

- 11.1. Introducción
- 11.2. El modelo cliente-servidor
- 11.3. Sistema de nombres de dominio (DNS)
- 11.4. Acceso remoto
- 11.5. Servicio de correo electrónico
- 11.6. WWW y transferencia de hipertexto: servicio HTTP
- 11.7. Aplicaciones multimedia en Internet

SERVICIOS INTERNET

11.1. Introducción

La capa superior de cualquier modelo de red es la de aplicación. En esta capa se proporcionan al usuario distintos servicios de comunicación de datos. Con el término *usuario* se alude tanto a humanos como otros servicios propiamente dichos. Este capítulo trata del estudio de las aplicaciones Internet más importantes. En particular, se incluyen los servicios de resolución de nombres, de acceso remoto, de correo electrónico, web y aplicaciones multimedia.

En nuestro estudio, el enfoque adoptado se centra en la especificación de los correspondientes protocolos y la explicación de las funcionalidades más relevantes, obviando las implementaciones software concretas de los diferentes servicios.

Una buena parte de las aplicaciones estandarizadas en Internet se basa en el modelo de interacción cliente-servidor. Por este motivo, como paso previo en el siguiente apartado se estudia este modelo, incluyendo la interfaz *socket* así como diferentes tipos de servidores. Posteriormente, en los apartados siguientes se explicarán las diferentes aplicaciones antes identificadas.

11.2. El modelo cliente-servidor

El término *servidor* está fuertemente arraigado entre los usuarios de Internet. A veces este término se identifica erróneamente como sinónimo de *host*. Un servidor es un proceso que permite el acceso remoto a ciertos recursos existentes, eso sí, en un *host*. Una aplicación servidora se caracteriza por

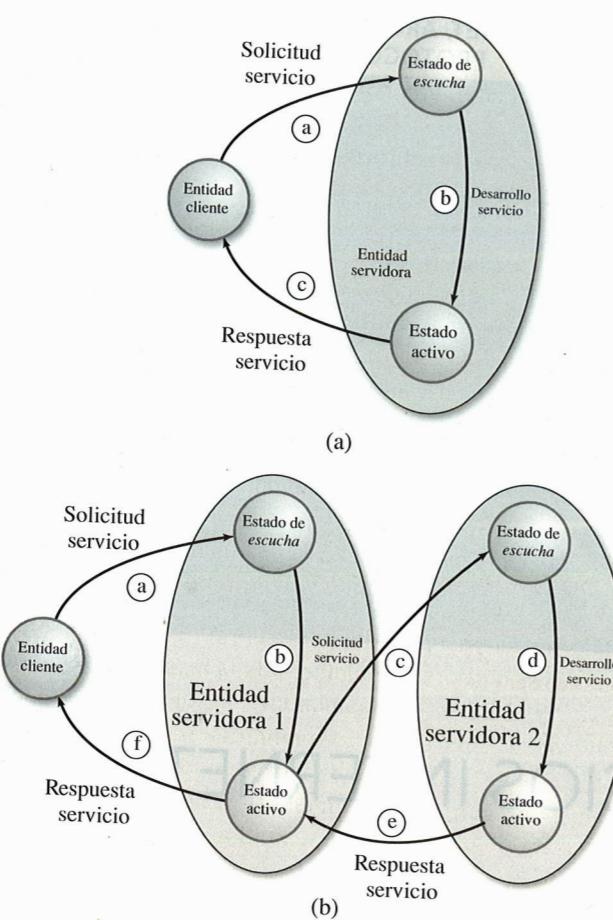


Figura 11.1. Desarrollo de un servicio Internet mediante el modelo cliente-servidor: simple (a) y recursivo (b).

encontrarse inicialmente en un estado latente de escucha, no realizando función alguna hasta que recibe una solicitud remota por parte del proceso remoto o *cliente*.

En la capa de transporte, y más concretamente en el contexto de TCP, es fácil diferenciar el papel de cliente frente al de servidor. Los primeros son los que explícitamente hacen la apertura activa (ver diagrama de estados de TCP, Figura 10.6), mientras que los servidores están permanentemente monitoreando el correspondiente puerto a la espera de recibir las solicitudes generadas por los clientes.

En la capa de aplicación, cliente es el proceso a través del cual el usuario interacciona con el servidor que proporciona el servicio concreto ofrecido. El modelo cliente-servidor está indisolublemente asociado a interacciones solicitud-respuesta. Así, el intercambio de información se realiza mediante mensajes originados en el cliente y enviados al servidor, es decir, las solicitudes ante las cuales los servidores, tras el procesamiento o acciones que hubieran lugar, devuelven un mensaje, es decir, la respuesta.

Como se muestra en la Figura 11.1(a), un servicio que adopte el modelo cliente-servidor se desarrolla en los siguientes pasos:

1. Inicialmente se debe ejecutar la aplicación servidora, pasando esta a modo de escucha y quedando a la espera de recibir solicitudes de servicio. Es lo que se conoce como una puesta en marcha *pasiva* del servicio. Usualmente este inicio se realiza explícitamente por parte del

administrador o a través de un *script*, dejando el proceso ejecutándose en *background*, es decir, sin tener asociada una consola explícita de entrada-salida.

2. La ejecución de la aplicación cliente dará lugar al envío de una solicitud de servicio hacia el servidor (paso (a) en Figura 11.1(a)). Esto, frente al caso del servidor, supone una puesta en marcha *activa* del servicio. Los procesos de cliente nacen y mueren a demanda del usuario, a diferencia de los servidores, que permanecen a la escucha incluso tras haber servido a uno o más clientes.
3. El servidor responderá a dicha solicitud de acuerdo con la aplicación desarrollada, llevándose a cabo el intercambio de información necesario para hacer efectivo el servicio concreto (pasos (b) y (c) en Figura 11.1(a)).
4. Tras el desarrollo del servicio, el servidor volverá al estado de escucha a la espera de nuevas solicitudes. Por su parte, el cliente finalizará su ejecución.

