HTTP

Sistemas Telemáticos para Medios Audiovisuales

Departamento de Sistemas Telemáticos y Computación (GSyC)

Marzo de 2012

Contenidos

Contenidos

Definiciones

HTTP *(HyperText Transfer Protocol)*

Protocolo entre navegadores y servidores WWW para transferir documentos hipermedia.

URL *(Universal Resource Locator)*

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

HTML *HyperText Markup Language*

Lenguaje que permite incluir en documentos enlaces a otros documentos mediante URLs.

HTTP

- Protocolo utilizado para transferir páginas web.
- Sigue el modelo Cliente-Servidor:
  - Cliente: navegador web que pide páginas y, al recibirlas, las muestra al usuario.
  - Servidor: servidor web en el que están alojadas páginas que piden los clientes.
- Funciona sobre TCP.
- Por defecto un servidor HTTP escucha en el puerto 80.
- HTTP puede servir tanto contenido estático (ficheros) como contenido dinámico (el resultado de ejecutar programas en el servidor).

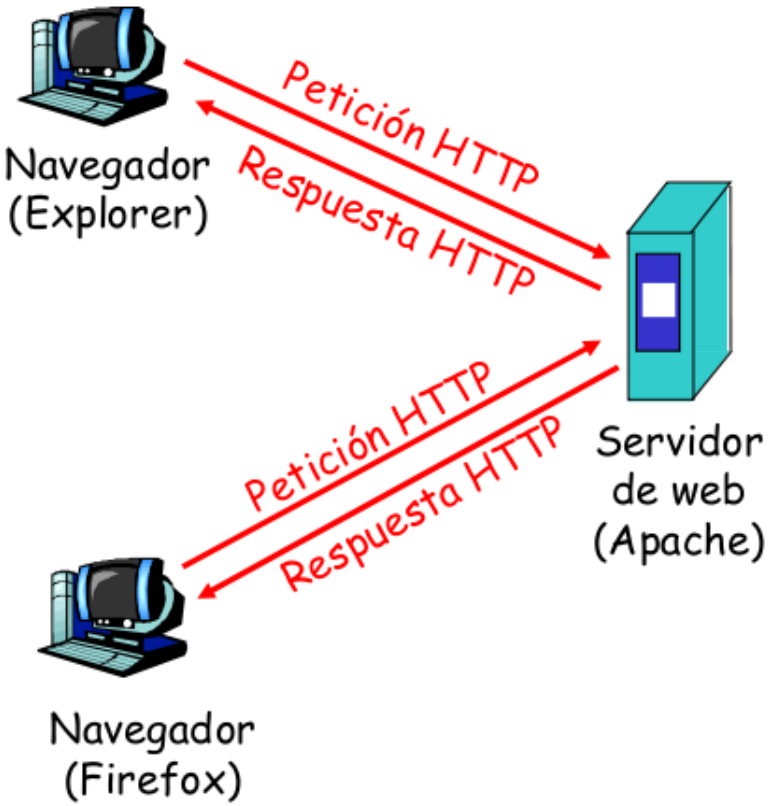
Páginas web

- Una página web consta de uno o más objetos.
- Un objeto es un archivo (un archivo HTML, una foto JPG, un applet Java, etc) que es direccionable a través de su URL.
- La mayoría de las páginas web están formadas por un archivo HTML base y diversos objetos referenciados dentro del archivo como parte de la misma página.
  - Ej: un fichero HTML y 5 imágenes JPG forman una página web compuesta por 6 objetos.

Contenidos

Interacción cliente-servidor en HTTP

- Pasos:
  1. El cliente abre una conexión TCP con el servidor.
  2. El cliente envía un mensaje de petición.
  3. El servidor responde con un mensaje de respuesta.
  4. El servidor cierra la conexión TCP.
- HTTP no mantiene estado (un servidor no se guarda información sobre las peticiones anteriores hechas por un cierto cliente).
  - El servidor podría guardar estado de forma externa a HTTP.



Conexiones No Persistentes y Conexiones Persistentes

HTTP sobre Conexiones No Persistentes

- Se envía un objeto como máximo por una conexión TCP.
- HTTP/1.0 utiliza sólo conexiones no persistentes.

HTTP sobre Conexiones Persistentes

- Se pueden enviar múltiples objetos por una sola conexión TCP entre el cliente y el servidor.
- HTTP/1.1 utiliza por defecto conexiones persistentes, aunque puede usar también conexiones no persistentes

Conexiones No Persistentes (I)

- Supongamos que un navegador solicita la URL: `http://www.uni.edu/departamento/index.html` que consta de texto HTML y 10 imágenes JPG

1a. El cliente HTTP inicia la conexión TCP con el servidor HTTP de `www.uni.edu` en el puerto 80.

1b. El servidor HTTP en `www.uni.edu` estaba esperando conexiones TCP en el puerto 80 y "acepta" esta conexión

2. El cliente HTTP envía un mensaje HTTP de petición (que contiene la URL) a través de la conexión TCP. El mensaje indica que el cliente quiere el objeto `/departamento/index.html`

3. El servidor HTTP recibe el mensaje de petición, compone un mensaje HTTP de respuesta que contiene el objeto solicitado, y lo envía al cliente.

Conexiones No Persistentes (II)

4. El servidor HTTP cierra la conexión TCP.

5. El cliente HTTP recibe el mensaje de respuesta que contiene el archivo html. Analizando el archivo HTML se encuentran referenciados 10 objetos JPG que forman parte de la página.

