POST para envío de datos al servidor
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP
  - Cache-Control y ETag en HTTP/1.1
  - Expires, Last-Modified en HTTP/1.0
  - Uso de Cache-Control en una petición
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias

# Formato respuestas



- La versión del protocolo indica la mayor versión que entiende el servidor, pero siempre se responde atendiendo a la versión de la petición recibida.
- Las respuestas normalmente sí llevan cuerpo: el recurso solicitado. Las cabeceras **Content-Length** y **Content-Type** son necesarias en este caso.

# Respuestas: Resultado

- Códigos de estado del resultado en los mensajes de respuesta:
  - 1xx: Mensaje informativo
  - 2xx: Resultado con éxito
    - 200 OK
  - 3xx: Redirección del cliente a otra URL
    - 301 Moved permanently
    - 304 Not Modified
  - 4xx: Error en el lado del cliente
    - 404 Not Found
  - 5xx: Error en el lado del servidor
    - 500 Internal Server Error

# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP**
  - Petición HTTP
  - Respuesta HTTP
  - **Líneas de cabecera en petición/respuesta HTTP**
  - GET y POST para envío de datos al servidor
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP
  - Cache-Control y ETag en HTTP/1.1
  - Expires, Last-Modified en HTTP/1.0
  - Uso de Cache-Control en una petición
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias

# Líneas de cabecera

- Mismo formato que las cabeceras del formato de los correos electrónicos (RFC 822, sección 3).
- En HTTP/1.0 se definen 16 cabeceras, ninguna obligatoria.
- En HTTP/1.1 se definen 46 cabeceras, **siendo obligatoria en las peticiones la cabecera Host**: nombre del servidor del recurso

Host: www.urjc.es

- Se recomienda incluir en las peticiones al menos:

- **User-Agent**: tipo de navegador

User-Agent: Mozilla/5.0

- Se recomienda incluir en las respuestas al menos:

- **Server**: tipo de servidor

Server: Apache/1.3

- **Last-Modified**: fecha de última modificación del recurso

Last-Modified: Wed, 28 May 2014 18:41:38 GMT

- Si un mensaje tiene cuerpo, es obligatorio que se incluyan las cabeceras:

- **Content-Type**: tipo MIME de lo que va en el cuerpo

Content-Type: text/html

- **Content-Length**: tamaño en bytes del cuerpo

Content-Length: 10726

# Cabecera para conexiones persistentes y no persistentes

- En HTTP/1.0 las conexiones, por defecto, son no persistentes.
- Si en HTTP/1.0 se quiere usar conexiones persistentes (los servidores no están obligados a soportarlas):
  - 1 el cliente incluirá en su petición la cabecera  
`Connection: Keep-Alive`
  - 2 si el servidor lo acepta incluirá en su respuesta la cabecera  
`Connection: Keep-Alive`
- En HTTP/1.1 y HTTP/2 las conexiones, por defecto son persistentes.
- Si en HTTP/1.1 se quiere usar conexiones no persistentes:
  - 1 el cliente incluirá en su petición la cabecera  
`Connection: close`
  - 2 el servidor incluirá en su respuesta la cabecera  
`Connection: close`

# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP**
  - Petición HTTP
  - Respuesta HTTP
  - Líneas de cabecera en petición/respuesta HTTP
  - **GET y POST para envío de datos al servidor**
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP
  - Cache-Control y ETag en HTTP/1.1
  - Expires, Last-Modified en HTTP/1.0
  - Uso de Cache-Control en una petición
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias



# Uso de GET y POST para envío de datos al servidor

- Los formularios se utilizan para enviar datos desde el navegador al servidor:

Nombre:

Edad:

- **POST:**

- Envía datos al servidor, normalmente los introducidos por el usuario en un formulario.
- **Los datos van en el cuerpo.**
- El *path* de la línea inicial (URL) se refiere normalmente al programa que tratará los datos que se envían.

- **GET:**

- GET también permite enviar los datos de un formulario. En este caso, **los datos van en el path de la línea inicial (URL), y no hay cuerpo.**
- El tamaño de los datos subidos con GET está limitado por el tamaño máximo de una URL (255 caracteres), por lo que NO se utiliza GET para subir datos de formularios grandes.