Además de las comentadas aperturas activa y pasiva que diferencian a los clientes de los servidores, otra característica diferenciadora es que, mientras que el servidor se suele asociar a un puerto conocido a priori (y reservado si se trata de un servicio estandarizado), los clientes usan puertos diferentes dependiendo de las disponibilidades del sistema operativo en cuestión.

Las aplicaciones cliente-servidor pueden implementarse en forma de cascada o *recursiva*. En este sentido, una entidad servidora puede actuar a su vez como cliente de otra servidora. En la Figura 11.1(b) se muestra un ejemplo de servicio en el que un cliente lleva a cabo la solicitud a un servidor, el cual, a su vez, consulta con otro servidor. Una vez recibida la respuesta, esta será transmitida al cliente inicial. Por supuesto, la recursividad aludida puede ser de orden superior a uno.

Como se ha comentado anteriormente, las entidades servidora y cliente son meras aplicaciones, pudiéndose encontrar implementadas en hosts distintos o en el mismo. Por lo general, tanto las aplicaciones clientes como servidoras se ejecutan en el plano de usuario. Esto es, el control del proceso correspondiente es del usuario. Por el contrario, los servicios de la capa de transporte (e inferiores) de la pila TCP/IP son por lo general parte integrante del sistema operativo (Figura 11.2).

Las aplicaciones (cliente o servidora) hacen uso de los servicios de TCP/IP (y en concreto de la capa de transporte) mediante un conjunto de llamadas al sistema. Estas no son sino un conjunto de procedimientos (rutas o métodos dependiendo del lenguaje de programación utilizado) que ceden el control al sistema operativo, a través de las cuales las aplicaciones pueden comunicarse entre sí. Este conjunto de llamadas al sistema define la interfaz entre la capa de aplicación y la pila TCP/IP.

Aunque existen muchos entornos de desarrollo de aplicaciones cliente-servidor, aquí vamos a destacar dos interfaces: Winsock, para entornos Windows, y la Interfaz Socket, para sistemas Unix.

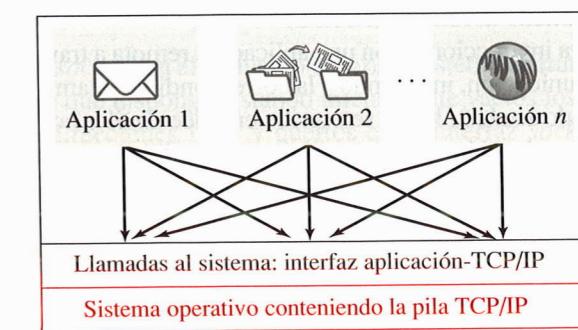


Figura 11.2. Implementación de aplicaciones TCP/IP a través de llamadas al sistema operativo.

Dada su inclusión en la distribución gratuita BSD («Berkeley Software Distribution») presente, por ejemplo, en el sistema operativo Linux, a continuación se describe brevemente la interfaz *socket* para el desarrollo de aplicaciones TCP/IP.

Antes de ello, comentar que no hay que confundir interfaz con protocolo. Nótese que la interfaz está íntimamente relacionada con el sistema operativo en particular e incluso con el compilador o entorno de desarrollo utilizado, mientras que protocolo es el conjunto de reglas sintácticas y funcionales que regulan el intercambio de información entre entidades pares (situadas en la misma capa) que en este caso, como ya hemos comentado, se denominan cliente y servidor.

Adicionalmente, es pertinente señalar que el modelo cliente-servidor no es el único posible. Es interesante mencionar que el modelo cliente-servidor acopla fuertemente tanto en el tiempo como en localización a las entidades que intercambian información. Es decir, las entidades que interactúan con este paradigma necesitan coincidir en el tiempo y tener perfectamente ubicada la localización (típicamente la dirección IP y puerto) del otro extremo.

Alternativamente hay otros paradigmas como la publicación-subscripción, en la que se definen unas entidades denominadas *publicadores* que generan información, mientras que otras, los *consumidores*, se suscriben a esta información. Hay distintas especificaciones de este paradigma, destacando CORBA y DDS («Data Distribution Service»), estándares especificados por el OMG («Object Management Group»), cuyas implementaciones están siendo cada vez más utilizadas. Concretamente, algunas implementaciones se basan en una aproximación centrada en datos, es decir, para intercambiar información se define un espacio de datos global a través del cual se realiza la comunicación mediante tópicos. Un tópico se puede definir como un canal virtual a través del cual los publicadores publican datos y los subscriptores acceden a ellos. El espacio de datos puede implementarse de forma centralizada (con las dificultades que ello puede generar) o distribuida, como una memoria cache residente en todas las entidades participantes. En este caso, la *cache* permite desacoplar tanto en tiempo como en espacio el intercambio de información.

11.2.1. La interfaz *socket*

Como su propio nombre indica, la interfaz entre las aplicaciones y la pila TCP/IP se basa en el concepto de *socket*. Para explicar este partimos del precedente de la gestión de entrada-salida I/O del sistema operativo, y concretamente de UNIX (o cualquiera de sus variantes). Como es sabido, la interacción con dispositivos o ficheros en este SO se basa en el modelo abrir-leer/escribir-cerrar. Es decir, antes de que la aplicación pueda interactuar con el fichero (o dispositivo), tiene que abrirlo (típicamente invocando a la llamada al sistema *open*) y, a través del descriptor del fichero correspondiente, lee y escribe (llamadas *read* y *write* o sus variantes) de acuerdo con las demandas de la aplicación, para finalizar cerrándolo (mediante la función *close*).

De igual manera, para interactuar con una aplicación remota a través de la interfaz *socket*, la aplicación debe abrir la comunicación, invocando a la correspondiente llamada al sistema, la cual devuelve un descriptor (denominado *socket*), luego se escribe en y lee de él y por último se cierra.

El descriptor de comunicación o *socket* consiste en realidad en una variable tipo puntero a una estructura, la cual tiene definida una serie de campos que caracterizan o definen todas las propiedades y atributos necesarios asociados a la comunicación. Concretamente, entre otras, esta estructura tiene definidos los siguientes campos (Figura 11.3):

- **Familia de protocolos:** referente al tipo de *socket* de que se trata, esta variable puede tomar los valores **PF_INET**, para indicar comunicaciones Internet IPv4, u otras en desuso como **PF_UNIX**, para procesos UNIX, **PF_APPLETALK**, para sistemas Apple, **PF_X25**, para entornos X.25, etc.