6. Se repiten los pasos del 1 al 4 para cada uno de los 10 objetos JPG.

Modelo del tiempo de respuesta

RTT

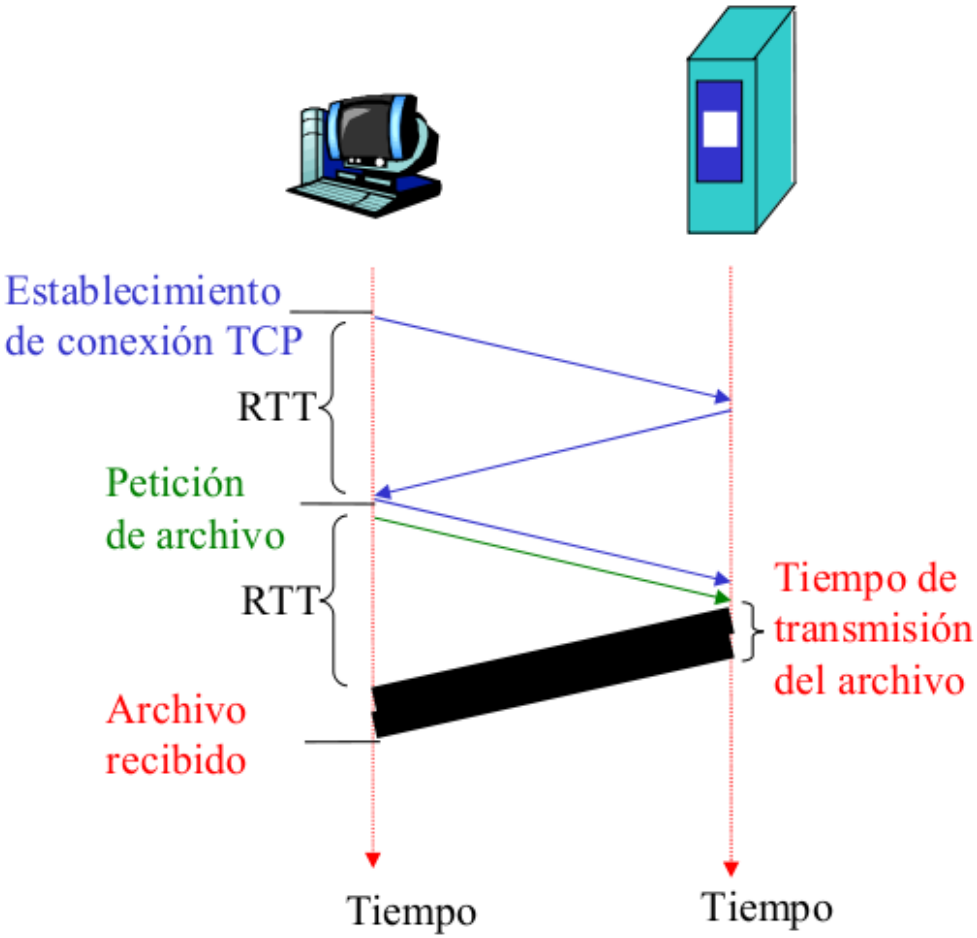
Tiempo necesario para enviar un paquete "pequeño" desde el cliente hasta el servidor y otro de vuelta al cliente.

Tiempo de Respuesta

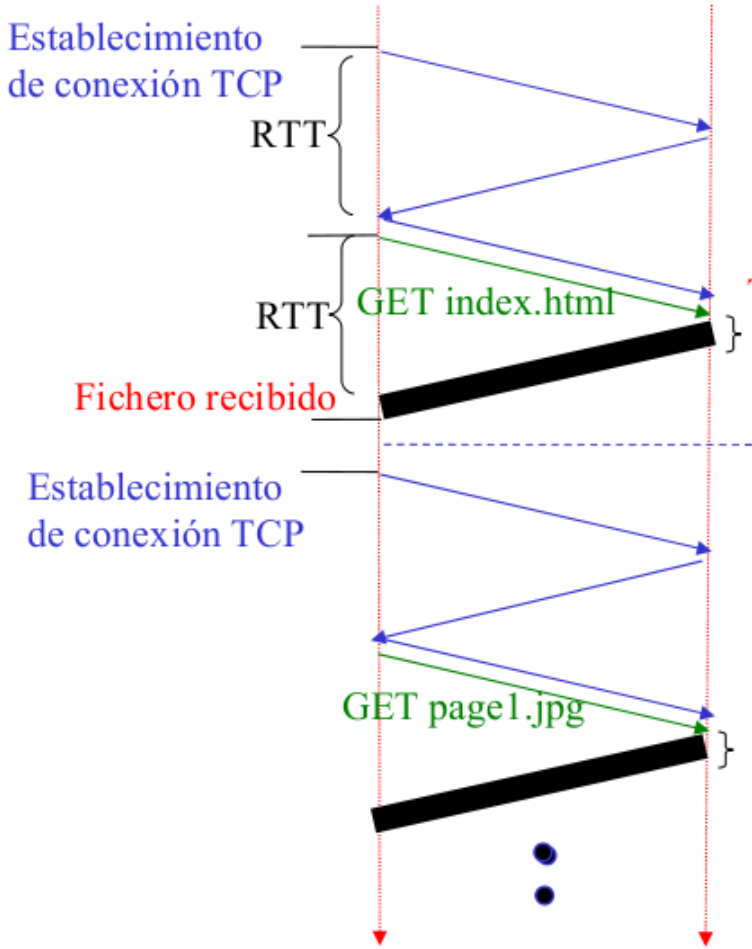
Incluye:

- Un RTT para iniciar la conexión TCP.
- Un RTT para la petición HTTP y los primeros bytes de respuesta HTTP de vuelta.
- El tiempo de transmisión del archivo

Total = 2 RTT + tpo. trans. archivo



Conexiones No Persistentes



http://www.somewhere.com/index.html

index.html referencia: page1.jpg, page2.jpg, page3.jpg.

