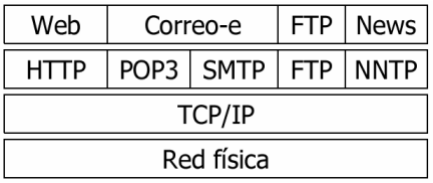
Tema 2: Protocolo HTTP.

1. **Introducción.**
2. **Mensajes HTTP.**
   1. Partes del mensaje.
   2. Primera línea del mensaje
   3. Cabeceras del mensaje.
   4. Cuerpo del mensaje.
3. **Elementos Avanzados.**
   1. Cookies
   2. Manejo de sesiones.
   3. Autentificación y Autorización del cliente.
   4. Seguridad
   5. Conexiones persistentes
   6. Caché.

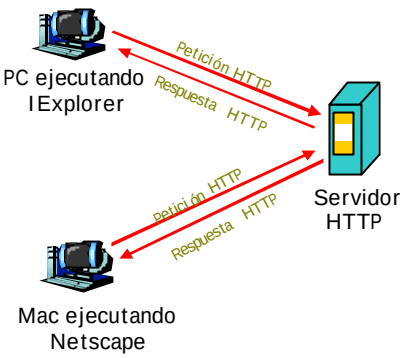
Descripción general (I)

- Los elementos software de la arquitectura web (clientes, servidores, proxies) utilizan el protocolo HTTP para comunicarse.
  - HTTP define la sintaxis y la semántica que utilizan estos elementos para comunicarse.
- Las últimas versiones HTTP/1.0 y HTTP/1.1
- Es un protocolo en la capa de aplicación. Por debajo está TCP/IP.



Descripción general (II)

- Protocolo de comunicaciones estándar que comunica servidores, proxies y clientes.
  - Permite la transferencia de documentos web, sin importar cual es el cliente o cual es el servidor.
- Es un protocolo basado en el esquema petición/respuesta.
  - El cliente envía un mensaje de petición y el servidor contesta con un mensaje de respuesta, cuyo contenido es función de la petición hecha por el cliente.



Descripción general (III)

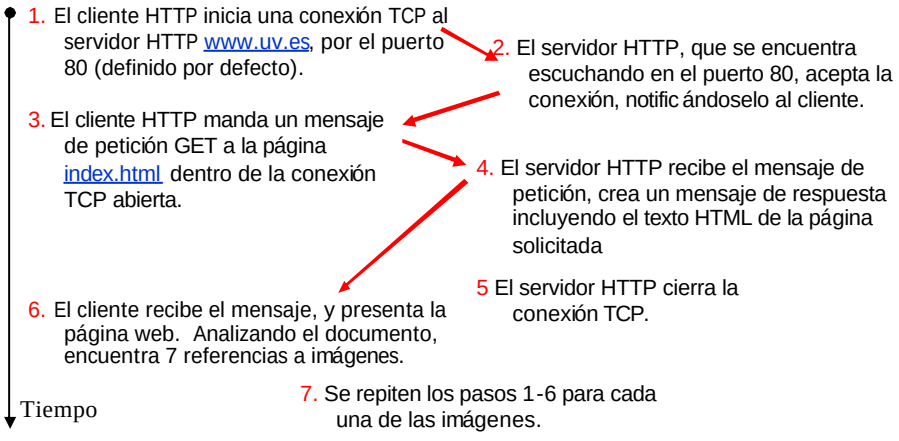
- El protocolo HTTP está basado en mensajes.
  - Texto plano.
    - Ventajas:
      - Legible.
      - Fácil de depurar.
    - Desventajas:
      - El mensaje es más largo.
- Es un protocolo sin manejo de estados.
  - Hay ausencia de estado tras cada par petición-respuesta
  - Tras la respuesta, el servidor cierra inmediatamente la conexión.
  - No existe el concepto de sesión.

Escenario típico (I)

- El usuario escribe en la barra de dirección del navegador el recurso al que desea acceder:
  - **`http://www.uv.es/~uvalen/cat/index.html`**
- El navegador descompone la URL en 3 partes:
  - El protocolo ("**`http`**")
  - El nombre del servidor ("**`www.uv.es`**")
  - El camino ("**`/~uvalen/cat/index.html`**")
- El navegador se comunica con servidor de nombres para traducir el nombre del servidor "**`www.uv.es`**" en una **Dirección IP**, que es utilizada para conectarse a la máquina servidora.

Escenario típico (II)

La página http://www.uv.es/~uvalen/cat/index.html contiene referencias a 7 imágenes.



2.1 Partes del mensaje

- Protocolo basado en mensajes texto, compuestos de una línea inicial, de una cabecera y de un cuerpo.
  - El mensaje es la unidad fundamental de la comunicación HTTP.
  - Se incluyen dentro de los paquetes TCP/IP
- Línea inicial del mensaje:
  - Primera línea del mensaje donde se indica que hacer (mensaje de petición) o que ha ocurrido (mensaje de respuesta).
- Cabecera del mensaje:
  - Bloque de campos terminados por una línea en blanco
  - Contienen los atributos del mensaje.
- Cuerpo del mensaje:
  - Es opcional. Su presencia depende de la petición y del resultado.
  - El contenido está determinado por el tipo de recurso.