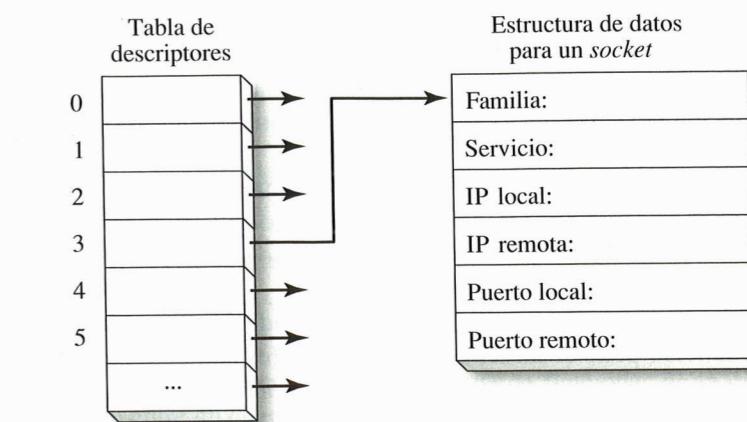


Figura 11.3. Descriptores y estructuras de datos para sockets.

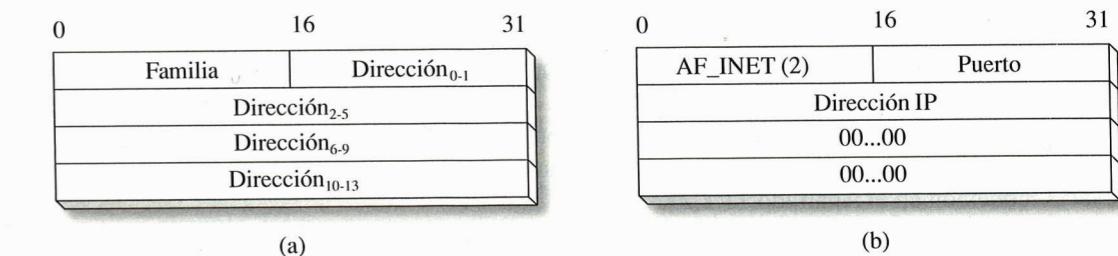


Figura 11.4. Estructura sockaddr_in genérica (a) y particular para comunicaciones IPv4 (b).

- **Servicio:** tipo de servicio a utilizar ofrecido por las capas inferiores, el cual puede ser, entre otros, **SOCK_STREAM**, que indica que está basado en TCP, **SOCK_DGRAM**, basado en UDP, o **SOCK_RAW**, para el acceso directo a los servicios de red o inferiores.
- **IP local:** dirección IP del *host* local.
- **IP remota:** dirección IP del *host* remoto.
- **Puerto local:** número de puerto asociado a la entidad local.
- **Puerto remoto:** número de puerto de la entidad remota.

Cuando se crea un *socket* con el comando *open*, el sistema operativo le asocia internamente el primer descriptor libre de que disponga, estando inicialmente vacíos los datos de la estructura apuntada.

Para manejar las direcciones IPv4 y puertos en la interfaz *socket* se define la estructura **sockaddr_in**, la cual tiene el formato indicado en la Figura 11.4. Para direcciones IPv6 la interfaz define una estructura parecida, denominada **sockaddr_in6**. La familia de direcciones puede ser, como ocurría con la familia de protocolos, de varios tipos: **AF_INET** (valor 2), para direcciones IPv4, **AF_INET6** (valor 10), para las direcciones IPv6, y otras en desuso como **AF_UNIX**, para direcciones propias de Unix, etc.

Las funciones de la interfaz *socket* más conocidas a través de las que se accede a la estructura **sockaddr_in** y, en definitiva, mediante las que se permite la comunicación extremo a extremo entre un cliente y un servidor son:

- `socket(familia, tipo, protocolo)`: función que crea un *socket* de familia de protocolos PF_INET, PF_UNIX, etc., de tipo SOCK_STREAM, SOCK_DGRAM, SOCK_RAW, etc. y correspondiente al protocolo (si *familia*=PF_INET) TCP, UDP, IP, ICMP, etc.
- Esta función devuelve un *socket*, es decir, un puntero a una estructura como la mostrada en la Figura 11.3.
- `bind(socket, dir_local, long_dir)`: asignación de la dirección local especificada en la estructura `sockaddr_in`, de nombre *dir_local* (IP más puerto) y longitud *long_dir* octetos, a un *socket*. Esta función suele aplicarse solo al servidor para asociarle una IP y un puerto en concreto.
- `listen(socket, lcola)`: aplicable solo a servidores orientados a conexión (basados en TCP), esta función pone el *socket* en modo pasivo (estado de escucha) e indica, a través de *lcola*, el número máximo de solicitudes de conexión que se permite almacenar en la cola mientras se está atendiendo a una conexión previa.
- `connect(socket, dir_remota, long_dir)`: función que establece la conexión de un *socket* dado con un proceso remoto de dirección *dir_remota*, de longitud *long_dir*, si el *socket* es de tipo SOCK_STREAM. En cambio, si este es de tipo SOCK_DGRAM, la función `connect` solo sirve para llenar localmente los campos de la estructura *socket* que aluden al extremo remoto (IP más puerto).
- `accept(socket, dir_remota, long_dir)`: función a través de la que se acepta una conexión entrante y se le asocia un *socket*. La dirección de la entidad remota queda almacenada en la estructura *dir_remota*, de longitud *long_dir*, de tipo `sockaddr_in`. Esta función devuelve un *socket* nuevo «conectado» al extremo remoto (ver servidores orientados a conexión en el Apartado 11.2.2).
- `read(socket, buffer, long)`: función que permite leer de un *socket*. Los datos, en una cantidad *long* octetos, quedan almacenados en *buffer*.
- `write(socket, buffer, long)`: función que permite escribir en un *socket* un total de *long* octetos de datos contenidos en *buffer*.
- `close(socket)`: función a través de la que se cierra un *socket* y, en consecuencia, se da por finalizada una comunicación.

