

Tema 2: Aplicaciones en red

Bibliografía: [Kurose10], Capítulo 2 (excepto 2.3, 2.6 – 2.8)



DOCENCIA VIRTUAL

Finalidad:

Prestación del servicio Público de educación superior (art. 1 LOU)

Responsable:

Universitat Politècnica de València.

Derechos de acceso, rectificación, supresión, portabilidad, limitación u oposición al tratamiento conforme a políticas de privacidad:

<http://www.upv.es/contenidos/DPD/>

Propiedad intelectual:

Uso exclusivo en el entorno de aula virtual.

Queda prohibida la difusión, distribución o divulgación de la grabación de las clases y particularmente su compartición en redes sociales o servicios dedicados a compartir apuntes.

La infracción de esta prohibición puede generar responsabilidad disciplinaria, administrativa o civil



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



- Aprender aspectos de la implementación de los protocolos de aplicación
 - Tipos de servicio del nivel de transporte
 - Modelo cliente servidor
 - Modelo P2P
- Aprender las características básicas de los protocolos empleados por los principales servicios en Internet
 - HTTP
 - SMTP/POP3/IMAP
 - DNS



- 1. Principios de las aplicaciones en red**
- 2. La Web y HTTP**
- 3. El correo electrónico**
- 4. DNS: servicio de directorio de Internet**



1. Principios de las aplicaciones en red

- Arquitectura de las aplicaciones en red
 - Cliente-servidor
 - Entre pares (p2p)
 - Híbrido
- Comunicación entre procesos
- Identificación de los procesos
- Interfaz entre el proceso y la red de computadores: Sockets
- Servicios de transporte en Internet
 - TCP
 - UDP
- Protocolo del nivel de aplicación

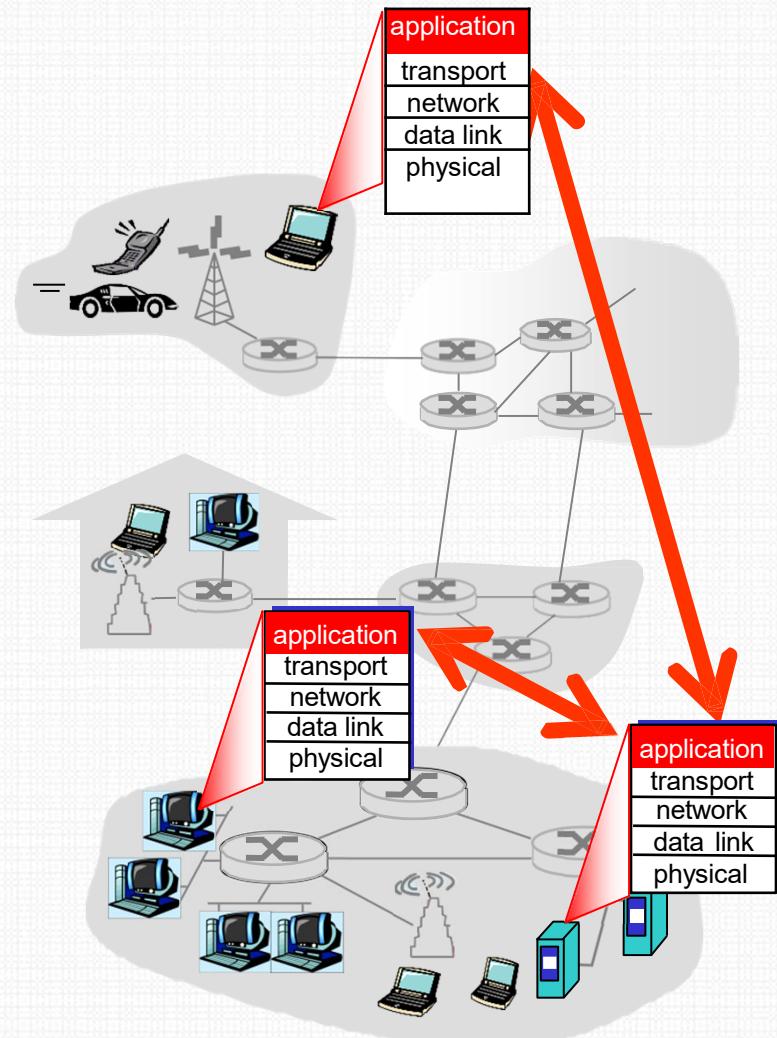
2. La Web y HTTP

- Servicio de transferencia de información multimedia WWW
 - Estructura básica de una página web
 - Identificación de los objetos: URL
 - Lenguaje HTML



Desarrollo de aplicaciones en red:

- Programas que se ejecutan en varios sistemas terminales
 - Pueden estar escritos en distintos lenguajes
- Se comunican a través de la red
 - ¿En qué elementos de la red tienen que ejecutarse?
- No es necesario desarrollar software para los dispositivos del núcleo de la red
 - No ejecutan aplicaciones de usuario

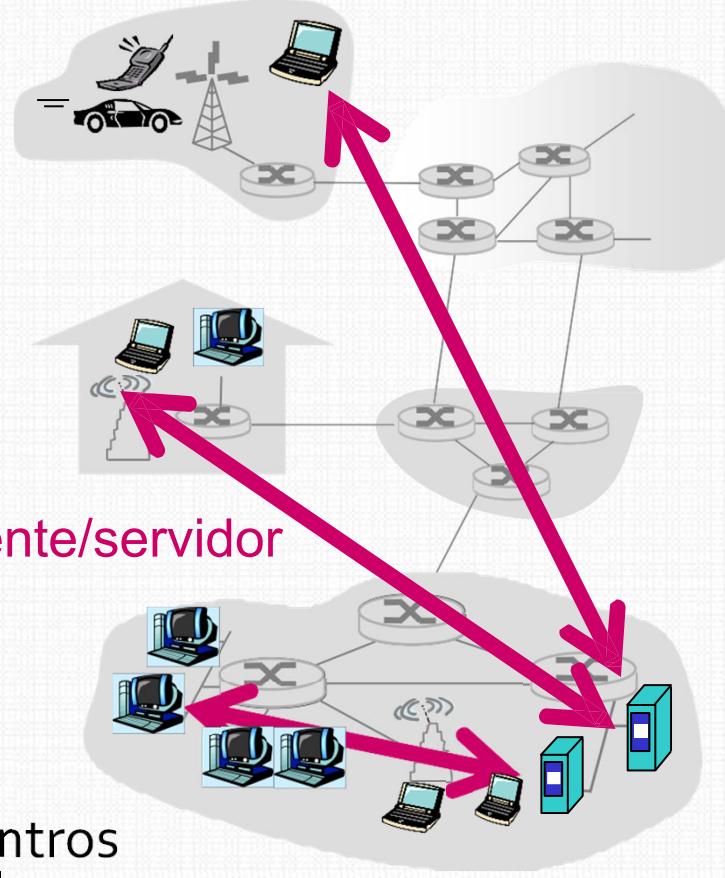


- Elegida por el desarrollador de la aplicación
- Define la estructura que tendrá la aplicación en los distintos sistemas terminales
 - Cliente–servidor
 - Entre pares (P2P)
 - Híbrido de cliente-servidor y P2P



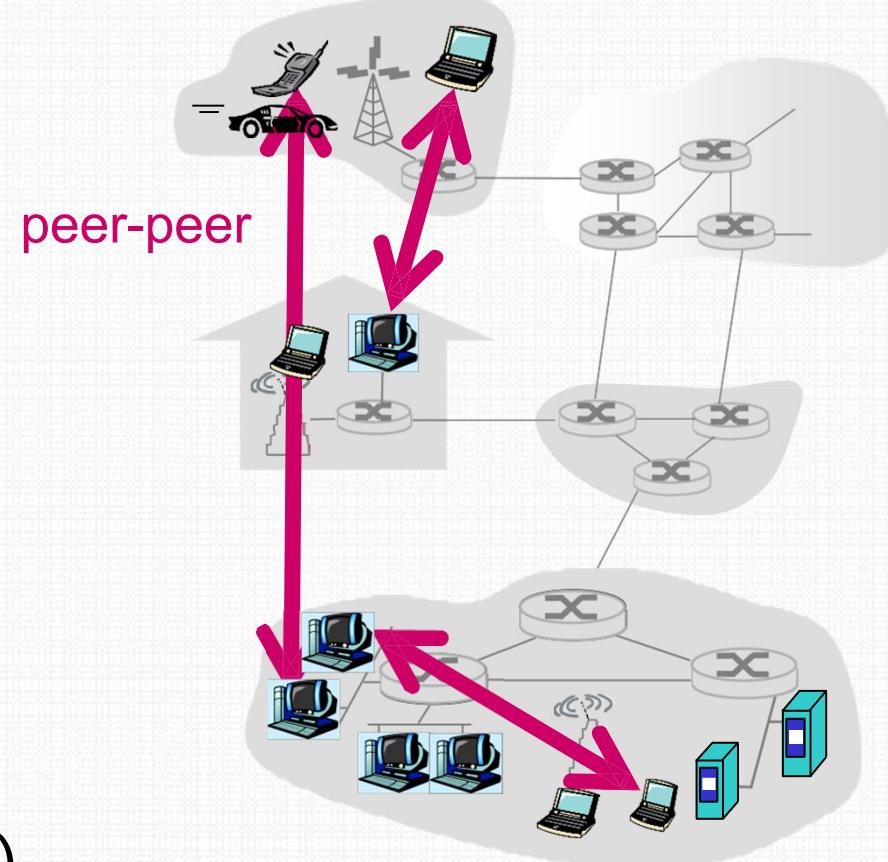
□ Arquitectura Cliente – Servidor

- Asimétrico
- Clientes
 - Inician la comunicación
 - Se conectan intermitentemente
 - Pueden tener IP dinámica
 - No comunican con otros clientes
- Servidor
 - Host siempre activo
 - Dirección IP fija
 - Escalado mediante granjas de servidores o centros de datos que simulan un único servidor virtual
- Ejemplos:
 - Correo electrónico
 - Web
 - ...



❑ Arquitectura P2P (entre pares)

- Los hosts se comunican directamente entre sí.
- Los pares se conectan de forma intermitente y pueden cambiar de dirección IP (IP dinámica)
- Modelo con una alta escalabilidad (auto-escalable), pero muy difícil de gestionar.
- Ejemplos: Gnutella



■ Modelo híbrido

- Los pares se registran previamente en un servidor
- Los pares buscan otros pares en el servidor
- Una vez localizado/s el/los otro/s par/es se sigue el modelo P2P

■ Skype

- Aplicación de voz y vídeo sobre IP
- Servidor centralizado para encontrar la IP del receptor
- Conexión cliente-cliente directa, no a través del servidor



■ Mensajería instantánea

- “chateo” entre dos usuarios es P2P
- Servidor centralizado para la detección/localización/desconexión de los clientes
 - Cada usuario registra su IP en servidor central al arrancar
 - Los usuarios contactan con servidor central para encontrar las IP de sus amigos

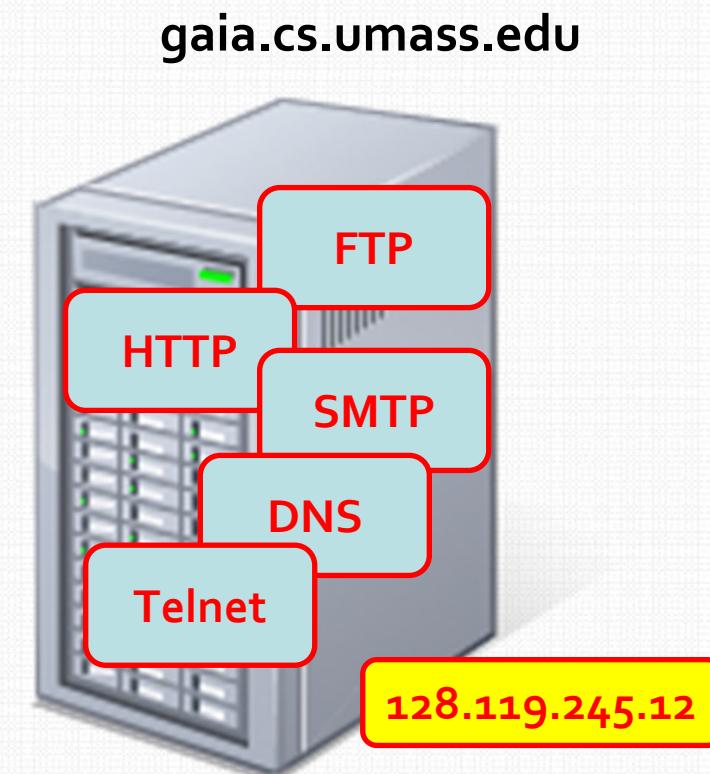
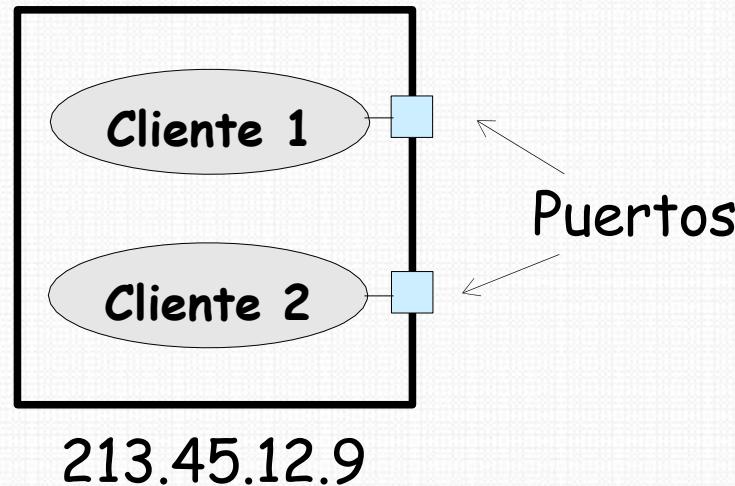


- **Proceso:** programa ejecutándose en un host
 - Dentro del mismo host, dos procesos se comunican por medio **de comunicación entre procesos** (definido por el SO).
 - Los procesos en hosts diferentes se comunican por **intercambio de mensajes** (regulado por un **protocolo de aplicación**).
 - Una **aplicación de red** consta de parejas de procesos que se intercambian mensajes a través de una red.

- **Proceso cliente:** proceso que inicia la comunicación
- **Proceso servidor:** proceso que espera ser contactado
- **Nota:** en las aplicaciones con arquitectura P2P los procesos pueden ser tanto clientes como servidores.



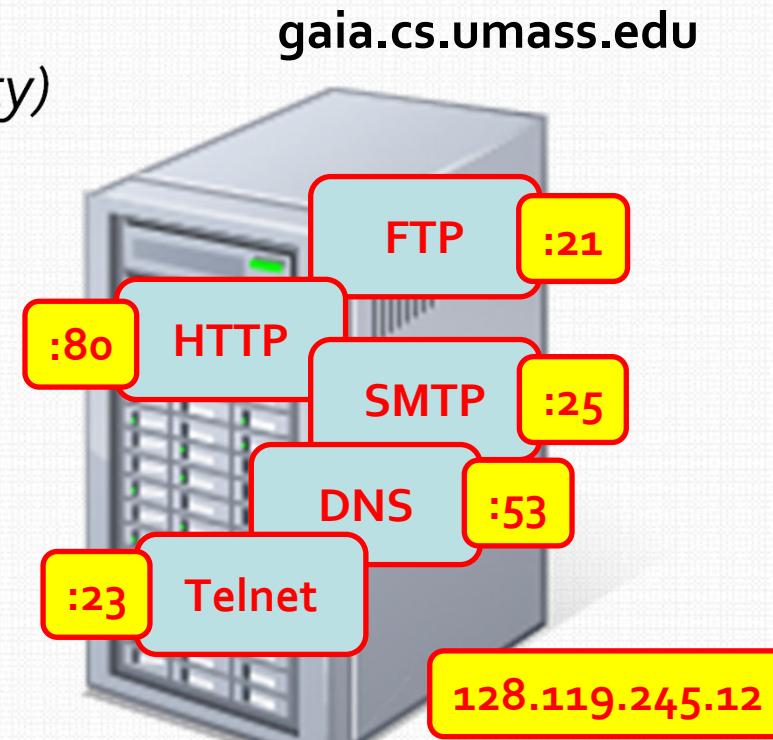
- Las direcciones IP identifican de forma única un computador en la red
- ¿Cómo distinguir entre distintos procesos dentro del mismo computador?
- Identificador de *puerto* (16 bits)



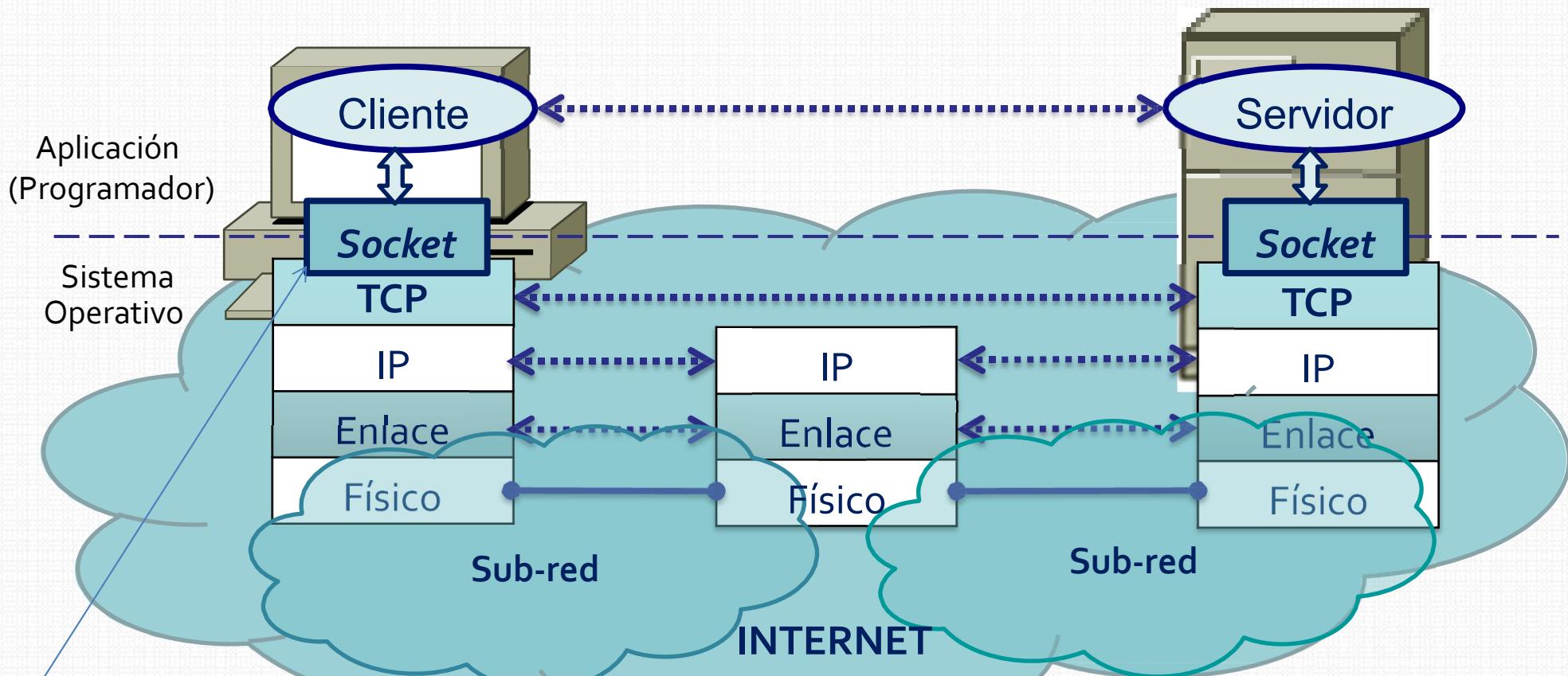
- El identificador debe incluir la dirección IP y el número de **puerto**
 - Puertos (16 bits = 65536 puertos)
 - Ejemplos de números de puerto:
 - Servidor HTTP: 80
 - Servidor Mail: 25
 - IANA (*Internet Assigned Numbers Authority*)
 - Lista de nº de puerto para protocolos estándar de Internet
- ❖ Para enviar un mensaje HTTP al servidor Web gaia.cs.umass.edu:

Dirección IP: 128.119.245.12

Número puerto: 80

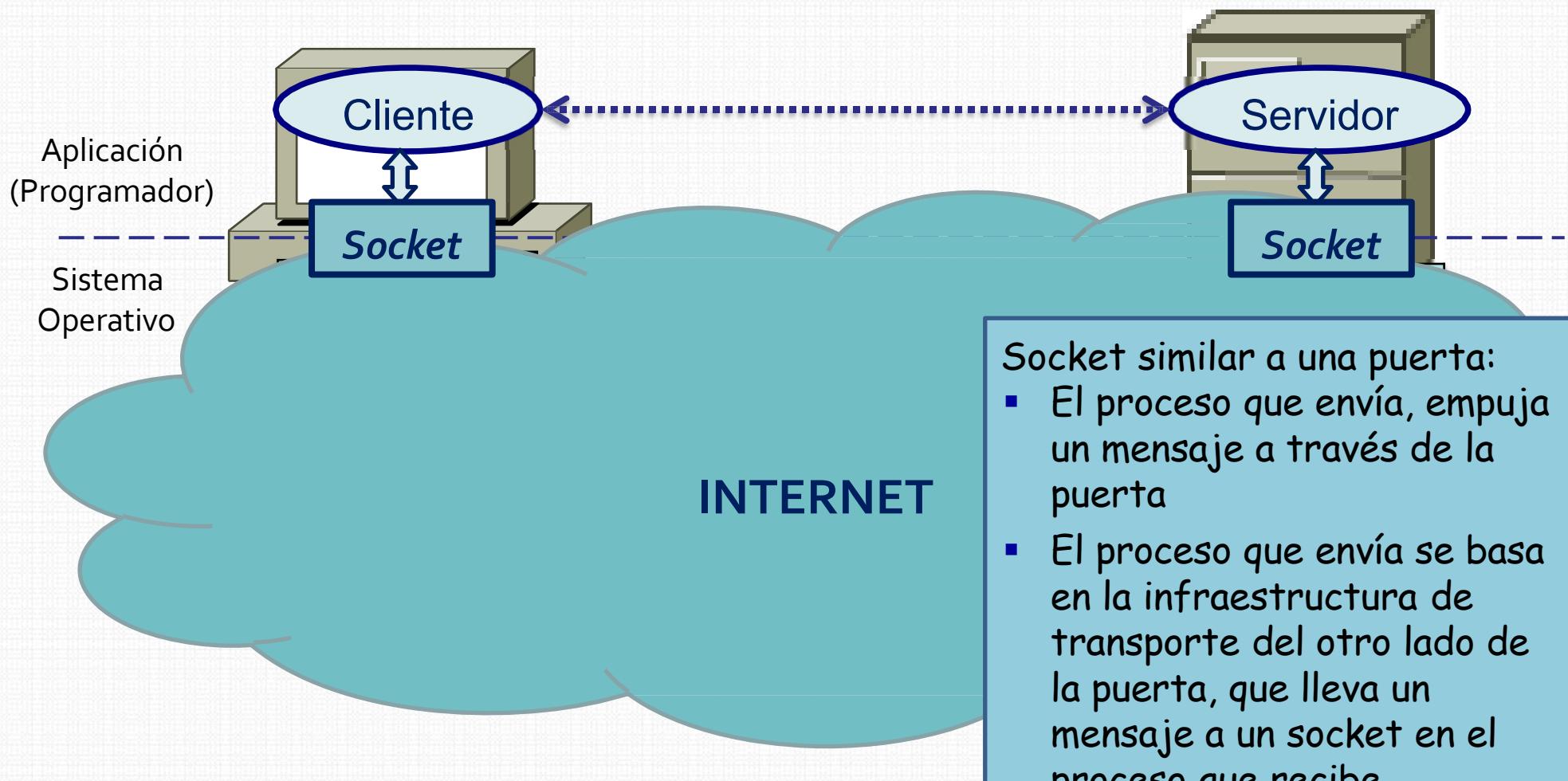


- Los procesos envían y reciben mensajes a través de sus *sockets*...



Interfaz entre la capa de aplicación y la capa de transporte o
Interfaz de programación de aplicaciones (*API, Application Programming Interface*)

- ...pero tienen poco control sobre los niveles inferiores



□ Servicios que puede ofrecer un protocolo de transporte

- Fiabilidad
 - Algunas aplicaciones (ej., audio) toleran parcialmente la pérdida de datos
 - Otras app (ej., transferencia de ficheros, telnet) requieren transferencia de datos 100% fiable
- Retardos (garantías de temporización)
 - Algunas aplicaciones (ej., telefonía, videoconferencia, juegos en red) requieren bajos retardos para ser “efectivas”.
- Productividad
 - Algunas aplicaciones (ej., Multimedia) requieren una mínima tasa de productividad para ser “efectivas”.
 - Otras app (“app elásticas”) hacen uso de la tasa que tengan disponible (ej., correo electrónico, transferencia de ficheros)
- Seguridad
 - ¿Encriptación? ¿Integridad de los datos?



- Dos tipos de servicio posibles ofrecidos a las aplicaciones en TCP/IP:
 - Orientado a la conexión (**TCP**)
 - Sin conexión (**UDP**)
- La elección depende de las necesidades de la aplicación



Servicio TCP (*Transmission Control Protocol*)

- Orientado a Conexión
 - Requiere el establecimiento de una conexión entre los procesos.
- Transporte fiable
 - Los errores de transmisión por la red son recuperados.
- Control de flujo
 - El emisor no agobia al receptor con un exceso de datos.
- Control de congestión
 - Se ralentiza al emisor cuando la red se sobrecarga.
- **No ofrece**
 - Garantías de tiempo
 - Productividad mínima
 - Seguridad

Servicio UDP (*User Datagram Protocol*)

- Servicio de transferencia de datos **no fiable**
- **No ofrece**
 - Conexión
 - Fiabilidad
 - Control de flujo
 - Control de congestión
 - Garantías de tiempo
 - Productividad mínima
 - Seguridad

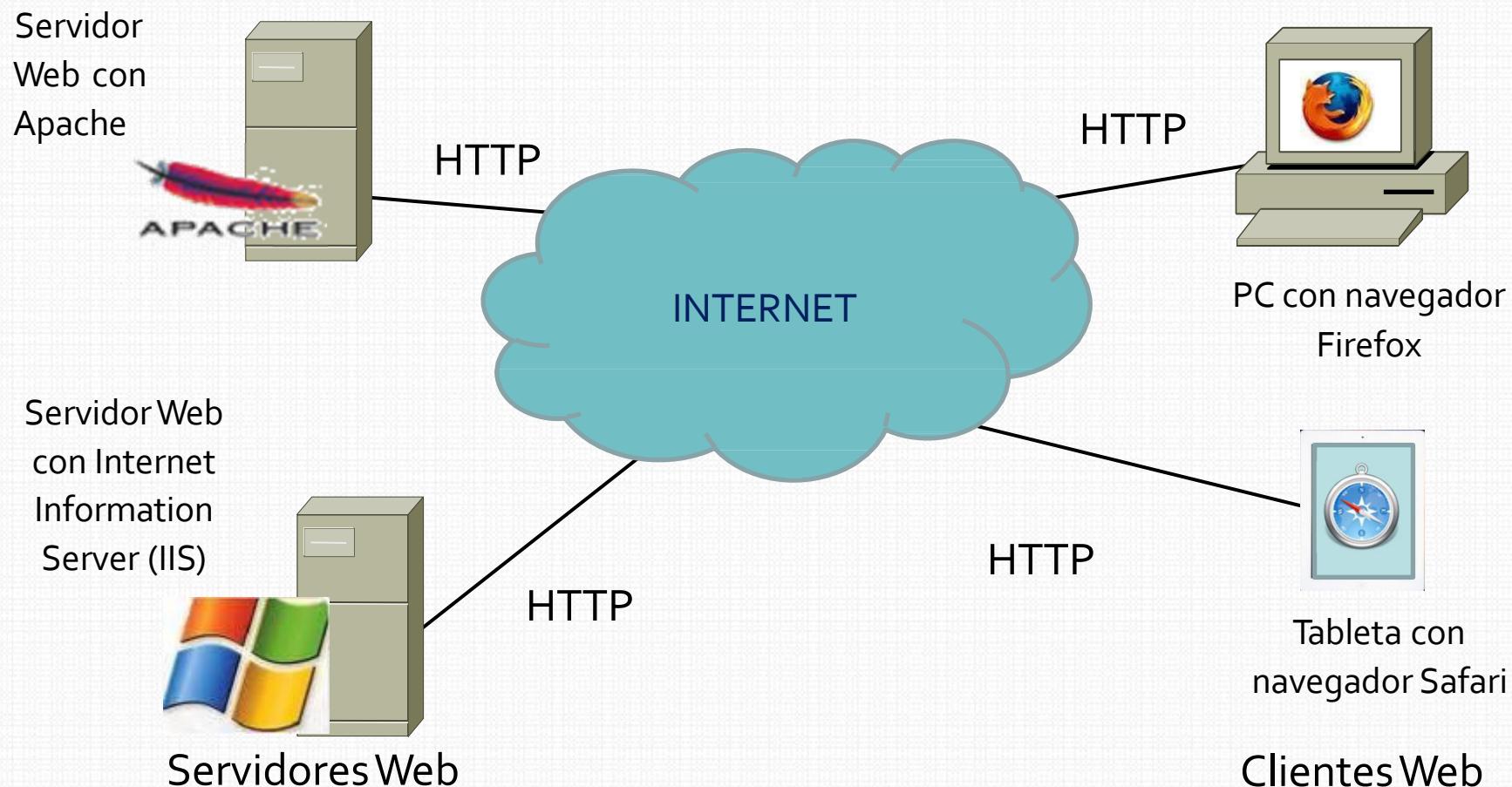


- Comunicación entre **procesos**:
 - En el mismo host, comunicación interproceso
 - En hosts diferentes, **Protocolo de nivel de aplicación**

Aplicación	Protocolo de aplicación	Protocolo de transporte
Correo electrónico	SMTP [RFC 2821]	TCP
Terminal remoto	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Transferencia de ficheros	FTP [RFC 959]	TCP
Multimedia streaming	HTTP (ej., YouTube), RTP [RFC 1889]	TCP o UDP
Telefonía	SIP, RTP, propietario (ej., Skype)	UDP
Juegos online	propietario	UDP preferentemente



- Diversas aplicaciones implementan un determinado servicio... pero el protocolo es siempre el mismo
- Web, ejemplo de aplicación de red, emplea como protocolo del nivel de aplicación HTTP



Define:

- Los tipos de mensajes intercambiados (de solicitud, de respuesta,...)
- La sintaxis de los mensajes
 - Tamaño de los campos
 - Delimitadores
- La semántica de los campos
- Las reglas para decidir cuándo y qué tipo de mensajes hay que enviar, las acciones a realizar, etc.

¿Quién define los protocolos de aplicación?

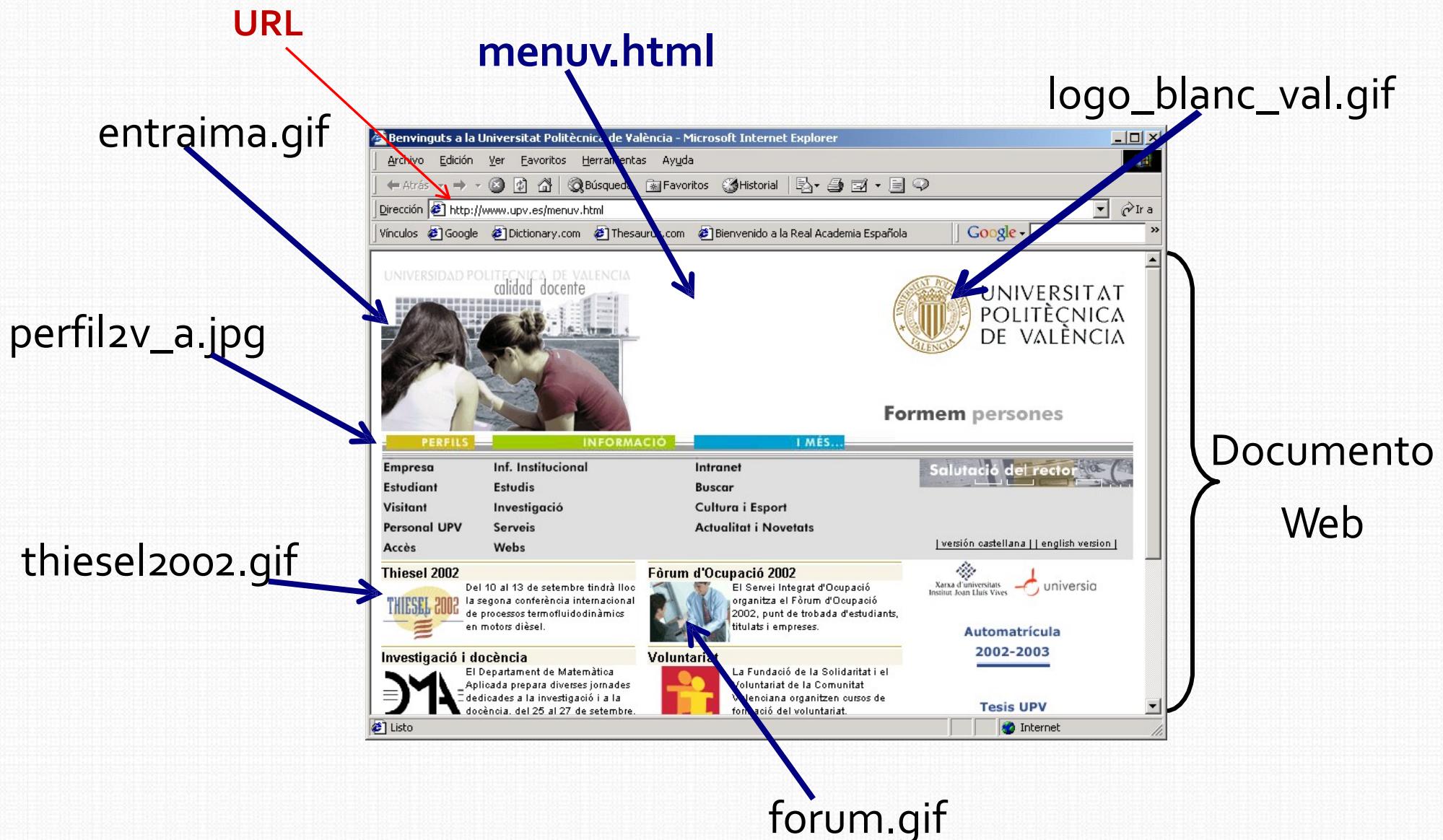
- Protocolos de dominio público:
 - Definidos en RFC's
 - Por ejemplo, HTTP, SMTP, ...
- Protocolos propietarios
 - Por ejemplo, Skype, SMB



- WWW (World Wide Web)
 - Servicio de transferencia de información hipertexto e hipermedia
- Protocolo HTTP (*HyperText Transfer Protocol*)
- Los documentos web constan de objetos.
 - Objetos: ficheros HTML, imágenes JPEG o GIF, applets Java, ficheros de audio o vídeo
- Un documento web se compone de un **fichero base HTML** que referencia a otros objetos
 - Cada objeto se referencia por su **URL** (*Uniform Resource Locator*)



Estructura de un documento web



■ ***URL (Uniform Resource Locator)***

Expresa de manera uniforme los distintos recursos que podemos acceder con el cliente Web (RFC 1738 y 1808, actualizados en RFC 3986)

■ Consta de varios campos:

<http://www.upv.es/valencia/index.html>

Ruta y nombre del documento

Protocolo:
http, ftp, news, etc

Nombre del servidor Web

■ También se puede especificar el puerto (por defecto se conecta al 80), usuario, etc.

protocolo://[usuario:contraseña@] host [:puerto] directorio.archivo

■ Las URL en un documento HTML pueden ser absolutas o completas (ejemplo anterior) o relativas o parciales:

[valencia/index.html](#)



- **HyperText Mark-up Language (RFC 1866)**
- Lenguaje de descripción de páginas web
 - Describe cómo se visualizarán por pantalla los elementos de texto: párrafos, listas, tablas, etc.
 - Basado en la referenciación
- Estándar a cargo de la W3C (HTML Working Group)
- Se basa en la inserción de **marcadores de control** o **etiquetas** que delimitan elementos de documento como cabeceras, párrafos, imágenes, etc



`<TITLE> Bienvenidos a la web </TITLE>`



```
<HTML><HEAD><TITLE>Índice de enlaces relacionados con RDC </TITLE></HEAD>
```

```
<BODY BACKGROUND="fondo.jpg">
<H2><FONT color=purple>Enlaces de interés</FONT></H2>
<UL>
<STRONG>
<LI><A HREF="is.html">Una buena introducción a Internet</A>
<LI><A HREF="htmlref.html">Introducción al HTML</A> (En castellano)
<LI><A HREF="html.html">Descripción detallada de los elementos HTML</A>
</STRONG>
</UL>

<P><IMG SRC="hola.jpg" ALIGN=BOTTOM></P>
</BODY>
</HTML>
```

Enlaces de interés

- [Una buena introducción a Internet](#)
- [Introducción al HTML \(En castellano\)](#)
- [Descripción detallada de los elementos HTML](#)

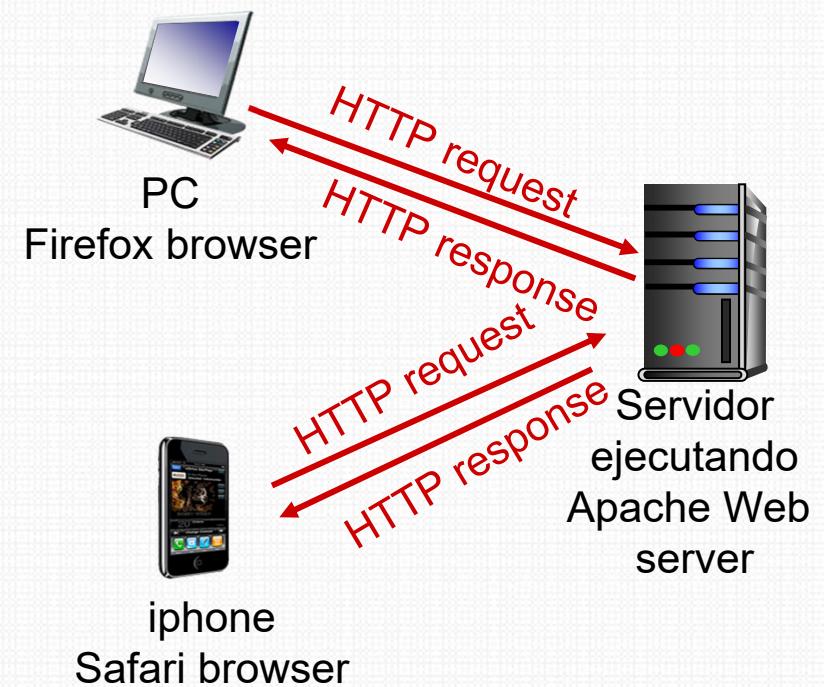


2. La Web y HTTP

- Generalidades sobre HTTP
- Formato de una transacción HTTP
 - Métodos de petición
 - Cabeceras
 - Mensajes de respuesta HTTP
 - Códigos de estado
- Tipos de conexiones
 - No persistentes
 - Persistentes
- ¿Por qué HTTP/2?
- Interacción usuario-servidor: Cookies
- Proxy (caché web)
- Peticiones GET condicionales



- HTTP: *HyperText Transfer Protocol*
 - Protocolo del nivel de aplicación
- Modelo cliente/servidor
 - **Cliente**: navegador que solicita, recibe y muestra objetos Web
 - **Servidor**: servidor web envía objetos en respuesta a las peticiones
- Versiones:
 - HTTP 1.0 (RFC 1945)
 - HTTP 1.1 (RFC 2616)
 - HTTP 2.0 (RFC 7540)



- Utiliza **TCP**:
 - El cliente inicia una conexión TCP (crea un socket) al servidor en el puerto **80**
 - El servidor acepta la conexión TCP del cliente
 - Intercambio de mensajes HTTP entre el navegador (cliente HTTP) y el servidor Web (servidor HTTP)
 - Cierre de la conexión TCP
- HTTP es un protocolo "**sin memoria de estado**"
 - El servidor no mantiene información sobre las solicitudes anteriores del cliente
 - Se simplifica la implementación



- Dos tipos de mensajes con el mismo formato: peticiones o solicitudes (cliente) y respuestas (servidor)
 - Línea inicial (petición/estado): **texto ASCII** (obligatoria, termina con <CR><LF>)
 - Líneas de cabecera: **texto ASCII** (opcionales, terminan con <CR><LF>)
 - Línea en blanco (obligatoria <CR><LF>)
 - Cuerpo: cualquier tipo de **texto o datos binarios** (opcional)
- Formato general **mensaje petición**:

Órdenes GET,
POST, HEAD,..

Método	sp	URL	sp	Versión	cr	lf
Cabecera	:	Valor	cr	If		
Cabecera	:	Valor	cr	If		
⋮						
Cabecera	:	Valor	cr	If		
cr	lf					
Cuerpo						

Línea de petición

Líneas de
cabecera

Línea en blanco

- Ejemplo **mensaje petición:**

Línea de petición
(órdenes GET,
POST, HEAD,...)