# Ejemplo de formularios HTML

- Código HTML de un formulario que enviará los datos mediante GET:

```
<FORM action="http://pc2.emp2.net/form.php" method=GET>
  <P>
    Nombre: <INPUT type="text" name="nombre"><BR>
    Edad: <INPUT type="text" name="edad"><BR>
    <INPUT type="submit" value="Enviar"><INPUT type="reset">
  </P>
</FORM>
```

- Código HTML de un formulario que enviará los datos mediante POST:

```
<FORM action="http://pc2.emp2.net/form.php" method=POST>
  <P>
    Nombre: <INPUT type="text" name="nombre"><BR>
    Edad: <INPUT type="text" name="edad"><BR>
    <INPUT type="submit" value="Enviar"><INPUT type="reset">
  </P>
</FORM>
```

# Ejemplo de envío de datos con GET y POST

- Cuando el usuario rellene los datos y pulse sobre el botón "Enviar", el navegador mandará un mensaje con los datos al servidor.

```
GET /form.php?nombre=Fulano+Mengano&edad=24 HTTP/1.0
Host: pc2.emp2.net
User-Agent: Mozilla/4.5 [en]
Accept: image/jpeg, image/gif, text/html
Accept-language: en
Accept-Charset: iso-8859-1
```

```
POST /form.php HTTP/1.0
Host: pc2.emp2.net
User-Agent: Mozilla/4.5 [en]
Accept: image/jpeg, image/gif, text/html
Accept-language: en
Accept-Charset: iso-8859-1
Content-Type: application/x-www-form-urlencoded
Content-Length: 26

nombre=Perico+Palotes&edad=24
```

- ? : separación entre el recurso y los parámetros
- = : separación entre nombre del campo del formulario y su valor
- & : separación entre parámetros
- + : espacio en blanco

# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP
- 4 Aplicaciones web**
- 5 Caché de contenidos en HTTP
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias

# Páginas web dinámicas: Aplicaciones Web

- Las URL ahora NO identifican un recurso almacenado en un fichero, sino que **el recurso pedido se crea dinámicamente**, ejecutándose un programa en el servidor.
- En la URL se especifica el nombre del programa a ejecutar y puede incluir argumentos que se le pasan al programa.
- Estos programas que se ejecutan en el servidor suelen estar escritos en lenguajes de *scripting*: *shell-scripts*, PHP, Perl, Python, Ruby
- El resultado de la ejecución de estos programas es una página web (o una parte de ella).

# Aplicaciones web que siguen el API REST

- **REST** (Representational State Transfer) es una forma de diseñar aplicaciones web que hace un uso de las URLs y de los métodos de HTTP totalmente diferente al prevista en el protocolo:
  - Las URLs no referencian recursos consistentes en archivos, ni recursos generados dinámicamente, sino objetos (elementos, componentes) de una aplicación.
  - Método POST: se invoca para crear un objeto
  - Método GET: se invoca para consultar el estado de un objeto
  - Método PUT: se invoca para modificar el estado de un objeto
  - Método DELETE: se invoca para borrar un objeto
- Este API REST ha hecho que se vuelvan a utilizar los métodos PUT y DELETE, abandonados en el HTTP convencional.

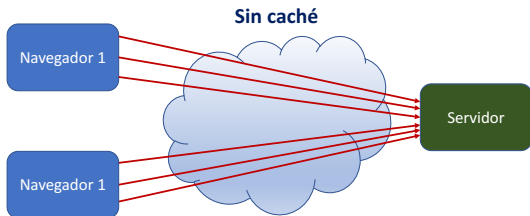
# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP**
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias

# Cachés en HTTP

## Objetivo

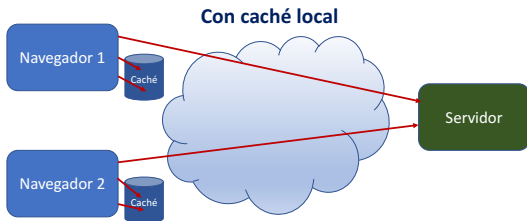
Minimizar las transferencias vía HTTP de recursos que no han cambiado.



- Cuando no se utilizan cachés:
  - si un mismo cliente pide varias veces un mismo recurso que no ha cambiado, todas las veces se transfiere su contenido.
  - si varios clientes piden el mismo recurso que no ha cambiado, todas las veces se transfiere su contenido.

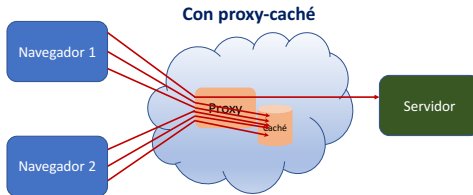


# Caché local (privada)



- Cada navegador mantiene una **caché local (privada)**, para cada usuario:
  - la primera vez que se pide un recurso, se almacena en la caché
  - las siguientes veces el recurso puede obtenerse de la caché sin volver a transferirlo desde el servidor.