Otras funciones de la interfaz *socket* de amplio uso son las siguientes:

- `recv / recvmsg / recvfrom`: funciones de lectura de un *socket*. Mientras que `recv`, al igual que `read`, solo puede utilizarse para servicios orientados a conexión, `recvmsg` y `recvfrom` pueden utilizar tanto para servicios orientados a conexión como no orientados a conexión.
- `send / sendmsg / sendto`: funciones de escritura en un *socket*. Mientras que `send`, al igual que `write`, solo se puede utilizar para servicios orientados a conexión, `sendmsg` y `sendto` pueden utilizarse tanto para servicios orientados a conexión como no orientados a conexión.
- `shutdown`: finaliza una conexión TCP en uno de los dos sentidos.
- `gethostbyname`: función que permite especificar un *host* mediante un nombre de dominio en lugar de mediante una dirección IP. La conversión se llevará a cabo internamente en base a algún mecanismo establecido al efecto (ver Apartado 11.3).
- `getservbyname`: permite especificar un servicio mediante un nombre en lugar de mediante el número de puerto, la conversión se llevará a cabo internamente.
- `getprotobynumber`: función que permite especificar un protocolo mediante un nombre en lugar de mediante el número correspondiente. La conversión se llevará a cabo, como antes,

- `htons / htonl`: convierten un entero corto/largo con formato de representación del orden de los octetos considerado en el *host* al definido en la red.
- `ntohs / ntohs`: convierten un entero corto/largo con formato de representación del orden de los octetos considerado en la red al definido en el *host*.

Los ficheros de cabecera básicos usuales necesarios para poder disponer de las funciones mencionadas anteriormente son:

```
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
```

En la Figura 11.5 se esquematiza una secuencia de funciones de la interfaz *socket* involucradas en la comunicación entre un cliente y un servidor en una comunicación TCP. Estas son las siguientes:

- Tanto en una entidad como en la otra se necesita la creación inicial de un *socket* mediante la función `socket`.
- Este se asocia a una dirección local en la parte del servidor mediante la función `bind`.
- La función `listen` en el servidor hará que el servicio pase a modo de escucha, definiendo el tamaño de la cola para almacenar solicitudes de conexión.
- Será el cliente quien solicite el servicio mediante la función `connect`.
- El servidor acepta esta a través de `accept`, llamada al sistema que devuelve un *socket* conectado con el cliente.
- Establecida la conexión, ambas entidades procederán al intercambio de datos mediante, por ejemplo, las funciones `read` y `write`. Entre ambas funciones, claro está, las dos partes realizarán los procesos necesarios correspondientes al servicio y al protocolo de la capa de aplicación implementados.
- Finalizado el servicio, ambas partes cerrarán la conexión mediante la liberación del *socket* correspondiente (función `close`). En el caso del servidor, el proceso quedará a la espera de nuevas solicitudes de servicio.

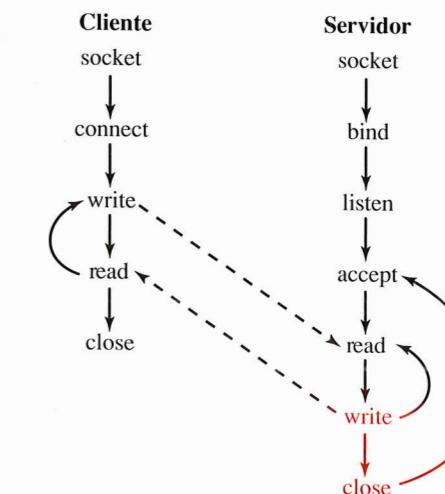


Figura 11.5. Ejemplo de secuencia de llamadas a funciones *socket* en una comunicación TCP cliente-servidor.

11.2.2. Tipos de servidores

Los servicios TCP/IP pueden clasificarse atendiendo a dos criterios distintos:

1. Tipo de comunicación: esta puede ser *orientada a conexión* o *no orientada a conexión*; es decir, basada en TCP o en UDP.
2. Número de clientes atendidos simultáneamente: desde este punto de vista, el servicio se clasifica como *iterativo*, cuando solo se acepta un cliente a la vez, o como *concurrente*, cuando se permiten varios clientes de forma simultánea en el tiempo.

De acuerdo con lo anterior, hay cuatro tipos de servidores, correspondientes a las posibles combinaciones que podemos hacer: iterativo no orientado a conexión, iterativo orientado a conexión, concurrente orientado a conexión y concurrente no orientado a conexión. A continuación se explica cada uno de ellos.

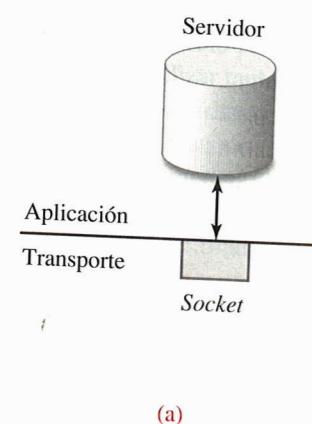
Servidores iterativos no orientados a conexión

Este tipo de servidores son los más simples. Basados en UDP (o sea, no existe conexión previa a la transmisión de los datos), disponen de un *socket* o puerto bien conocido (del tipo de los mostrados en la Tabla 10.1) sobre el que se desarrolla el servicio con el cliente en forma secuencial (Figura 11.6(a)). Las funciones involucradas en este tipo de servicios son las indicadas en la Figura 11.6(b), donde hemos de destacar:

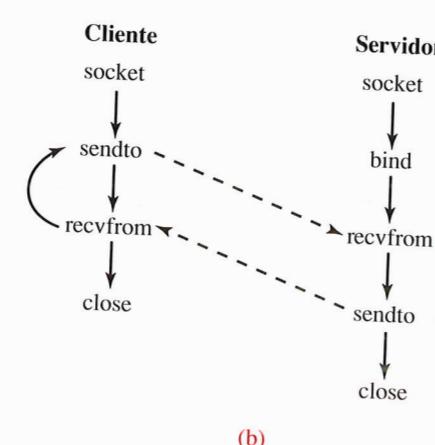
- a) No es necesario utilizar la función `connect` en el cliente, y no se usa `accept`, para el establecimiento de la conexión, puesto que esta no existe.
- b) Las funciones de lectura/escritura son `recvfrom/sendto` (o `recvmsg/sendmsg`) en lugar de `read/write` (o `recv/send`) como sucede con TCP. La función `recvfrom` devuelve los parámetros necesarios para identificar la IP y puerto del cliente, los cuales se pasan a la función `sendto`.