Mensajes de petición y respuesta

- El cliente envía una petición al servidor en forma de mensaje texto, incluyendo:
  - Una línea inicial con el ***método*** de solicitud, la ***URL*** del recurso solicitado y la ***versión del protocolo***.
  - Una lista de campos, consistente en modificadores de la petición, información del cliente, etc.
  - Un posible cuerpo de contenido.
- El servidor responde con un mensaje donde se incluye:
  - Una línea de ***estado***, con la ***versión del protocolo*** y un ***código*** de éxito o error.
  - Una lista de campos, donde se incluyen entre otras cosas: el ***tipo MIME*** de la respuesta, información del servidor, entidades de meta-información, etc.
  - Un cuerpo con el contenido del recurso solicitado (opcional).

Ejemplo

- Ejemplo de petición:

```
GET /saludo.html HTTP/1.1
Host www.uv.es
```

- Ejemplo de respuesta:

```
HTTP/1.1 200 OK
Date: Sun, 01 May 2003 12:00:01 GMT
Content-Type: text/html
Content-Length: 45

<HTML>
  <BODY> Hola Mundo! </BODY>
</HTML>
```


2.2 Primera línea del mensaje

- Línea inicial en las peticiones:
  - Especifica el recurso que se solicita, y qué se precisa de él:
    - Nombre de método (GET, POST, HEAD, etc.).
    - Recurso (URL completa o el camino de la URL)
  - Versión del protocolo HTTP (HTTP/x.x).
  - Ejemplo:
    - **`GET /directorio/otro/fichero.html HTTP/1.0`**
- Línea inicial de la respuesta:
  - Versión de HTTP (HTTP/x.x).
  - Código de estado:
    - Código numérico de estado.
    - Comentario descriptivo de estado.
  - Ejemplo:
    - **`HTTP/1.1 401 Anauthorized`**

Métodos de envío de los datos

- **GET:** Solicita un documento al servidor.
  - Se pueden enviar datos en la URL
- **HEAD:** Similar a GET, pero sólo pide las cabeceras HTTP.
  - Comprobar enlaces
  - Para consultar información sobre el fichero (fecha de modificación, tamaño, tipo de servidor, tipo de documento solicitado) antes de solicitarlo.
- **POST:** Manda datos al servidor para su procesado.
  - Similar a GET, pero además envía datos en el cuerpo del mensaje.
  - La URL corresponde a un página dinámica que trata los datos enviados.
- **PUT:** Almacena el documento enviado en el cuerpo del mensaje.
- **DELETE:** Elimina el documento referenciado en la URL.
- **TRACE:** Rastrea los intermediarios por los que pasa la petición.
- **OPTIONS:** Averigua los métodos que soporta el servidor.
- En una caché sólo se guardan las respuestas de las peticiones realizadas con GET y HEAD (POST no).

Método GET

- Sintaxis:
  - **`GET <URL> <VERSION>`**
- Solicita el recurso nombrado en la URL
  - Recurso estático o dinámico (con o sin parámetros)
- Variantes (para reducir el trafico en la red):
  - GET condicional
    - Baja el recurso sólo bajo ciertas condiciones
    - Añadiendo las cabeceras:
      - **`If-Modified-Since, If-Match, If-Range, etc.`**
  - GET parcial
    - Baja sólo ciertas partes del recurso
    - Añadiendo la cabecera:
      - **`Range: bytes=...`**

Ejemplos

```
GET http://www.uv.es/index.html HTTP/1.1
Host: www.uv.es
If-Modified-Since: Fri, 1 Feb 2004 13:53:40 GMT
```

```
HTTP/1.0 304 Not Modified
Date: Thu, 1 Mar 2004 13:55:13 GMT
Content-Type: text/html
Expires: Fri, 30 Apr 2004 13:55:13 GMT
```

```
GET /Default.htm HTTP/1.1
Host: www.microsoft.com
Range: bytes=0-80
```

```
HTTP/1.1 206 Partial content
Server: Microsoft-IIS/5.0
Content-Type: text/html
Content-Length: 81
Content-Range: bytes 0-80/19618
<HTML> ....
```


POST

- Sintaxis:

  POST <URL> <VERSION>
  <CABECERA>
  <CRLF>
  <CUERPO DEL MENSAJE>

- Proporciona datos al recurso nombrado en la URL.
  - Los datos son enviados en el cuerpo del mensaje.
- Códigos de respuesta:
  - **200 OK**
  - **204 No Content**
  - **201 Created (cabecera location)**

Ejemplo POST

```
POST /test.cgi HTTP/1.0
Host: www.teco.edu:8080
User-Agent: Mozilla/4.7 (compatible; MSIE 5.0; Windows 5.0)
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 39

name=Marie&path=%2F&ort=Karlsruhe&submit=Submit+Request
```

```
HTTP/1.0 200 OK
Date: Wed, 27 Oct 1999 14:13:43 GMT
Server: Apache/1.2.1
Content-Type: text/html
Content-Length: 380

<html><head>
<title>CGI-Script</title> ...
```


Códigos de estado

- **1xx**: Mensaje informativo.
- **2xx:** Exito
  - 200 OK
  - 201 Created
  - 202 Accepted
  - 204 No Content
- **3xx:** Redirección
  - 300 Multiple Choice
  - 301 Moved Permanently
  - 302 Found
  - 304 Not Modified
- **4xx:** Error del cliente
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
- **5xx:** Error del servidor
  - 500 Internal Server Error
  - 501 Not Implemented
  - 502 Bad Gateway
  - 503 Service Unavailable

2.3 Cabeceras del mensaje

- HTTP/1.0: 16 cabeceras, ninguna obligatoria.
- HTTP/1.1: 46 cabeceras, "**Host**:" obligatoria en las peticiones (usada por los "servidores virtuales" y proxies).
- Formato de las cabeceras
  - **`Nombre`**`: sp` **`VALOR`** `CRLF`
  - Mismo formato que las cabeceras de correo y News (MIME).
- Clasificación:
  - Genéricas: cliente y servidor
  - Exclusivas de la petición (información del cliente)
  - Exclusivas de la respuesta (información del Servidor)
  - Entidad del cuerpo del mensaje
- Se recomienda incluir en las peticiones al menos **`User-Agent`** **`y`** en las respuestas **`Server`** y **`Last-Modified`**.