Líneas de
cabecera

```
GET /index.html HTTP/1.1
Host: www-net.cs.umass.edu
User-Agent: Firefox/3.6.10
Accept-Language: en-us,en
Connection: close
```

← Línea en
blanco



Solicitud al servidor

- GET es el método más común
- Significa “dame este recurso”

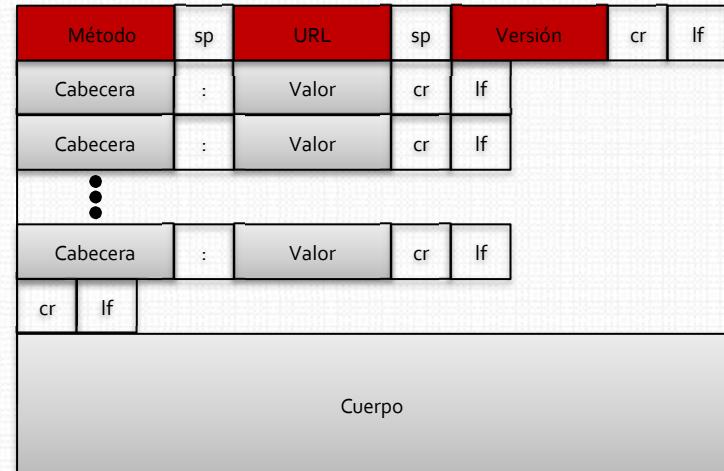
`GET /path/to/file/index.html HTTP/1.0`

Envío de información al servidor (cargar entrada de formulario)

- Mediante GET en el URL

`/search?client=ubuntu&q=telefonica+peerings`

- Mediante POST en el cuerpo de la petición



```
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
```

← Línea en blanco

```
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
```

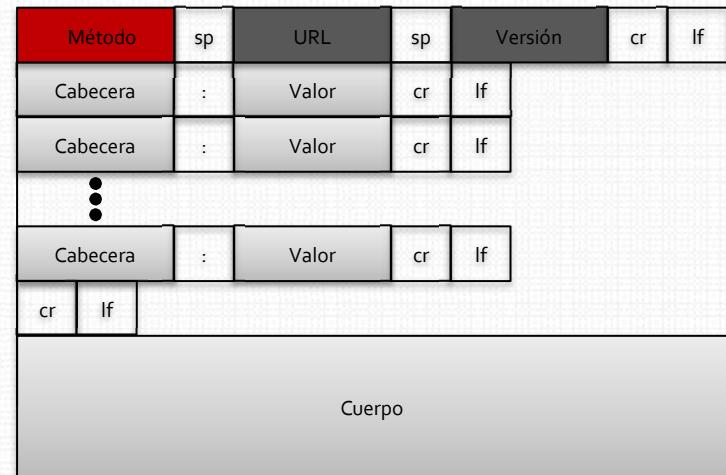
? : separación entre el
recurso y los parámetros
=: separación entre
nombre del campo del
formulario y su valor
& : separación entre
parámetros
+: espacio en blanco

← Línea en blanco



HTTP /1.0

- GET
- POST
- HEAD
 - Solo las cabeceras. Pregunta al servidor por un objeto, pero la respuesta es sin objeto.

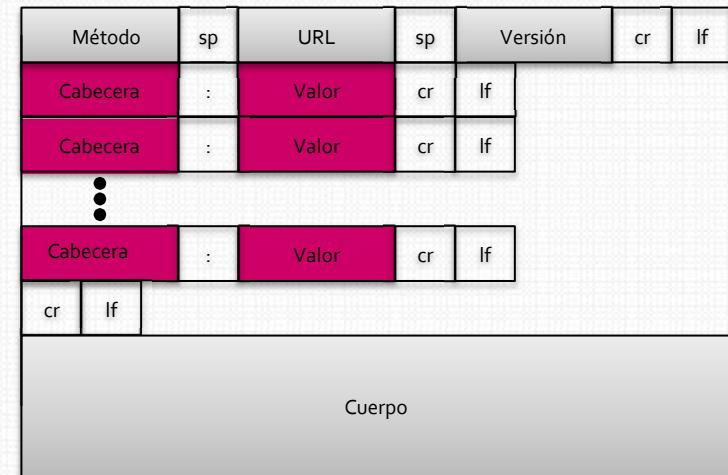


HTTP /1.1

- GET, POST, HEAD
- PUT
 - Carga el fichero del cuerpo del mensaje en la ruta especificada en el URL
- DELETE
 - Borra el fichero especificado en el URL



- Dan información
 - sobre la petición o la respuesta
- El formato de una cabecera es:
Nombre-Cabecera: valor <CR><LF>
- Acaban con una línea en blanco
<CR><LF>
- HTTP 1.0 define 16 cabeceras,
ninguna es obligatoria
- HTTP 1.1 define 46 cabeceras, solo una (**Host:**) es
obligatoria en las peticiones
- Cuando un servidor no entiende una
cabecera, simplemente la ignora

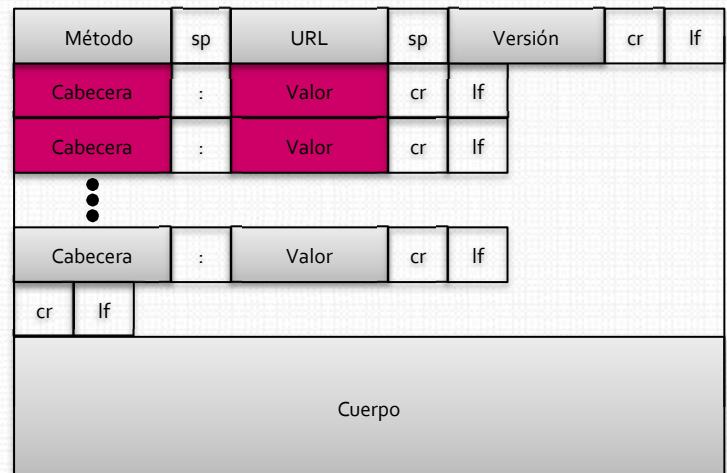


Petición

- **User-Agent**: el navegador que usa el cliente
- **Accept**: tipo de contenidos que son aceptados (**1.1**)
- **Host**: nombre del servidor al que va dirigida la petición (**obligatoria en 1.1**)

Respuesta

- **Content-Type**: el tipo MIME de los datos en el cuerpo, p.e.: **text/html** o **image/gif**
- **Content-Length**: el tamaño en bytes del cuerpo
- **Content-Encoding**: indica una codificación adicional aplicada al contenido
- **Date**: Fecha de la respuesta (**obligatoria en 1.1**)



- Formato general **mensaje respuesta**:

Línea de estado



Versión	sp	Código	sp	Frase estado	cr	lf
Cabecera	:	Valor	cr	lf		
Cabecera	:	Valor	cr	lf		
●						
Cabecera	:	Valor	cr	lf		
cr	lf					
Cuerpo						

Línea de estado

(protocolo,
código de estado,
frase de estado)

HTTP/1.1 200 OK

Cabeceras

Línea en blanco

Datos:
fichero HTML
solicitado

Connection: close
Date: Thu, 09 Sep 2010 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 2008
Content-Length: 6821
Content-Type: text/html

<html><body><h1>Bienvenido a Redes</h1>
Esto fue breve.
</body></html>



- Primera línea de la respuesta del servidor al cliente
- Códigos de estado:
 - **1xx** Información al cliente
 - El servidor recibió la petición, pero el resultado no está disponible
 - **2xx** Éxito en la operación
 - **200 OK**
 - **3xx** Redirección a otra URL
 - **301 Moved Permanently**
 - **4xx** Error por parte del cliente
 - La petición contenía un error que impidió un resultado satisfactorio
 - **400 Bad Request**
 - El servidor no comprende el mensaje de petición
 - **404 Not Found**
 - El documento solicitado no se encuentra en este servidor
 - **5xx** Error por parte del servidor
 - **500 Internal Server Error, 501 Not Implemented, 505 HTTP Version Not Supported**
 - Petición correcta pero el servidor no pudo servirla

Versión	sp	Código	sp	Frase estado	cr	lf
Cabecera	:	Valor	cr	lf		
Cabecera	:	Valor	cr	lf		
⋮						
Cabecera	:	Valor	cr	lf		
cr	lf					

Cuerpo



Petición {

GET /img/sic/pixelb.gif HTTP/1.1
User-Agent: Mozilla/4.76 (Windows NT 5.0; U)
Host: sic.uji.es
Accept: image/gif, image/x-xbitmap, image/jpeg
Accept-Charset: iso-8859-1,* ,utf-8

Respuesta {

HTTP/1.1 200 OK
Date: Fri, 25 Jun 2010 13:02:54 GMT
Server: Apache/1.3.12 (Unix) mod_perl/1.21
Last-Modified: Fri, 11 Jun 2010 16:49:46 GMT
Content-Length: 799
Content-Type: image/gif

bla bla bla



- Probar HTTP desde telnet (como cliente):
 - Realiza una sesión de telnet a tu servidor Web preferido:

```
telnet www.upv.es 80
```

Abre una conexión TCP con el puerto 80 (valor por defecto de los servidores HTTP) de www.upv.es. Cualquier cosa que se teclee se enviará por la conexión.

- Escribe una petición GET HTTP:

```
GET / HTTP/1.0
```

Envía una petición de la página /. Es necesario dejar una línea en blanco para terminar la cabecera HTTP.

- Observa el resultado...



No persistentes

- Una conexión TCP por objeto
 - Empleadas **por defecto** en **HTTP/1.0**
 - Puede indicarse/modificarse mediante la cabecera **Connection**

Persistentes

- Pueden enviarse varios objetos por la misma conexión TCP
 - Conexión **por defecto** en **HTTP/1.1**
 - Puede modificarse mediante la cabecera **Connection**



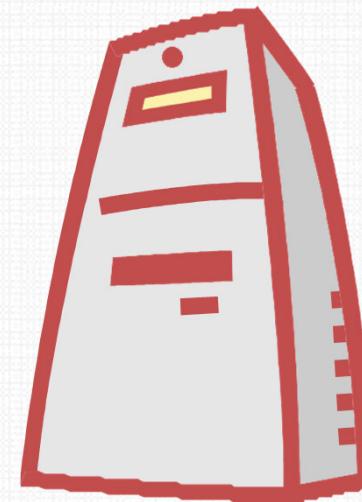
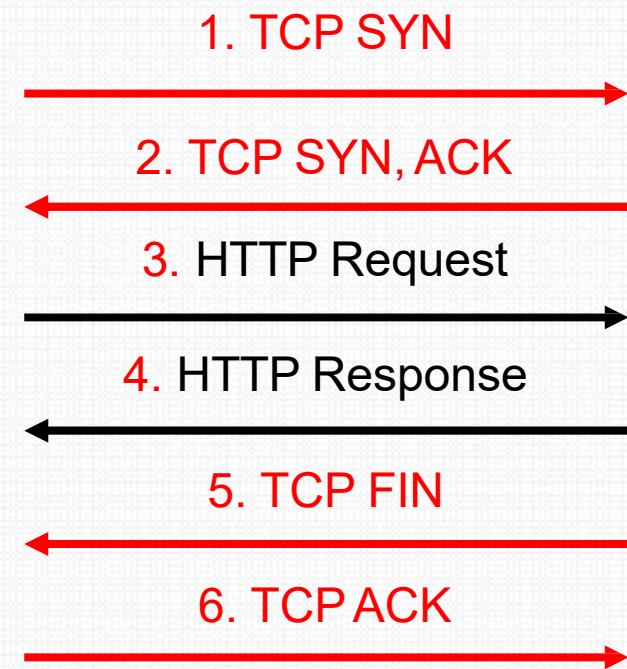
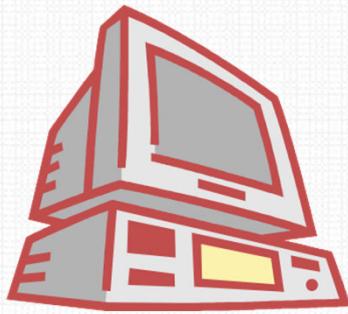
- El cliente se conecta con el servidor (puerto 80)
- El cliente envía la orden:

```
GET /valencia/index.html HTTP/1.0 <CR><LF> <CR><LF>
```

- El servidor envía el documento HTML y cierra la conexión
- Para conseguir **cada uno de las objetos** incluidos en la página (iconos, imagen de fondo, dibujos, etc.) el cliente debe establecer una **nueva conexión TCP**

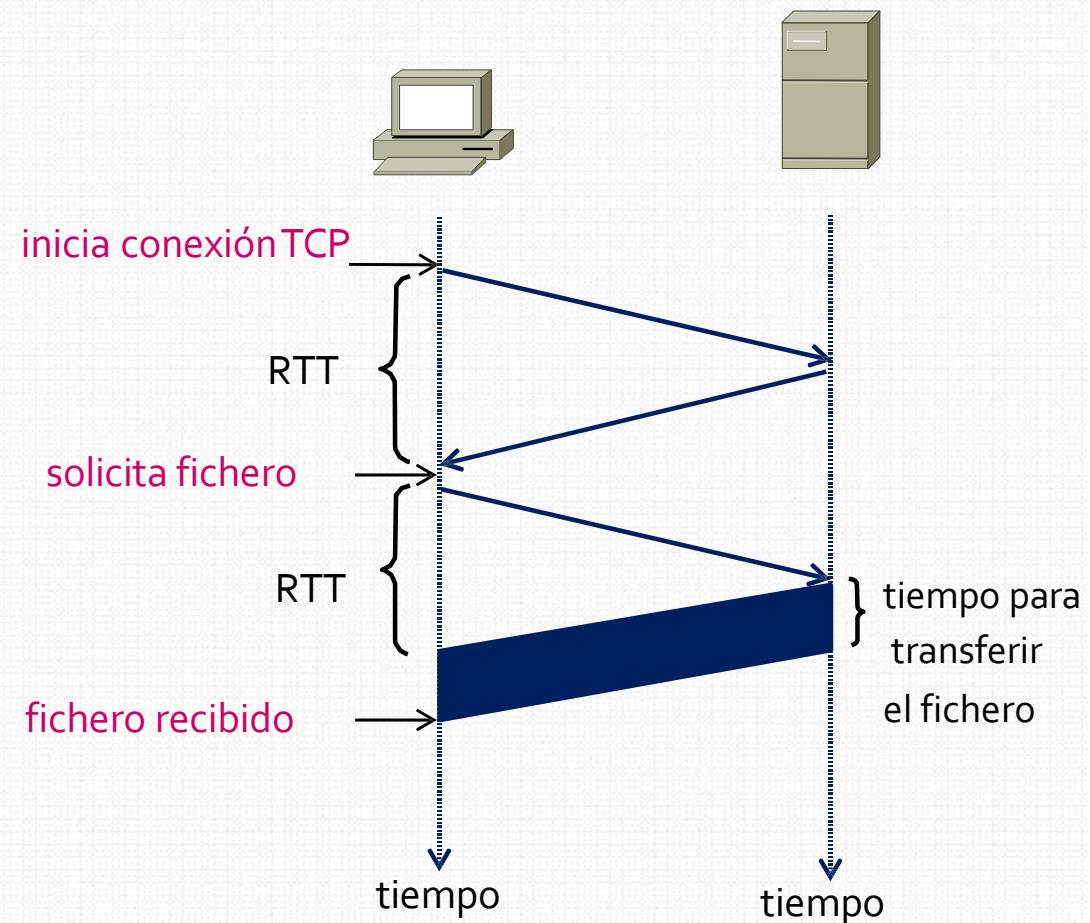


■ HTTP 1.0



- **RTT (Round Trip Time):**
tiempo para enviar un paquete
y recibir su respuesta asociada
- **Tiempo de respuesta:**
 - Un RTT para establecer la conexión TCP
 - Un RTT para la petición HTTP y los primeros bytes de la respuesta
 - Tiempo de transmisión

$$T_{\text{total}} = 2RTT + T_{\text{trans}}$$



- Sobrecarga y retardo:
 - Apertura y cierre de conexiones TCP
 - *Arranque lento* en TCP
- Soluciones
 - Los navegadores con frecuencia abren conexiones TCP paralelas (simultáneas) al mismo servidor para buscar objetos referenciados
 - Sobrecarga en los servidores
 - Se aconseja un máximo de 4 conexiones simultáneas al mismo servidor
 - El cliente puede solicitar que el servidor mantenga la conexión abierta
 - Cabecera *Connection: Keep_alive*
 - El servidor puede aceptar o no la petición
 - Cabecera *Connection: close*



- El servidor deja la conexión TCP abierta después de enviar una respuesta
- Por defecto en HTTP 1.1
- La conexión se mantiene abierta hasta que servidor o cliente decidan cerrarla
 - Cabeceras:
Connection: keep-alive <CR><LF>
Keep-alive: valor_segs <CR><LF>
 - ¿Cuándo se cierra la conexión?
 - Por ejemplo, después de un tiempo de inactividad
 - Empleando la cabecera: *Connection: close <CR><LF>*
- Mensajes HTTP subsecuentes entre los mismos cliente/servidor son enviados por la **misma conexión**.



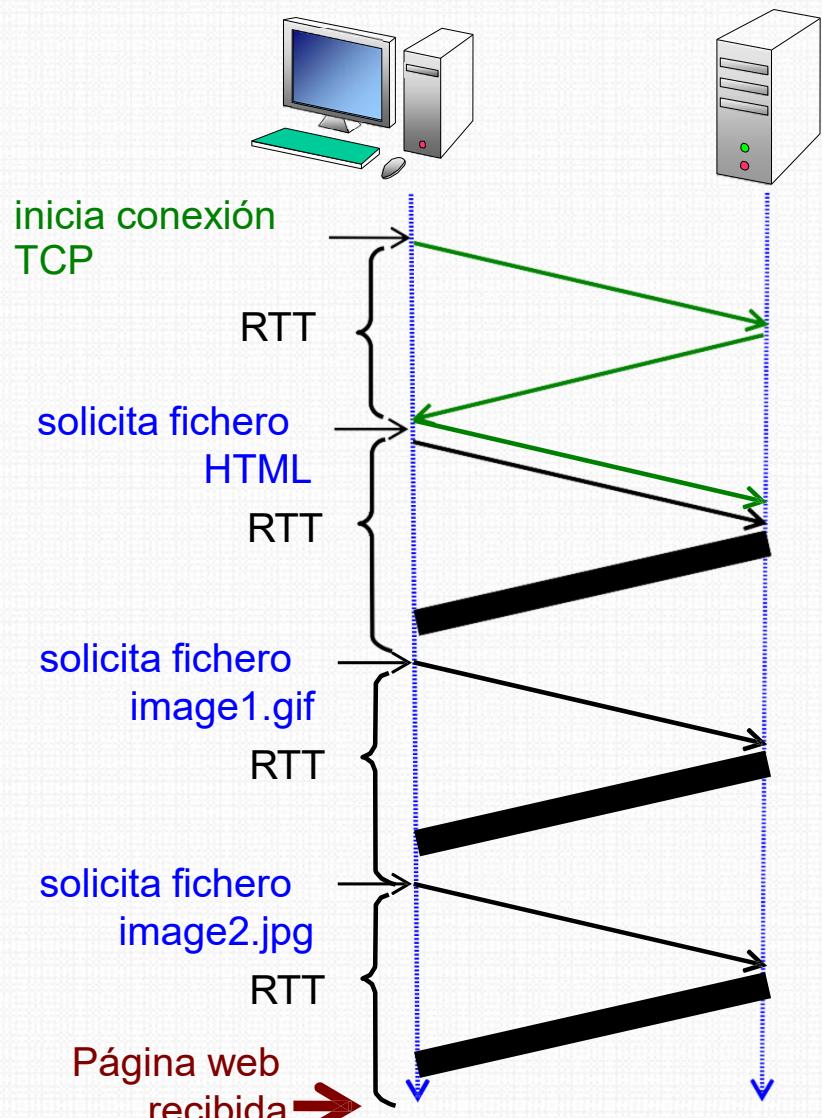
■ Persistencia sin pipelining

- El cliente realiza una nueva solicitud solo cuando la previa ha sido recibida.
- Se aconseja un máx. de 2 conexiones simultáneas al mismo servidor
- **1 RTT por cada objeto referenciado**