Servidores iterativos orientados a conexión

En este caso sí se establece una conexión previa a la transmisión de los datos; es decir, este tipo de servidores usa TCP. Como se muestra en la Figura 11.7(a), ahora aparecen involucrados dos *sockets*: uno para atender las solicitudes de conexión, el cual es bien conocido por los clientes (del tipo de los



(a)



(b)

Figura 11.6. Servidor iterativo no orientado a conexión: visión conceptual (a) y llamadas a funciones de la interfaz socket (b).

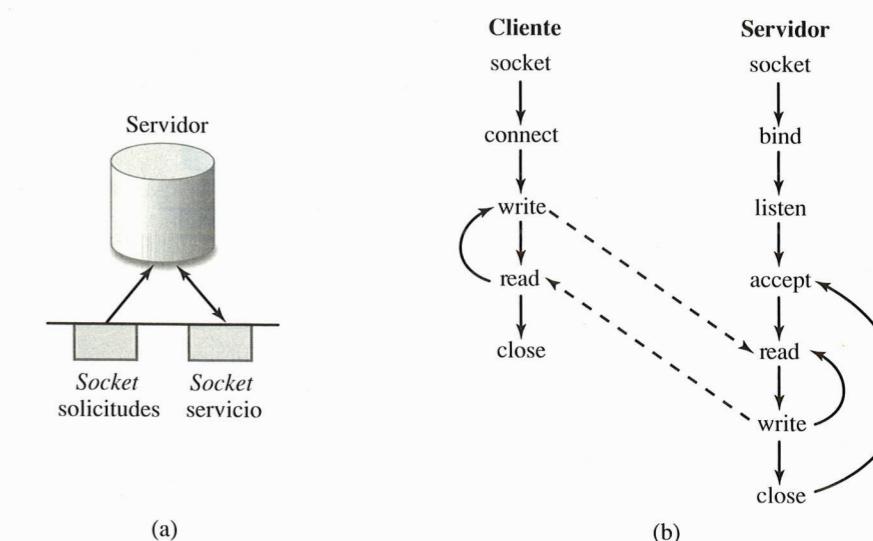


Figura 11.7. Servidor iterativo orientado a conexión: visión conceptual (a) y funciones de la interfaz socket involucradas (b).

indicados en la Tabla 10.2), y otro sobre el que se realiza el intercambio de datos propiamente dicho y se desarrolla el servicio como tal. Este segundo *socket* es el que devuelve la función `accept` tras aceptar la solicitud de conexión (ver Apartado 11.2.1). Como servidor iterativo que es, hasta que no finalice el servicio en curso no se pueden volver a atender nuevas solicitudes, las cuales, si las hay, estarán pendientes de ser atendidas en la cola creada al efecto con `listen`.

Las funciones de la interfaz *socket* involucradas en la implementación de este tipo de servicios son las mostradas en la Figura 11.7(b), las cuales ya han sido comentadas como ejemplo en la Figura 11.5.

Servidores concurrentes orientados a conexión

Como en el caso anterior, en este tipo de servidores existe un *socket* bien conocido sobre el que se reciben todas las solicitudes, las cuales se desvían a *sockets* de servicio sobre los que se desarrollarán las conexiones individuales. Sin embargo, como concurrentes que son, frente a los iterativos comentados anteriormente, estos servidores pueden dar servicio a varios clientes de forma simultánea en el tiempo. Como se muestra en la Figura 11.8(a), cada una de las conexiones correspondientes a los distintos clientes está gestionada por un proceso esclavo dependiente del proceso servidor o maestro.

Las funciones involucradas en la implementación de este tipo de servidores son las mismas que las indicadas en la Figura 11.7(b), con la diferencia de que el desarrollo de cada servicio aceptado se llevará a cabo mediante la ejecución de un proceso hijo (por ejemplo, usando la función `fork()`). Sin embargo, el alto coste que conlleva la creación de nuevos procesos, unido al hecho de que en ocasiones puede requerirse, por ejemplo, el intercambio de información entre las diversas conexiones desarrolladas, provoca que en ciertos casos se realice una simple simulación de la concurrencia de los servicios. Este tipo de procesamiento, por contraposición a la concurrencia efectiva, real, se conoce como *concurrencia aparente* (Figura 11.8(b)) y puede implementarse a través del uso de la función de entrada/salida síncrona `select` de la interfaz *socket*, la cual permite al servidor atender de forma pseudo-concurrente a cada uno de los *sockets* de servicio asociados a los clientes.

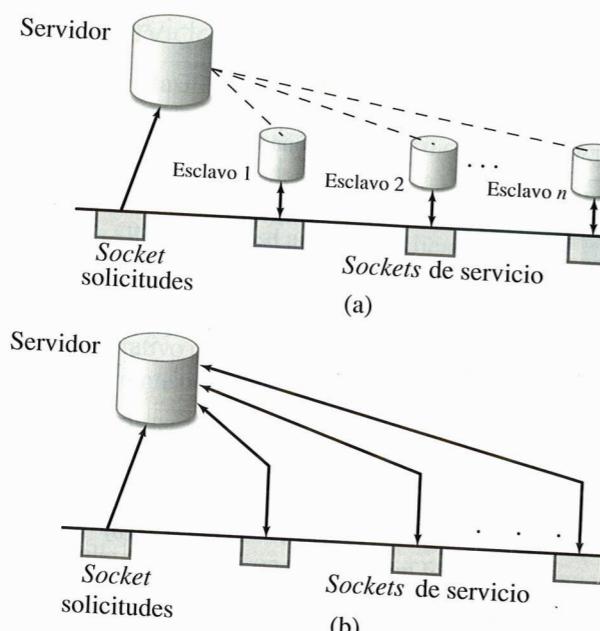


Figura 11.8. Servidor concurrente orientado a conexión (a). Concurrencia aparente (b).