Cabeceras Genéricas

- Cabeceras generales para la solicitud y la respuesta. No tienen relación directa con la entidad.
- HTTP/1.x
  - **`Date`**: fecha de creación del mensaje
    - **`Date: Tue, May 16 12:39:32 2001 GMT`**
  - **`Pragma: no-cache`**.
    - No envíar la copia guardada en la caché.
- HTTP/1.1
  - **`Cache-Control`**: Controla el comportamiento de la caché
  - **`Connection`**:
    - **`close`**
    - **`Keep-Alive`** (HTTP/1.0)
  - **`Via`**: Información sobre los intermediarios.

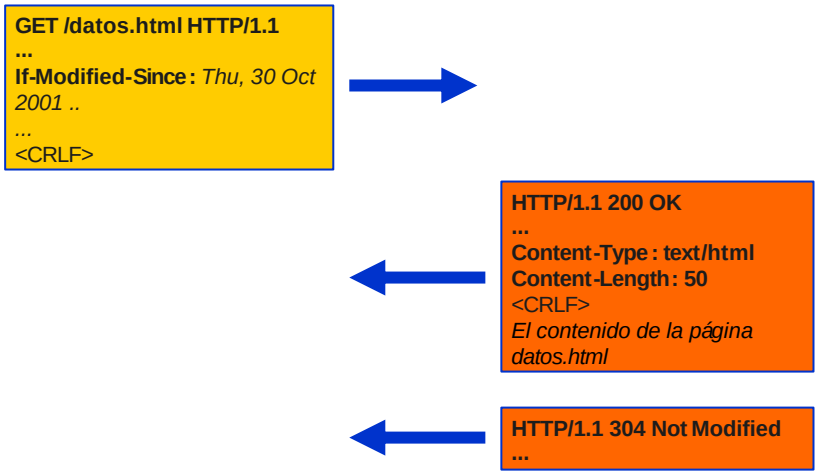
Cabeceras de la petición (I)

- Exclusivas de la petición.
- Preferencias en la respuesta (HTTP/1.1):
  - **`Accept`**: Tipos MIME aceptados por el navegador.
  - **`Accept-Charset`**: Preferencias en el conjunto de caracteres.
  - **`Accept-Encoding`**: Preferencias en la codificación del contenido.
  - **`Accept-Language`**: Preferencia en el Lenguaje del documento.
- Peticiones condicionales (HTTP/1.1):
  - **`If-Modified-Since`**: fecha (tb. HTTP/1.0)
  - **`If-Unmodified-Since`**: fecha
  - **`If-Match`**: etiqueta.
    - La petición será efectiva si coinciden las etiquetas
  - **`If-None-Match`**: etiqueta.

Cabeceras de la petición (II)

- Restricciones en el servidor (HTTP/1.1):
  - **`Max-Forward`**: límite de cabeceras añadidas en TRACE
  - **`Range`**: rango (de bytes de la entidad).
    - Se emplean para GET parciales.
- Otra información enviada con la petición:
  - **`Host`**: Nombre y puerto del servidor al que se dirige la petición
  - **`From`**: e-mail del solicitante.
  - **`User-Agent`**: Identificación del programa del cliente
    - **`Mozilla/4.7 (compatible; MSIE 6.0; Windows 5.0)`**
  - **`Referer`**: URL origen de la petición
  - **`Authorization`**:Tipo sp Credenciales
    - **`Authorization: Basic B64(login:password)`**
  - **`Cookies`**: nombres y valores de las cookies.

Ejemplos de petición condicional



Cabeceras de la respuesta

- Redirecciona:
  - **`Location`**: localización real del recurso
- Seguridad:
  - **`WWW-Authenticate`** (se solicita autentificación)
    - **`WWW-Authenticate: Basic realm="ambito"`**
- Caché (HTTP/1.1):
  - **`Etag`**: etiqueta.
  - **`Age`**: tiempo (desde que fue generada la respuesta).
- Otra información relacionada con la respuesta:
  - **`Server`**: versión del software del servidor
    - **`Server: Apache/1.3.12 (Win32)`**
  - **`Retry-After`**: tiempo. (HTTP/1.1)
    - Tiempo de espera antes de solicitar el recurso de nuevo.
  - **`Accept-Ranges`** (HTTP/1.1)
  - **`Set-Cookies`**

Cabeceras Entidad

- Mensajes de solicitud y respuesta
- HTTP/1.0
  - **`Allow`**: métodos permitidos para el recurso
  - **`Content-Encoding`**: codificación de la entidad (p.e. compresión)
    - **`Content-Encoding: gzip`**
  - **`Content-Length`**: longitud de la entidad (importante en solicitud)
  - **`Content-Type`**: tipo MIME de la entidad
    - **`Content-Type: text/html; charset=iso-latin-1`**
  - **`Expires`**: fecha tope de validez en la caché
  - **`Last-Modified`**: fecha de la última modificación de la entidad
- HTTP/1.1
  - **`Content-Language`**: Lenguaje natural de la entidad.
  - **`Content-Location`**: localización URL alternativa.