# Caché global: Proxy-caché



- Los navegadores pueden estar configurados para realizar las peticiones HTTP a través de una máquina intermedia, el **proxy-caché**. Los clientes no se comunican directamente con los servidores que tienen los recursos, es el proxy-caché el que hace de intermediario y se comunica por una parte con los clientes y por otra con los servidores finales.
- Las peticiones a un proxy se distinguen porque incluyen la URL completa en la primera línea del mensaje de petición. Ejemplo:  
`GET http://gsyc.escet.urjc.es/index.html HTTP/1.0`
- Las peticiones y respuestas que envía un proxy incluyen la línea de cabecera **Via**. Ejemplo:  
`Via: 1.0 myproxy.com`
- Un proxy HTTP casi siempre tiene asociada una **caché global**:
  - la primera vez que un cliente pide al Proxy un recurso, se almacena en la caché
  - las siguientes veces el recurso puede obtenerse de la caché sin volver a transferirlo desde el servidor.

# Funcionamiento general de las cachés

- Un servidor puede controlar si se puede o no almacenar cada recurso que envía, y durante cuánto tiempo, para según qué **tipo de caché** (privada o pública).
- Un recurso cacheado tiene un **tiempo de vigencia**, durante el cuál el recurso es servido directamente de la caché sin consultar al servidor que originalmente lo envió.
- Un recurso **caducado** provoca que quien lo tiene intente **revalidarlo**, es decir, comprobar si su contenido ha cambiado en el servidor original:
  - si NO hubiera cambiado, NO se vuelve a transferir, se sirve desde la caché, y se define un nuevo tiempo de vigencia
  - si SÍ hubiera cambiado, SÍ se vuelve a transferir, y si el nuevo contenido puede volver a cachearse, se definirá un nuevo tiempo de vigencia.
- Bajo determinadas circunstancias (ej: desconexión de la red), se puede servir contenidos caducados desde una caché, es decir, sin revalidar.

# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP
  - Petición HTTP
  - Respuesta HTTP
  - Líneas de cabecera en petición/respuesta HTTP
  - GET y POST para envío de datos al servidor
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP**
  - Cache-Control y ETag en HTTP/1.1
  - Expires, Last-Modified en HTTP/1.0
  - Uso de Cache-Control en una petición
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias

# Cabecera **Cache-Control** en respuestas

- Introducida en HTTP 1.1.
- Enviada por los servidores en las respuestas para controlar si un recurso puede cachearse o no, cómo y cuánto tiempo.
- Principales valores (pueden concatenarse varios):
  - **no-store**: el recurso no puede almacenarse
  - **no-cache**: si el recurso se almacenara, no podría servirse desde la caché sin revalidarlo previamente
  - **must-revalidate**: cuando el recurso caduque, es obligatorio revalidarlo (es decir, no puede servirse caducado en ninguna circunstancia)
  - **public**: el recurso puede ser almacenado en cachés de cualquier tipo
  - **private**: el recurso sólo puede ser almacenado en cachés privadas
  - **max-age=X**: número X de segundos que el recurso tiene de vigencia
- Ejemplos:

```
Cache-Control: no-cache, must-revalidate
```

```
Cache-Control: max-age=86400
```

```
Cache-Control: private, max-age=600
```

# Cabecera ETag

- Introducida en HTTP 1.1.
- Identifica de manera única un recurso para su posterior validación.
- Lo genera el servidor cuando envía la respuesta con el recurso solicitado. No se especifica la forma de generar estos identificadores, pero típicamente serán un *hash* del recurso.

ETag: "33a64df551425fcc55d9f25f89d4"

- El ETag se utilizará para revalidar un recurso caducado. El cliente solicitará el recurso y usará la cabecera `If-None-Match` cuyo valor contendrá el ETag de dicho recurso (que recibió previamente y desea saber si se ha modificado):

`If-None-Match: "33a64df551425fcc55d9f25f89d4"`

- En el servidor se comprobará el valor recibido en la cabecera `If-None-Match`:
  - Si el recurso ha cambiado tendrá un valor diferente ETag y por tanto no coincidirá con el enviado en la cabecera `If-None-Match`. El servidor enviará de nuevo el recurso.
  - Si el recurso no ha cambiado, el ETag enviado para revalidar será el mismo que el almacenado en el servidor y por tanto, el servidor no necesita enviar de nuevo el recurso.
- La validación con ETag se considera **validación fuerte**.