Servidores concurrentes no orientados a conexión

La principal ventaja de los servicios no orientados a conexión, frente a los orientados a conexión, radica en el menor consumo de recursos en el servidor y su mayor interactividad. Esta ventaja se contrarresta con el coste usual involucrado en la creación de procesos concurrentes, por lo que no es habitual encontrar implementados en la práctica servidores concurrentes no orientados a conexión.

11.2.3. Otros tipos de servidores

Los tipos de servidores comentados anteriormente son los básicos, a partir de los cuales podemos encontrar otros más complejos. Estos son los siguientes:

- **Servidores multiprotocolo.** Este tipo de servidores se implementan tanto sobre TCP como sobre UDP con un único proceso; es decir, permiten que el servicio proporcionado pueda ser tanto orientado a conexión como no orientado a conexión. Para ello, como se indica en la Figura 11.9, existen dos *sockets* para atender las solicitudes de servicio, uno para TCP y otro para UDP. Es claro pues que, dependiendo del puerto por el que se reciba la solicitud, el servicio desarrollado será orientado a conexión o no orientado a conexión. En la Figura 11.9 se muestra el esquema conceptual de un servidor multiprotocolo iterativo, pudiéndose recurrir a un desarrollo concurrente para ciertos servicios.
- **Servidores multiservicio.** Según es fácilmente deducible del nombre, este tipo de servidores utiliza varios *sockets* para atender diferentes solicitudes, uno por servicio ofrecido. En este caso, el servidor selecciona periódicamente un *socket* mediante la función `select` y se pone a escuchar en él durante un intervalo de tiempo dado. Si recibe una solicitud de servicio sobre el mismo, se procederá a atenderla de la forma conveniente dependiendo de si el servicio es iterativo no orientado a conexión (Figura 11.10(a)), iterativo orientado a conexión (Figura 11.10(b)) o concurrente orientado a conexión (Figura 11.10(c)).

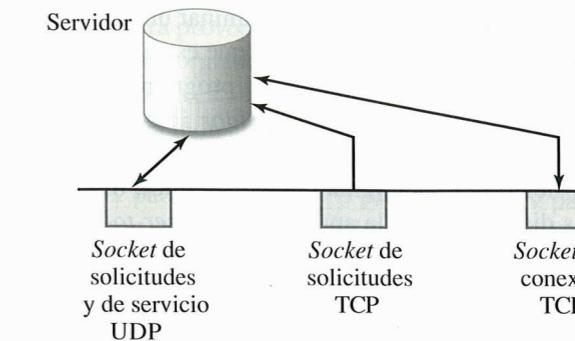


Figura 11.9. Servidor iterativo multiprotocolo.

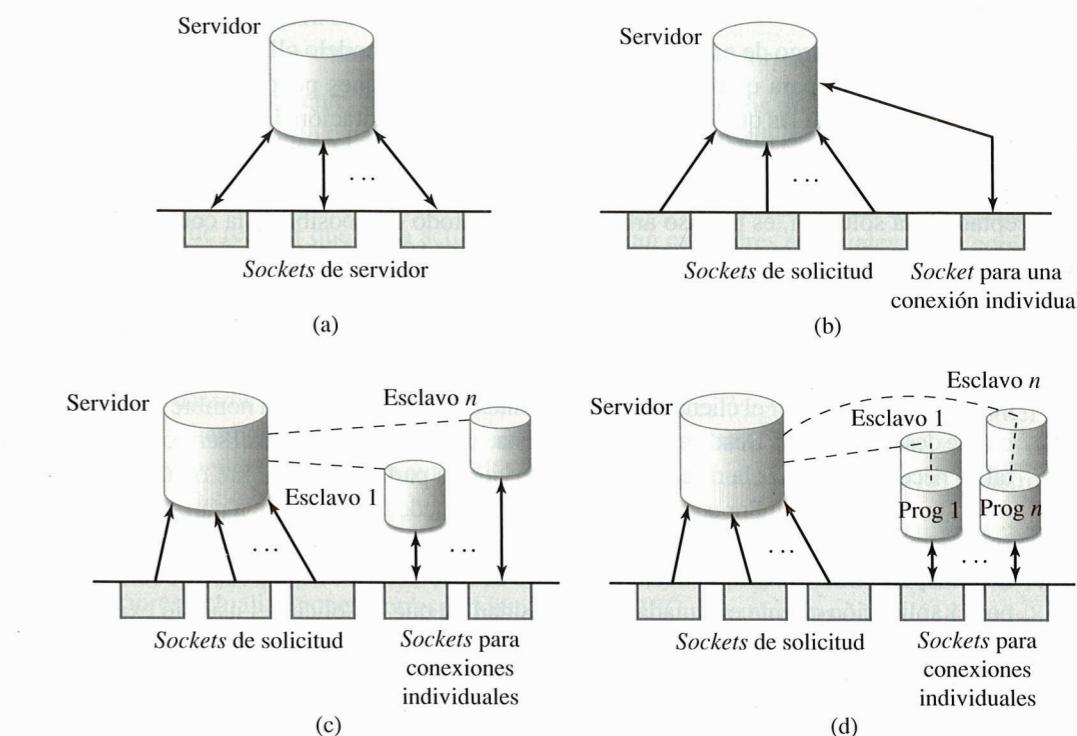


Figura 11.10. Servidores multiservicio: iterativo no orientado a conexión (a), iterativo orientado a conexión (b), concurrente orientado a conexión (c) y concurrente orientado a conexión basado en `execv` para ejecutar programas independientes para cada aplicación aceptada (d).