2.4 Cuerpo del mensaje

- En las respuestas contiene el recurso pedido o texto explicando un error.
- En las peticiones contiene datos de usuario o ficheros para subir.
- Si hay cuerpo, deben aparecer al menos las siguientes cabeceras relativas a él:
  - **Content-Type**: tipo MIME de los datos (ej: text/html, image/png).
  - **Content-Length**: número de bytes en el cuerpo.

Internacionalización del contenido

- Cada día aparecen en la web millones de documentos escritos en cientos de lenguajes.
- HTTP da soporte para el transporte y procesado de documentos en lenguajes y alfabetos distintos.
  - Los clientes mandan las cabeceras **`Accept-Charset`** y **`Accept-Language`** para indicarle al servidor los sistemas de codificación y lenguajes que el cliente entiende, así como cuales prefiere:
    - **`Accept-Charset`**: **`iso-8859-1, utf-8`**
    - **`Accept-Language`**: **`es, en`**
  - El servidor le indica al cliente el alfabeto y lenguaje utilizado con el parámetro **`charset`** de **`Content-Type`** y con la cabecera **`Content-Language`**.
    - **`Content-Type: text/html; charset=iso-2022-jp`**
    - **`Content-Language: jp`**

Ejemplo de petición

```
GET /~pdi/test.html HTTP/1.1 <CRLF>
Connection: close <CRLF>
User-Agent: Mozilla/5.0 [en] (X11; I; Linux 2.2.15 i586;
   Nav ...) <CRLF>
Host: www.uv.es <CRLF>
Accept: image/gif, image/x-xbitmap, image/jpeg,
   image/pjpeg, */* <CRLF>
Accept-Encoding: gzip <CRLF>
Accept-Language: es <CRLF>
Accept-Charset: iso-8859-1 <CRLF>
<CRLF>
```

Ejemplo de respuesta

```
HTTP/1.1 200 OK <CRLF>
Date: Tue, 23 Jan 2001 12:44:27 GMT <CRLF>
Server: Apache/1.3.9 (Unix) Debian/GNU <CRLF>
Last-Modified: Tue, 23 Jan 2001 12:39:45 GMT <CRLF>
ETag: "19e89f-22-3a6d7b91" <CRLF>
Content-Length: 34 <CRLF>
Content-Type: text/html; charset=iso-8859-1 <CRLF>
<CRLF>
<html>Esto es una prueba</html>
```


3.1 Cookies

- Las cookies son información que el navegador guarda en memoria o en el disco duro dentro de ficheros texto, a solicitud del servidor.
  - Incluyen datos generados por el servidor, o datos introducidos en un formulario por el usuario, enviados al servidor y reenviados por éste al cliente.
- HTTP es un protocolo sin estados (no almacena el estado de la sesión entre peticiones sucesivas)
  - Las cookies pueden usarse para asociar estado.
    - Proporcionan una manera de conservar cierta información entre peticiones del cliente.

Usos

- Almacenar información importante para el servidor.
  - Constituyen una potente herramienta empleada por los servidores Web para almacenar y recuperar información acerca de sus visitantes.
- Ejemplos de uso:
  - Guarda información de la sesión.
  - Comercio electrónico.
    - Carrito de la compra.
  - Personalización de páginas
    - Idiomas
  - Seguimiento de las visitas a un Web
    - Carteles publicitarios
  - Almacenamiento del login y password

Almacenamiento de la cookies

- El hecho de ser almacenadas en el lado del cliente, libera al servidor de una importante carga
  - El cliente devuelve la información al servidor en siguientes peticiones.
- Tipos: cookies de sesión y cookies persistentes.
- Las cookies persistentes son almacenadas en disco por el propio navegador.
  - Internet explorer:
    - Un archivo para cada cookie:
      - <identificador de usuario>@<dominio.txt>
  - Netscape:
    - Todas en el mismo archivo: cookie.txt

Cookie enviada por el servidor (I)

- Cabecera HTTP: “Set-Cookie”
  - Cabecera incluida por el servidor en el mensaje de respuesta, cuando quiere enviar una cookie.
  - Formato:
    - “**`Set-Cookie`**:”
    - “**`nombre=valor`**”: Nombre de la cookie y valor
    - “**`;expires=fecha`**”: Fecha de caducidad
    - “**`;path=camino`**”: Camino
    - “**`;domain=dominio`**”: Dominio
    - “**`;secure`**”: sólo se transmite sobre canales seguros (HTTPS).
  - Ejemplo:

```
Set-Cookie: unnombre=unvalor; expires=Mon, 30-Jan-2001
  12:35:23 GMT; path=/dir; domain=mi.dominio.com;
  secure
```


Cookie enviada por el servidor (II)

- Información incluida:
  - Nombre y Valor de la cookie.
  - Fecha de caducidad.
    - El navegador conservará y recuperará la cookie sólo si su fecha de caducidad aún no ha expirado.
    - Si no se especifica, caduca cuando el usuario salga de la sesión.
  - Camino de las aplicaciones con acceso a la cookie.
    - Si no se especifica, toma como camino el directorio de la aplicación que la originó.
  - Dominio (completo o parcial) de los servidores con acceso a la cookie.
    - Si no se especifica ningún dominio, el navegador sólo devolverá la cookie a la máquina que la originó.
  - Atributo **`secure`** indicando que la cookie sólo será transmitida a través de un canal seguro, con SSL.

