REDES: Tema 2. Capa de aplicación

(Grado en Ingeniería Informática)

Francisco Argüello Pedreira y José Carlos Cabaleiro Domínguez
Departamento de Electrónica y Computación
Universidad de Santiago de Compostela

27 de septiembre de 2011

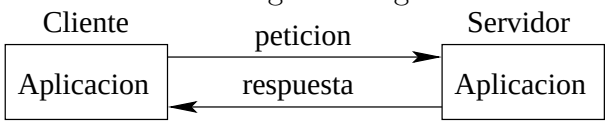
Índice general

Capítulo 2

Capa de aplicación

2.1. Introducción

La capa de aplicación se ocupa de la comunicación entre procesos. El esquema típico considerado se muestra en la siguiente figura.



2.1.1. Protocolo de comunicación

La comunicación de dos procesos, tanto en el mismo host como en hosts diferentes, se hace enviando mensajes. Para que los mensajes sean comprendidos hay que establecer un protocolo.

El protocolo facilita la programación. Hay que programar tanto las funciones de envío como las de recepción y si tenemos un protocolo claro la programación será más fácil.

El protocolo debe especificar:

1. Qué mensajes hay, típicamente qué peticiones se pueden hacer y qué respuestas les corresponden.
2. Las reglas que especifican cuando se envían o responden mensajes.
3. La sintaxis del mensaje, es decir, los campos de que consta.
4. La semántica de cada campo, es decir, su significado.

Por ejemplo, en una aplicación ftp:

1. Mensajes: por ejemplo el envío del nombre del usuario (login), de la clave, la petición de un fichero, etc, y sus correspondientes respuestas.
2. Reglas: por ejemplo, el envío del login debe preceder a todo lo demás y si la clave no es correcta debe repetirse el envío del login.

3. Sintaxis: en ftp la sintaxis de los mensajes es típicamente de una palabra de 4 (a veces 3) caracteres, un espacio, una frase y un retorno de línea.

XXXX	sp	frase	cr+lf

4. Significado: la palabra inicial de 4 caracteres de ftp puede tomar los siguientes significados:

USER	nombre de usuario
PASS	palabra clave
RETR	petición de un fichero

2.1.2. Servicios que necesita la aplicación de red

Los protocolos de la capa de aplicación utilizan por debajo los protocolos básicos proporcionados por la capa de transporte. Como vimos en el tema anterior, estos son: TCP que proporciona un servicio fiable y UDP que proporciona un servicio sencillo y rápido pero que no garantiza la recepción.

En este tema vamos a ver los protocolos de aplicación siguientes:

TCP	HTTP (web) SMTP, POP3 (correo) FTP (ficheros)
UDP	DNS (traducciones)

Los cuatro primeros protocolos usan TCP porque debemos tener la seguridad de que las páginas web, los correos y los ficheros se reciben sin errores. En el caso del DNS (servicio de nombres de dominio), utilizado para realizar traducciones de nombres de hosts/direcciones IP, se usa UDP porque se prefiere cambiar de servidor en caso de no obtener respuesta. Cada host suele tener una lista de servidores de nombres y cuando se precisa una traducción se va probando sucesivamente con los servidores de la lista. Si el primero no responde porque el mensaje se pierde, se pasa el siguiente y así sucesivamente. Si hay problemas, es mejor cambiar de servidor antes que reintentarlo siempre con el mismo (algún servidor puede estar fuera de uso), por eso va bien UDP.

Por el otro lado, se denomina *agente de usuario* a la interfaz entre el usuario y la aplicación. O sea, el programa gráfico que permite al usuario trabajar cómodamente con la aplicación, por ejemplo el navegador o el lector de correo.

2.2. HTTP: Protocolo Transfer. HiperTexto

Este protocolo define la comunicación entre un servidor y un cliente web.

2.2.1. Tipos de conexiones HTTP

Una página web (o documento) usualmente consta de varios objetos, es decir, de varios ficheros, por ejemplo:

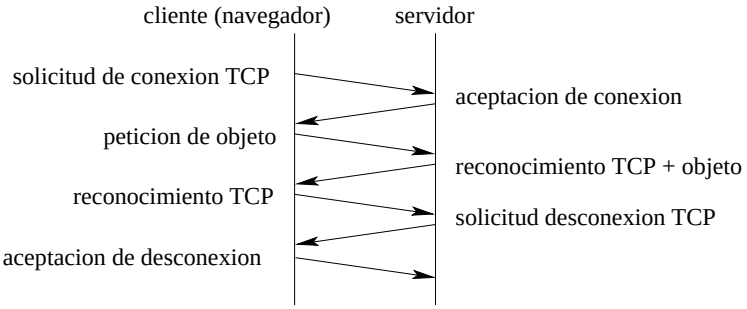
objetos:	documento base imágenes animaciones flash programas java

Cuando un navegador solicita una página web, el primer objeto que se devuelve es el documento base, que contiene el layout de la página y usualmente también el texto. El layout indica cómo conseguir los restantes objetos y dónde han de colocarse.

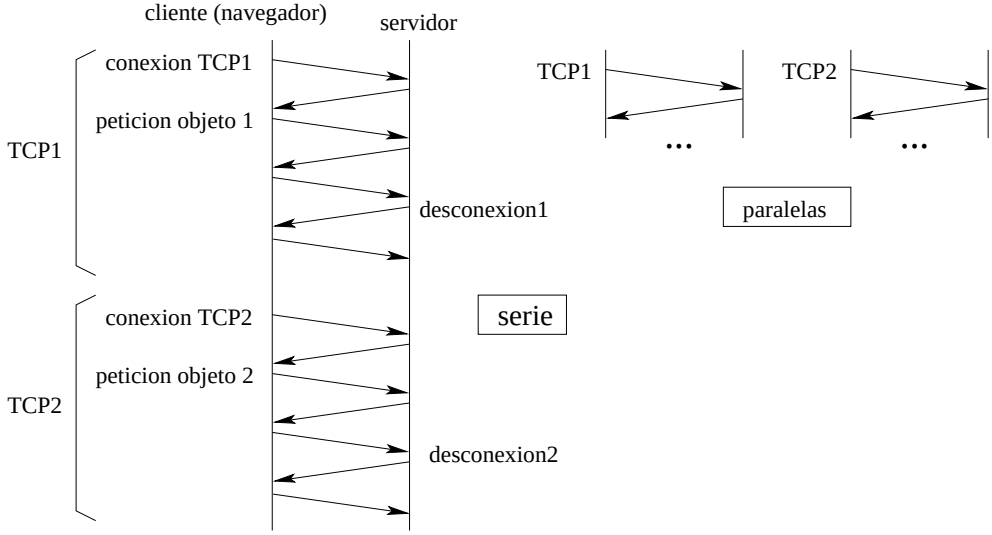
Dependiendo de como se realicen las conexiones para conseguir los distintos objetos se distinguen las siguientes conexiones HTTP:

Conexiones HTTP	
no persistentes	serie
	paralelas
persistentes	sin entubamiento
	con entubamiento

Con **conexiones no persistentes** se utiliza una conexión TCP distinta para transferir cada uno de los objetos. Con **conexiones persistentes** se pueden transferir varios objetos e incluso varias páginas con la misma conexión TCP.

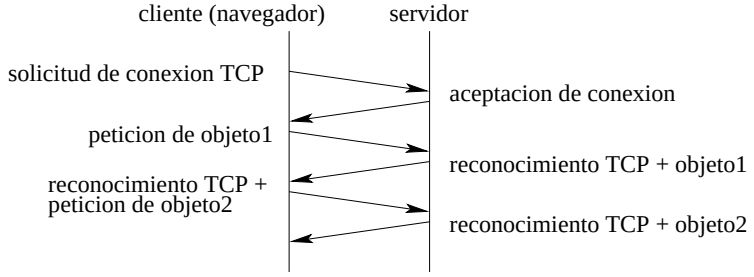


Las conexiones no persistentes pueden ser **serie** si se espera a que acabe la conexión TCP previa antes de que comience la siguiente o bien **paralelas** si se inician varias conexiones TCP a la vez.

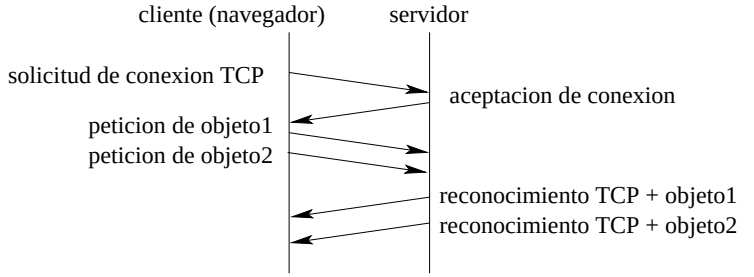


Con **conexiones persistentes** se utiliza la misma conexión TCP para conseguir varios objetos o incluso varias páginas web distintas. Típicamente el servidor cierra la conexión TCP cuando no ha sido utilizada durante cierto tiempo (configurable por el administrador).

Existen dos versiones de conexiones persistentes. En la versión **sin entubamiento**, el cliente sólo pide un nuevo objeto cuando el previo ha sido recibido. En la versión **con entubamiento** el cliente puede hacer peticiones de varios objetos antes de recibir los anteriores. El modo por defecto de HTTP/1.1 es persistente con entubamiento.



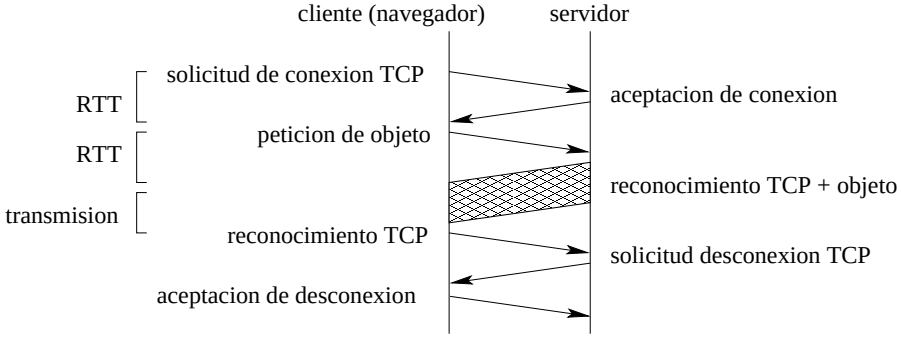
sin entubamiento



con entubamiento

2.2.2. Tiempo de transferencia de una página web

Vamos a calcular el tiempo que se necesita para conseguir el primer objeto de un página web. Vamos a considerar *conexiones no persistentes serie*. Como detalle adicional vamos a tener en cuenta que el objeto es un fichero de un determinado tamaño.



Hemos definido:

- *Tiempo de ida y vuelta* (RTT round-trip time). Es el tiempo necesario para que un paquete pequeño (de tamaño despreciable, por ejemplo de un 1 bit) vaya del cliente al servidor y de nuevo al cliente. Incluye todos los posibles retardos: tiempo de propagación en los enlaces, espera en las colas y procesamiento en los routers. Ya tenemos el tiempo de un bit.
- *Tiempo de transmisión del fichero*. Es el tiempo necesario para almacenar y reenviar un fichero y como es lógico depende del tamaño del fichero.

Resulta un tiempo de transferencia del primer objeto $2RTT + t_{transmision}$, que puede comprobarse que es el mismo en los cuatro tipos de conexiones. En cambio, el tiempo de transferencia de los siguientes objetos es dependiente del tipo de conexión (obtenerlos como ejercicio).