# Ejemplo

- Respuesta de un servidor a un recurso:

```
HTTP/1.1 200 OK
Date: Mon, 3 Apr 2017 13:19:41 GMT
Server: Apache/1.3.3 (Unix)
Cache-Control: max-age=3600, must-revalidate
ETag: "3e86-410-3596fbbc"
Content-Length: 1040
Content-Type: text/html
```

- Significado:
  - **Cache-Control**: puede cachearse, vigencia de 1 hora, no puede devolverse sin revalidar (cuando caduque).
  - **Etag** para validación
- Mientras no caduque (durante 1 hora), se servirá de la caché.
- El valor por defecto para Cache-Control es **private**, por tanto, solo puede ser almacenada en una caché privada.
- Tras caducar, el cliente tendrá que revalidar el recurso utilizando la cabecera If-None-Match.

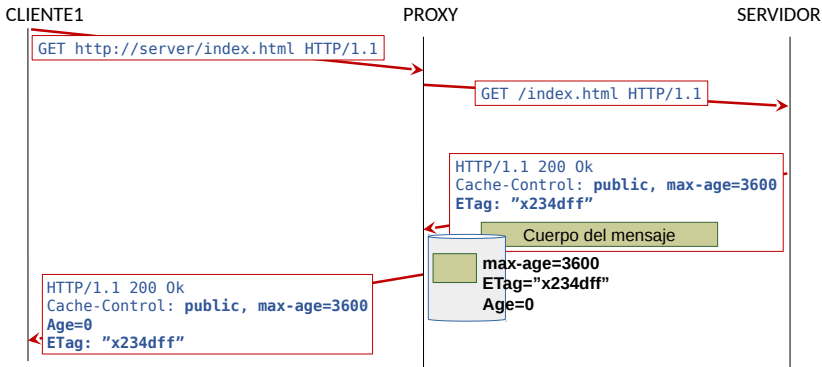
# Respuestas servidas por un Proxy desde su caché

- La respuesta incluye la cabecera **Age**, que indica el tiempo en segundos de vigencia ya consumido por el recurso desde que está en la caché (o ha sido revalidado):  
**Age: 20**
- Si tiene valor 0 indica que el recurso está recién obtenido.



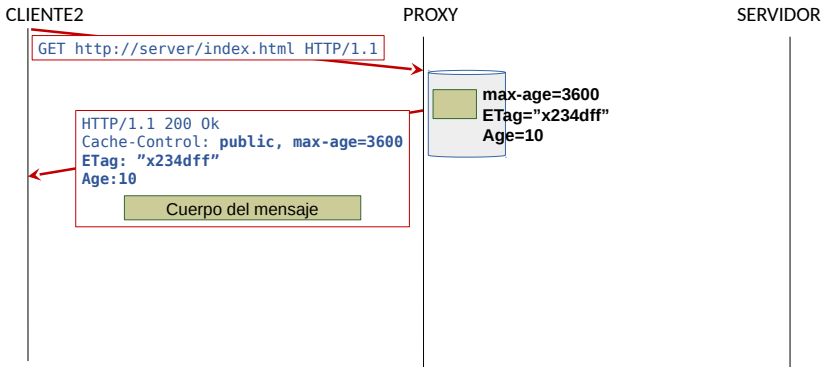
# Caché pública: cabeceras Cache-Control y ETag (I)

- El cliente1 solicita recurso a un servidor a través de un proxy que no lo tiene en su caché.



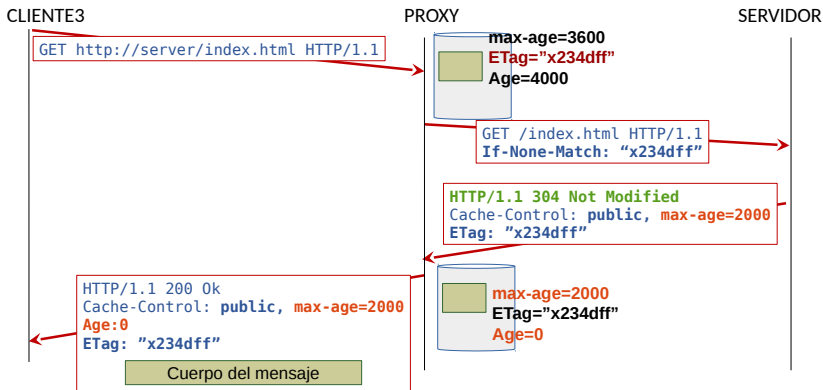
# Caché pública: cabeceras Cache-Control y ETag (II)

- El cliente2 solicita recurso a un servidor a través de un proxy que lo tiene vigente en su caché.