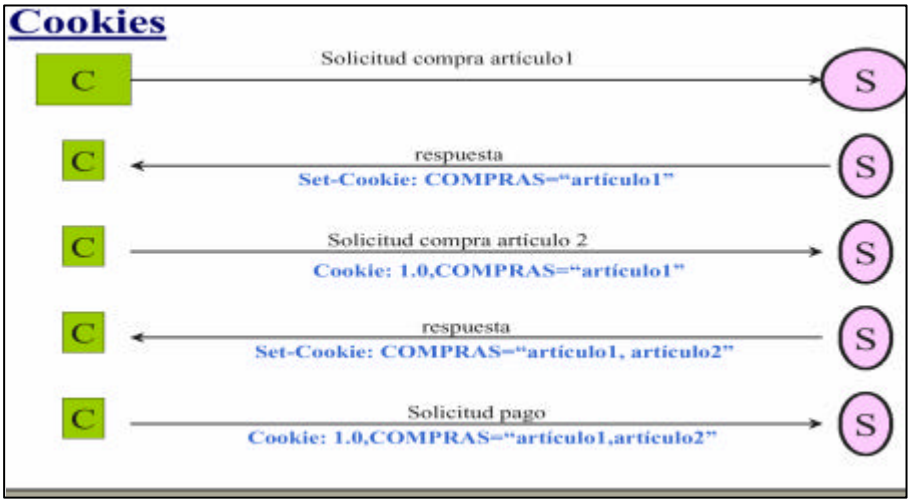
Cookie enviada por el cliente

- Cabecera HTTP: “Cookie”.
  - Enviará todas las cookies en una única cabecera HTTP:
    - `Cookie: nombre1=valor1; nombre2=valor2; ...`
- Proceso:
  - Cuando un cliente solicita una URL, buscará en su lista de cookies aquellas que coincidan con ese dominio y con ese camino.
  - Dentro de la cabecera “Cookie”, las cookies se ordenan de más a menos específicas (según camino).
  - No se consideran las cookies caducadas (de hecho, se eliminan).

Limitaciones

- Por su diseño, las cookies tienen las siguientes limitaciones:
  - Máximo de trescientas cookies en el disco.
    - Si llega la número 301, se borra la más antigua.
  - Tamaño máximo de 4 Kbytes por cookie (nombre y valor).
  - Veinte cookies máximo por servidor o dominio.
  - Ninguna máquina que no encaje en el dominio de la cookie podrá acceder a ella.

Ejemplo



Seguridad

- Son simples ficheros texto almacenados por el navegador.
  - Son elementos pasivos que no pueden emprender ninguna acción.
  - No pueden infectar el ordenador con ningún tipo de virus.
- No pueden ser usados para extraer datos de nuestro disco duro.
  - Solo almacenan la información confidencial que previamente haya sido enviada al servidor.
- Sin embargo:
  - No son un buen elemento de seguridad, ya que cualquier usuario que tenga acceso a ellas (tal vez a través de la red local, nunca a través de Internet) puede extraer sus datos.
  - Pueden ser utilizadas por los servidores para hacer un seguimiento oculto de las páginas visitadas por un usuario y crearse un perfil del usuario.

3.2 Sesiones

- HTTP es un protocolo sin manejo de estados.
  - Tras la respuesta, el servidor cierra inmediatamente la conexión (HTTP/1.0).
  - Los servidores HTTP responden a cada solicitud del cliente sin relacionar tal solicitud con alguna solicitud previa o siguiente.
    - El protocolo HTTP no maneja un estado de cada conexión realizada por el mismo usuario, sea cual sea la versión HTTP.
  - No existe el concepto de sesión.
- Las sesiones son fundamentales en las aplicaciones Web. Permiten:
  - Definir varios estados distintos en la aplicación.
  - Colocar las solicitudes y respuestas dentro de un contexto más amplio.
    - Los clientes y servidores intercambian información sobre el estado de la aplicación.

Datos asociados a la sesión.

- El servidor almacenará la información necesaria para llevar el seguimiento de la sesión.
  - Identificador de la sesión.
  - Identificador del usuario en sesión.
  - Tiempo de expiración de la sesión.
  - Dirección donde se encuentra localizado el cliente.
  - Variables asociadas a la sesión.
  - Otras variables temporales.
- Por la misma naturaleza del HTTP es imposible asegurar la existencia o la ausencia de una sesión.
  - Establecer un proceso que revise periódicamente los tiempos de expiración de cada proceso.
  - Eliminar los datos asociados a la sesión si ya excedió el tiempo.

Mecanismos.

- Se deben establecer mecanismos ajenos al protocolo HTTP para llevar el control de la sesión.
  - A través de **Cookies**
  - Reescribiendo (inflando) las direcciones URL.
  - A partir de controles HTML ocultos.
    - <INPUT type="hidden" name="session" value="1234">
  - A partir de la dirección IP del cliente.
    - El servidor mantiene la información de la sesión en:
      - Memoria RAM.
      - Archivos.
      - Una base de datos.
- Los más utilizados son los tres primeros.
- Pocas aplicaciones Web hacen uso exclusivo de un tipo
  - Generalmente se mezclan, obteniendo las ventajas de cada uno y tratando de evitar sus desventajas.