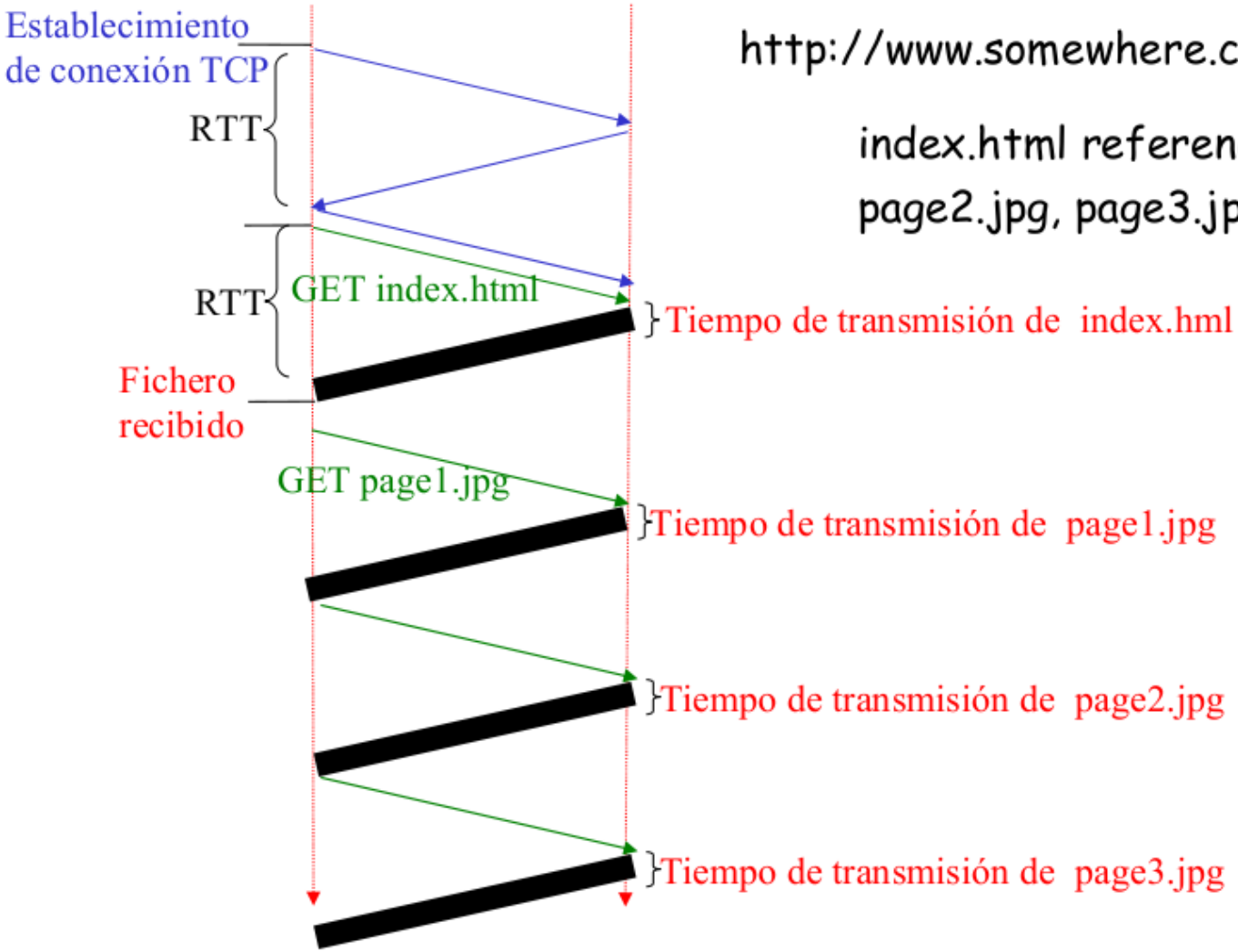
Tiempo de transmisión de index.hml

Cierre de conexión TCP
(en paralelo con la nueva)

Tiempo de transmisión de page1.jpg

Las conexiones para obtener page2.jpg y page3.jpg podrían abrirse en paralelo con la que pide page1.jpg

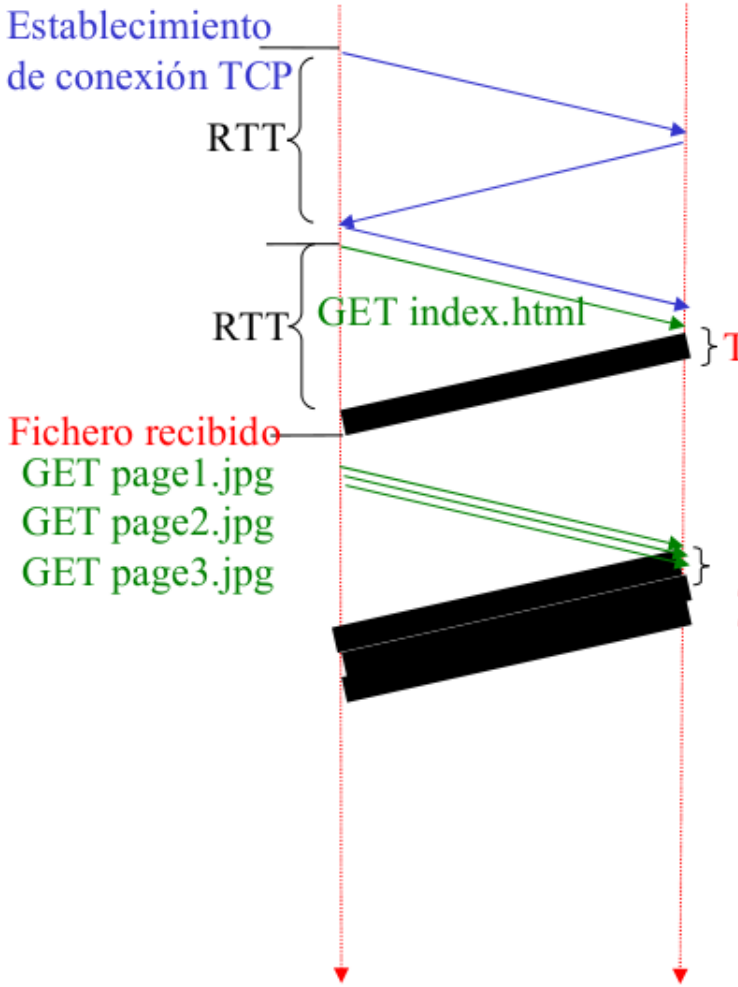
Conexiones Persistentes Sin *Pipelining*



http://www.somewhere.com/index.html

index.html referencia: page1.jpg, page2.jpg, page3.jpg.

Conexiones Persistentes Con *Pipelining*



http://www.somewhere.com/index.html

index.html referencia: page1.jpg, page2.jpg, page3.jpg.