2.2.3. Mensajes HTTP

Separación de los campos. Antes de nada, veamos como se separan los campos en los mensajes. En ASCII se consideran los siguientes caracteres:

sp	espacio
cr	carriage return (retorna a la columna 0)
lf	line feed (avance de una línea)

Los protocolos de las aplicaciones de red, incluido HTTP, definen la separación de líneas en un mensaje con la pareja de caracteres *cr+lf*. Sin embargo, no todos los sistemas operativos utilizan por defecto estos separadores en los textos. Windows si lo hace, mientras que Linux sólo separa las líneas con un *lf*.

Mensajes	cr+lf
Textos en Windows	cr+lf
Textos en Linux, UNIX	lf
Textos en Mac	?

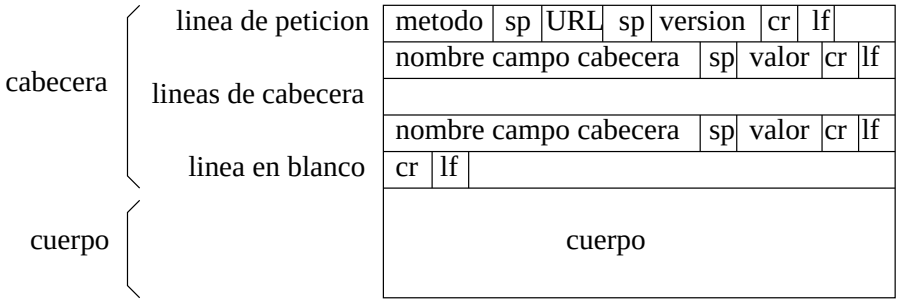
En cuanto a la programación en lenguaje C, la sentencia `printf("hola\n");` en Windows inserta *cr+lf*, mientras que en Linux inserta sólo *lf*. Por tanto, en Linux tendremos que escribir `printf("hola\r\n");` si queremos insertar *cr+lf*. De todas formas, los programas de la capa de aplicación son bastante robustos y suelen comprender los separadores de línea en cualquiera de las versiones.

Peticiones y respuestas. El protocolo HTTP es extremadamente simple. Sólo hay dos tipos de mensajes: petición y respuesta. Las peticiones se usan para pedir objetos, por ejemplo, páginas web o imágenes, mientras que las repuestas contienen los objetos.

Cabecera y cuerpo. Ambos tipos de mensajes constan de una cabecera y de un cuerpo. La cabecera contiene información de control y es texto legible en ASCII de 7 bits (letras, números, espacios, retornos y poco más, no incluye la ñ puesto que ésta sólo está incluida en el ASCII de 8 bits). El cuerpo contiene la información, que puede ser un objeto, el contenido de un formulario, etc, y son datos binarios, es decir, no tienen el significado de los caracteres ASCII.

Mensaje HTTP de petición

Estos mensajes tienen una cabecera que consta de 3 partes y de un cuerpo que consta sólo de una.



- *Línea de petición*. Con ella pedimos un objeto, como un documento base o una imagen. Tiene 3 campos separados por espacios: primero el *método*, que puede ser GET, para pedir una página de web normal o POST cuando se rellena un cierto tipo de formulario. El segundo campo es la *URL* (localizador de recursos uniforme) que indica el directorio y el nombre del objeto que queremos obtener. Por último, el tercer campo es la versión del protocolo que se desea usar, actualmente HTTP/1.1.

- *Líneas de cabecera*. Puede haber el número de líneas que haga falta. Cada línea contiene una opción y consta de dos campos: el nombre de la opción y su valor. Algunos ejemplos:

Host:	nombre servidor web (puede haber varios para una IP)
Connection:	close (si queremos conexiones no persistentes)
User-agent:	mozilla/4.0 (el navegador)
Accept-languaje:	es (el idioma preferido de la página, si la hay)

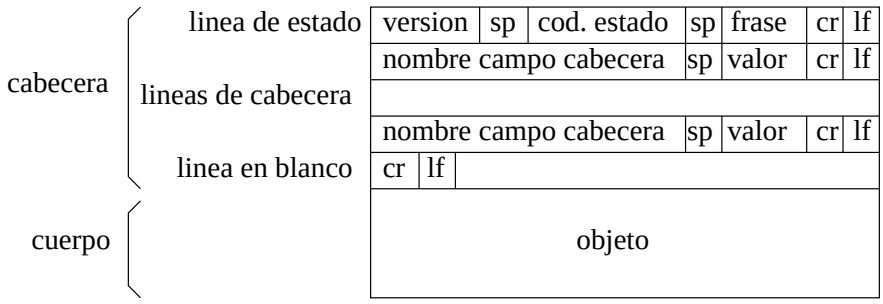
- *Línea en blanco*. Separa la cabecera del cuerpo.

- *Cuerpo*. Cuando se pide una página o un objeto con el método GET el cuerpo va vacío. Cuando se rellena un formulario con el método POST, el cuerpo contiene los datos del formulario. En este caso el servidor procesará los datos introducidos y devolverá una página personalizada.

  Nota: hay más formas de enviar los datos de un formulario, por ejemplo, incluirlos en la URL, como hace, por ejemplo, google.

Mensaje HTTP de respuesta

También tienen una cabecera que consta de 3 partes, mientras que el cuerpo sólo consta de una.



- *Línea de estado*. Tiene 3 campos: el primero es la *versión* que está utilizando el servidor, actualmente HTTP/1.1, el segundo es el *código de estado*, que indica el éxito o fracaso de la petición, mientras que el tercero es una *frase* con una pequeña explicación de este código. Por ejemplo, algunos códigos y frases:

200	OK (el objeto o página se sirve sin problemas)
404	Not found (la página no existe)
400	Bad Request (no se entendió el formato de la petición)

- *Líneas de cabecera*. Son las opciones y puede haber el número que haga falta. Cada línea consta de dos campos: el nombre de la opción y su valor. Por ejemplo:

Connection:	close (indica que se usan conexiones no persistentes)
Date:	fecha de envío de la página
Server:	Apache/1.3.0 (el servidor web)
Last-Modified:	cuando se creó o modificó por última vez la página
Content-Length:	tamaño en bytes del objeto, página o imagen
Content-Type:	text/html, image/gif, image/jpeg

- *Línea en blanco*. Separa la cabecera del cuerpo.