Ventajas/desventajas.

- Cada uno de los anteriores mecanismos tienen sus ventajas y desventajas:
  - La **dirección IP** no distingue usuarios, sólo máquinas.
    - La dirección IP está oculta si hay un proxy por medio.
  - Las **cookies** pueden ser leídas por terceros.
    - Se debe utilizar exclusivamente cookies de sesión.
    - Algunos usuarios no aceptan cookies de ningún tipo.
  - Los **controles HTML ocultos** y las **URL infladas**, se vuelven más complicados de mantener cuando la información persistente crece en tamaño.
  - Las **URL infladas** sólo funcionan si el acceso a las páginas se realiza activando enlaces (es decir, si no se introduce la URL directamente)

Identificadores de la sesión.

- Para que la aplicación Web identifique cada petición HTTP dentro de una sesión, las peticiones deben contener un identificador pasado a través de:
  - Parámetros en la URL (método GET)
  - Parámetros incluidos en el cuerpo del mensaje (método POST)
  - **Cookie**
- Esto, entre otras cosas, evita que el usuario deba autentificarse en cada petición.
- Los identificadores de la sesión deben ser únicos y difíciles de adivinar.
  - Existe la posibilidad de que agentes externos quieran entrar de manera fraudulenta al sistema.
  - Es necesario algún mecanismo que provea identificadores aleatorios y con un gran periodo de repetición

Autentificación del usuario

- La manera predeterminada de trabajar de la Web es anónima.
  - Lo único que se puede saber con seguridad es el número IP del cliente (si no hay un proxy por medio).
- A veces, debido a cuestiones de personalización o a políticas de restricción, las aplicaciones Web deben conocer y verificar la identidad del usuario:
  - A través de un nombre de usuario y una palabra secreta.
  - A este proceso de identificación se le conoce como **autentificación**.

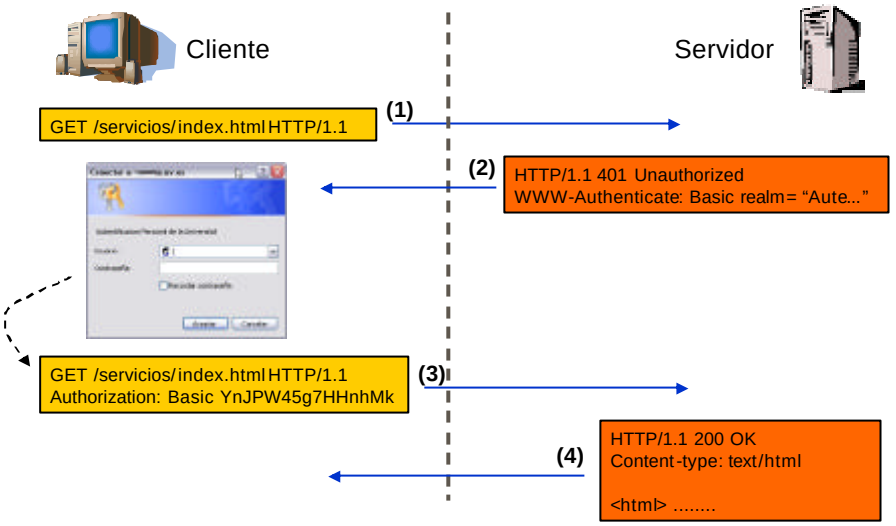
Autorización del usuario

- Una vez identificado, se comprueba si el usuario y palabra clave enviados son válidos, así como sus restricciones de uso.
  - La lista de usuarios y sus restricciones de uso se encuentra normalmente en una base de datos.
  - A este proceso se le conoce como **autorización**.
- Si la respuesta coincide, el servidor transfiere el recurso.

Autentificación con HTTP

- Existen múltiples métodos para la autenticación de usuarios.
- El protocolo HTTP provee un mecanismo para la autenticación de un usuario.
  - Cabeceras: **`WWW-Authenticate`** y **`Authoritation`**
  - Los navegadores se encargan de manejar la petición al usuario por parte del servidor para que se identifique, presentándole un cuadro de diálogo:
- Autentificación con HTTP/1.1
  - La autentificación **Basic** (Ya existía en la versión HTTP/1.0).
  - La autentificación **Digest** (Mejora la Basic).

Procedimiento



Autentificación Basic

- Codificación simple de 6 bits.
  - Une en una cadena el login y password separado por ":"
  - Divide la secuencia de bits de la cadena en grupos de 6 bits.
  - A cada trozo le asigna una letra (extraída de una alfabeto especial de 64 caracteres).
- El servidor devuelve el mensaje:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Autentificación perso.. "
```

- El cliente reenvía el mensaje añadiendo:

```
Authorization: Basic JYnWp4tdG90dHk6T3ch
```

- Es muy simple. Se puede decodificar en pocos segundos a mano.

Autentificación Digest

- Alternativa mucho más segura que Basic.
- Utiliza algoritmos de 128 bits (MD5) para hallar el compendio del password.
- El servidor responde con el mensaje:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest realm=".."
  nonce="HK6TP4C.."
```

- El cliente reenvía el mensaje añadiendo:

```
Authorization: Digest username="pepe" realm=".."
  nonce="HK6TP.." uri="/cgi-bin/servicios.cgi"
  response="66C4EH87SK3JHH33833HDLSDJKD38838JJ5G3"
```