Tiempo de Respuesta: Resumen

Conexiones no persistentes:

- 2 RTT por cada objeto.
- El sistema operativo gasta tiempo en asignar los recursos del host para cada conexión TCP.
- Los navegadores suelen abrir conexiones TCP paralelas para traer los objetos referenciados.

Conexiones persistentes:

- El servidor deja la conexión TCP abierta tras enviar la respuesta.
- Los mensajes HTTP posteriores entre el mismo cliente/servidor se envían por la misma conexión.
- El servidor cerrará la conexión inactiva pasado un plazo.

Conexiones persistentes sin pipelining

- El cliente sólo emite una nueva petición una vez que ha recibido la anterior respuesta.
- 1 RTT por cada objeto.

Conexiones persistentes con pipelining

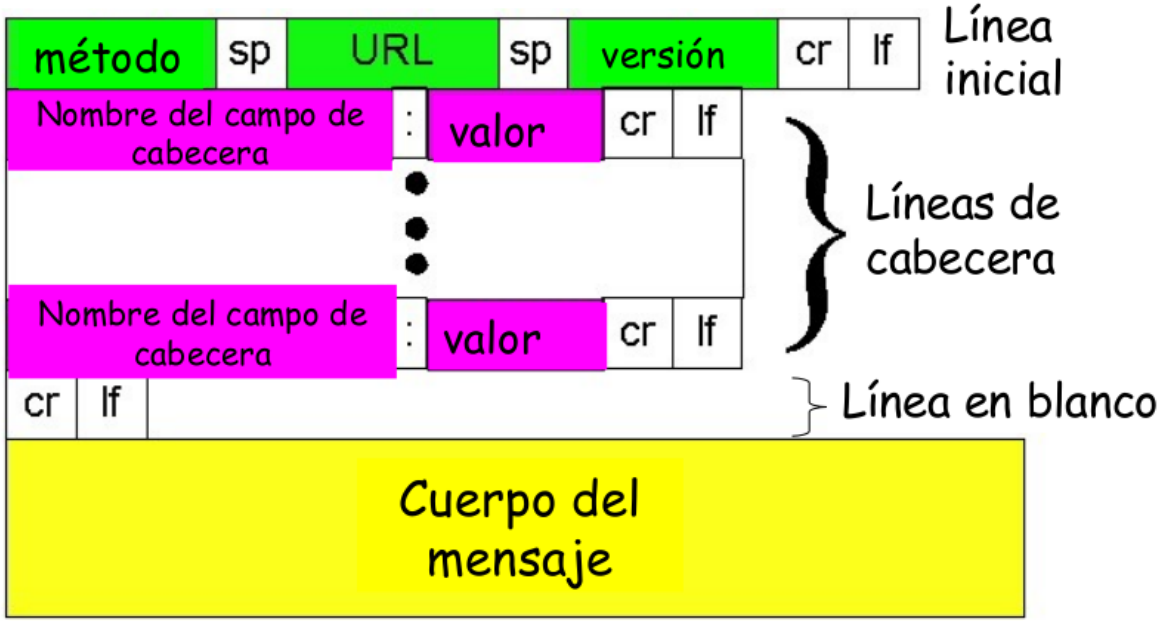
- Por defecto en HTTP/1.1.
- El cliente hace su petición tan pronto como encuentra un objeto referenciado.
- Tan sólo 1 RTT para todos los objetos.

Contenidos

Formato general de los mensajes

- Mensajes compuestos por líneas de texto:
  - Línea inicial (diferente para peticiones y respuetas), terminada en <CR><LF> *(Carriage Return + Line Feed)*.
  - Líneas de cabecera (0 ó más), cada una terminada en <CR><LF>, con el siguiente formato:

    `Cabecera-X: Valor-X <CR><LF>`
  - Línea en blanco (<CR><LF>).
  - Cuerpo del mensaje (opcional).
- Todas las líneas pueden terminar <LF> en vez de <CR><LF>,

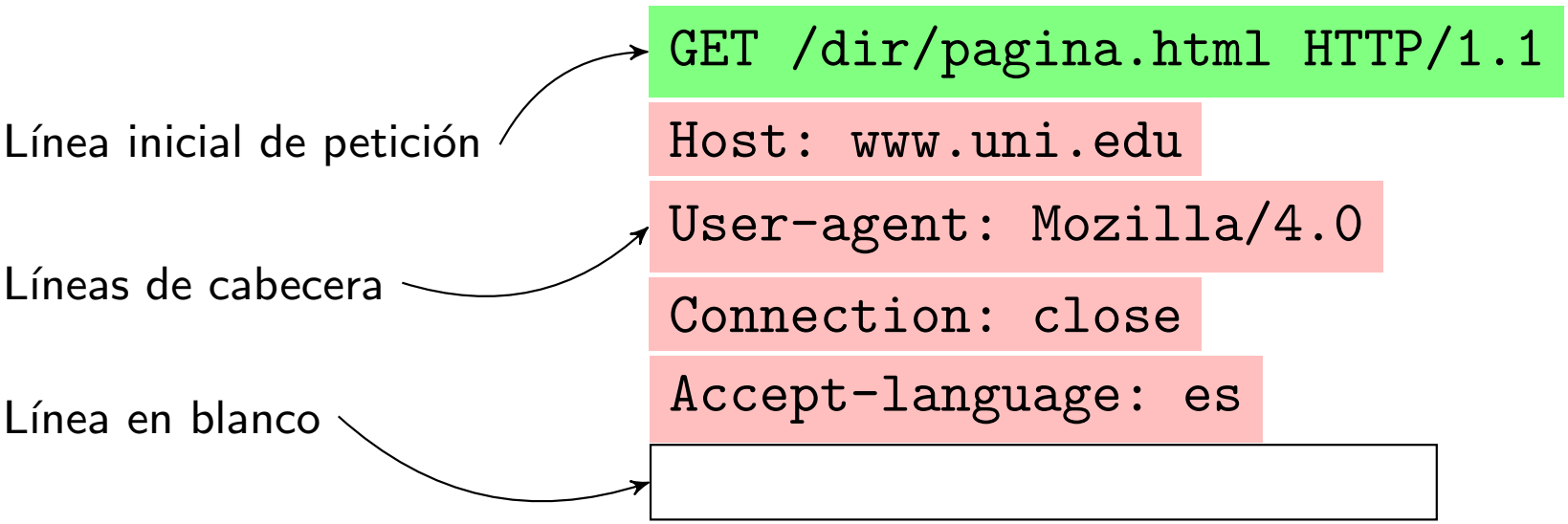


Línea inicial en peticiones

- Especifica el recurso que se solicita y qué se quiere de él:
  - Nombre del método (GET, POST, HEAD)
  - Trayecto de acceso (path)
  - Versión de HTTP (en la forma `HTTP/x.y`)
- Ejemplo:

```
GET /directorio/otro/fichero.html HTTP/1.0
```