- *Cuerpo*. Los datos, en este caso el objeto, página web, imagen, programa java, animación flash, etc.

Puerto. HTTP utiliza por defecto el puerto 80. En una de las prácticas veremos el contenido de los mensajes que se envían y reciben en HTTP con el comando *telnet*. A este comando hay que indicarle el nombre de un servidor web y especificarle el puerto 80; por ejemplo, *telnet www.usc.es 80*.

Protocolo sin estado. Cada petición de página se resuelve sin tener en cuenta las peticiones de las páginas previas. El servidor HTTP no almacena ninguna información de los clientes. Sin embargo opcionalmente a HTTP se le pueden incluir mecanismos adicionales para mantener información de estado tales como autorizaciones y cookies.

2.3. FTP: Protocolo Transferencia Ficheros

Este protocolo define la comunicación con un servidor de ficheros. Tiene la particularidad de que utiliza dos conexiones TCP paralelas: una para control y otra para datos.

- *Conexión de control.*

  (1) Usa el puerto 21.
  (2) Se utiliza para enviar los comandos: nombre de usuario, la clave, *dir, get, put*, etc, y recibir las respuestas.
  (3) La conexión TCP de control es *persistente*, es decir, dura todo el tiempo de la sesión FTP.
  (4) Usa ASCII de 7 bits, es decir, su contenido es legible.

- *Conexión de datos.*

  (1) Usa el puerto 20.
  (2) Se utiliza, por ejemplo, para transmitir la lista de ficheros en respuesta al comando *dir*, el contenido de los ficheros que se transfieren, etc.
  (3) La conexión TCP de datos es *no persistente*, es decir se abre y cierra una conexión TCP nueva por cada fichero que se transfiere.
  (4) Los datos se transmiten en binario.

Protocolo con estado. Mientras dura la sesión el servidor FTP ha de mantener información de estado, como el nombre del usuario, directorio actual, etc.

Comandos y respuestas FTP. Ambos van por la conexión de control. Cada comando que teclea el usuario y cada respuesta que devuelve el servidor corresponden normalmente a un mensaje transmitido.

Los *comandos* son líneas terminadas en *cr+lf* y constan de una palabra de 4 caracteres en mayúsculas y campos adicionales. Por ejemplo:

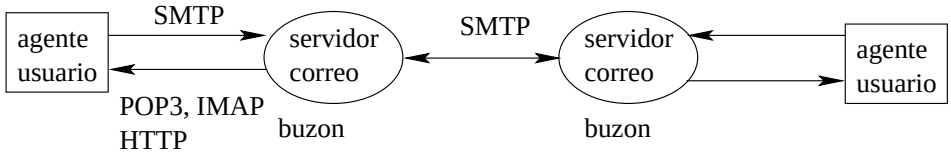
USER	nombre de usuario
PASS	clave
LIST	
RETR	nombre de fichero (traer fichero)
STOR	nombre de fichero (almacenar fichero)

Las *respuestas* también son líneas y constan de dos campos: un código de tres dígitos y una frase explicativa. Por ejemplo:

331	Username OK, password required
125	Data connection already open; transfer starting
425	Can't open data connection
452	Error writing file

Transmisión sin cifrar. Una cuestión a tener en cuenta es que tanto FTP, como telnet y HTTP transmiten todo sin cifrar, incluido el nombre de usuario y la clave. Existen aplicaciones alternativas que realizan la misma función pero que envían la información cifrada tales como *ssh*, *sftp* y *https*.

2.4. SMTP: Protocolo Transf. Correo Sencillo



Existen dos tipos de protocolos de correo:

Para enviar correo al/entre servidores: SMTP.
Para el acceso al correo por el agente de usuario: POP3, IMAP, HTTP.

El funcionamiento del sistema de correo tiene las siguientes características:

1. El agente de usuario es el programa de correo que el usuario ejecuta en su ordenador local y que le permite descargar o enviar correo a un servidor. El buzón de correo no está en el ordenador local sino en un servidor. La razón es que los correos se pueden recibir a cualquier hora del día mientras que el ordenador local usualmente no está siempre encendido o conectado a Internet. Es preferible que cuando estos lleguen se almacenen en el servidor.

2. Cuando enviamos un correo, nuestro agente de usuario se lo pasa al servidor de correo y nuestro ordenador local se olvida ya de todo. El servidor de correo origen examina el destino del correo y selecciona el servidor de correo destino apropiado. Este lo guarda en el buzón de correo del destinatario. El protocolo SMTP se ha diseñado para que haya sólo dos servidores de correo, aunque en la práctica puede haber también servidores intermedios.

3. Puede haber problemas, por ejemplo, que el servidor destino este fuera de servicio. En este caso, el servidor origen lo guarda en la cola de mensajes para intentar enviarlo posteriormente (usualmente cada 30 minutos). Si después de varios días no ha habido éxito el servidor se da por vencido y devuelve al remitente un mensaje de error. Hasta aquí sólo se ha utilizado el protocolo SMTP.

4. Si por el contrario el correo llegó bien, el agente de usuario del destino puede descargarlo en cualquier momento usando alguno de los protocolos de acceso al correo.

Otras características:

- Usa el puerto 25.
- Usa conexiones TCP persistentes: se mantiene abierta la sesión TCP mientras dura la sesión, además si se envían varios mensajes todos irán por la misma conexión TCP.
- Es bastante arcaico puesto que emplea ASCII de 7 bits tanto para las cabeceras como para los cuerpos. Cuando transfiere una imagen, por ejemplo, el agente de usuario la transforma en ASCII.

Mensajes SMTP. Hay tres tipos de mensajes: *comandos*, *respuestas* y *datos*. Comandos y respuestas son mensajes de una línea terminada en *cr+lf*. Dentro de esta línea, los campos se separan con espacios.