Alternativas a la autentificación HTTP

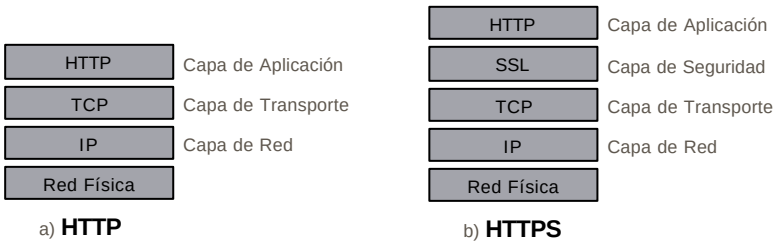
- La autentificación con HTTP presenta varias desventajas:
  - No termina la sesión hasta que es cerrado el navegador.
  - No se puede modificar la presentación de la ventana de diálogo, donde se le solicita al usuario que se identifique.
    - Este formulario es manejado por el navegador autónomamente.
- La otra alternativa es que la aplicación Web se haga cargo de la autenticación, integrándose a la autorización del usuario y al mecanismo de sesiones.
  - La presentación se hace a través de formularios HTML.
  - Otorga más flexibilidad para modificar el método de autentificación cuando se necesite.
  - La comunicación se cifra utilizando HTTPS.

El protocolo HTTPS

- HTTPS: protocolo que utiliza SSL (o TSL) para transportar mensajes HTTP (puerto 443).
- SSL asegura que la conexión TCP está cifrada, de forma que una tercera parte no puede espiar su contenido.
- Esta ampliamente implementado
  - Tanto en los navegadores como en los servidores actuales.
- La URL comienza por “https://”.
- Aunque la conexión HTTP es sin estado, la información SSL se puede retener y reutilizar.
  - Cliente y servidor pueden transmitir nuevos mensajes de forma segura utilizando la misma conexión SSL.

El protocolo SSL (I)

- SSL: Secure Socket Layer.
  - Trabaja sobre TCP y debajo de los protocolos de alto de nivel tales como HTTP.



El protocolo SSL (II)

- Permite:
  - Al cliente SSL, **autentificar la identidad del servidor SSL**.
    - Utilizando técnicas de criptografía de llave pública, comprueba que el certificado del servidor es válido y ha sido avalado por una autoridad certificadora (CA).
  - Al servidor SSL, **autentificar la identidad del cliente SSL**.
    - Usando las mismas técnicas.
  - Establecer una **conexión encriptada** entre ambas máquinas.
    - Encriptación asimétrica RSA (clave pública).
    - Encriptación simétrica.
    - Proveen un alto grado de confidencialidad.
  - Asegurar la **integridad** de los mensajes.
    - Además, la información encriptada es protegida con un mecanismo para detectar si ésta fue alterada durante su tránsito por la red.

3.5 Conexiones Persistentes

- Permiten que varias peticiones y respuestas sean transferidas usando la misma conexión TCP.
- Se usan por omisión en HTTP 1.1.
- Si se envía la cabecera “Connection: close”, el servidor cierra la conexión después de la respuesta.
- Un servidor puede cerrar la conexión antes de enviar todas las respuestas.
- El servidor cerrará las conexiones inactivas pasado un plazo de tiempo (ej: 30 segundos?).

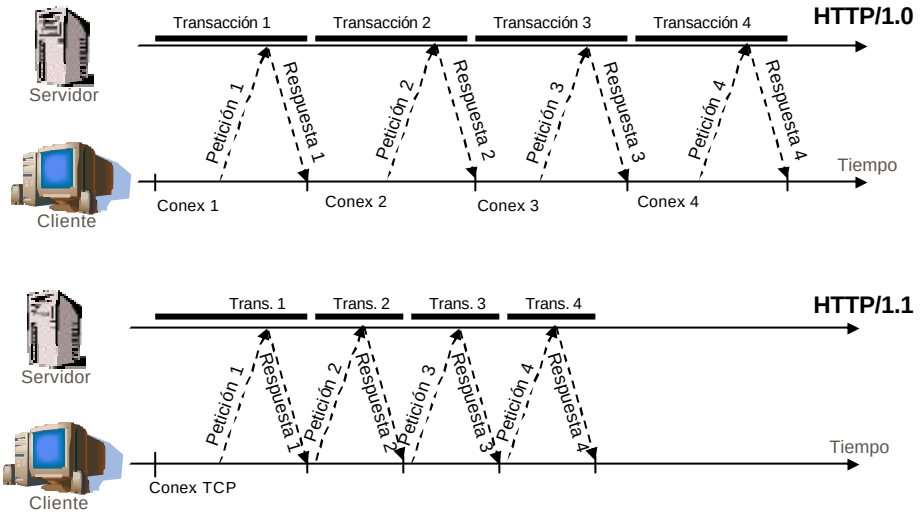
Conexiones con HTTP/1.0

- Crea y cierra una conexión TCP por cada petición/respuesta HTTP.
  - No es del todo cierto
    - **`Connection: Keep-Alive`**
- Desventajas:
  - Se incrementa la carga para múltiples peticiones al mismo servidor.
    - Establece una nueva conexión TCP para cada objeto
    - La conexión TCP tarda relativamente bastante tiempo en establecerse.
  - Las conexiones keep-Alive dan problemas
- Ventajas:
  - Es fácil de implementar.