Ejemplo de mensaje HTTP de petición

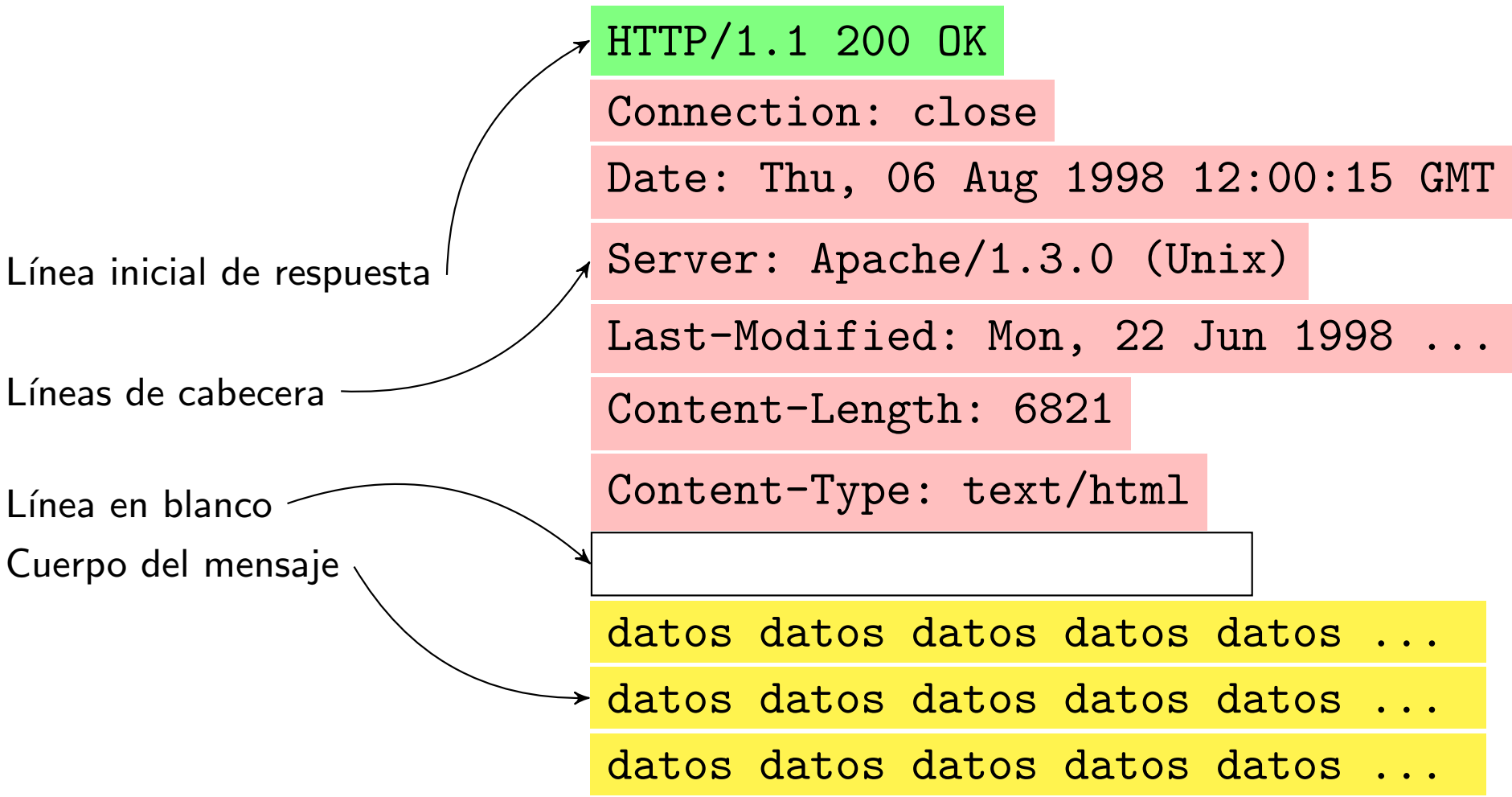


Línea inicial en respuestas

- Proporciona información de estado:
  - Versión de HTTP (siempre HTTP/x.y).
  - Código de estado numérico.
  - Código de estado textual.
- Códigos de estado:
  - 1xx: Mensaje informativo.
  - 2xx: Resultado exitoso (200 OK).
  - 3xx: Redirección del cliente a otra URL (301 Moved permanently, 303 See Other).
  - 4xx: Error en el lado del cliente (404 Not Found).
  - 5xx: Error en el lado del servidor (500 Server Error).
- Ejemplo:

```
HTTP/1.0 200 OK
```

Ejemplo de mensaje HTTP de respuesta



Líneas de cabecera

- Mismo formato que las cabeceras de correo y News (RFC 822, sección 3).
- En HTTP/1.0 se definen 16 cabeceras, ninguna obligatoria.
- En HTTP/1.1 se definen 46 cabeceras, siendo la cabecera `Host:` obligatoria en las peticiones.
- Se recomienda incluir en las peticiones al menos:
  - `User-Agent:` (ej: Mozilla/4.7).
- Se recomienda incluir en las respuestas al menos:
  - `Server:` (ej: Apache/1.3).
  - `Last-Modified:` (fecha GMT, usada por las caches).