- *Comandos*. Constan de una palabra en mayúscula que indica el comando seguida de los parámetros que necesita. Por ejemplo:

HELO	nombre servidor
MAIL FROM:	dirección de correo del remitente
RCPT TO:	dirección de correo del destinatario
DATA	(el contenido del correo)
QUIT	

- *Respuestas*. Constan de un código numérico seguida optativamente de una frase aclarativa. Por ejemplo:

220	nombre del servidor
250	comando que se ejecutó satistactoriamente
354	envíe el correo, terminando en .
221	cierre de conexión

- *Datos*. El contenido de los correos, texto, imágenes, etc. Todos los objetos que forman parte de un mensaje van encapsulados en un sólo fichero codificado en ASCII de 7 bits (con las llamadas extensiones MIME).

Un ejemplo de transacción típica entre cliente (C) y servidor (S) de correo:

```
S:  220 smtp.usc.es SMTP USC Server
C:  HELO smtp.usc.es
S:  250 smtp.usc.es
C:  MAIL FROM: pepito@usc.es
S:  250 Ok
C:  RCPT TO: juanito@yahoo.es
S:  250 Ok
C:  DATA
S:  354 Enter data, end data with <CR><LF>.<CR><LF>
C:  Hola
C:  Esto es una prueba, adios
C:  .
S:  250 Ok: message queued as C5F0B13210D
C:  quit
S:  221 Bye
```

Protocolo completamente inseguro. Como puede verse en el ejemplo anterior, en ningún momento se ha pedido el nombre del usuario y la clave. El servidor permite enviar correos a cualquiera sin pedir identificación. Esto facilita la proliferación del correo basura (spam).

Comparación con HTTP

1. HTTP es un protocolo de demanda mientras que SMTP es un protocolo de oferta. Ambos transfieren ficheros, pero el cliente (el que inicia la transacción) HTTP demanda el fichero (esto es la página web), mientras que en SMTP el cliente pone el correo, esto es, oferta el fichero.

2. SMTP requiere que ambos, cabecera y cuerpo sean ASCII de 7 bits (si se manda un archivo binario hay que convertirlo previamente a ASCII). En HTTP la cabecera es de 7 bits pero el cuerpo puede contener datos binarios.

3. Otra diferencia es la forma de envío de varios objetos: HTTP los envía en ficheros diferentes. En SMTP todos los objetos van encapsulados en el mismo fichero que contiene el texto del correo (usando extensiones MIME).

4. HTTP no tiene estado (cada mensaje es independiente de los demás). SMTP ha de recordar en que fase de la sesión se encuentra.

2.5. Protocolos de acceso al correo

Una vez el correo ha llegado al buzón destino, veamos como el agente de usuario lo transfiere al ordenador local. Para esto no se usa SMTP puesto que es un protocolo de oferta y ello implicaría que nuestro ordenador debería estar siempre encendido y conectado a Internet a la espera de recibir los correos. Lo más práctico es diseñar un protocolo distinto y adecuado para descargar los correos. Estos son los protocolos POP3, IMAP y HTTP. Sólo vamos a ver el primero.

POP3: Protocolo de Oficina Postal, versión 3

```
S:  +OK POP3 pop.usc.es server ready
C:  user pepito
S:  +OK User name accepted, password please
C:  pass miclave
S:  +OK Mailbox open, 2 messages
C:  list
S:  +OK Mailbox scan listing follows
S:  1 10064
S:  2 1054
S:  .
C:  retr 1
S:  (texto ASCII del correo)
S:  .
C:  dele 1
S:  +OK
C:  quit
S:  +OK POP3 server bye
```

Como podemos ver en el ejemplo anterior, POP3 también tiene 3 tipos de mensajes: comandos, respuestas y datos.

- *Comandos*. Constan de una palabra de 4 caracteres seguida por los campos que precisa. Por ejemplo:

user	nombre del usuario
pass	palabra clave
list	
retr	número de correo (traer correo)
dele	número de correo (borrar correo)
quit	

- *Respuestas*. Básicamente las hay de dos tipos:

+OK frase explicativa
−ERR frase explicativa

- *Datos*. Los datos de la respuesta pueden ser: la lista de mensajes, los contenidos de los correos, etc. El correo completo compuesto por la cabecera y el cuerpo se envía en un único mensaje. En esto difiere de SMTP, en el cual cada campo de la cabecera del correo generaba un mensaje independiente.

POP3 se puede usar para descargar y borrar (retr+dele) los correos del servidor o para descargar y mantener en el servidor (retr).

Otras características:

- Usa el puerto 110.
- POP3 también usa *conexiones persistentes*, pues se usa la misma conexión TCP mientras dure la sesión en la que pueden descargarse varios correos.
- Una diferencia con respecto a SMTP es que para usar este protocolo se necesita una cuenta con clave. En SMTP cualquiera puede enviar correo tenga cuenta o no, porque el servidor no comprueba nada; POP3 sí lo hace.

2.6. DNS: Servicio de Nombres de Dominio

Esta aplicación se utiliza para traducir nombres de host a direcciones IP y viceversa. DNS consta de los siguientes elementos:

1. Numerosos servidores de nombres distribuidos por Internet.
2. La base de datos del DNS está distribuida de forma jerárquica entre un gran número de servidores. Por ejemplo, los servidores que participan en la traducción del nombre de host *jerez.dec.usc.es* son los siguientes: el dominio *es* está contenido un servidor de primer nivel, el dominio *usc* está contenido en un servidor de segundo nivel, *dec* en un servidor de tercer nivel, etc.

3. Un protocolo que permite a los hosts pedir traducciones a los servidores y que los servidores intercambien datos entre ellos.

Protocolo sin conexión. DNS usa UDP, utiliza el puerto 53 y no tiene estado. Cada host contiene una lista de direcciones de servidores de nombres, que se suele introducir en el momento de instalación del sistema operativo. Cuando el host precisa hacer una traducción se dirige al primer servidor de la lista. Si por alguna razón no llega la respuesta, pasa al siguiente y así sucesivamente. Esta la razón por la que usa UDP en vez de TCP: si la respuesta no llega o llega mal se prefiere probar con el siguiente servidor antes que reintentarlo con el primero.