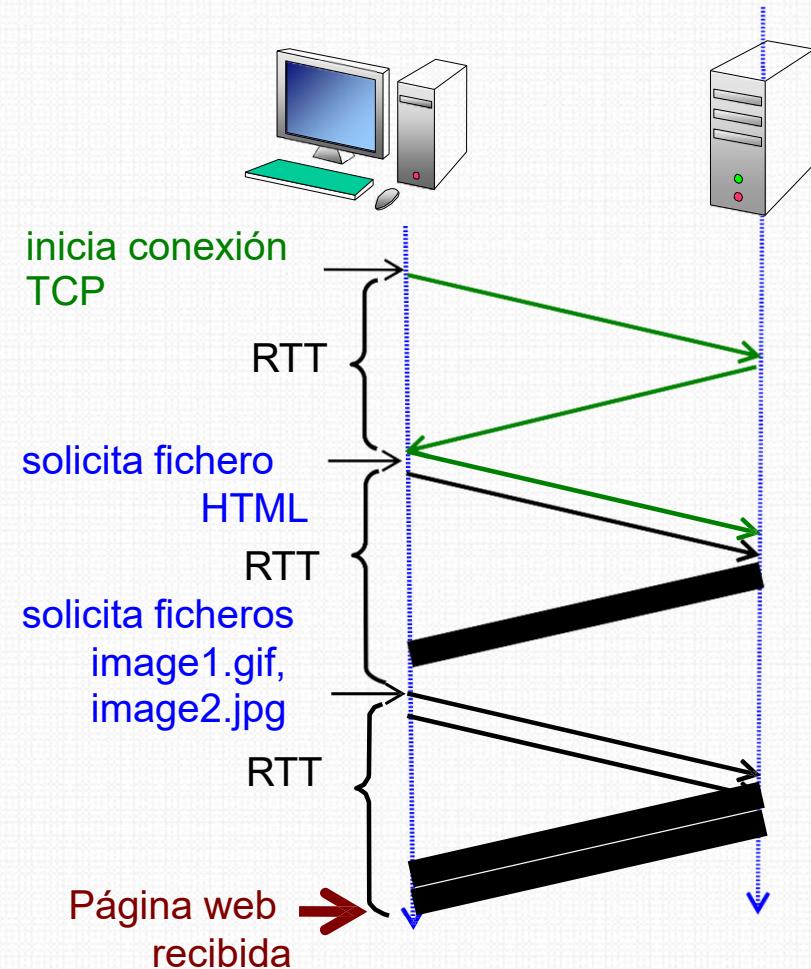
■ Persistencia con pipelining

- Optimización
- **Por defecto en HTTP/1.1**
- El cliente envía una petición tan pronto como encuentra la referencia a un objeto
- Se aconseja máx. 1 conexión al mismo servidor
- **1 RTT para todos los objetos referenciados**





Conexiones HTTP persistentes
sin pipelining



Conexiones HTTP persistentes
con pipelining

2. La Web y HTTP

- Generalidades sobre HTTP
- Formato de una transacción HTTP
 - Métodos de petición
 - Cabeceras
 - Mensajes de respuesta HTTP
 - Códigos de estado
- Tipos de conexiones
 - No persistentes
 - Persistentes
- **¿Por qué HTTP/2?**
- Interacción usuario-servidor: Cookies
- Proxy (caché web)
- Peticiones GET condicionales



■ Problemas en HTTP/1.1

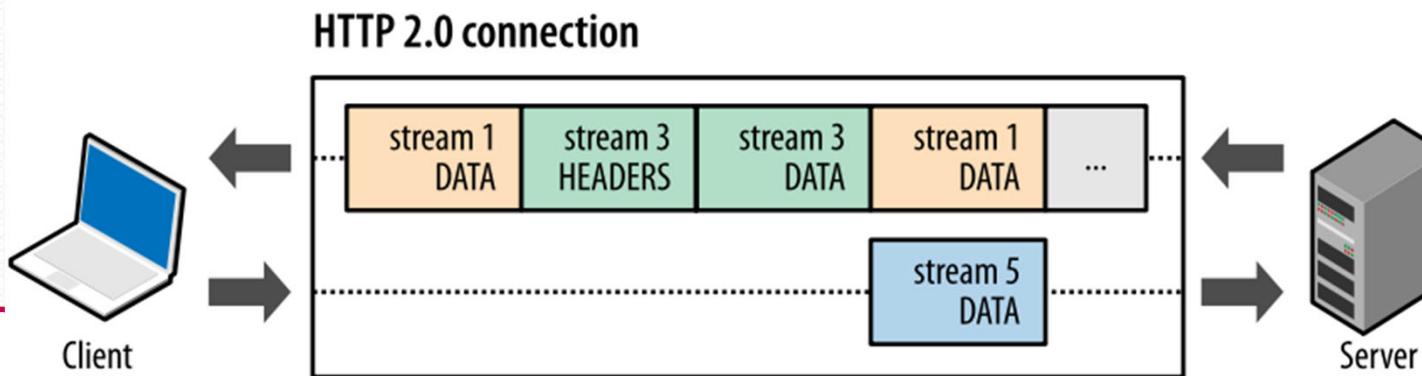
- El formato de los mensajes HTTP se diseñó para su implementación con las herramientas de la década de los 90's, pero no es adecuado para el rendimiento de las aplicaciones web actuales.
- HTTP/1.1 con pipelining aborda parcialmente la concurrencia de peticiones, pero se puede producir bloqueo en la cabecera de línea (las respuestas se sirven en serie). Por lo que se suelen utilizar varias conexiones persistentes sin pipelining.
- Algunos campos de las cabeceras son repetitivos y prolijos lo que da como resultado una latencia excesiva.



- HTTP/2 soporta todas las características básicas de HTTP/1.1, pero pretende ser más eficiente.
- En lugar de crear una conexión diferente para enviar cada objeto se crea **una sola conexión para enviar toda la información**
- Es un protocolo binario
 - Más eficiente, compacto, menos propenso a errores
- Codificación eficiente de los campos de cabecera
 - Compresión basada en eliminar elementos redundantes
- Realiza control de flujo
 - Solo se transmite información que pueda ser utilizada por el receptor
- Permite priorizar las solicitudes para mejorar el rendimiento
- Empleo de *Server push*
 - El servidor se anticipa y envía información antes de que el usuario la solicite

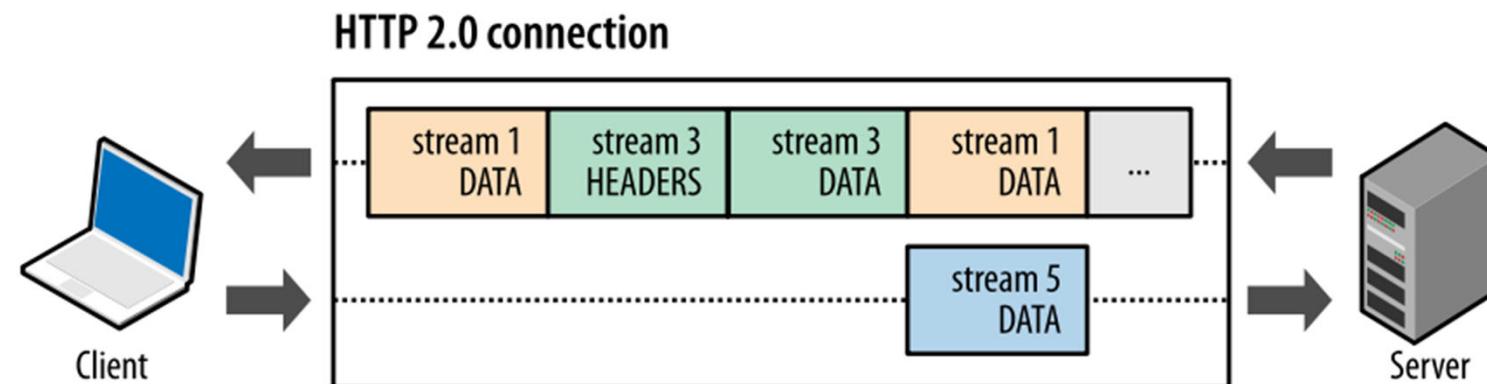


- La unidad básica del protocolo es una trama
 - Cada trama tiene un tipo y propósito distinto (HEADERS, DATA, SETTINGS, WINDOW_UPDATE, PUSH_PROMISE,...)
- **Multiplexado** para enviar y recibir diversos mensajes al mismo tiempo sobre una misma conexión
 - El multiplexado se logra a través de los **flujos (streams)**
 - Una misma conexión puede contener múltiples flujos concurrentes
 - Un flujo es una secuencia independiente bidireccional de tramas
 - Cada par petición-respuesta se asigna a un flujo
 - Los flujos son independientes entre sí (no se bloquean entre sí)



■ Por tanto:

- El cliente y servidor desglosan un mensaje HTTP en tramas diferentes, con capacidad de intercalarlas y reensamblarlas en el otro extremo



- Ej: el cliente transmite una trama DATA (stream 5) al servidor, mientras éste transmite una secuencia intercalada de tramas al clientes para las transmisiones 1 y 3
 - → 3 transmisiones paralelas



<https://http2.akamai.com/demo>



2. La Web y HTTP

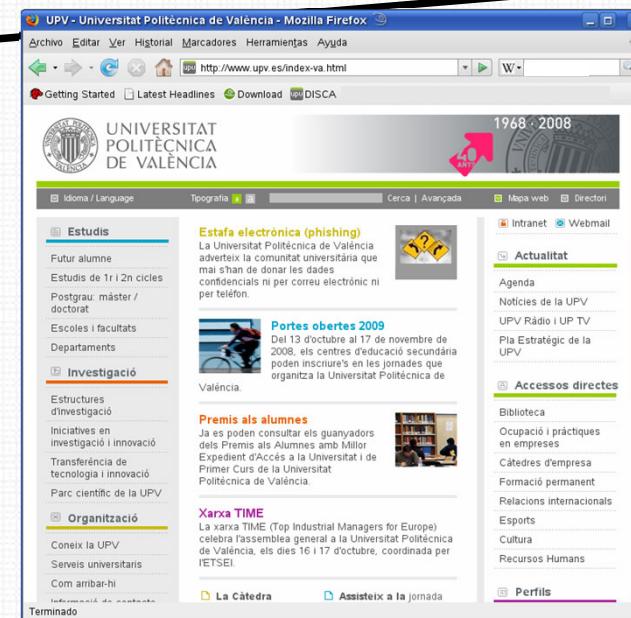
- Generalidades sobre HTTP
- Formato de una transacción HTTP
 - Métodos de petición
 - Cabeceras
 - Mensajes de respuesta HTTP
 - Códigos de estado
- Tipos de conexiones
 - No persistentes
 - Persistentes
- ¿Por qué HTTP/2?
- **Interacción usuario-servidor: Cookies**
- Proxy (caché web)
- Peticiones GET condicionales



- En HTTP los servidores **no guardan información de estado**
 - No mantienen información del cliente
 - Simplifica el diseño de los servidores
- Con las ***cookies*** se consigue mantener información del cliente
 - Con la colaboración del cliente
 - Con poco impacto sobre HTTP y los agentes de usuario
- ¿Qué son las *cookies*?
 - Los cookies son datos (identificadores numéricos, cadenas de caracteres,...)
 - No son obligatorias ... pero un servidor puede negarse al diálogo con un cliente que rechace sus cookies
 - Originalmente definidas por Netscape, después estandarizadas en el RFC 2965. El RFC actual es el 6265.



Funcionamiento de las cookies

Cliente HTTPGET www.upv.es/index-es.html HTTP/1.1**Servidor HTTP****Set-cookie
idioma=es****Cookie
idioma=es**GET entidades/ABDC/index-es.html HTTP/1.1

■ Cuatro componentes:

1. Línea de cabecera en la respuesta HTTP:

Set-cookie: <nombre>=<valor>[;...;]

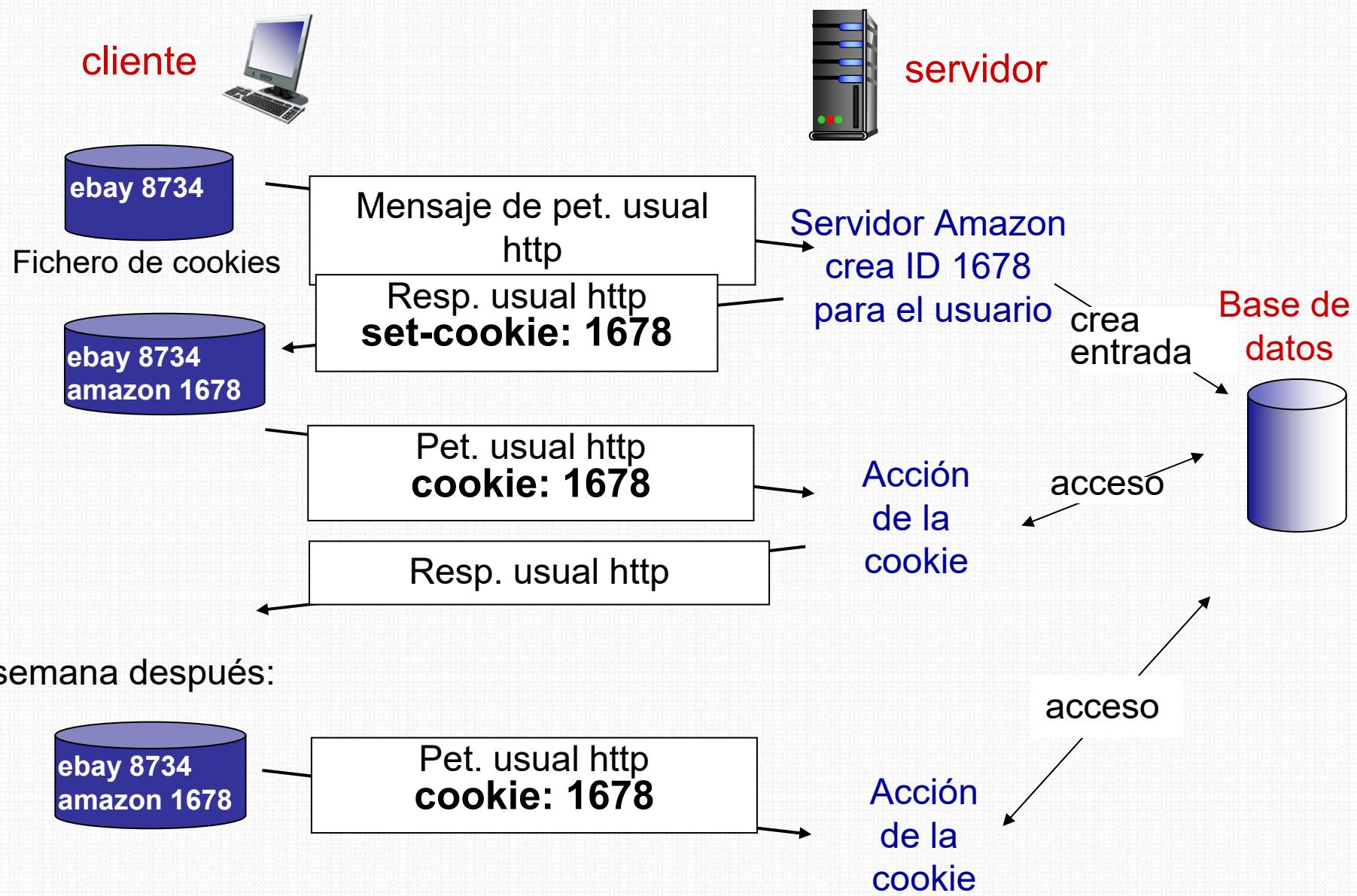
Opcionalmente otros campos: dominio, fecha de caducidad de la *cookie*, etc.

2. Línea de cabecera en la petición HTTP:

Cookie: <nombre>=<valor>

3. Fichero de *cookies* en el disco del usuario, gestionado por el navegador (*cookies.txt* en Firefox)
4. Una base de datos en el servidor Web





- Las *cookies* sirven para:
 - Identificación de usuarios
 - Cestas de compra virtuales
 - Recomendaciones
 - Personalización de páginas
 - Sesiones de usuario (*Webmail*)
 - Obtención de perfiles de usuarios
- Cookies y privacidad:
 - Las cookies permiten a los servidores aprender sobre tí
 - Tu nombre y el correo electrónico pueden ser suministrados a los sitios



```
GET /jsp/portalgv.jsp HTTP/1.1
```

```
Host: www.gva.es
```

```
User-Agent: Mozilla/5.0 ...
```

```
...
```

```
HTTP/1.1 200 OK
```

```
Date: Mon, 20 Oct 2008 21:09:58 GMT
```

```
Server: Apache/2.0.53 (Unix) mod_webapp/1.2.0-dev
```

```
Content-Type: text/html; charset=ISO-8859-1
```

**Set-Cookie: chsesion=EE79E6672999;Expires=Tue, 20-Oct-2009
21:09:58 GMT;Path=/**

```
...
```

```
GET /jsp/xper.jsp HTTP/1.1
```

```
Host: www.gva.es
```

```
User-Agent: Mozilla/5.0 ...
```

Cookie: chsesion=EE79E6672999

```
...
```

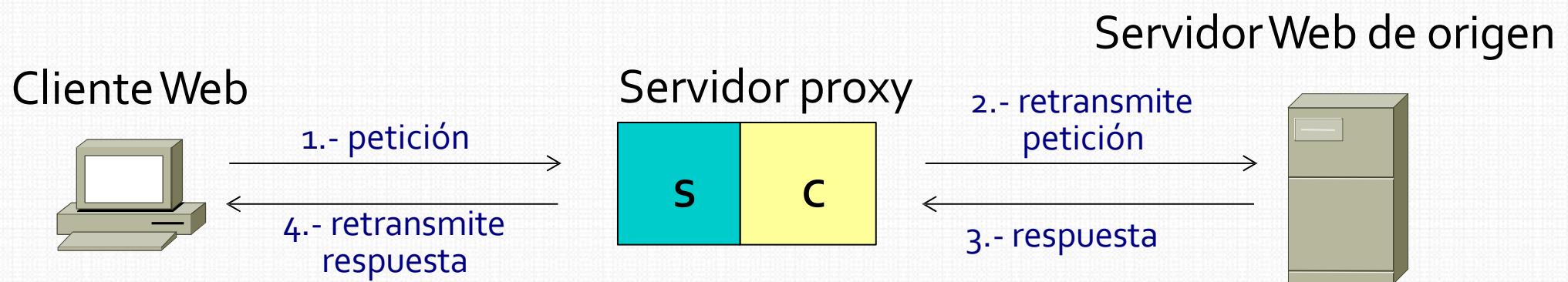


2. La Web y HTTP

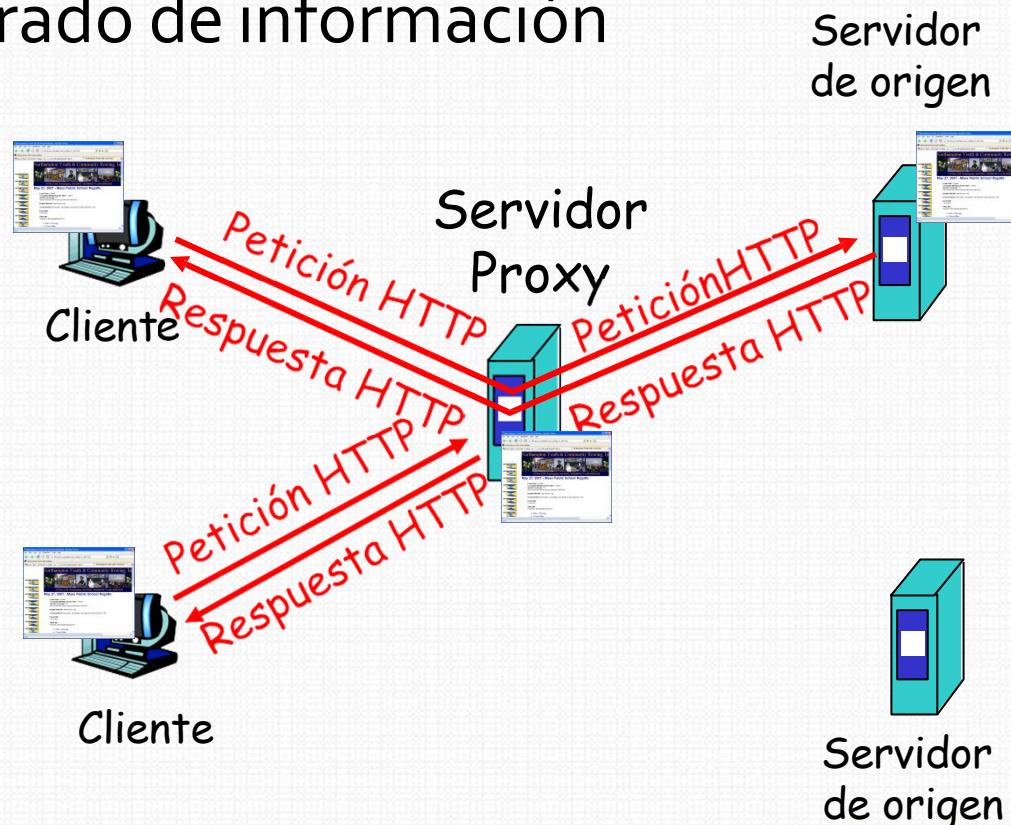
- Generalidades sobre HTTP
- Formato de una transacción HTTP
 - Métodos de petición
 - Cabeceras
 - Mensajes de respuesta HTTP
 - Códigos de estado
- Tipos de conexiones
 - No persistentes
 - Persistentes
- ¿Por qué HTTP/2?
- Interacción usuario-servidor: Cookies
- **Proxy (caché web)**
- Peticiones GET condicionales



- **Objetivo:** satisfacer la petición del cliente sin la participación del servidor de origen
- El navegador envía todas las solicitudes http al Proxy
 - El objeto está en la caché: la caché devuelve el objeto
 - Si no está: la caché pide el objeto al servidor de origen, lo almacena localmente y lo devuelve al cliente
- Actúa tanto como un servidor como un cliente
- Suele instalarla el ISP (universidad, compañía, ISP residencial)



- **Eficiencia:** caché de páginas visitadas. Se reduce el tiempo de respuesta al cliente y se reduce el tránsito.
- **Economía:** compartir conexión externa. Se reduce el tráfico externo de las instituciones
- **Seguridad:** filtrado de información