Si bien este tipo de servidores es muy potente por cuanto que permite el desarrollo de múltiples servicios, hay que señalar que presenta un inconveniente importante: cada vez que deseemos añadir, modificar o eliminar un servicio hemos de actualizar el programa servidor completo. Este problema se puede resolver mediante la consideración de programas independientes para cada aplicación aceptada. De esta forma, cuando se reciba una solicitud de servicio por uno de los *sockets* al efecto, el servidor se limitará a ejecutar un programa independiente asociado a dicho servicio a través de los recubrimientos `execv` del sistema (Figura 11.10(d)).

Por otra parte, para facilitar la tarea de añadir o eliminar un servicio se suele recurrir a un fichero de configuración donde se indica al servidor, cada vez que este se inicia, cuáles son los *sockets* de servicio a atender y, como hemos dicho antes, cuáles los programas asociados a ejecutar en cada caso. Este es el modo de proceder en el conocido *superservidor inetd* del sistema Unix, cuyo fichero de configuración asociado es, en la mayoría de las distribuciones, */etc/inetd.conf*.

Para finalizar la exposición del paradigma cliente-servidor, diremos que una alternativa para el desarrollo de aplicaciones distribuidas es la aproximación *peer-to-peer*. Aunque caracterizada por el tratamiento de igualdad o paridad entre entidades finales, la mayor parte de las implementaciones *peer-to-peer* actuales en realidad son un *overlay* o recubrimiento a los servicios de transporte basados en el paradigma cliente-servidor.

11.3. Sistema de nombres de dominio (DNS)

Según lo explicado a lo largo de esta segunda parte del texto, el modelo cliente-servidor está acoplado espacialmente; es decir, para que las correspondientes aplicaciones puedan comunicarse, necesitan saber las direcciones IP en las que están ubicadas. A pesar de la notación decimal con puntos adoptada para las direcciones IP, es claro que el manejo directo de este tipo de direccionamiento por parte del usuario es incómodo. Resulta mucho más fácil el manejo de «nombres» asociados a dichas direcciones. Aceptada esta solución, es preciso arbitrar algún método que posibilite la conversión automática de nombres a direcciones IP, servicio que se conoce como de *resolución de nombres*. Con este objetivo principal surgió el denominado *sistema de nombres de dominio* (DNS, «Domain Name System»), cuyas dos características más sobresalientes son:

1. Se suele invocar como paso previo en la ejecución de otros servicios, puesto que la activación de cualquiera de ellos por el cliente correspondiente utiliza usualmente el nombre del servidor con el que se desea contactar. Desde este punto de vista, antes de llevar a cabo el servicio solicitado como tal es necesario obtener la dirección IP del servidor a partir de su nombre o, lo que es lo mismo, «resolver su nombre». Por supuesto, todo servicio puede desarrollarse a partir de la dirección IP, en cuyo caso, como es obvio, no se procederá a la resolución del nombre del *host* servidor.
2. Es transparente al usuario; es decir, el proceso de resolución se lleva a cabo de forma automática por la aplicación cliente ejecutada, sin necesidad de que lo solicite el usuario explícitamente.

Como en el caso de las direcciones IP públicas, los nombres deben ser únicos y universales. Por ello, para facilitar la tarea de asignación de nombres, el espacio diseñado al efecto no es plano, sino jerárquico, refiriéndose a cada nivel en la jerarquía con el término *dominio*. En este sentido, el nombre de un *host* IP es de la forma *parte_local.dominio_niveln... dominio_nivel2.dominio_nivel1...*. Así, por ejemplo, el nombre *luna.fcien.ugr.es* consta de 4 dominios: el raíz o «.» tácito en todo nombre aunque no se incluya explícitamente, *es*, *ugr* y *fcien*, siendo *luna* la parte local e identificativa, en último término, del *host* dentro del dominio. El dominio situado más a la derecha del nombre, es decir, el de nivel 1 (superior) por debajo del raíz, se conoce como *dominio genérico* (*.es* en el ejemplo anterior), mientras que a los de nivel inferior (2, 3, etc.) se les llama en ocasiones *subdominios*. Inicialmente se definieron nueve dominios genéricos (ver RFC 1591):

- **.com** → dominio dedicado a organizaciones comerciales.
- **.edu** → instituciones educativas, tales como universidades, de los Estados Unidos.
- **.gov** → instituciones gubernamentales estadounidenses.
- **.mil** → dominio destinado a grupos militares de los Estados Unidos.

- **.net** → dominio para proveedores de red.
- **.org** → dominio para organizaciones diversas diferentes a las anteriores.
- **.int** → organizaciones establecidas por tratados internacionales entre gobiernos.
- **.xy** → dos caracteres indicativos de la zona geográfica, según ISO-3166. Algunos ejemplos son: **.eu** para Europa, **.es** para España, **.us** para Estados Unidos, **.fr** para Francia, **.pt** para Portugal, **.gi** para Gibraltar, **.jp** para Japón, **.tv** para Tuvalu, etc.

Con posterioridad se añadieron algunos dominios genéricos adicionales, como:

- **.biz** → del término inglés «business», este dominio ha sido definido para negocios.
- **.info** → dominio de uso no restringido.
- **.name** → dominio de nombres personal; por ejemplo, *juan.perez.name*.
- **.pro** → para actividades profesionales.
- **.coop** → dominio ideado para cooperativas.
- **.aero** → para compañías aéreas.
- **.museum** → como su nombre indica, este dominio corresponde a museos.

Y más recientemente, se han definido dominios genéricos asociados a comunidades autónomas como por ejemplo **.cat** para Cataluña, **.eus** operativo a partir de marzo o abril de 2014 para el País Vasco, y para mejorar la visibilidad de ciertas ciudades importantes se han definido dominios genéricos como **.paris**, **.berlin**, entre otros. Además de la definición de los dominios genéricos anteriores, hemos de reseñar que en la actualidad se aceptan caracteres especiales, tales como **ñ** o caracteres acentuados, en los nombres de dominio (IDN, «Internationalised Domain Names», RFC 3490).