Servicios proporcionados por DNS. Son los siguientes:

- Traducciones de nombres de hosts/direcciones ip y también obtener alias de hosts (por ejemplo, *www.elmundo.es* es un alias para *wwwb2.elmundo.es*).
- *Servidores autorizados para un dominio*. Informa qué servidor autorizado es el encargado de proporcionar información sobre un determinado dominio.
- *Alias de servidores de correo*. Permite simplificar las direcciones de correo. Por ejemplo, *usc.es* es un alias para el servidor de correo de la universidad de santiago es *smtp.usc.es*. Así las direcciones de correo pueden escribirse en la forma *pepito@usc.es*.
- *Distribución de la carga*. A un mismo nombre de host se le puede asignar varias IPs en el DNS. Se suele usar con servidores espejo, es decir, con el mismo contenido y que pueden usarse de forma indistinta. El DNS se puede configurar para devolver de forma rotatoria cada una de la direcciones IP del conjunto.

Tipos de servidores. Se distinguen los siguientes:

1. *Servidores locales*. Son los servidores de nombres que atienden a las consultas de nuestros hosts. Cada host contiene una lista de direcciones de servidores de nombres locales, que se suele introducir en el momento de instalación del sistema operativo. Típicamente estos servidores nos los proporciona el ISP (proveedor de servicios de Internet). Cuando nuestro host necesite una consulta, empezará preguntando al primer servidor de la lista.

   Nota: un servidor local puede ser tanto autorizado como no autorizado.

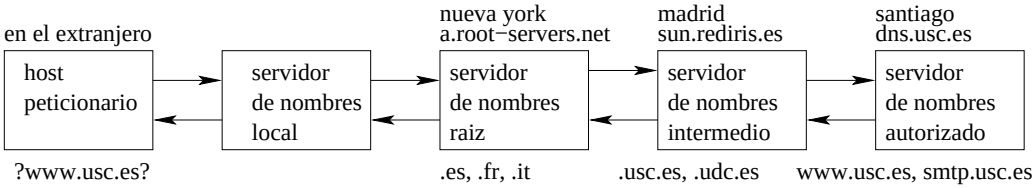
2. *Servidores autorizados/no autorizados*. Los servidores de nombres no autorizados son los que instala un particular o empresa con la lista de los hosts de su red y que probablemente no estén dados de alta en el sistema mundial del DNS. En este caso, sólo se conocen internamente y sólo serán utilizados desde dentro de la red local.

Si queremos que los nombres de nuestros hosts sean accesibles desde Internet tenemos que darlos de alta en un servidor autorizado. Normalmente los servidores autorizados pertenecen al ISP y cada host debe estar incluido en al menos dos servidores autorizados, por razones de fiabilidad.

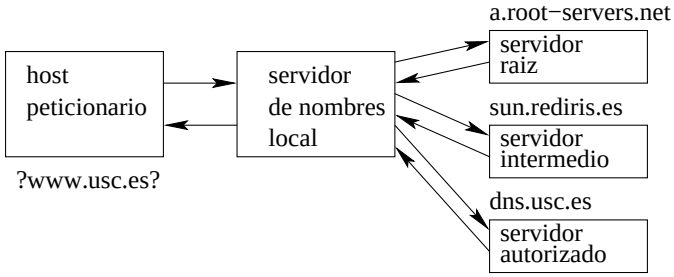
3. *Servidores raíz*. Hay sobre una docena de servidores raíz en Internet. Proporcionan la información de los dominios de primer nivel (*es, uk, fr, com, org*). Si le preguntamos qué servidor se encarga de las consultas del dominio *es* nos dará algunos servidores en Madrid y Barcelona.

4. *Servidores intermedios*. Mientras que los servidores autorizados proporcionan información sobre hosts y los servidores raíz sobre dominios de primer nivel, los servidores intermedios (o servidores autorizados intermedios) proporcionan información sobre los niveles intermedios (*usc.es, udc.es, elmundo.es*). Si al servidor de Madrid le preguntamos qué servidor se ocupa del dominio *usc.es* nos dará un servidor en la Universidad de Santiago.

Consultas recursivas. El siguiente esquema ilustra la cadena de peticiones que se realizan en una consulta del DNS, usando consultas recursivas. En una consulta recursiva se le indica a cada servidor de nombres que se encargue de interrogar al siguiente servidor de la cadena.



Consultas iterativas. Las consultas que hemos visto antes son recursivas porque cada servidor de nombres se encarga de interrogar al siguiente. Otra posibilidad son las consultas *iterativas* en las que el servidor de nombres local es el encargado de ponerse en contacto con todos los servidores.



Caché DNS. Cuando un servidor DNS obtiene una correspondencia además de servirla, hace una copia para ella en su memoria local (disco o RAM). Esta copia se utilizará cuando se vuelva a hacer la misma consulta. Las entradas de la cache se borran después de cierto tiempo sin utilizarse (usualmente en dos días). Esto se hace a todos los niveles de la jerarquía, hasta los propios hosts lo hacen: si una aplicación en un ordenador obtiene una traducción, más adelante otra aplicación podrá beneficiarse de la misma.

Mensajes DNS. Sólo hay dos tipos de mensajes: *Consultas* y *Respuestas*. Ambos tienen el mismo formato, una cabecera con información de control y un cuerpo que contiene las consultas y respuestas.

La cabecera consta de 6 campos (algunos de ellos son en binario):

- *Identificación*. Es un número de 16 bits que identifica la consulta, puesto que un host puede hacer varias consultas a la vez. Este número será el mismo en un mensaje de consulta y en su correspondiente respuesta.

- *Señales*. Son 4 bits que indican si el mensaje es una consulta o respuesta, si la respuesta la dio un servidor autorizado, si el cliente desea consultas recursivas, si el servidor puede realizar recursión, etc.