Conexión directa al servidor

{

GET /index.html HTTP/1.1
Host: www.nla.net
Accept: */*
Connection: Keep-alive

Conexión a través de un *proxy*

{

GET http://www.nla.net/index.html HTTP/1.1
Host: www.nla.net
Accept: */*
Proxy-connection: Keep-alive

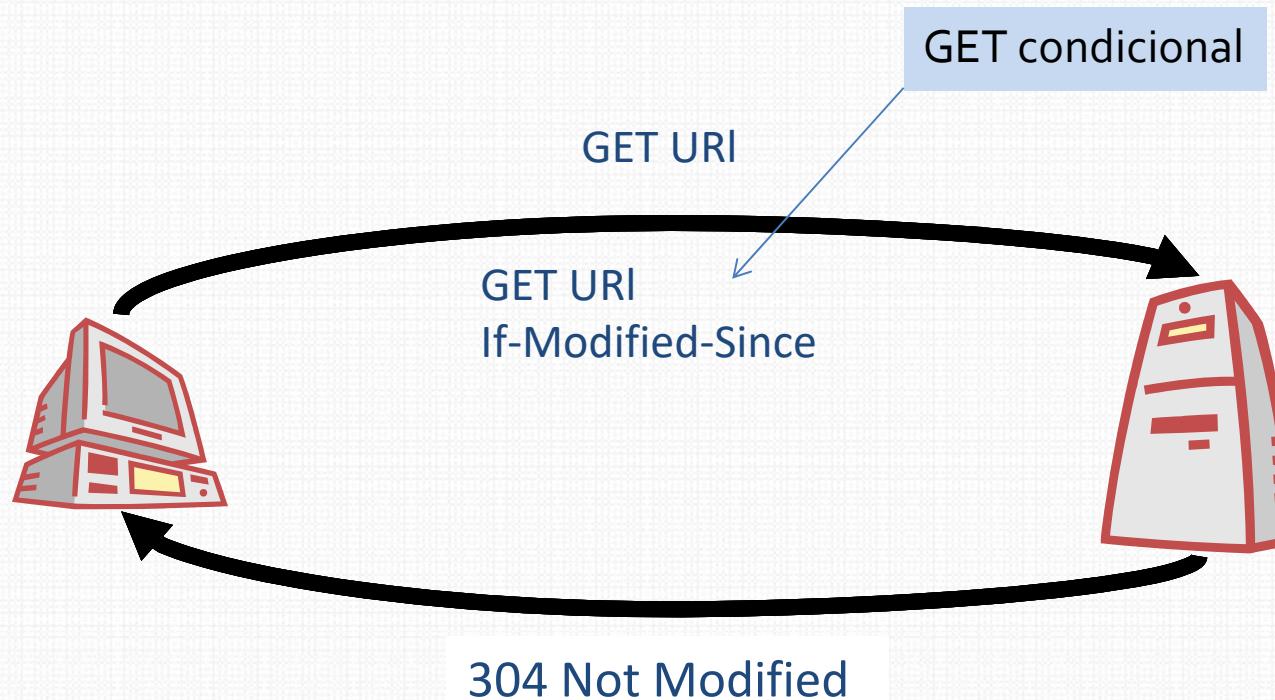


2. La Web y HTTP

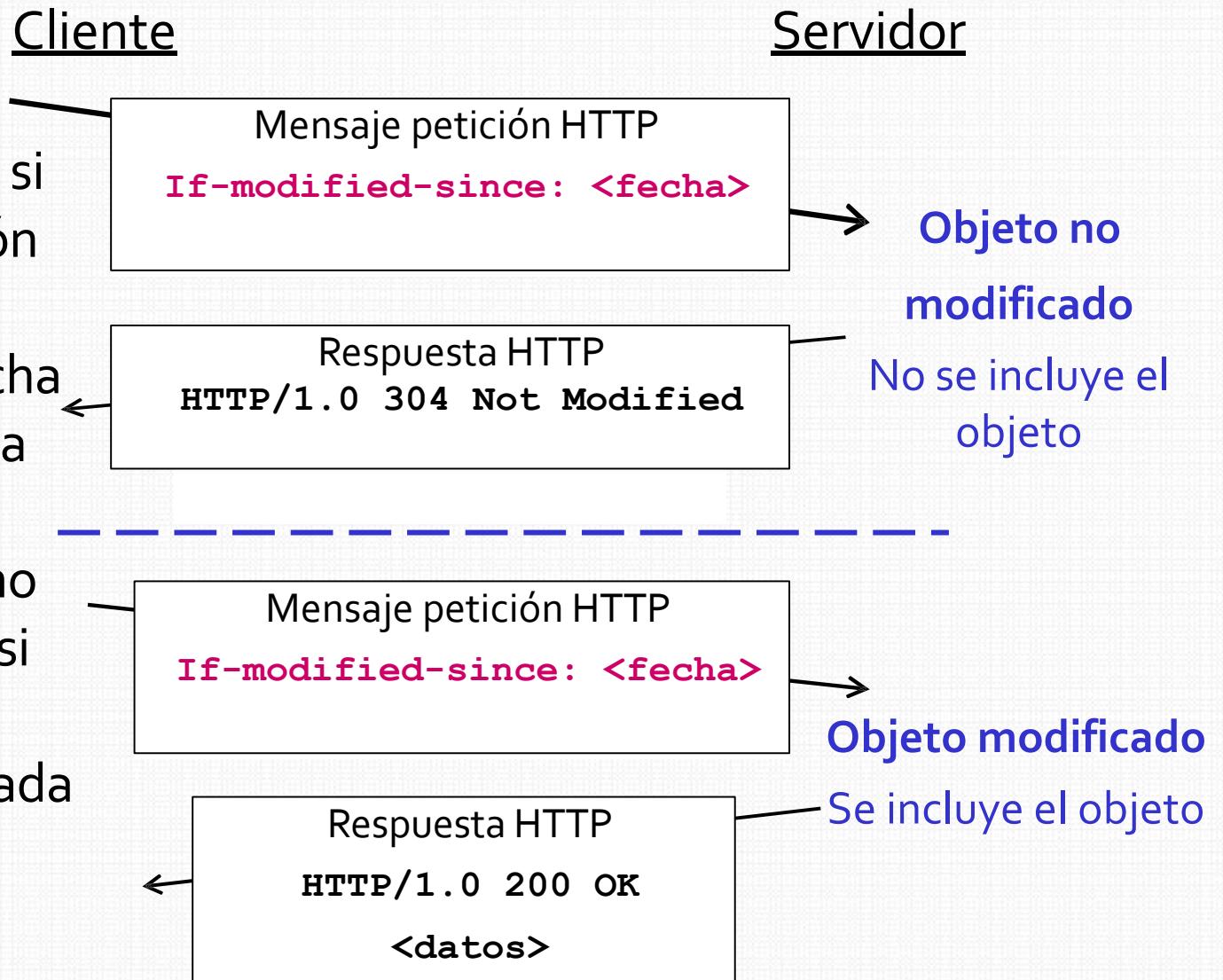
- Generalidades sobre HTTP
- Formato de una transacción HTTP
 - Métodos de petición
 - Cabeceras
 - Mensajes de respuesta HTTP
 - Códigos de estado
- Tipos de conexiones
 - No persistentes
 - Persistentes
- ¿Por qué HTTP/2?
- Interacción usuario-servidor: Cookies
- Proxy (caché web)
- **Peticiones GET condicionales**



- Existen cachés tanto en el cliente como en los proxies.
 - El contenido de las cachés puede no estar actualizado
 - Mecanismo para que la caché verifique que sus objetos están actualizados → GET condicional



- **Objetivo:** no enviar el objeto desde el servidor si la caché tiene una versión puesta al día.
- **Cliente:** especifica la fecha de la copia en caché en la petición HTTP.
- **Servidor:** su respuesta no contiene ningún objeto si no ha sido modificado desde la fecha especificada en la petición.



- Ejemplo cabeceras en la petición:

GET /easy/http/ HTTP/1.1

If-Modified-Since: Wed, 13 Sep 2009 22:51:57 GMT; length=45531

Referer: http://www.google.com/search?q=http+tutorial

User-Agent: Mozilla/4.76 (Windows NT 5.0; U)

Host: www.jmarshall.com

Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg

Accept-Encoding: gzip

Accept-Charset: iso-8859-1,*,utf-8

- Ejemplo cabeceras en la respuesta:

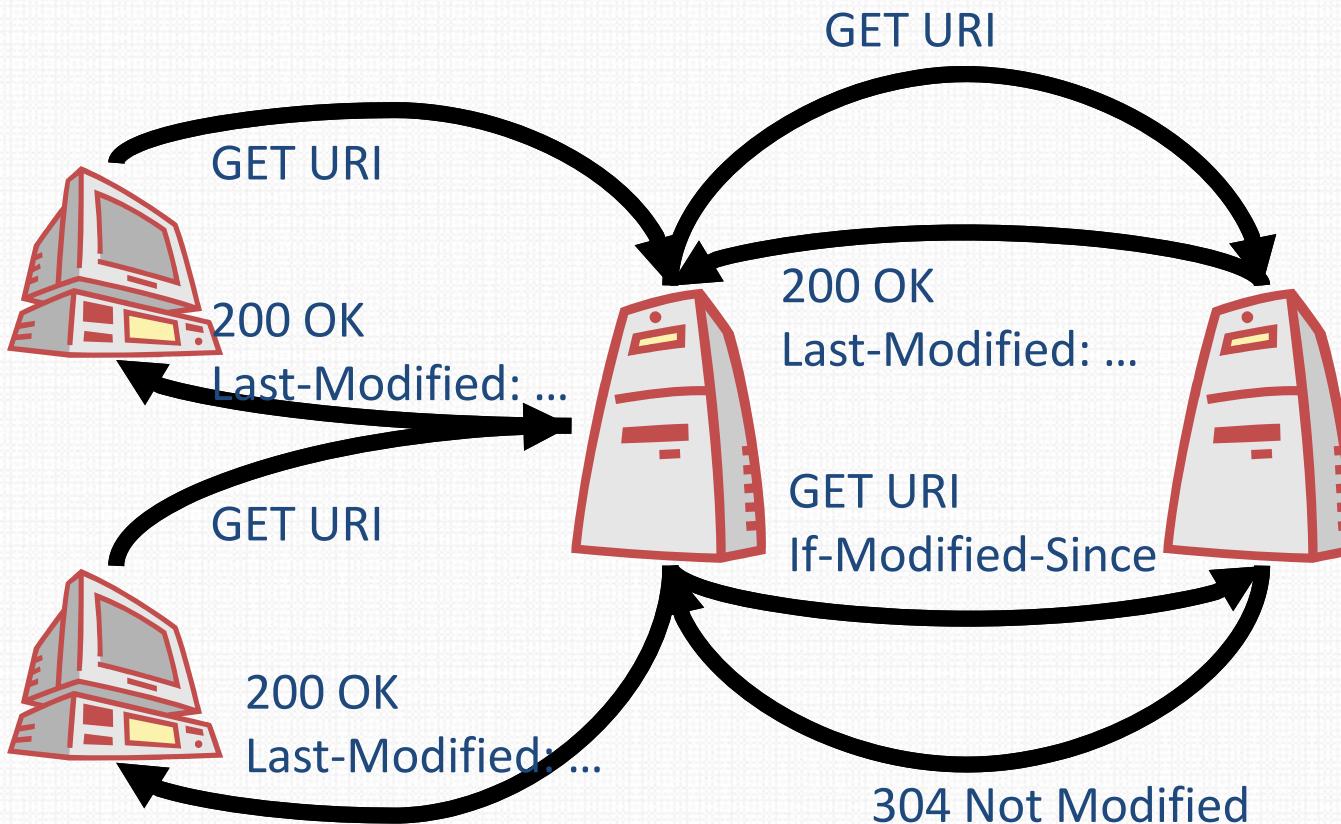
HTTP/1.1 304 Not Modified

Date: Fri, 07 Dec 2009 11:29:00 GMT

Server: mod_jk FrontPage/4.0.4.3 Confluence Apache/1.3.20 (Unix)



- GET condicional empleado tanto por clientes como por servidores proxy

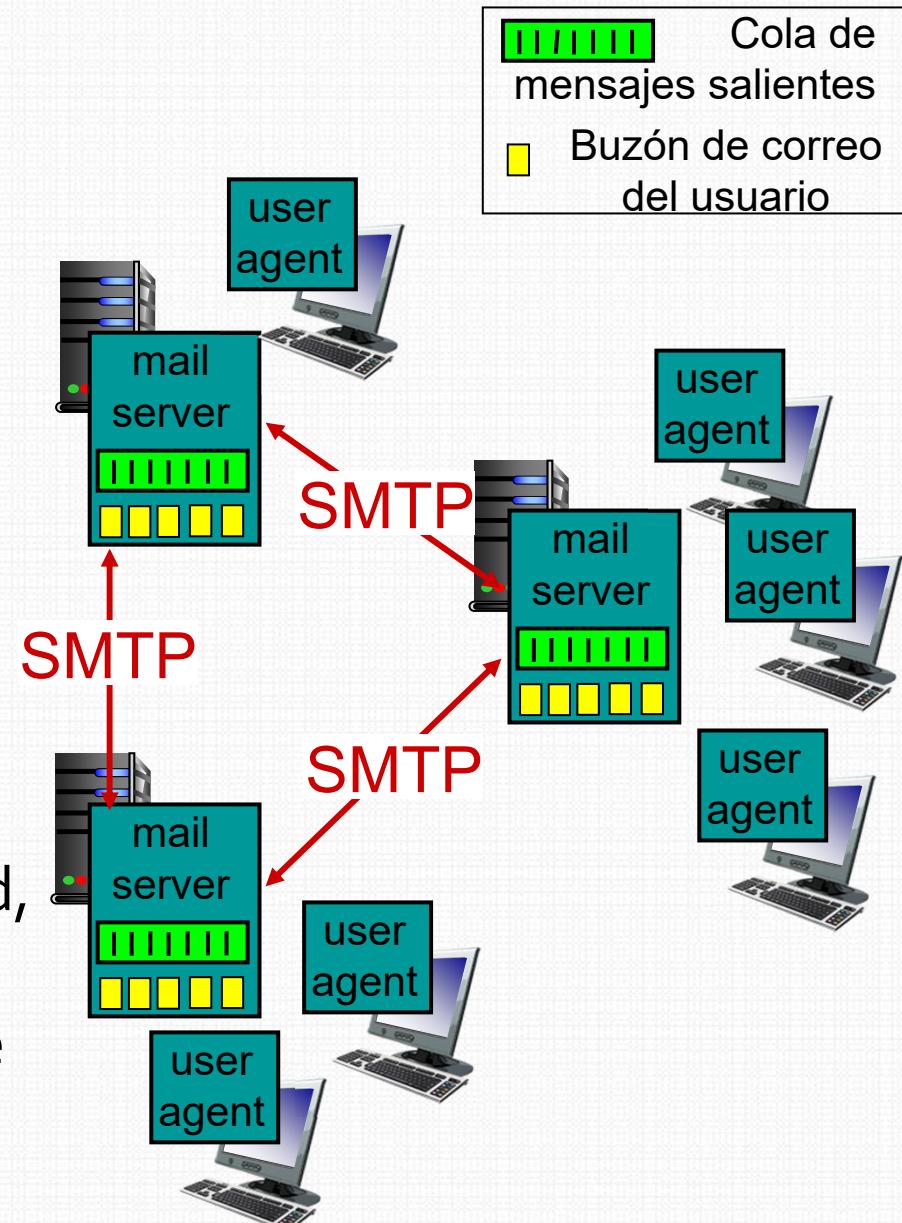


3. El correo electrónico

- Componentes y protocolos implicados
- Esquema de funcionamiento
- Direcciones de correo
- Formato de los mensajes de correo
- Protocolo SMTP
 - Esquema de funcionamiento
 - Comparación con HTTP
 - Órdenes del protocolo
 - Respuestas SMTP
 - Extended SMTP (ESTMP)
- Protocolos de acceso al correo
 - Post Office Protocol (POP3)
 - Internet Message Access Protocol (IMAP)
- Multipurpose Internet Mail Extensions (MIME)
 - Necesidad y funcionamiento
 - Cabeceras relacionadas
 - Tipos de contenidos MIME

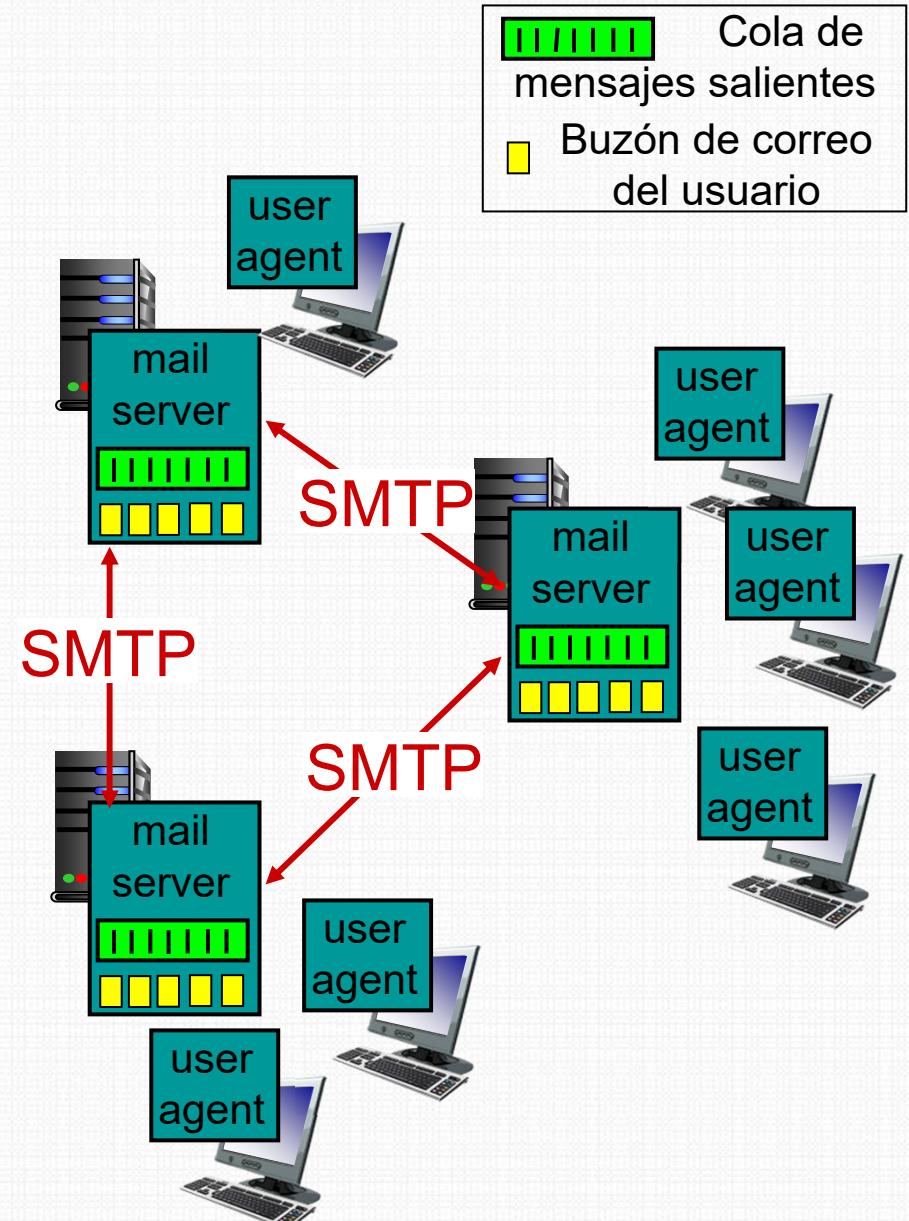


- Servicio asíncrono
- **Tres componentes principales:**
 - Cliente de correo
 - Servidores de correo
 - Protocolo de transferencia simple de correo: SMTP
- **Cliente de correo** (o agente de usuario)
 - “Lector de correo”
 - Compone, edita y lee los mensajes de correo
 - Ej.: Outlook, elm, Mozilla Thunderbird, iPhone mail client,...
 - Recupera los mensajes entrantes que se almacenan en el servidor