Cabeceras comunes para peticiones y respuestas

- `Content-Type:`
  - Descripción MIME de la información contenida en este mensaje.
  - MIME (Multipurpose Internet Mail Extensions): Estándar que especifica como debe un programa transferir archivos de cualquier tipo (no necesariamente ASCII). Los tipos MIME se especifican con "tipo-de-contenido/subtipo":
    - `text/html`, `text/plain` ...
    - `image/gif`, `image/jpeg`, `image/tiff` ...
    - `video/mpeg`, `video/quicktime` ...
- `Content-Length:`
  - Longitud en bytes de los datos enviados.
- `Content-Encoding:`
  - Formato de codificación de datos enviados en el mensaje. Para enviar datos comprimidos (z-gzip, o z-compress).
- `Date:`
  - Fecha local de la operación, incluye zona horaria.

Cabeceras sólo para peticiones

- `Accept:`
  - Lista de tipos MIME aceptados por el cliente. Se puede utilizar `*` para indicar rangos de tipos de datos; `tipo/*` indica todos los subtipos de un determinado medio, mientras que `*/*` representa a cualquier tipo de dato disponible.
- `Authorization:`
  - Clave de acceso que envía un cliente para acceder a un recurso de uso protegido o limitado.
- `From:`
  - Dirección de correo electrónico del usuario del cliente Web que realiza el acceso.
- `If-Modified-Since:`
  - Permite realizar operaciones GET condicionales, en función de si la fecha de modificación del objeto requerido es anterior o posterior a la fecha proporcionada.
- `Referer:`
  - Contiene la URL del documento desde donde se ha activado este enlace. De esta forma, un servidor puede informar al creador de ese documento de cambios o actualizaciones en los enlaces que contiene.
- `User-agent:`
  - Cadena que identifica el tipo y versión del cliente que realiza la petición. Ejemplo: un navegador tipo Mozilla envíaría `User-Agent: Mozilla/4.5`

Cabeceras sólo para respuestas

- `Allow:`
  - Informa de los comandos HTTP opcionales que se pueden aplicar sobre el objeto al que se refiere la respuesta
- `Expires:`
  - Fecha de expiración del objeto enviado.
- `Last-modified:`
  - Fecha local de modificación del objeto devuelto
- `Location:`
  - Informa sobre la dirección exacta del recurso al que se ha accedido. Cuando el servidor proporciona un código de respuesta de la serie 3xx, este parámetro contiene la URL necesaria para accesos posteriores a este recurso.
- `Server:`
  - Cadena que identifica el tipo y versión del servidor:
    `Server: Apache/1.3.0 (Unix)`
- `WWW-Authenticate:`
  - Cuando se accede a un recurso protegido o de acceso restringido, el servidor devuelve un código de estado 401, y utiliza este campo para informar de los modelos de autenticación válidos para acceder a este recurso.

Cabeceras 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 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`

Cuerpo del mensaje

- En las respuestas contiene el recurso pedido o texto explicando un error.
- En las peticiones contiene datos (por ejemplo, los introducidos por el usuario en un formulario) o ficheros para subir.
- Si hay cuerpo, normalmente algunas cabeceras son relativas a él:
  - `Content-Type:` tipo MIME de los datos que van en el cuerpo.
  - `Content-Length:` número de bytes que hay en el cuerpo.

Métodos GET, HEAD, y POST

- GET:
  - Solicita un objeto al servidor especificando su URL.
- HEAD:
  - Igual que un GET, pero sólo pide las cabeceras.
  - Con este método se pueden consultar las características sin descargar el objeto:
    - Permite que los clientes puedan comprobar si ha habido modificaciones en un objeto sin necesidad de transferirlo (para comparar, por ejemplo, si ha habido modificiaciones frente a una versión en caché).
- 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 envian.
  - NOTA: También se pueden enviar datos con un GET. En este caso, los datos van codificados en el *path* de la línea inicial (URL), y no hay cuerpo.

Ejemplo de formularios HTML

- 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>
```

- 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

```
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

Otros métodos

- PUT:
  - Actualiza información sobre un objeto del servidor. Similar a POST, pero el servidor debe almacenar en el *path* que acompaña al método en la línea inicial el contenido del mensaje.
  - Originalmente ideado para subir a un servidor páginas WWW, pero hoy no se usa (se suben páginas WWW por mecanismos externos a HTTP).
- DELETE:
  - Elimina en el servidor el documento especificado.
  - También en desuso.
- ...

Usando HTTP desde `telnet`

1. En un terminal, se realiza un `telnet` al puerto 80 de la máquina del servidor:

```
telnet www.urjc.es 80
```

Abre una conexión TCP con el puerto 80 de `www.urjc.es`. Cualquier cosa que se teclee se enviará por la conexión.

2. En un terminal, se realiza un `telnet` al puerto 80 de la máquina del servidor:

```
GET /index.html HTTP/1.0

```

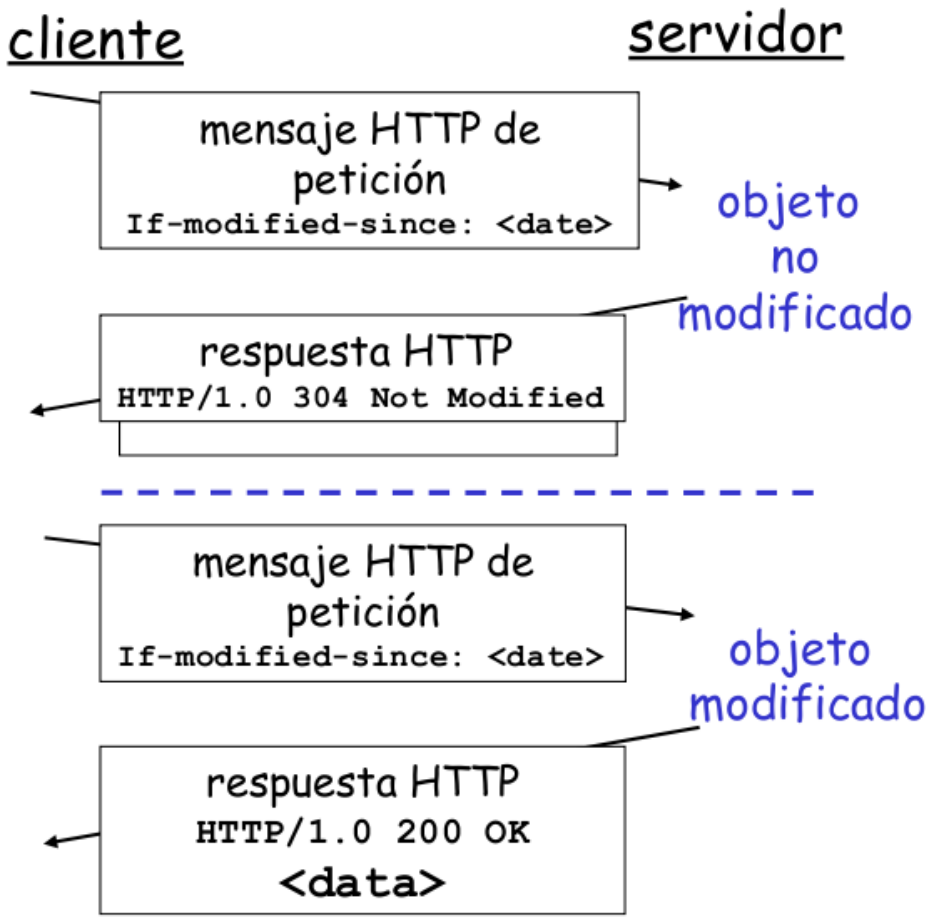
Envía una petición de la página `index.html`. Es necesario dejar una línea en blanco para terminar la cabera HTTP.