- *Tamaño de los campos del cuerpo* (4 campos). Un mismo mensaje DNS puede ser utilizado para realizar varias consultas. Estos cuatro números indican: (1) cuantas cuestiones contiene el mensaje, (2) cuantas respuestas se han obtenido, (3) cuantos servidores autorizados están indicados en el mensaje y (4) cuanta información adicional hay.

El cuerpo consta de 4 campos:

- *Cuestiones*. Una o varias preguntas. Aquí se incluye el nombre o dirección IP a traducir, la dirección de correo que se quiere convertir, etc

- *Respuestas*. Una o varias respuestas que se han obtenido, por ejemplo, una pareja nombre de host/dirección IP. Hay que tener en cuenta que cada cuestión puede originar varias respuestas puesto que, por ejemplo, un nombre de host puede corresponder a varias direcciones IP.

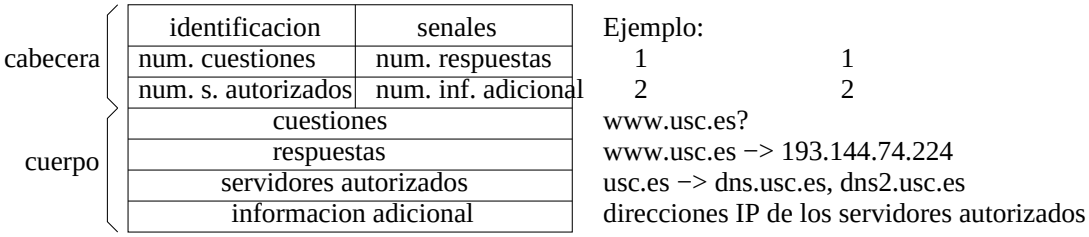
- *Servidores autorizados*. Aquí se indican los servidores de nombres autorizados a los que se puede consultar para un determinado dominio. Esto permite hacer una cadena de consultas.

- *Información adicional*. Otra información relativa a las consultas que no ha sido explícitamente solicitada.

  Cualquiera de estos cuatro campos es una 4-tupla que se denomina *registro de recurso* y tiene el siguiente formato:

Tipo	Nombre	Valor	TTL
A	nombre de host	dirección IP	
NS	dominio	servidor autorizado para el dominio	
CNAME	alias	nombre de host	
MX	alias de correo	servidor de correo	

  TTL (time to live) es el tiempo durante el cual es válida la respuesta.



2.7. Distribución de contenidos

2.7.1. Introducción

Los accesos a servidores centralizados con contenidos web, ficheros, audio, vídeo, etc, pueden ser lentos debido principalmente a dos causas:

- El camino que tienen que seguir los mensajes es lento o está congestionado.
- El servidor puede estar sobrecargado.

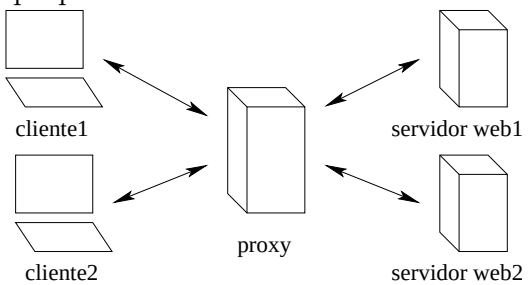
En este apartado veremos formas alternativas a los servidores centralizados para proporcionar contenidos. Es decir, que en vez de tener todos los contenidos en el mismo servidor veremos como distribuirlos (y duplicarlos) en distintas zonas geográficas y dirigir cada petición al servidor con menor tiempo de respuesta.

Para hacer esto tenemos 3 opciones:

Método de distribución	Quién proporciona la infraestructura
Caché web	ISP
Redes de distribución de contenidos	distribuidor de contenidos
Redes P2P (de igual a igual)	usuario

2.7.2. Caché web (o servidor proxy)

La caché web consiste en poner un servidor intermedio (denominado proxy) por el cual pasarán todas las peticiones web de los hosts de una red. El servidor proxy típicamente lo proporciona el ISP.

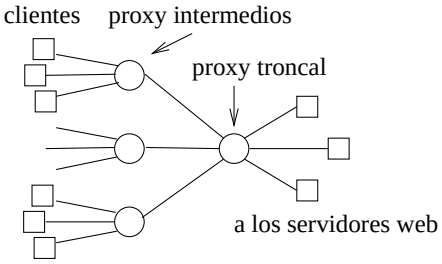


Así, todos los host de la red dirigen primero sus peticiones web al proxy, el cual las redirige a los servidores web destino. Los contenidos devueltos por el servidor pasan a través del proxy, el cual, a la vez que los redirige a los hosts que han

hecho las peticiones, almacena una copia en su caché. Las siguientes peticiones a este contenido serán a partir de ahora locales. El servidor proxy mantiene las copias de los contenidos durante un cierto tiempo; cuando vence este tiempo las borra.

Para usar el servidor proxy el usuario ha de configurar su navegador y rellenar el campo de proxy caché en el menú de preferencias. Por ejemplo, en la Universidad de Santiago hay que indicar el servidor *proxy.usc.es*.

Por último, la caché web admite un esquema jerárquico:



2.7.3. Redes de distribución de contenidos (CDN)

Ciertas empresas denominadas CDN poseen cientos de servidores distribuidos a lo largo del mundo y conectados a Internet por enlaces de alta velocidad, en lo que se denominan *centros de host de Internet*. Estas empresas, sin embargo, no ofrecen contenidos, sino que alquilan su infraestructura a otras empresas. Por ejemplo, la empresa Akamai proporciona su infraestructura al periódico El País, como puede observarse tecleando el comando `host www.elpais.es`. Akamai replica los contenidos de El País en sus centros de host de Internet.

El acceso a los contenidos almacenados en un centro de host de Internet puede hacerse mediante dos técnicas:

- *Redirección de los objetos*. Con esta técnica los documentos base, que es lo primero que busca un navegador cuando se solicita un contenido, estarán en el sitio web del cliente. Pero estos documentos base serán mínimos, conteniendo prácticamente sólo una redirección a los objetos almacenados en los servidores de la empresa CDN.