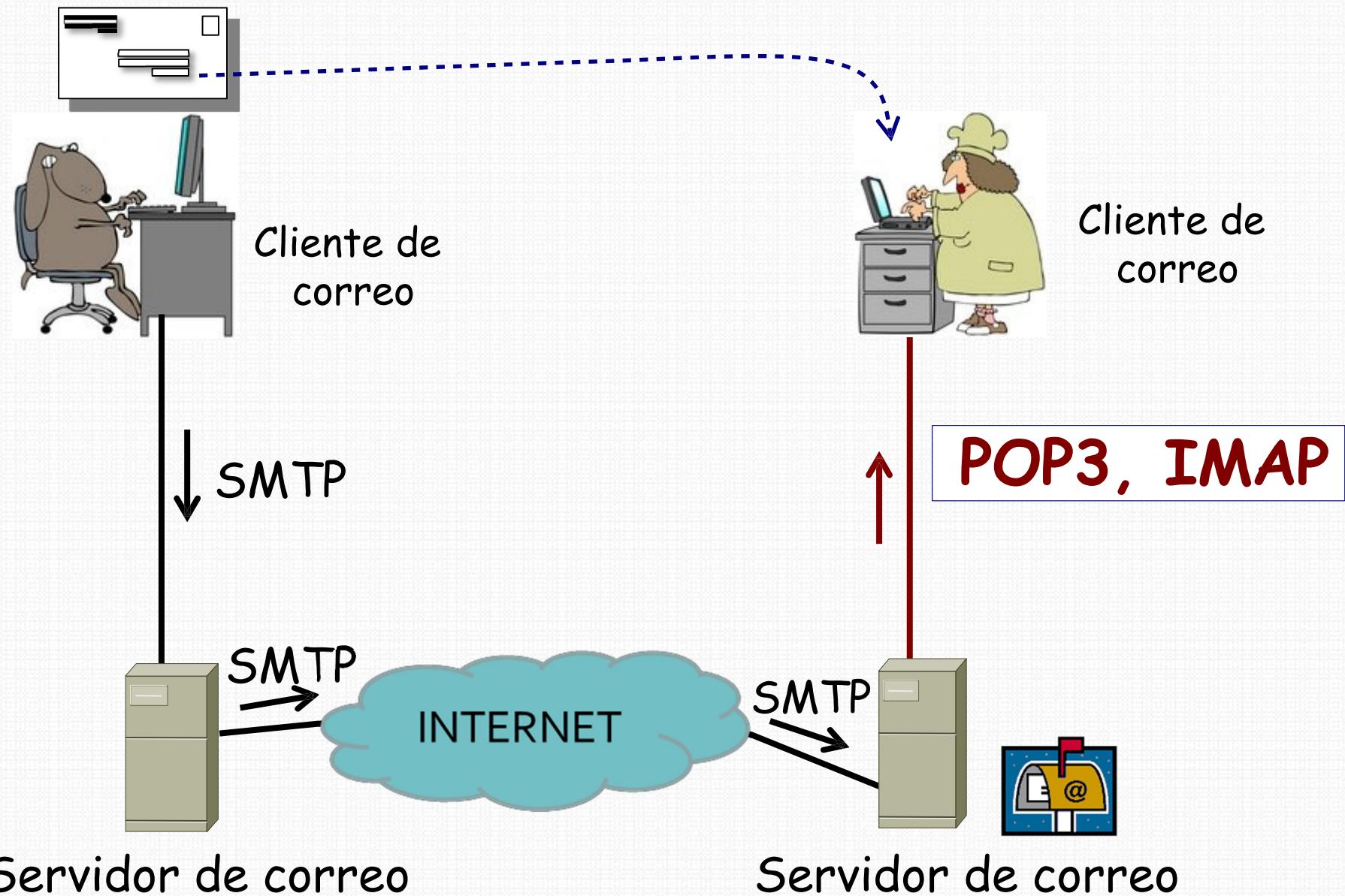


Servidor de correo

- El buzón de correo (mailbox) contiene los mensajes entrantes para el usuario
- Cola de mensajes salientes
- Protocolo SMTP: se utiliza para enviar mensajes entre servidores de correo
 - Cliente: “servidor” que envía correo
 - Servidor: “servidor” que recibe correo



Esquema de funcionamiento



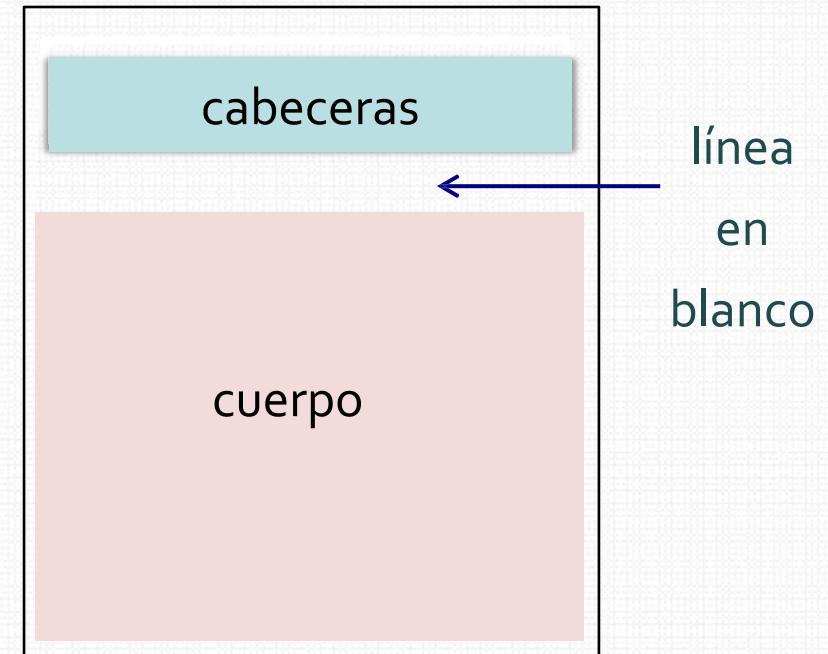
- Identifican los buzones de correo de los usuarios
- Tienen el formato:

buzón@dominio_de_correo

- Normalmente el **buzón** es un nombre de usuario del sistema destino
- Se pueden crear alias
 - profes@redes.upv.es = {joan, pepe, ana}@redes.upv.es
- Formato de **dominio_de_correo**: nombres separados por puntos
 - Puede ser el nombre de un computador
 - Ejemplo: zoltar.redes.upv.es
 - Pero no necesariamente
 - Ejemplos: disca.upv.es, gmail.com
 - ¿Dónde está el buzón de un dominio en este caso?



- Dos partes :
 - Cabeceras: siguen un estándar
 - Destinatario:
 - Origen:
 - Asunto:
 - Cuerpo: texto arbitrario (opcional)
- Cabeceras y cuerpo codificados en ASCII 7 bits y separadas por una línea en blanco (<CR><LF>)



Líneas de cabecera:

To:	Destinatarios principales
Cc:	Destinatarios secundarios
Bcc:	Copias ocultas
From:	Remitente
Received:	Cada agente que retransmitió el mensaje
Subject:	Título del mensaje
Date:	Fecha y hora de envío
Message-Id:	Identificador del mensaje
In-Reply-To:	Identificador del mensaje que provocó la respuesta

- Cada línea de cabecera debe terminar con la secuencia de caracteres <CR><LF>
- Los usuarios pueden inventar nuevas cabeceras que empiecen por **X-**
 - **X-mailer: Pegasus Mail por Windows (v2.23)**



cabecera

Return-Path: <pepe@disca.upv.es>
Received: from vega.upv.es by disca.upv.es (SMI-8.6/SMI-SVR4) id CAA03766; Wed, 24 Jan 2010 02:13:22 GMT
Received: from mailman.endymion.com (antares.cc.upv.es [158.42.3.1])
by vega.upv.es (8.8.4/8.8.5) with SMTP id BAA00996
for <juan@disca.upv.es>; Wed, 24 Jan 2010 01:56:46 +0100 (MET)
From: pepe@disca.upv.es
Message-Id: <201001240056.BAA00996@vega.upv.es>
To: juan@disca.upv.es
Subject: leeme
Date: Wed, 24 Jan 2010 01:56:46 +0000

Hola,
Esto es una prueba.

Línea en blanco



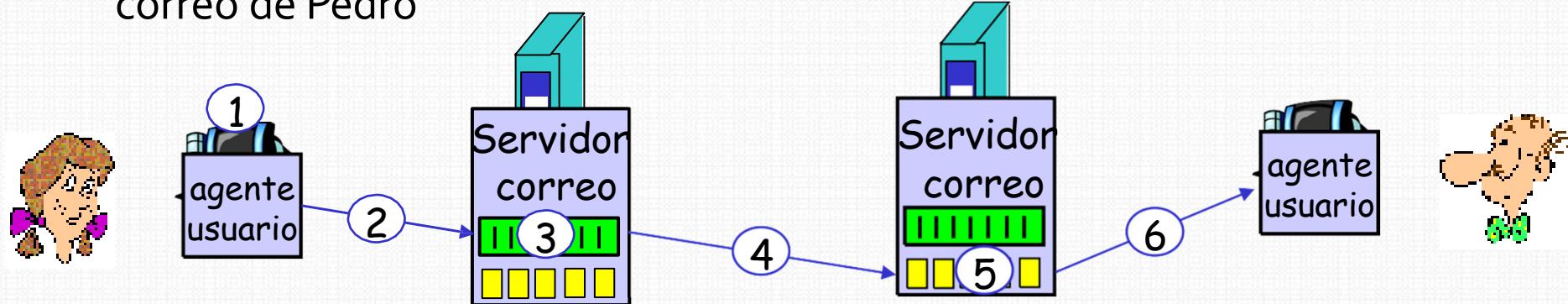
■ Simple Mail Transfer Protocol (RFC 5321)

- Protocolo de transmisión de correo electrónico
- Usa **TCP (conexiones persistentes)** para transferir mensajes de correo electrónico de forma **fiable** desde el cliente al servidor (puerto **25**)
- Tres fases en la transferencia:
 - Establecimiento (saludo)
 - Transferencia de mensajes
 - Cierre de conexión
- Modelo petición/respuesta
- **Los mensajes deben codificarse en ASCII de 7 bits**



- Escenario: Ana envía un mensaje a Pedro

- 1) Ana utiliza su agente de usuario para componer el mensaje y mandarlo a `pedro@etsinf.upv.es`
- 2) El agente de usuario de Ana envía un mensaje a su servidor de correo; el mensaje se coloca en la cola de mensajes
- 3) La parte **cliente de SMTP** abre conexión TCP con el servidor de correo de Pedro
- 4) El cliente de SMTP manda el mensaje de Ana a través de la conexión TCP
- 5) El servidor de correo de Pedro coloca el mensaje en el buzón de correo de Pedro
- 6) Pedro invoca su agente de usuario para leer el mensaje



- Órdenes y respuestas en formato ASCII 7-bits
- Cada orden en una línea
- Conjunto de órdenes (**cliente**):

HELO nombre de dominio <CR><LF>	Identifica origen de la conexión
MAIL FROM: <dirección origen> <CR><LF>	Remitente
RCPT TO: <dirección destino> <CR><LF>	Destinatario
DATA <CR><LF>	Introducción de datos
QUIT <CR><LF>	Cierra la conexión con servidor SMTP
RSET <CR><LF>	Aborta la conexión con servidor SMTP
HELP <CR><LF>	Muestra ayuda sobre órdenes aceptadas

- El mensaje (cabeceras + cuerpo) se transfiere con la orden DATA
- **No confundir las cabeceras “From:” y “To:” de los mensajes de correo con las órdenes MAIL FROM y RCPT TO**



- Formato de las respuestas del **servidor**:
 - Tres dígitos, espacio en blanco y texto con información adicional acabado en <CR><LF>
Ejemplo: 250 OK
 - Suele ir en una línea de texto, pero algunas respuestas son multilínea
- Procesamiento de la orden DATA
 - El servidor envía un mensaje al cliente para que comience el envío del mensaje
 - El cliente indica el fin del mensaje mediante la secuencia <CR><LF>.<CR><LF>



Fase de
saludo

S: 220 smtp.sudominio.com ESMTP Sendmail

C: HELO miequipo.midominio.com

S: 250 Hello, please to meet you

C: MAIL FROM: <yo@midominio.com>

S: 250 Ok

C: RCPT TO: <destinatario@sudominio.com>

S: 250 Ok

C: DATA

S: 354 End data with <CR><LF>.<CR><LF>

C: Subject: Campo de asunto

C: From: yo@midominio.com

C: To: destinatario@sudominio.com

C:

C: Hola,

C: Esto es una prueba.

C: Hasta luego.

C:

C: .

S: 250 Ok: queued as 12345

C: QUIT

S: 221 Bye

Fase de
transferencia

Fase de
cierre

SMTP no requiere
autentificación

Fin del mensaje mediante
la secuencia
<CR><LF>.<CR><LF>

SMTP usa conexiones persistentes.
Si hubiera más mensajes para ese
servidor de correo, se empezaría de
nuevo con la orden MAIL FROM:...

- ESMTP = Extended SMTP
 - Añade nuevas características a SMTP
 - El cliente inicia la sesión con **EHLO** (en vez de **HELO**)
- Si el servidor es ESMTP, responde con código **250** y la lista de extensiones ESMTP que soporta

```
220 vega.upv.es ESMTP Sendmail ; Fri, 5 Feb 2010 20:32:22 +0100 (MET)
```

```
EHLO jaume.disca.upv.es
```

```
250-vega.upv.es Hello jaume.disca.upv.es [158.42.53.234], pleased to meet you
```

```
250-8BITMIME
```

```
250-SIZE 5000000
```

```
250-DSN
```

```
250 HELP
```



■ Comparación de SMTP con HTTP:

- Ambos tienen interacción mediante órdenes/resuestas ASCII, códigos de estado
- Ambos emplean TCP
- SMTP: mensaje, incluyendo el cuerpo, en formato ASCII de 7 bits.
- HTTP: cuerpo del mensaje formado por cualquier tipo de texto o datos binarios.
- HTTP: **protocolo pull** (de extracción)
- SMTP: **protocolo push** (de inserción)
- HTTP: cada objeto es encapsulado en su propio mensaje de respuesta.
- SMTP: múltiples objetos enviados en un único mensaje

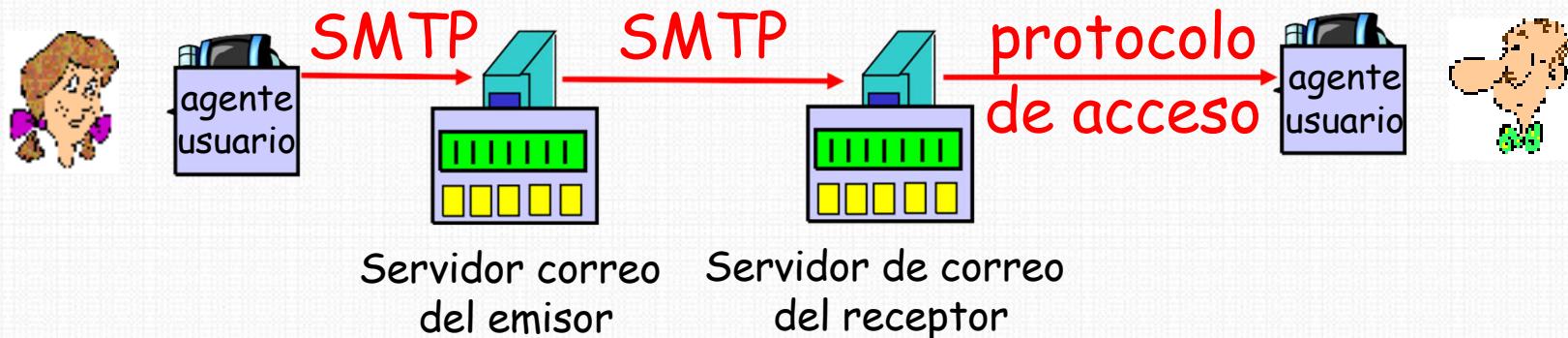


SMTP:

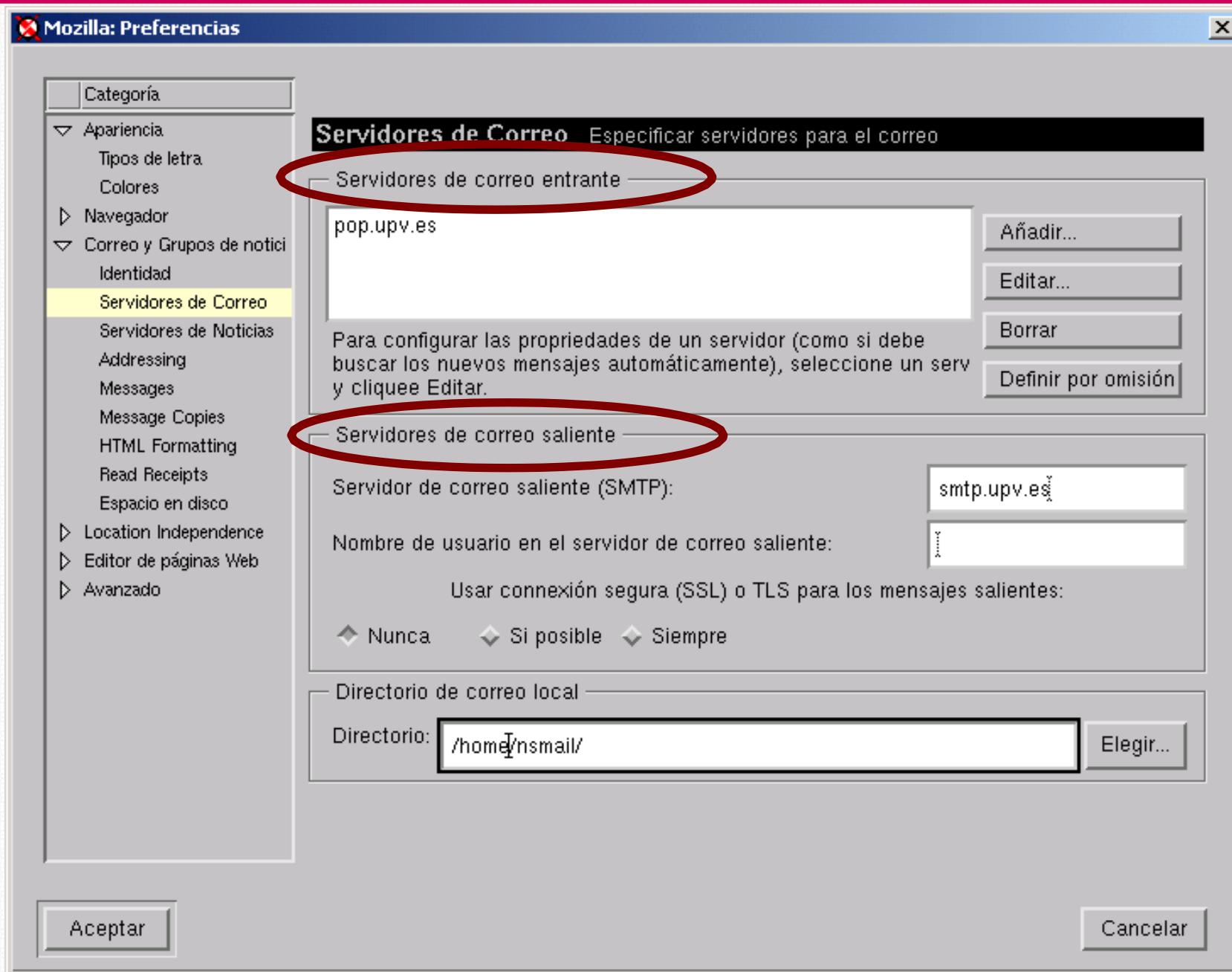
- entrega / almacenamiento hasta el servidor destino

 Protocolos de acceso al correo en el servidor

- Recuperación desde el servidor
 - **POP**: Post Office Protocol [RFC 1939]: autorización, descarga.
 - **IMAP**: Internet Mail Access Protocol [RFC 3501]: más facilidades, incluyendo manipulación de los mensajes en el buzón.