3. Se muestra la respuesta recibida del servidor.

Contenidos

Caché en un cliente de HTTP

- Objetivo: no enviar objetos si el cliente tiene una versión actualizada en su caché.
- Cliente: especifica la fecha de la copia en caché en la petición HTTP:
  `If-modified-since: <date>`
- Servidor: su respuesta no contiene ningún objeto si no ha sido modificado desde la fecha especificada en la petición:
  `HTTP/1.0 304 Not Modified`



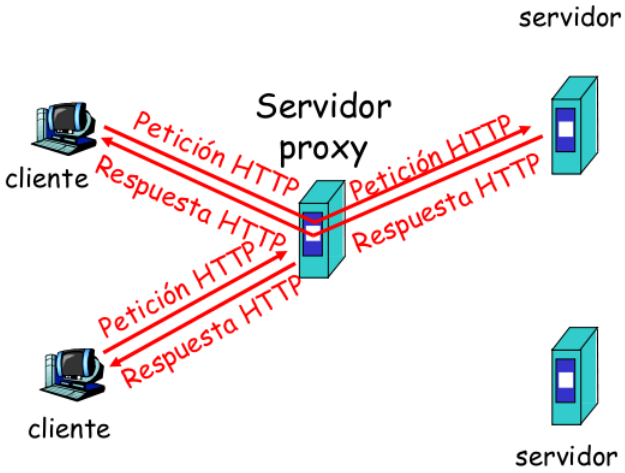
Otras condiciones

- Los servidores deben responder siempre con la cabecera `Date` indicando la fecha y hora actual (en GMT).
- Los servidores han de entender `If-Modified-Since` y `If-Unmodified-Since`.
- Los clientes pueden usar o no usar `If-Modified-Since` y `If-Unmodified-Since`.
- Respuesta a `If-Modified-Since`, si no se ha modificado el objeto desde esa fecha: `Not Modified`.
- Respuesta a `If-Unmodified-Since`, si se ha modificado el objeto desde esa fecha: `Precondition Failed`.

Contenidos

Representantes *(proxies)* de HTTP

- Un *proxy* HTTP es un intermediario entre un cliente y un servidor.
- Funcionamiento para un cliente que tenga configurado un *proxy*:
  1. El cliente envía su petición al *proxy*
  2. El *proxy* hace la petición al servidor
  3. El servidor envía la respuesta al *proxy*
  4. El *proxy* envía la respuesta al cliente
- El *proxy* actúa, por tanto, como cliente y como servidor.



- Normalmente un *proxy* lo es de varios clientes y tiene una caché asociada
- Pueden encadenarse varios proxies.
- Usos: cortafuegos, aumento de velocidad por uso de la caché
- Las peticiones a un proxy incluyen la URL completa en la primera línea del mensaje de petición
- Ejemplo de petición a un proxy:
  `GET http://gsyc.escet.urjc.es/index.html HTTP/1.0`

Contenidos

Virtual Hosts

- Virtual Hosts: Una misma máquina, y un mismo servidor HTTP, responde peticiones dirigidas a nombres de máquina diferentes.
  - Ejemplo: un mismo servidor responde a peticiones dirigidas a `www.zapatos.com` y a `www.camisas.com`.
- Además del soporte de HTTP, se necesita una de estas cosas:
  - que la máquina tenga varias direcciones IPs, que el DNS asociará a los distintos nombres
  - que el DNS asocie la misma IP a los diferentes nombres
- Al introducir el soporte para *Virtual Hosts* en HTTP 1.1 se hizo obligatoria en las peticiones el uso de la cabecera `Host`:
  - La línea inicial sólo lleva el *path*, sin en nombre de máquina.
  - Gracias al nombre que aparece en la cabecera `Host`, el servidor puede servir el árbol de páginas adecuado según el nombre de máquina que usa el cliente.
  - Si un servidor recibe una petición HTTP 1.1 sin cabecera `Host` debe devolver un mensaje de error `400 Bad Request`.
- Los servidores también han de aceptar líneas iniciales de petición con URLs completas, incluyendo el nombre de máquina (en lugar de sólo el *path*): será obligatorio en versiones futuras.
- Ejemplo de petición "mínima" en HTTP 1.1:

```
GET /dir/index.html HTTP/1.1
Host: gsyc.escet.urjc.es

```

Contenidos

Persistencia de estado en HTTP

- HTTP se diseña de forma que los servidores no almacenen estado sobre los clientes (pues un servidor tendrá muchos clientes).
- Sin embargo, es muy frecuente la necesidad 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:
  - El estado es mantenido por el servidor de forma externa a HTTP (basándose en la IP del cliente, o en otros datos)
  - 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).