- *Balanceo de las peticiones usando el DNS*. Esta técnica permite seleccionar el servidor CDN más cercano o rápido al hacer la traducción del nombre de host a una dirección IP.

2.7.4. Redes P2P (de igual a igual, peer to peer)

Por último, una forma de distribución de contenidos proporcionada por los propios usuarios la constituyen las redes P2P. Todos los usuarios de una red P2P son a la vez servidores y clientes de los contenidos. Son servidores pues comparten

los ficheros almacenados en su disco duro y son clientes cuando descargan algo. Un usuario se une a la red P2P ejecutando la correspondiente aplicación: napster, gnutella, kaaza, emule, bittorrent, ...

Cuando el usuario ejecuta la aplicación, lo primero que hace ésta es buscar a otros usuarios P2P. Se necesita por lo menos un *nodo de arranque*, es decir por lo menos conocer a un usuario que ya esté en la red. Esto puede conseguirse, por ejemplo, a través de una página web.

Una vez conectado a la red P2P, el usuario da a conocer públicamente el contenido de su disco duro. La red P2P necesita construir un *directorio* con la totalidad de los contenidos y su ubicación. Este directorio se utilizará durante las búsquedas. La construcción del directorio es la operación más crítica de la red P2P. Las posibilidades son:

Tipo de directorio	Ejemplo
centralizado	napster
no centralizado	kaaza
inundación de consultas	gnutella

Directorio centralizado

Un ejemplo es *napster*. Con este método, un único host en Internet centraliza la construcción y almacenamiento del directorio. Cuando un usuario ejecuta la aplicación P2P se dirige en primer lugar al host central y le proporciona su lista de contenidos. El host central construye el directorio con todos los datos proporcionados por los usuarios y atiende a todas las consultas. Las consultas devuelven las direcciones de los usuarios que disponen del contenido solicitado.

El directorio se actualiza constantemente. Cuando un usuario consigue un contenido pasa a ser servidor de ese contenido. Cuando un usuario deja de tener actividad es borrado del directorio, etc.

El directorio centralizado tiene varios problemas:

1. *Un único punto de fallo*. Si falla el host central, toda la aplicación P2P falla.

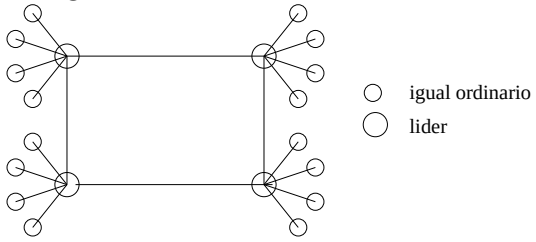
2. *Un cuello de botella*. El host central tiene que mantener el directorio de millones de usuarios y millones de consultas. Se genera un tráfico enorme hacia este punto.

3. *Infracción de copyright*. Al ser el host central propiedad de alguien es susceptible de demandas judiciales.

Directorio descentralizado

Un ejemplo es *kaaza*. El método consiste en distribuir el directorio entre algunos de los usuarios P2P. Ciertos usuarios P2P son designados como líderes en función de ciertos criterios (velocidad de las conexiones, antigüedad en la red,

etc). El directorio se divide en partes que se asignan a los líderes. El esquema se muestra en la siguiente figura.



Cada líder es responsable de atender las consultas de un grupo de usuarios que tiene a su cargo. Las consultas que no puede resolver por sí mismo las redirecciona a los otros líderes con los que está conectado. Esto origina una red de superposición (red virtual sobre la red real) con la correspondencia siguiente:

Red virtual	Red real
hosts	iguales P2P
routers	líderes P2P
enlaces	conexiones P2P

Así se resuelven los problemas de directorio centralizado: único punto de fallo, el cuello de botella y la infracción de copyright. La parte de diseño más delicada es la gestión de los líderes de grupo. Hay que tener en cuenta que un líder puede desaparecer (apagar) sin previo aviso. Entonces habría de elegir como líder a otro y reconstruir la parte del directorio que se haya perdido.

Inundación de consultas

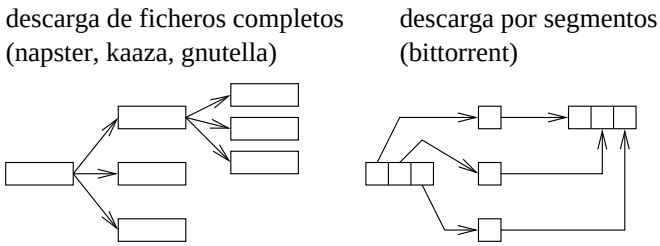
Un ejemplo es *gnutella*. Con este método, la red de superposición es plana, con todos los usuarios al mismo nivel. No se utiliza directorio, sino inundación de consultas. Las consultas se retransmiten de usuario en usuario hasta que se encuentra el contenido solicitado. El problema que tiene es que origina un tráfico enorme de mensajes. Esto se resuelve parcialmente limitando la profundidad de la inundación: se cuenta el número de nodos por los que pasa la consulta y cuando llegue a un cierto valor se da por terminada la consulta.

Descarga de ficheros

Vamos a ver ahora una técnica muy eficiente para la descarga de ficheros enormes y que originan en poco tiempo una avalancha de peticiones. Esta técnica es usada, por ejemplo, por *bittorrent*.

- En una aplicación P2P clásica, una vez que un usuario ha descargado un fichero completo puede convertirse a su vez en servidor de dicho fichero.
- Este método de descarga puede mejorarse si la descarga de los ficheros se hace por segmentos. Un usuario que ha descargado un segmento de un

fichero puede convertirse a su vez en servidor de dicho segmento antes de que complete la descarga del fichero.



Puesto que las aplicaciones P2P son unos 30 años más jóvenes que el TCP/IP, como conclusión podemos decir que TCP/IP fue muy bien diseñado para la tarea que se le encomendó: transportar datos a través de la red independientemente de la aplicación y del hardware.