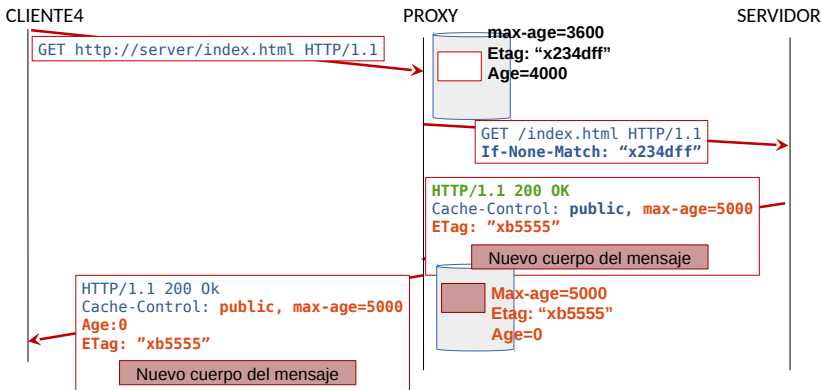
# Caché pública: cabeceras Cache-Control y ETag (III)

- El cliente3 solicita recurso a un servidor a través de un proxy que lo tiene caducado en su caché y tiene que revalidarlo. El recurso no ha cambiado.



# Caché pública: cabeceras Cache-Control y ETag (IV)

- El cliente4 solicita recurso a un servidor a través de un proxy que lo tiene caducado en su caché y tiene que revalidarlo. El recurso ha cambiado.



## Caché privada: cabeceras Cache-Control y ETag (IV)

- Los ejemplos de caché pública y cabeceras Cache-Control y Etag pueden aplicarse también a las cachés privadas.
- Con las cachés privadas es el propio navegador (cliente) el que almacena los recursos y comprueba su tiempo de vigencia para solicitar la revalidación al servidor.
- En entornos reales se producen situaciones que combinan tanto la caché privada del navegador como cachés públicas (pueden encadenarse proxys HTTP).

# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP
  - Petición HTTP
  - Respuesta HTTP
  - Líneas de cabecera en petición/respuesta HTTP
  - GET y POST para envío de datos al servidor
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP**
  - Cache-Control y ETag en HTTP/1.1
  - **Expires, Last-Modified en HTTP/1.0**
  - Uso de Cache-Control en una petición
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias

# Cabecera Expires

- Introducida en HTTP 1.0.
- Establece el límite de vigencia de un recurso (equivalente a max-age pero con una fecha absoluta en vez de un período).
- Ejemplo:

```
Expires: Wed, 21 Oct 2017 07:28:00 GMT
```

- Si se envía esta cabecera junto con `Cache-Control`, `Cache-Control` tiene precedencia.

# Cabecera Last-Modified

- Introducida en HTTP 1.0.
- Si no existe NI `Cache-Control` NI `Expires`, permite calcular el tiempo de vigencia de un recurso usando también la cabecera `Date` (hora en la que el servidor envía la respuesta), mediante la fórmula:

$$\text{tiempo\_de\_vigencia} = \frac{\text{Date} - \text{Last-Modified}}{10}$$

- Ejemplo:

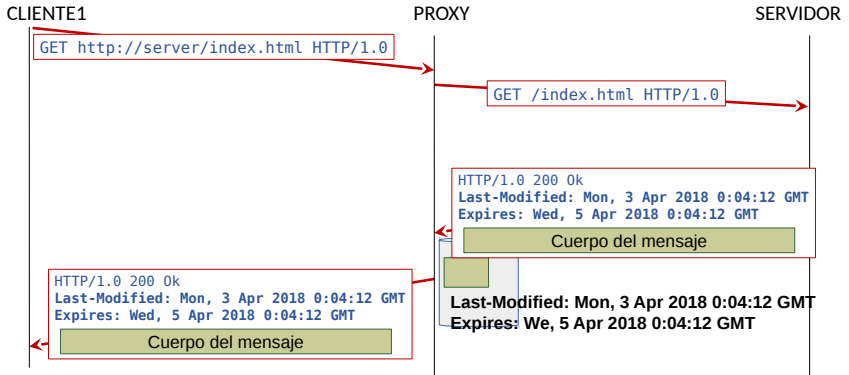
`Last-Modified: Wed, 20 Nov 2016 17:32:23 GMT`

- La cabecera `Last-Modified` se utiliza también para validación de recursos:
  - Para revalidar un recurso que tiene `Last-Modified`, el cliente usará la cabecera `If-Modified-Since`.
  - La validación mediante `Last-Modified` se considera **validación débil**.