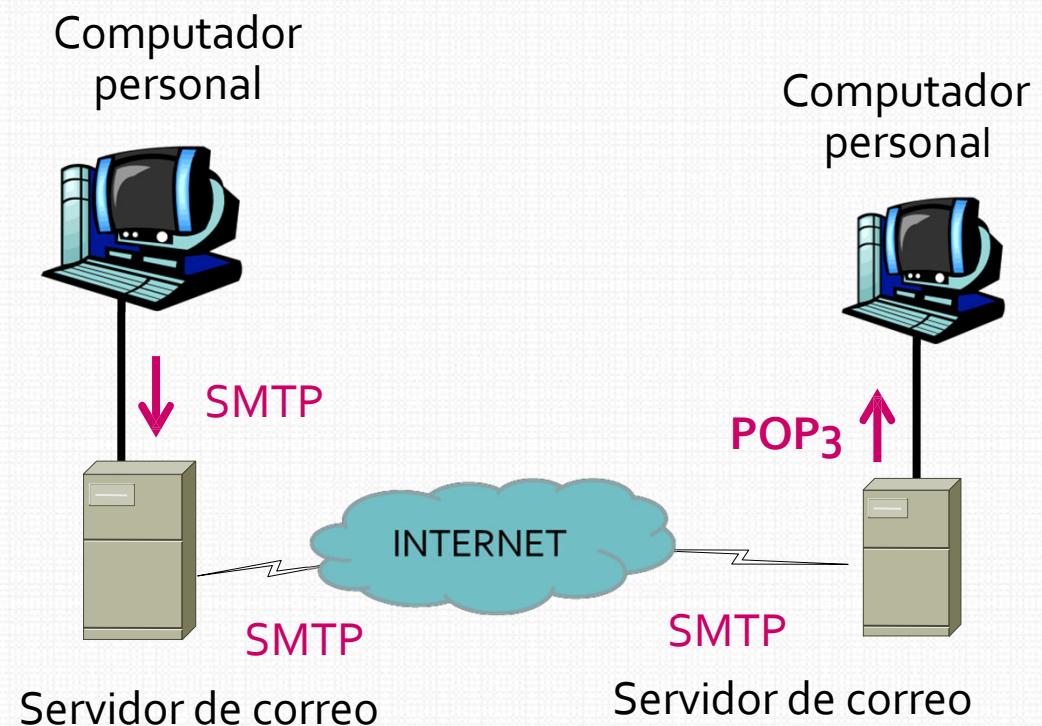


Configuración de un cliente



POP3: Post Office Protocol

- Permite al usuario consultar (leer) su correo previamente recibido y almacenado en su buzón
- Utiliza una conexión TCP al puerto **110** del servidor de correo
- Protocolo simple de funcionalidad limitada
- **No sirve para enviar mensajes**



- Autorización
 - Identificación del usuario
- Transferencia o transacción
 - Manipulación del contenido del buzón del usuario
- Actualización
 - Todas las modificaciones se realizan cuando el cliente finaliza el servicio



■ Órdenes del cliente

USER usuario <CR><LF>	Identificación usuario y contraseña
PASS password <CR><LF>	
STAT <CR><LF>	Devuelve núm. mensajes y tamaño en bytes
LIST <CR><LF>	Lista mensajes en el buzón y sus tamaños
RETR nº mens <CR><LF>	Solicita el envío del mensaje especificando el nº. No se borra del buzón
DELE nº mens <CR><LF>	Marca el mensaje (nº) para borrar (al salir)
QUIT <CR><LF>	Termina. Borra todos los mensajes marcados
RSET <CR><LF>	Abandona. No se borran los mensajes marcados

■ Orden opcional

TOP nº mens líneas <CR><LF>

Lista las cabecera de los mensajes más las líneas del cuerpo que se indican



■ Respuestas del **servidor**:

- +OK o -ERR, un espacio en blanco y un texto con información adicional (indica el resultado de la orden)
- Suelen ir en una línea de texto
- En las **respuestas multilínea** se utiliza la secuencia
`<CR><LF>. <CR><LF>`
para indicar el final de la respuesta

1. Fase de autorización

- **USER, PASS**

S: +OK POP3 server ready

C: USER bob

S: +OK

C: PASS hungry

S: +OK user successfully logged on



2. Fase de transacción

C: LIST

S: 1 498

S: 2 912

S: .

C: RETR 1

S: <message 1 contents>

S: .

C: DELE 1

C: RETR 2

S: <message 2 contents>

S: .

C: DELE 2

C: QUIT

S: +OK POP3 server signing off

Respuesta
multilínea



3. Fase de actualización

- Tras recibir el **QUIT**



C: RETR 2

S: <mensaje 2
contenidos>

S: .

RETR 2

+OK 563 octets

Received: from disca.upv.es by disca.upv.es

id LAA01086; Wed, 24 Feb 2010 11:52:55 GMT

Message-ID: <3A8A5F66.BCB70342@disca.upv.es>

Date: Wed, 24 Feb 2010 11:35:19 +0100

From: Ovidi Peix <ovidi@disca.upv.es>

MIME-Version: 1.0

To: pau@disca.upv.es

Subject: leeme

Content-Type: text/plain; charset=us-ascii

Content-Transfer-Encoding: 7bit

Hola,

Te apetece venirte al cine?

Bye



- Modos de funcionamiento (configurable por el usuario)
 - Descargar y borrar
 - No se puede recuperar el correo si se cambia de agente de usuario. ¿Usuarios nómadas?
 - Descargar y guardar
 - Copia los mensajes de correo sobre agentes de usuario diferentes

POP3

- POP3 no guarda el estado del cliente entre sesiones
- El usuario no puede organizar un sistema de carpetas en el servidor

IMAP (RFC 3501)

- *Internet Message Access Protocol*
- Más complejo que POP3
- Mantiene los mensajes en el servidor
- Permite al usuario organizar los mensajes en carpetas
- IMAP guarda el estado del cliente entre sesiones:
 - Nombres de las carpetas y qué mensaje está en qué carpeta
 - Utiliza conexiones TCP en el puerto 143



- Limitaciones de correo electrónico
 - El cuerpo del mensaje está en ASCII 7-Bits (EEUU)
 - ¿Y el texto con acentos?
 - ¿Y los mensajes en idiomas sin alfabeto (chino, japonés,...)?
 - ¿Y los cuerpos no textuales (audio, vídeo, imágenes)?
 - ¿Y los cuerpos con múltiples partes?

- **MIME**: extensiones para correo multimedia (RFC 2045-2049)
 - *Multipurpose Internet Mail Extensions*
- Las extensiones MIME van encaminadas a soportar:
 - Texto en conjuntos de caracteres no ASCII-7 en el cuerpo del mensaje (texto en distintos idiomas y alfabetos)
 - Adjuntos con información binaria (no ASCII-7)
 - Cuerpos de mensajes con múltiples partes (multi-part)
- Los datos no-ASCII a enviar (texto no ASCII-7-Bits, imágenes, vídeos) se codifican en ASCII-7-Bits
 - **Codificación Base64**: mapea grupos de 3 bytes en grupos de 4 caracteres ASCII 7-Bits



24 bits se codifican como **4 caracteres de 6 bits** según la tabla (ASCII)

00010000 10010100 01110010



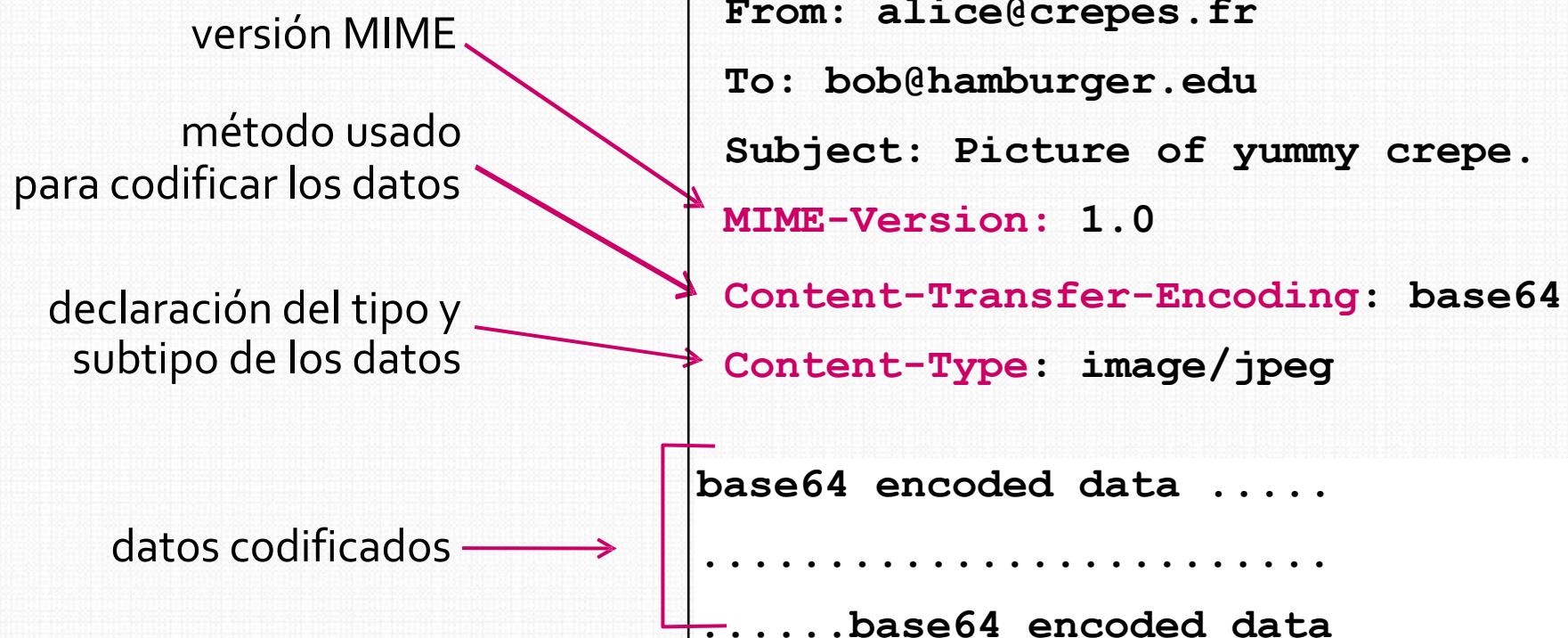
00010000 010001 110010
 ↓ ↓ ↓
 4 9 17 50
 E J R y



EJ Ry

0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w		
15	P	32	g	49	x		
16	Q	33	h	50	y		

- Para declarar el tipo MIME del contenido se incluyen líneas adicionales específicas en la cabecera
- Se pueden enviar mensajes con diversos contenidos en formatos diferentes



Content-type: tipo/subtipo; parámetros

- **Text**
 - Subtipos: **plain, html**
- **Image**
 - Subtipos: **jpeg, gif**
- **Audio**
 - Subtipos: **basic, mpeg, mp4,...**
- **Video**
 - Subtipos: **mpeg, quicktime,...**
- **Application**
 - Otros tipos de datos que deben ser procesados por el lector antes de ser visibles
 - Subtipos: **msword, octet-stream**
- **Multipart**
 - Mensajes con varios objetos
 - Subtipos: **mixed, alternative**

From: "Roser Peix" <roser@upvnet.upv.es>

To: profes_eui@upv.es

MIME-Version: 1.0

Content-type: Multipart/Mixed; boundary=Message-Boundary-15761

Subject: datos

--Message-Boundary-15761

Content-type: text/plain; charset=ISO-8859-1

Content-transfer-encoding: Quoted-printable

Adjunto os remitimos los datos en el fichero enero.pdf

Un saludo, Roser

Define etiqueta
de “frontera”

Cada parte con su propio
encabezado de contenido

--Message-Boundary-15761

Content-type: Application/Octet-stream; name=enero.pdf; type=Unknown

Content-transfer-encoding: BASE64

JVBERi0xLjlgDQoI4uPP0w0KIA0KOCAwIG9iag0KPDwNCi9MZW5ndGggOSAwIFINCi9GaWx0ZXIg
L0ZsYXRIRGVjb2RIIA0KPj4NCnN0cmVhbQ0KSInMI19vpDgWxT9BfQdLq5F6pYTBfzDm0QWuChk
KaEPVtFr9tppdaR9G6n7Zr7/XYBuoogKVriStlqJWI

--Message-Boundary-15761 --



4. Servicio de directorio de Internet (DNS)

- Componentes
- Sintaxis de los nombres
- Dominios de primer nivel (Top Level Domain, TLD)
- Servicios DNS
- Servidores de nombres DNS
 - Estructura de los servidores DNS
 - Servidores de nombre raíz
 - Servidores TLD y servidores autoritativos
 - Servidor de nombres local
- Resolución de nombres de dominio
 - Consulta iterativa
 - Consulta recursiva
- Caché de nombres
- Registros DNS
- Formato de los mensajes DNS
- Registrar un dominio



5. Servicio de nombres de dominio



NIF: 23.789.065-M

Nombre: Cosme García



Dir. IP: 158.42.4.2

Nombre: www.upv.es

¿Traducción entre direcciones IP y nombres?

- Resolución de nombres de dominio o de direcciones
- Tarea principal del sistema de nombres de dominio (DNS)

Domain Name System (RFCs 1034 y 1035) incluye:

- Una **sintaxis** para los nombres
- Una **base de datos distribuida** implementada como una jerarquía de **servidores de nombres**
- Función del núcleo de Internet implementada como un **protocolo de nivel de aplicación**
 - Los hosts contactan con los servidores de nombre para “resolver” los nombres (traducción dirección/nombre)
 - Sirve de apoyo a otros protocolos de aplicación

- Los nombres de dominio poseen una estructura jerárquica
- Son una secuencia de etiquetas separadas por puntos

Máximo 255 caracteres

- Ejemplo: **zoltar.redes.upv.es**

Nombre
del
equipo (host)

Subdominios
(hasta 127 niveles)

Dominio de nivel superior o
de primer nivel
(Top level domain)

- El nivel más alto de los nombres se divide en los siguientes dominios (**Top Level Domain o TLD**)

<http://www.iana.org/domains/root/db/>

com	organización comercial
edu	institución educativa USA
net	organización relacionada con la red
org	Otras organizaciones
info	Uso no restringido
...	...



Internet Assigned Numbers Authority

por
organización

es	España
uk	Reino Unido
fr	Francia
...	Otros países

Geográfico
(Códigos de dos
letras ISO3166-1)

Servicios DNS

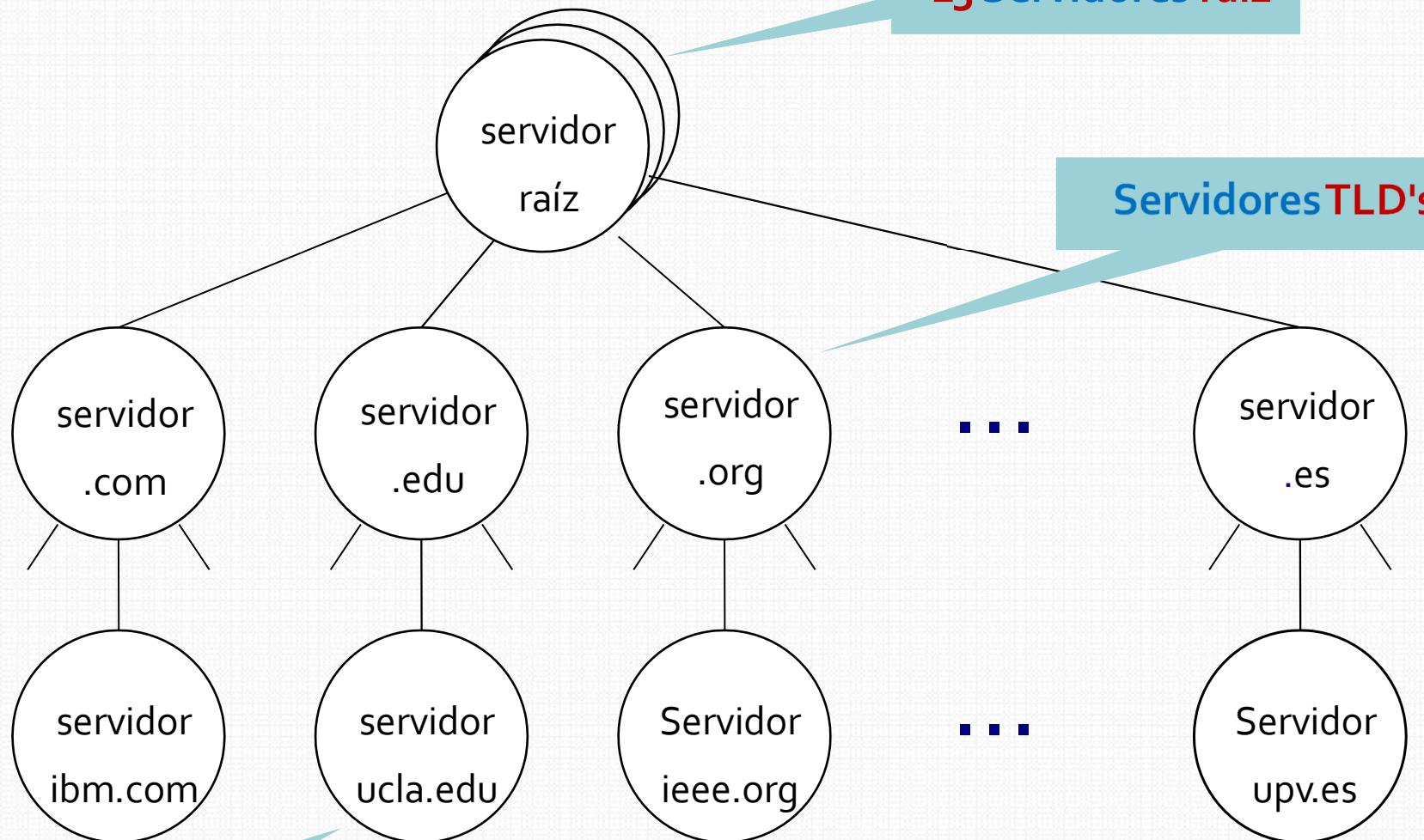
- Traducción de *nombre_de_host* a dirección IP
- Alias de hosts
 - Nombre canónico y alias
- Alias de servidor de correo
 - Registros especiales MX
- Distribución de carga entre servidores replicados (rotación DNS)
 - Servidores Web replicados: un nombre canónico asociado a varias IP



- Para asociar nombres de dominio a direcciones IP se utilizan **servidores de nombres**
- ¿Un DNS centralizado?
 - Punto singular de fallo
 - Volumen de tráfico alto
 - Distante de muchos puntos
 - Mantenimiento complejo
- El sistema DNS utiliza un diseño distribuido:
 - Los servidores DNS están organizados en una **estructura de árbol**
 - Físicamente los servidores están distribuidos alrededor del mundo

*¡No es
escalable!*





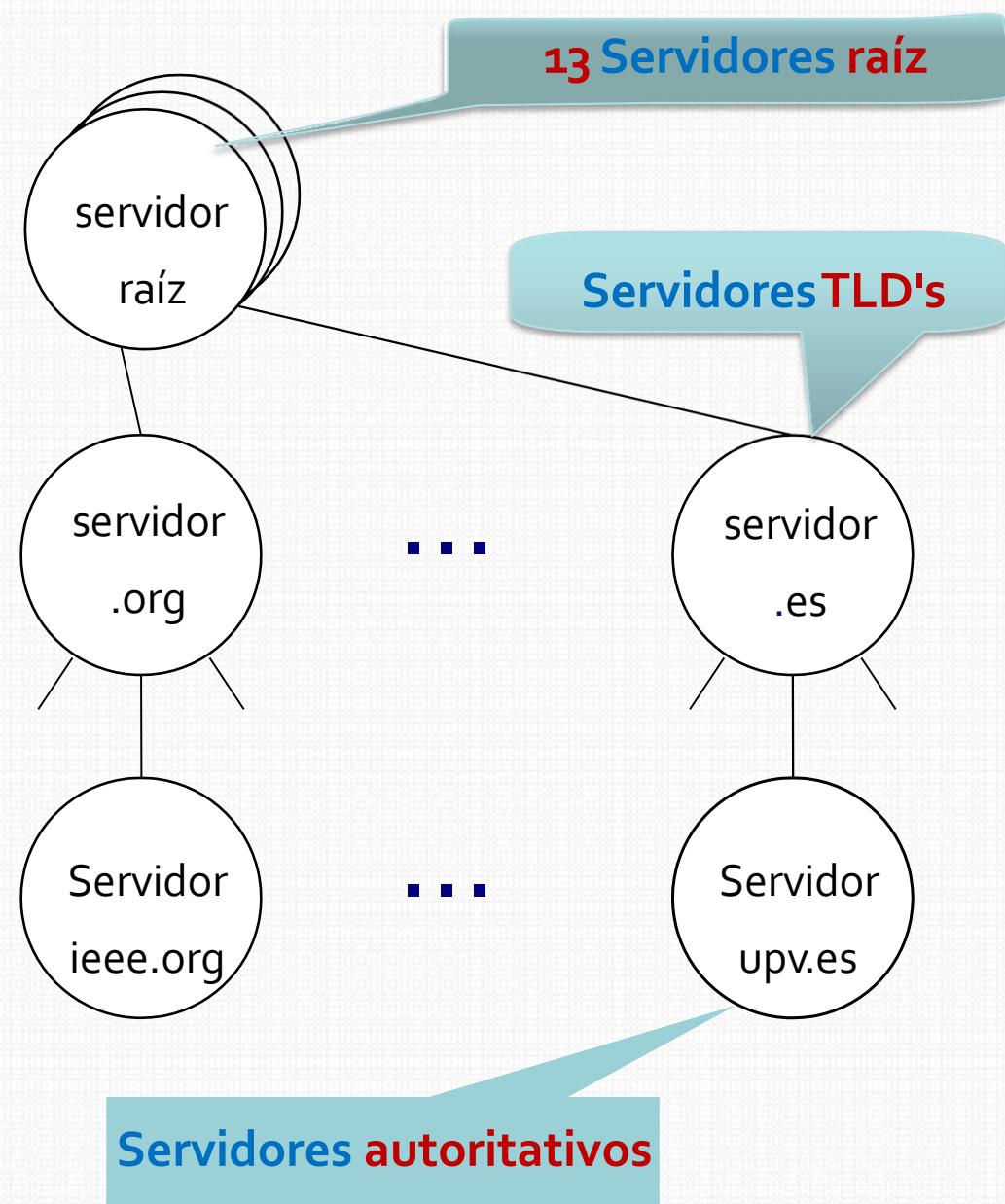
Servidores autoritativos

Un cliente desea la IP de

www.upv.es:

Primera aproximación:

- El cliente contacta con un **servidor raíz** para encontrar el servidor DNS “.es”
- El cliente contacta con el **servidor DNS “.es”** para encontrar el servidor DNS “upv.es”
- El cliente contacta con el **servidor DNS “upv.es”** para encontrar la IP de “www.upv.es”



- Cada servidor local debe conocer, al menos, uno de los servidores raíz
- 13 Servidores de nombre raíz en todo el mundo:

<http://www.root-servers.org/>



- Servidores de dominio de nivel superior: **TLD** (**Top-Level Domain**)
 - Responsables de los dominios com, org, net, edu, aero, jobs, museums y todos los dominios de nivel superior de los diferentes países, ej: uk, es, fr, ca, jp
- Servidores **autoritativos**
 - Servidores DNS de las organizaciones
 - Suministran mapeos de nombres de host a IP
 - Pueden estar mantenidos por la propia organización o por un proveedor de servicios
 - Suelen existir al menos dos (principal y secundario)



■ Servidor DNS **local**

- No pertenece estrictamente a la jerarquía
- Cada ISP (residencial, compañía, universidad, ...) tiene uno
 - Llamado también ***Default name server***
- Cuando un host hace una consulta DNS, la realiza a su servidor DNS local
- Actúa como un Proxy, enviando la consulta a la jerarquía de servidores DNS
- El DNS local mantiene una caché local de pares (nombre/IP)

- El cliente elabora una consulta de nombre de dominio que incluye:
 - El nombre a resolver
 - El tipo de nombre
- Envía la consulta a su **servidor de nombres local**
- La consulta suele hacerse por **UDP**
 - Al puerto 53



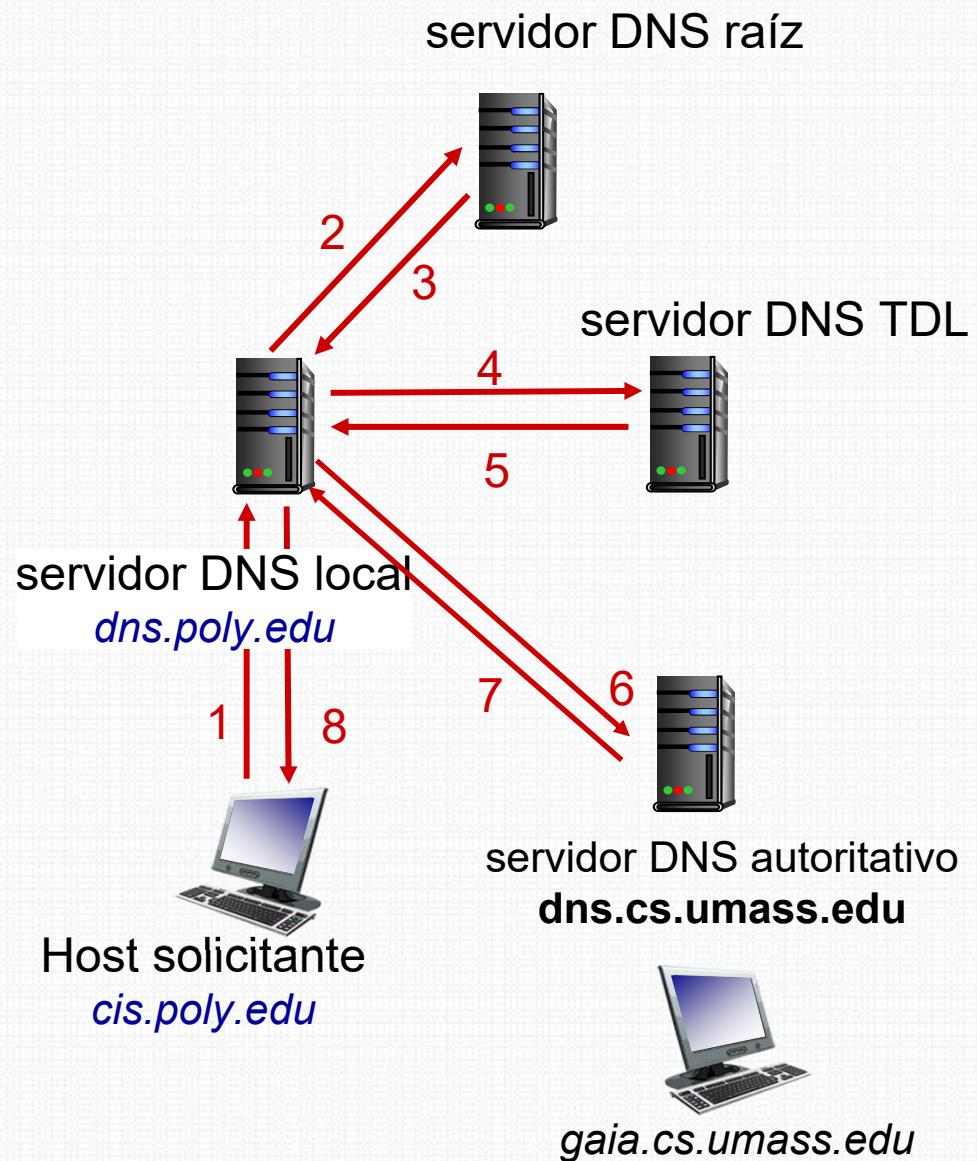
- El **servidor de nombres local** recibe una petición:
 - Si el nombre pertenece a su dominio, traduce el nombre y envía la respuesta al cliente
 - Si no puede resolver el nombre:
 - Contacta con un servidor raíz (actuando ahora como cliente)
 - El servidor raíz le indicará los servidores TLD apropiados
 - El servidor local contacta con un servidor TLD, si éste no puede resolver la consulta le indica el siguiente en jerarquía, y así sucesivamente



■ Consulta iterativa:

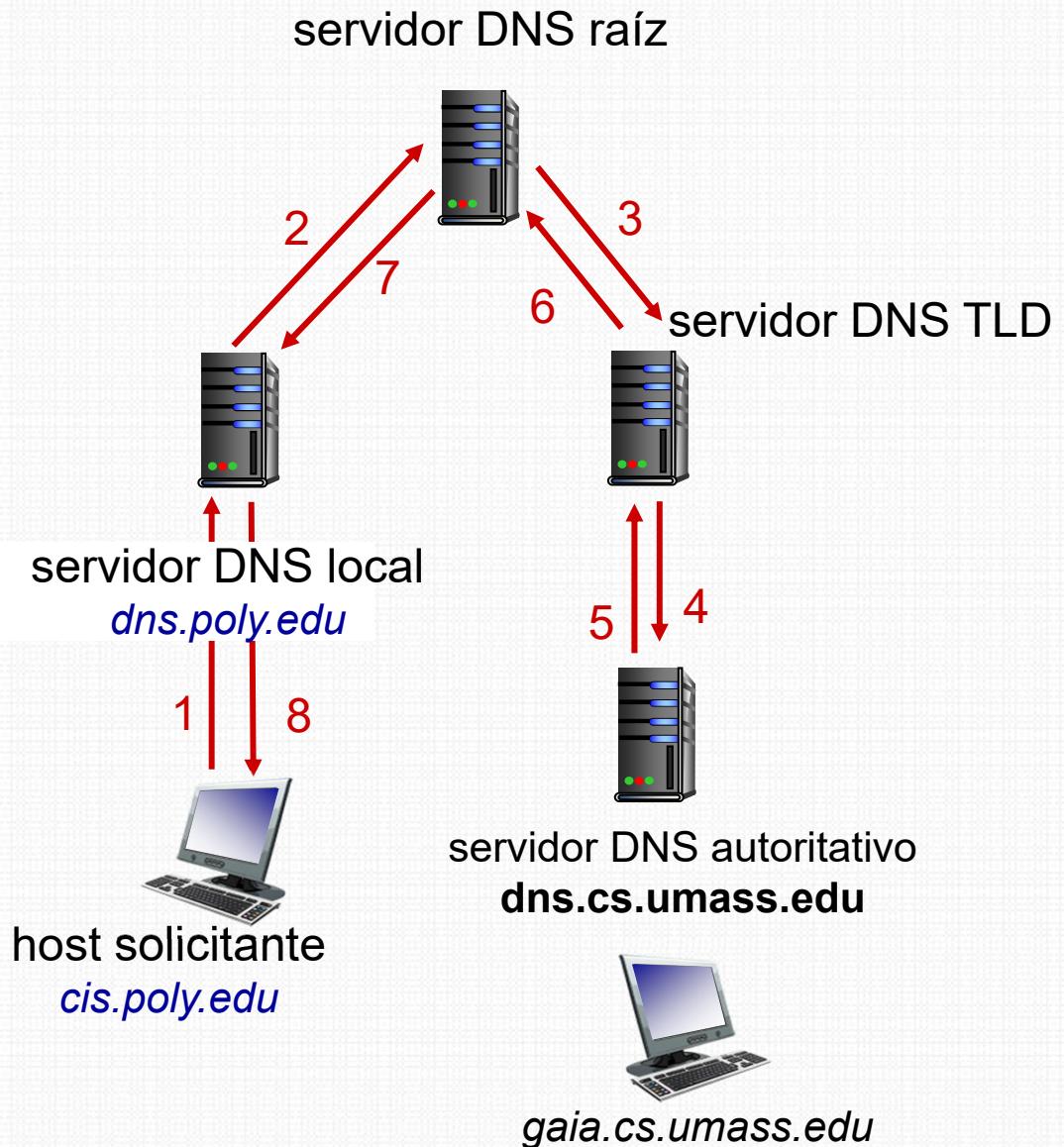
- El servidor contactado devuelve el nombre del servidor a contactar.
- “No conozco ese nombre, pero puedes preguntarle a este servidor”

El host cis.poly.edu solicita la IP de gaia.cs.umass.edu



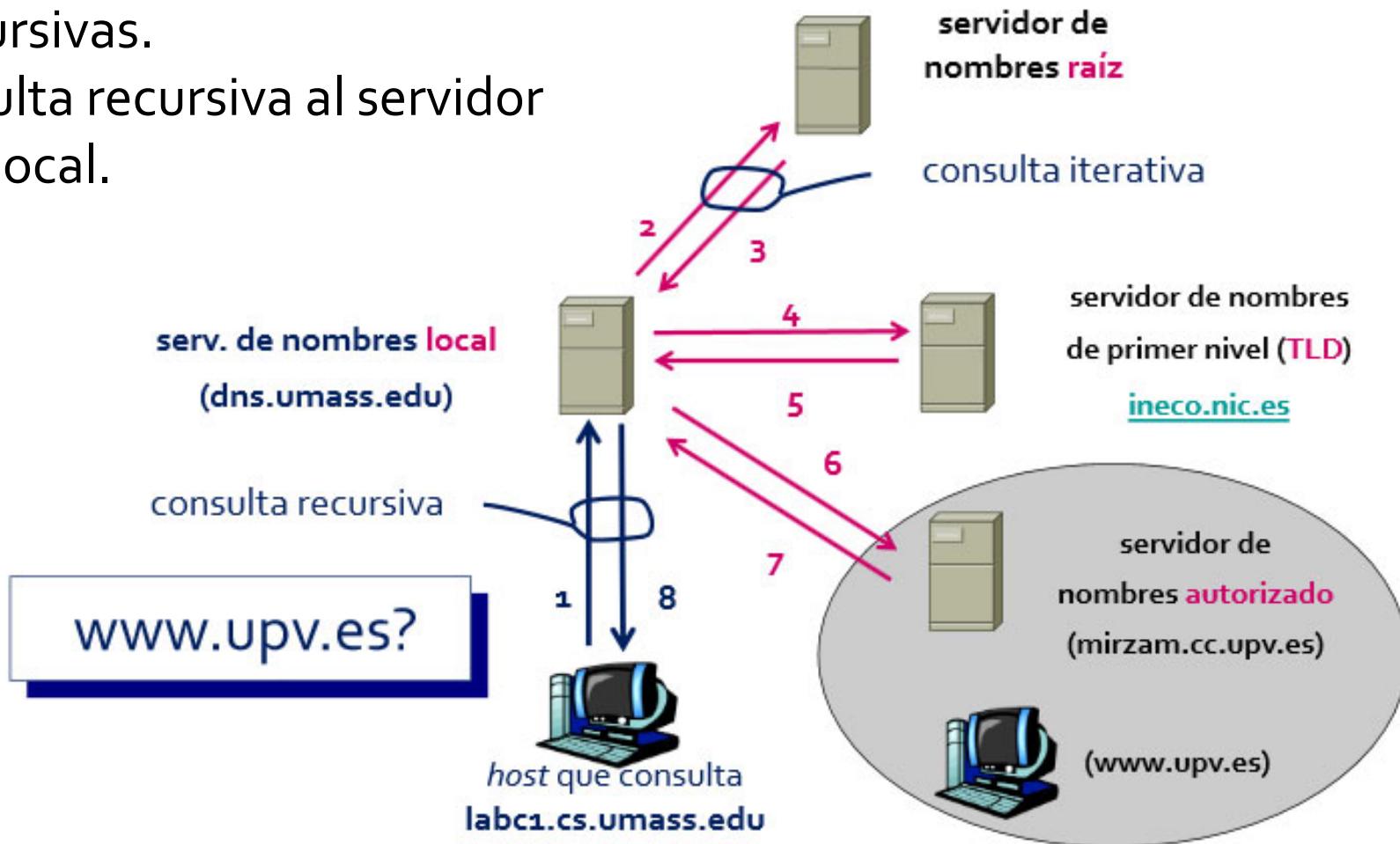
■ Consulta recursiva:

- Pone el peso de la resolución del nombre en el servidor contactado.
- Gran carga en los niveles superiores de la jerarquía.



El host `cis.poly.edu` solicita la IP de `gaia.cs.umass.edu`

- En la práctica, se emplea una mezcla de consultas iterativas y recursivas.
- Consulta recursiva al servidor DNS local.



- Cada servidor de nombres mantiene una caché de los nombres ya resueltos
 - Se registra dónde y cuándo ha conseguido esa información
- Cuando el servidor no puede resolver un nombre busca en su caché
 - Si lo resuelve en caché, envía la respuesta al cliente pero...
 - Le dice que está obtenida de la caché (*non authoritative*)
 - Le indica el nombre y dirección IP del servidor que la proporcionó
 - Si no, pasa la consulta a otro servidor
- Los servidores TLD normalmente están cacheados en los servidores DNS locales
 - Los servidores de nombre raíz no se visitan tan a menudo



- Los mapeos de la caché tienen un *timeout*, desaparecen después de un tiempo (TTL, *Time To Live*)
- Los mapeos de la caché pueden estar caducados
 - Si un nombre de host cambia su IP puede que el cambio no se conozca hasta que todos los TTL expiren
- En la RFC 2136 se proponen mecanismos de update/notify

- Los servidores de nombres almacenan la información en registros de recursos (RR):
 - **Formato RR: (*nombre, valor, tipo, ttl*)**
- **Tipo = A (o AAAA para IPv6) (*Address*)**
 - **Nombre** de un host
 - **Valor:** Dirección IP
- **Tipo = NS (*Name Server*)**
 - **Nombre** de dominio
 - **Valor:** Nombre del servidor de nombres autorizado (autoritativo) para el dominio
- **Tipo = CNAME (*Canonical name*)**
 - **Nombre** es un alias de un nombre canónico (real)
- **Valor:** nombre canónico
 - Ej: www.upv.es es realmente ias.cc.upv.es
- **Tipo = MX (*Mail Exchange*)**
 - **Nombre** de un dominio de correo
 - **Valor:** Nombre canónico del servidor de correo

Al hacer una consulta se indica el tipo de nombre a resolver

- Ejemplos:

(relay1.bar.foo.com, 145.37.93.126, A)

(foo.com, dns.foo.com, NS)

(foo.com, relay1.bar.foo.com, CNAME)

(foo.com, mail.bar.foo.com, MX)



```
ovidi@zoltar:~> host -v www.irisa.fr
```

```
Trying "www.irisa.fr"
```

```
;-->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13519
```

```
; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 1
```

;; QUESTION SECTION:

```
;www.irisa.fr. IN A
```

;; ANSWER SECTION:

```
www.irisa.fr. 6765 IN A 131.254.254.46
```

;; AUTHORITY SECTION:

```
irisa.fr. 72962 IN NS dns2.cso.uiuc.edu.
```

```
irisa.fr. 72962 IN NS presto.irisa.fr.
```

```
irisa.fr. 72962 IN NS dns.irisa.fr.
```

```
irisa.fr. 72962 IN NS dns1.cso.uiuc.edu.
```

;; ADDITIONAL SECTION:

```
presto.irisa.fr. 65752 IN A 131.254.6.128
```

Received 151 bytes from 158.42.249.8#53 in 2 ms

- Averiguar qué máquina atiende el dominio de correo disca.upv.es

nslookup

```
> set type=MX  
> disca.upv.es
```

Menor preferencia =
mayor prioridad

```
disca.upv.es      MX preference = 10, mail exchanger = smtp1.cc.upv.es  
disca.upv.es      MX preference = 15, mail exchanger = smtp2.cc.upv.es  
disca.upv.es      MX preference = 25, mail exchanger = mail.rediris.es  
smtp1.cc.upv.es  internet address = 158.42.250.32  
smtp2.cc.upv.es  internet address = 158.42.250.31
```



- Dos tipos: **consultas y respuestas** con el mismo formato
- **Cabecera** (12 bytes iniciales)
 - **Identificación:** 16 bits para asociar consultas y respuestas
 - **Flags:**
 - QR: Consulta (0) o respuesta (1)
 - AA: Respuesta autoritativa
 - RD: Recursión solicitada (deseada)
 - RA: Recursión disponible



■ Campos:

■ Preguntas:

- Consultas (nombre y tipo de consulta)

■ Respuestas:

- RRs en respuesta a la consulta

■ Autoridad

- Registros de otros servidores autoritativos

■ Información adicional:

- Otros registros útiles
- Ej.: si se preguntaba por MX, puede figurar el registro A adicional.

↔ 2 bytes ↔ ↔ 2 bytes ↔

Identificación	Flags
Nº de preguntas	Nº de resp
Nº regs autoriz.	Nº reg. adicionales
Preguntas (número variable de registros)	
Respuestas (número variable de registros)	
Autoridad (número variable de registros)	
Información adicional (número variable de registros)	

```
redes12@zoltar:~/Escritorio$ dig www.upv.es
; <>> DiG 9.7.0-P1 <>> www.upv.es
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53576
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 4, ADDITIONAL: 6

;; QUESTION SECTION:
;www.upv.es.      IN  A

;; ANSWER SECTION:
www.upv.es. 10800 IN CNAME ias.cc.upv.es.
ias.cc.upv.es. 10800 IN A      158.42.4.23

;; AUTHORITY SECTION:
upv.es.          10800 IN  NS   mirzam.ccc.upv.es.
upv.es.          10800 IN  NS   chico.rediris.es.
upv.es.          10800 IN  NS   sun.rediris.es.
upv.es.          10800 IN  NS   vega.cc.upv.es.

;; ADDITIONAL SECTION:
mirzam.ccc.upv.es. 10800 IN  A   158.42.1.5
vega.cc.upv.es.   10800 IN  A   158.42.4.1
sun.rediris.es.   2665  IN  A   130.206.1.2
sun.rediris.es.   5364  IN  AAAA 2001:720:418:caf1::2
chico.rediris.es. 2669  IN  A   130.206.1.3
chico.rediris.es. 4607  IN  AAAA 2001:720:418:caf1::3

;; Query time: 2 msec
;; SERVER: 158.42.249.8#53(158.42.249.8)
```



- Se solicita un nombre bajo uno de los dominios de primer nivel a un **agente registrador autorizado**
 - Ejemplo: **churreria.es** o **bunyuelos.com**
- La empresa solicitante puede nombrar sus máquinas como estime conveniente (introduciendo una nueva jerarquía o no):
 - **porras.madrid.churreria.es**
 - **calabaza.bunyuelos.com**

- Información para registrar en el dominio .es
 - <http://www.dominios.es>
- Para registrar otros dominios en España (.com, .net, .org, ...):
 - <http://www.icann.org/es/resources/registrars>
- Algunas compañías acreditadas:
 - Arsys: <http://www.arsys.es>
 - Interdomain: <http://www.interdomain.org>
 - Nominalia: <http://www.nominalia.com>



•Nominalia•

- Ejemplo: “Network Utopia” networkutopia.eu
- Se registra el dominio **networkutopia.eu** en un *DNS registrar*
 - Ej.: 1&1 (<http://www.1and1.es/>) por solo 0.99€ el primer año
 - El *DNS registrar* inserta dos RR en el servidor TLD de .eu:
 - (networkutopia.eu, dns1.networkutopia.eu, NS)
 - (dns1.networkutopia.eu, 212.212.212.1, A)
 - Si tenemos servidor web, en dns1.networkutopia.eu deberíamos insertar un RR
 - (www.networkutopia.eu, 212.212.212.2, A)
 - Si tenemos un servidor de correo, además
 - (networkutopia.eu, smtp.networkutopia.eu, MX)
 - (smtp.networkutopia.eu, 212.212.212.3, A)