Cookies

- Las *cookies* son datos (identificadores numéricos, cadenas de caracteres. . .).
- Funcionamiento:
  1. el servidor genera una *cookie* para representar el estado asociado a un cliente que ha hecho una petición
  2. el servidor envía la *cookie* al cliente
  3. el cliente almacena la *cookie* como asociada a ese servidor (y, en su caso, a un *path* de URL determinado)
  4. el cliente reenvía la *cookie* al servidor en las futuras peticiones que le realice
- Especificación original de Netscape, luego propuesta como RFC 2109, ampliada en RFC 2965.

Cabecera Set-Cookie

- Cabecera puesta por un servidor cuando quiere enviar una *cookie*
- El formato incluye:
  - Nombre de la cabecera: `Set-Cookie`
  - Nombre de la *cookie* y valor: `<nombre>=<valor>`
  - Fecha de caducidad: `expires=<fecha>`
  - Dominio y trayecto para el que es válida (se enviará para todos los trayectos de ese dominio que empiecen por el *path* especificado en la *cookie*):
    `domain=<dominio>; path=<trayecto>`
  - Si debe ser transmitida sólo sobre canales seguros (HTTPS): `secure`
- Ejemplo:

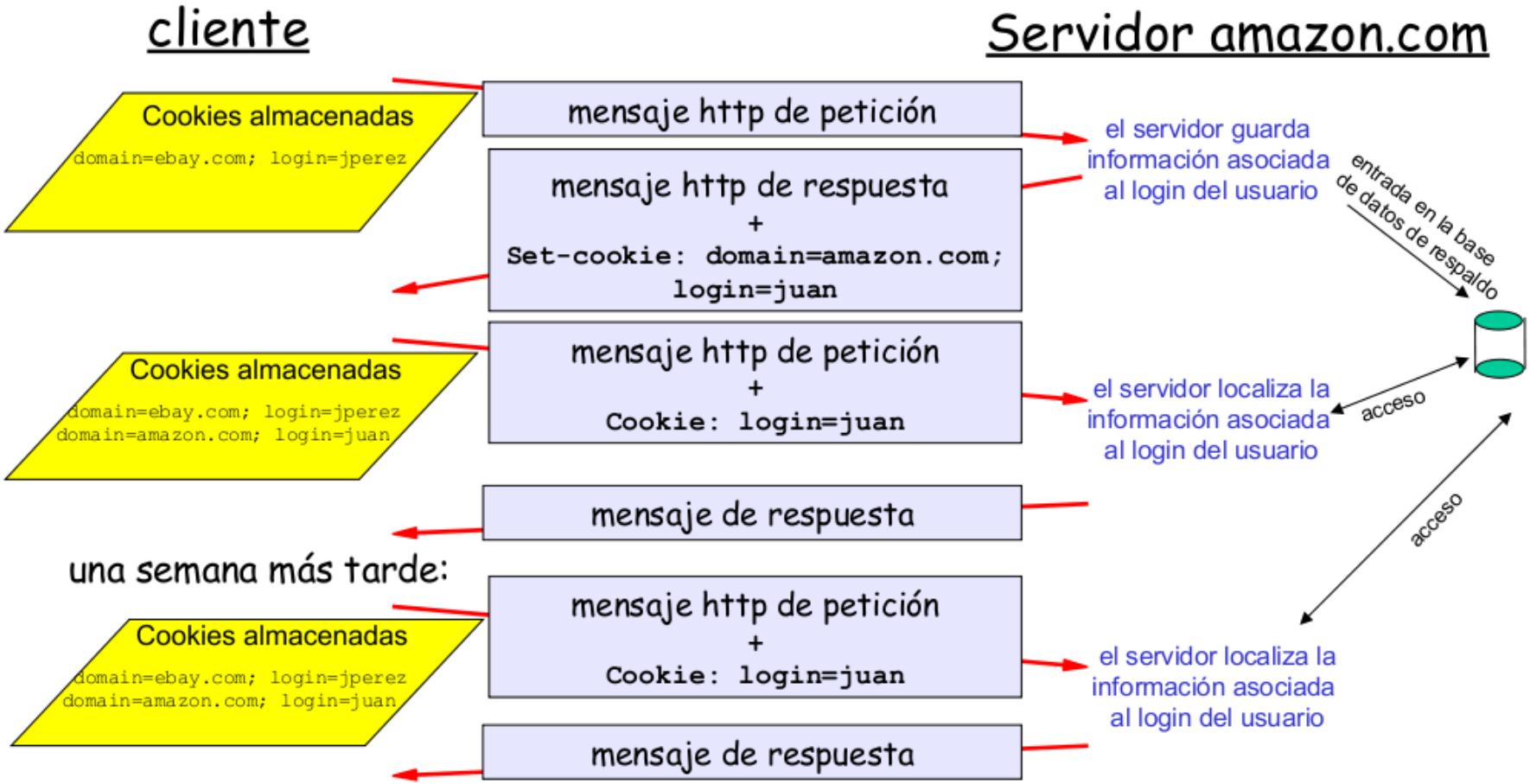
```
Set-Cookie: login=pepe; expires=Mon, 30-Jan-2009 12:35:23 GMT;
domain=www.myserver.com; path=/dir; secure
```

Cabecera Cookie

- Cuando un cliente pide una URL, buscará en su lista de *cookies* almacenadas si hay alguna que tenga que enviar (según el *domain* y el *path*).
- El cliente enviará todas las *cookies* en una única cabecera `Cookie`.
- Dentro de esta cabecera, las *cookies* se ordenarán de más a menos específicas (según su *path*).
- No se consideran las *cookies* con caducidad en el pasado (y se eliminan periódicamente).
- Ejemplo:

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

Ejemplo de funcionamiento



Contenidos

HTTPS

- HTTP sobre SSL *(Secure Socket Layer)*.
- La conexión TCP está cifrada, de forma que una tercera parte no puede conocer su contenido.
- Permite enviar datos "sensibles" a un servidor web, y recibirlos de él.
- Necesita de otros mecanismos (certificados, etc.) para ofrecer un nivel de seguridad razonable.
- Las URLs comienzan por `https://`

Contenidos

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**, Addisson 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 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`