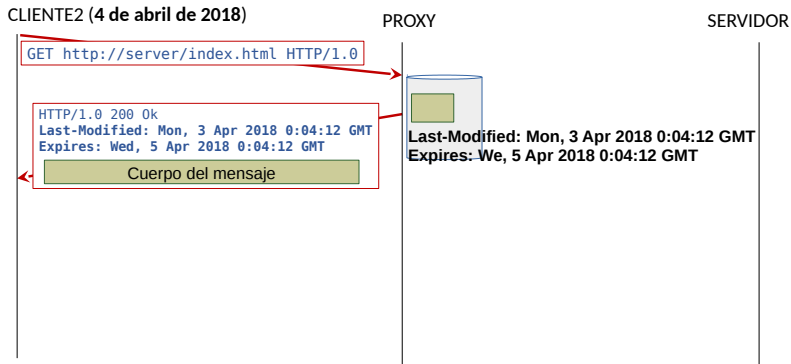
# Caché pública: cabeceras Expires y Last-Modified (I)

- El cliente1 solicita una página a un servidor a través de un proxy.



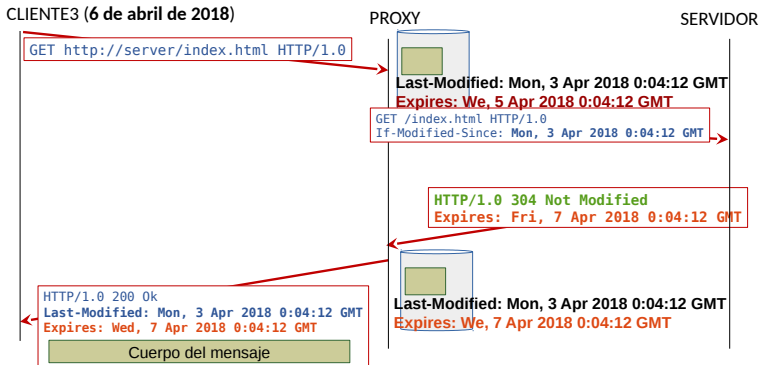
# Caché pública: cabeceras Expires y Last-Modified (II)

- El cliente2 solicita una página a un servidor a través de un proxy. El proxy se la sirve de su caché.



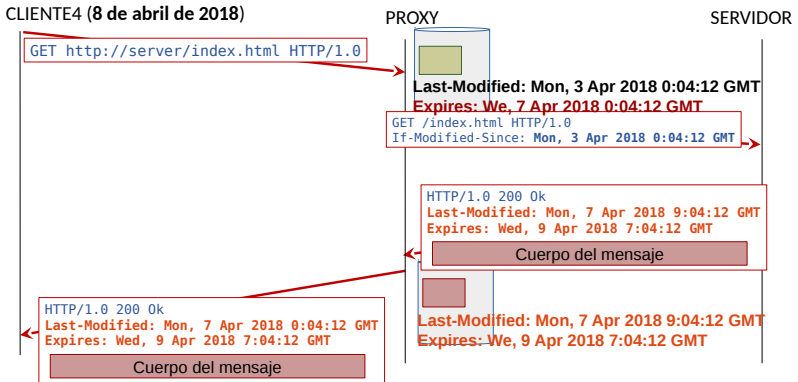
# Caché pública: cabeceras Expires y Last-Modified (III)

- El cliente3 solicita una página a un servidor a través de un proxy. El proxy la tiene caducada y necesita revalidarla. La página no ha cambiado



# Caché pública: cabeceras Expires y Last-Modified (IV)

- El cliente4 solicita una página a un servidor a través de un proxy. El proxy la tiene caducada y necesita revalidarla. La página ha cambiado.



# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP
  - Petición HTTP
  - Respuesta HTTP
  - Líneas de cabecera en petición/respuesta HTTP
  - GET y POST para envío de datos al servidor
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP**
  - Cache-Control y ETag en HTTP/1.1
  - Expires, Last-Modified en HTTP/1.0
  - **Uso de Cache-Control en una petición**
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias

# Cabecera **Cache-Control** en las peticiones

- La cabecera **Cache-Control** puede incluirse en las peticiones.
- Principales valores (pueden concatenarse varios):
  - **no-cache**: el recurso no puede provenir de una caché sin haber sido revalidado
  - **no-store**: quien reciba la petición no puede almacenar la respuesta
  - **max-age=X**: el cliente sólo quiere una respuesta cacheada si su **Age** es menor o igual que X segundos.

# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP
- 6 Cookies**
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias

# Persistencia de estado en HTTP

- HTTP se diseña de forma que **un servidor no almacena estado por cada petición**. Es decir: cada petición es completamente independiente de otras peticiones que haya hecho antes el mismo cliente.
- Sin embargo, es muy frecuente que aplicaciones web necesiten de mantener un estado persistente entre distintas operaciones de un mismo cliente con un mismo servidor
  - Ejemplo: datos asociados a un usuario (carro de la compra, login de usuario...)
- Soluciones:
  - 1 El estado es mantenido por el servidor de forma externa a HTTP (basándose en la IP del cliente, o en otros datos)
  - 2 Se utiliza HTTP para que el estado se mantenga en los clientes:
    - Mediante URLs incluidas en las páginas que va devolviendo el servidor: se incrusta el estado como parte de la URL
    - En campos (ocultos) de formularios que envía el servidor con el formulario para que posteriormente viajen como parámetros (con GET o POST) al mandar el formulario relleno el cliente al servidor.
    - **Mediante cookies** (RFCs 2109 y 2965).
  - 3 Combinaciones de las dos anteriores: se guardan datos de forma externa (en una base de datos) referenciados por una cookies.



# Cookies

- Las **cookies** son datos asociados a identificadores.
- Funcionamiento:
  - 1 el servidor genera una *cookie* para representar un estado asociado a un cliente que ha hecho una petición
  - 2 el servidor envía la *cookie* al cliente
  - 3 el cliente almacena la *cookie*
  - 4 el cliente reenvía la *cookie* al servidor en las futuras peticiones que le realice
- Especificación original de Netscape, luego propuesto como RFC 2109, ampliada en RFC 2965.

# Cabecera Set-Cookie

- Cabecera con la que un servidor que ha creado un cookie se la envía a un cliente.
- El formato incluye:
  - Nombre de la cabecera: `Set-Cookie`
  - Nombre de la *cookie* y valor: `<nombre>=<valor>`
  - Fecha de caducidad: `expires=<fecha>`
  - Dominio (servidor) y trayecto (*path*) para el que es válida: El cliente la reenviará a ese servidor para todas las peticiones de recursos que **empiecen por ese path**:  
`domain=<dominio>; path=<trayecto>`
  - Si debe ser transmitida sólo sobre canales seguros (HTTPS):  
`secure`
- Ejemplo:

```
Set-Cookie: login=pepe; expires=Mon, 30-Jan-2009 12:35:23 GMT;  
domain=www.myserver.com; path=/dir; secure
```

# Cabecera Cookie

- Cuando un cliente pide una URL a un servidor, buscará en su lista de *cookies* almacenadas de peticiones anteriores, las que cumplen **simultáneamente** las siguientes condiciones:
  - 1 Aún no han expirado
  - 2 Son del mismo *domain* (servidor) al que va se va a hacer la nueva petición
  - 3 La URL que se va a pedir empieza por el *path* para el que es válida
- El cliente enviará todas las *cookies* que cumplen las condiciones, en una o más cabeceras **Cookie**.
- Dentro de esta cabecera, las *cookies* se ordenarán de más a menos específicas (según su *path*).
- Las *cookies* con caducidad en el pasado se eliminarán periódicamente para liberar espacio en el cliente.
- Ejemplo:

```
Cookie: login=pepe; theme=basic
```

# Ejemplo: el servidor envía Cookie a cliente

- Un cliente solicita un recurso a un servidor y éste guarda información de esta petición asociándole la cookie `session-id=11111`.
- El cliente almacenará dicha cookie.



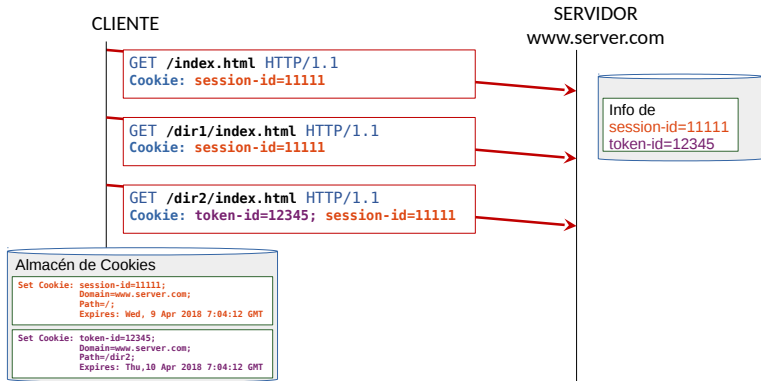
# Ejemplo: el cliente envía Cookie al servidor

- El cliente enviará la cookie en posteriores accesos si se cumplen las condiciones: Expires, Domain, Path.