Conexiones con HTTP/1.1

- HTTP/1.1 introduce las conexiones persistentes.
  - Se realizan múltiples peticiones/respuestas sobre la misma conexión TCP.
  - El cliente y el servidor mantienen por defecto abiertas las conexiones con caches y servidores.
  - Para que la conexión se cierre (cliente o servidor):
    - **`Connection: close`**
- Ventajas:
  - Reduce el número de conexiones y los consiguientes retrasos y gastos de memoria y CPU.
- Desventajas:
  - Es más complejo que el modelo HTTP/1.0

Comparativa



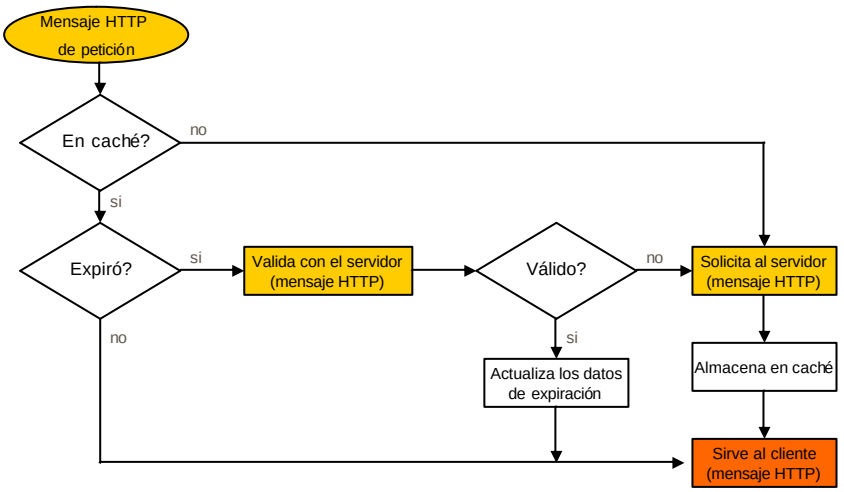
Problemas.

- Manejo de la conexiones:
  - Se suele limitar la conexión para un número finito de recursos.
  - Caches y servidores deben mantener sólo un tiempo limitado la conexión abierta, si está inactiva.
    - ¿Cuánto tiempo?
- Las respuestas son servidas en serie, ordenadas de acuerdo a las peticiones (FIFO)
  - Problema de bloqueo en la cabeza de la lista.
    - Recursos de gran tamaño provocan que los siguientes recursos tarden en servirse, aunque sean de pequeño tamaño.
  - El problema se agudiza si hay cachés intermedias.
    - El problema persiste aunque los recursos provengan originalmente de servidores distintos.

3.6 Cachés

- Almacenan respuestas (susceptibles de ser guardadas) con la intención de reducir el tiempo de respuesta y la carga de la red.
- Necesitan asegurar que los contenidos guardados en la caché son equivalentes a los almacenados en el servidor.
  - Dos modelos:
    - Expiración (consistencia débil)
      - Reduce las peticiones al servidor.
    - Validación (consistencia fuerte)
      - Reduce la cantidad de datos a transmitir.
- Es imprescindible (para la caché) que el cliente identifique al servidor original.
  - Campo de la cabecera HTTP: **`Host`**

Pasos (consistencia débil)



Cabeceras HTTP para controlar la caché

- Los mecanismos en HTTP 1.0 eran muy pobres.
  - **`Pragma, Expires, If-Modified-Since y Last-Modified`**
- HTTP 1.1 añade mecanismos para permitir a las caches ser más consistentes con los servidores.
  - **`Cache-Control, Age, Etag e If-...`**
- El cliente puede forzar a la caché a que actualice siempre el objeto:
  - **`Pragma: no-cache`** (HTTP/1.0)
  - **`Cache-Control: no-cache`** (HTTP/1.1)
- El servidor puede evitar que la caché guarde el objeto.
  - **`Cache-Control: no-store`** (HTTP/1.1)

Expiración del documento

- Determina si el tiempo que lleva almacenado el objeto en la caché ha sobrepasado el máximo permitido. Si el objeto guardado en la caché no es lo suficientemente reciente, la caché **valida** el recurso con el servidor.
- Cabeceras HTTP involucradas, enviadas por el servidor:
  - **`Expires`**: fecha (de expiración)
  - **`Age`**: segundos (que el objeto llevaba almacenado en el servidor)
  - **`Cache-Control: max-age = segundos`**
    - Vida máxima del objeto.
- La caché decide que el recurso ha expirado si:
  - La edad del recurso es mayor que la vida máxima (max-age).
    - Edad = Age + tiempo de respuesta + tiempo en la caché.
  - La fecha de expiración (Expires) ha sido sobrepasada.

Validación con el servidor

- La caché contrasta con el servidor si el contenido del objeto ha cambiado, antes de descargar dicho recurso.
- Utiliza peticiones condicionales:
  - Basadas en la etiqueta (**Etag**) del objeto guardado en caché
    - **`If-Match:`** etiqueta (Etag del objeto en la caché)
    - **`If-None-Match:`** etiqueta (idem)
  - Basadas en la fecha de la última modificación (**Last-Modified**)
    - **`If-Modified-Since:`** fecha (Last-Modified del objeto en caché)
    - **`If-UnModified-Since:`** fecha (idem)
- Un objeto guardado en la caché será actualizado si:
  - La fecha incluida en **`If-Modified-Since`** es posterior a la fecha de la última modificación del objeto en el servidor origen.
  - La etiqueta incluida en **`If-None-Match`** no coincide con la del servidor origen.
- El servidor puede forzar la validación de la caché:
  - **`Cache-Control: must-revalidate`**