# Ejemplo: el cliente envía Cookie al servidor

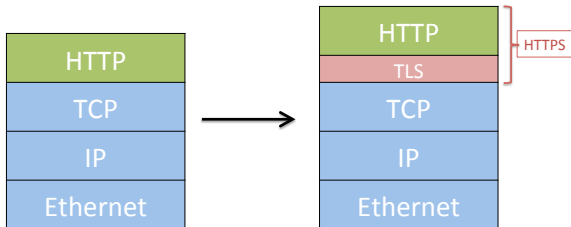
- El cliente enviará la cookie en posteriores accesos si se cumplen las condiciones: Expires, Domain, Path.
- Primero se envían las cookies más específicas.



# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP
- 6 Cookies
- 7 HTTPS**
- 8 HTTP 2.0
- 9 Referencias

# HTTPS



- El término **HTTPS** se refiere a colocar HTTP sobre **TLS** (*Transport Layer Security*).
- TLS aporta características de seguridad a las conexiones TCP. Antiguamente se llamaba SSL (*Secure Sockets Layer*)
- Permite confidencialidad en la conexión (mediante cifrado), autenticación de los extremos e integridad del contenido.
- Las URLs comienzan por <https://>



# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0**
- 9 Referencias

# HTTP 2.0

- **HTTP/2** (versión 2.0) es la nueva versión de HTTP.
- Principales características:
  - Se mantiene la semántica de HTTP/1.1 en cuanto a métodos, códigos de respuesta, URLs, cabeceras. . .
  - Se especifica cómo será la interacción entre clientes 1.1 y servidores 2.0 o viceversa
  - Una sola conexión para transmitir todos los recursos del mismo servidor, cada petición/respuesta de un recurso será un flujo (*stream*) dentro de la conexión
  - Cabeceras binarias (no de texto), mediante compresión (el tamaño medio de cabeceras en 1.1: 800 bytes (!!!)).
  - Paquetes de datos y paquetes de control
  - Prioridades en los flujos para evitar tener que esperar a completar los recursos previos para que llegue un recurso imprescindible para ir mostrando todo el contenido.
  - Sobre TLS.
  - Extensible para poder cubrir futuras necesidades.

# SPDY

- HTTP/2.0 está basado (como punto de partida) en el protocolo **SPDY**.
- SPDY es un protocolo desarrollado por Google para reemplazar HTTP de forma más eficiente.
  - SPDY se puede usar hoy
  - Reduce el tiempo de carga de página un 65 % utilizando flujos, compresión y formato binario de paquetes.
  - Necesita clientes y servidores que lo soporten:
    - Clientes: Chrome, Firefox 13+, Opera 12.10+, Explorer 11+
    - Servidores: Todas las aplicaciones web de Google, Twitter, Facebook, Wordpress. . .
    - <http://spdycheck.org/> para comprobar si un sitio soporta SPDY.

# Contenidos

- 1 Introducción
- 2 Relación entre HTTP y conexiones TCP
- 3 Formato de mensajes HTTP
- 4 Aplicaciones web
- 5 Caché de contenidos en HTTP
- 6 Cookies
- 7 HTTPS
- 8 HTTP 2.0
- 9 Referencias**

# Referencias

- J.J. Kurose y K.W. Ross, **Redes de Computadores: un enfoque descendente basado en Internet**, Pearson Educación, 2ª edición.
- W. Richard Stevens, **TCP/IP Illustrated, vol 3**, Addison Wesley.
- James Marshall, **HTTP Made Really Easy. A Practical Guide to Writing Clients and Servers**,  
<http://www.jmarshall.com/easy/http/>
- RFC 1945, **HTTP 1.0**,  
<http://www.faqs.org/rfcs/rfc1945.html>
- RFC 2068, **HTTP 1.1**,  
<http://www.faqs.org/rfcs/rfc2068.html>
- RFC 7540, **HTTP 2.0**,  
<http://www.faqs.org/rfcs/rfc7540.html>
- RFC 2964, **Use of HTTP State Management**,  
<http://www.faqs.org/rfcs/rfc2964.html>
- RFC 2965, **HTTP State Management Mechanism**,  
<http://www.faqs.org/rfcs/rfc2965.html>