La reserva y asignación de nombres de dominio los gestionan organismos como INTERNIC (<http://www.internic.org>) o ICANN («Internet Corporation for Assigned Names and Numbers», <http://www.icann.org>), dependientes del IANA (<http://www.iana.net>). Dada la extensión de Internet, el ICANN delega en centros regionales. En el caso de España, Red.es (organismo dependiente del Ministerio de Industria, Energía y Turismo del Gobierno de España), a través de www.dominios.es (y una serie de agentes registradores), es la autoridad nacional encargada de la gestión de los registros de dominios **.es**, estando en la actualidad registrados más de millón y medio de dominios.

El servicio DNS se implementa tanto sobre UDP como sobre TCP (puerto 53), es decir, es un ejemplo se servidor multiprotocolo (ver Apartado 11.2.3) y está especificado en los RFC 1034 y 1035, y posteriores actualizaciones correspondientes a los RFC con números 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2137, 2181, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604. DNS es un servicio fundamental en el despliegue de Internet, y se ha implementado como una aplicación siempre residente en *hosts* finales; es decir, respeta el principio de diseño de situar la complejidad en los extremos (ver Apartado 8.1).

En la Figura 11.11 se muestra el esquema básico de un procedimiento de resolución de nombres:

- a) Cuando un programa de usuario precisa de la resolución de un nombre, contacta con un proceso del sistema operativo local denominado *resolver*, el cual realiza llamadas al sistema operativo para llevar a cabo la consulta de un fichero o base de datos de conversión definido al efecto y/o de una memoria *cache* donde se almacenan temporalmente resoluciones previas.
- b) Si la resolución no puede hacerse efectiva localmente, se contacta con un servidor DNS remoto. La localización o identificación de este servidor se hace mediante configuración manual (cuyos detalles dependerán del SO en particular) o de forma automática (mediante, por ejemplo, el protocolo DHCP, ver Apartado 8.8). El servidor DNS proporcionará la respuesta tras la consulta de los ficheros oportunos o, en caso de necesidad, de un servidor de nombres de nivel superior.

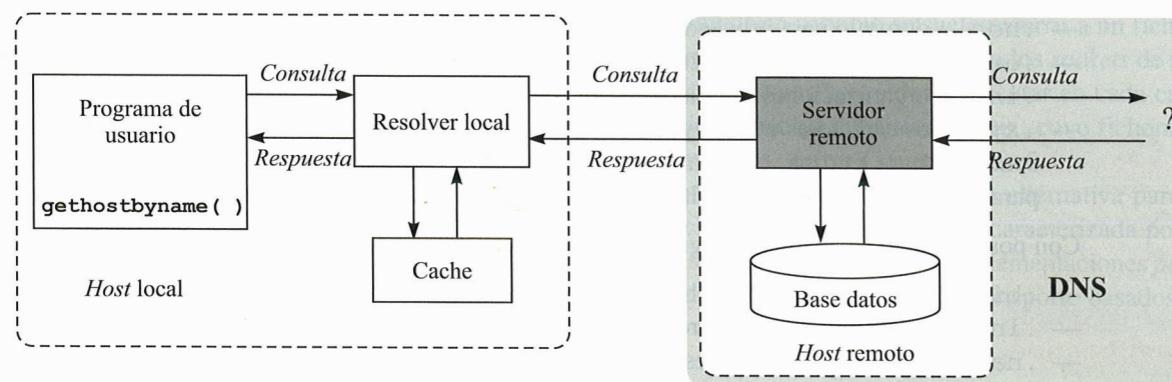


Figura 11.11. Procedimiento básico de resolución de nombres.

Al igual que el espacio de nombres de dominio, el servicio de resolución es jerárquico, tal que los distintos servidores (cooperativos entre sí) se organizan en un árbol lógico con una estructura coincidente con la jerarquía o árbol formado por el espacio de nombres de dominio. Así, un *host* contactará con el servidor DNS establecido al efecto, el cual, a su vez, podrá contactar (si no conoce la respuesta a la consulta recibida) con un servidor DNS:

- De nivel superior, si el dominio sobre el que se realiza la consulta no depende jerárquicamente de él (en principio, estas consultas se redirigen al servidor raíz, es decir, al nivel más alto de la jerarquía);
- De nivel inferior, en caso contrario (en este caso, la consulta se redirigirá hacia el servidor del que dependa el dominio a consultar; este podrá, a su vez, redireccionarlo hacia otros inferiores, según la jerarquía establecida, hasta alcanzar algún servidor que conozca la respuesta).

Desde este punto de vista, la resolución de nombres puede ser *recursiva* o *no recursiva (iterativa)*. En el primer caso, ver Figura 11.12(a), tanto la consulta como la respuesta se trasladan recurrentemente de servidor en servidor, dejando la interacción abierta durante todo el procedimiento. En cambio, la resolución no recursiva se caracteriza por que cuando un servidor no conoce la respuesta a la resolución planteada, este responde proporcionando la identidad del servidor siguiente en la jerarquía, con el que el primero debe contactar directamente para hacer la consulta pertinente (Figura 11.12(b)). En este caso, como se observa en la figura, la interacción no se mantiene durante todo el procedimiento, sino que cesa una vez proporcionada la respuesta. La resolución recursiva consume más recursos, si bien tiene como ventaja que la respuesta puede ir actualizando las memorias *cache* de todos los servidores involucrados; por el contrario, la resolución iterativa consume menos recursos, si bien las memorias *cache* no pueden ser actualizadas.

En el árbol jerárquico de nombres de dominio se define el concepto de *zona* como el conjunto completo de nombres de dominio que están por debajo de un nodo del árbol. Así por ejemplo, la zona asociada al dominio raíz («.» situado en el nivel más alto) incluirá a todos los nombres de dominio del espacio de nombres, mientras que, por ejemplo, la zona asociada a `.ugr.es` contendrá solo los nombres de dominio cuyos dos primeros niveles sean `.es` y `.ugr`, respectivamente. En cada zona existe un servidor DNS responsable que tiene la obligación de conocer la resolución de todos los posibles nombres que pertenezcan a esa zona. Este servidor se conoce como la *autoridad*. Las respuestas que proporciona la autoridad se denominan «*autorizadas*» (frente a estas, las resoluciones obtenidas de

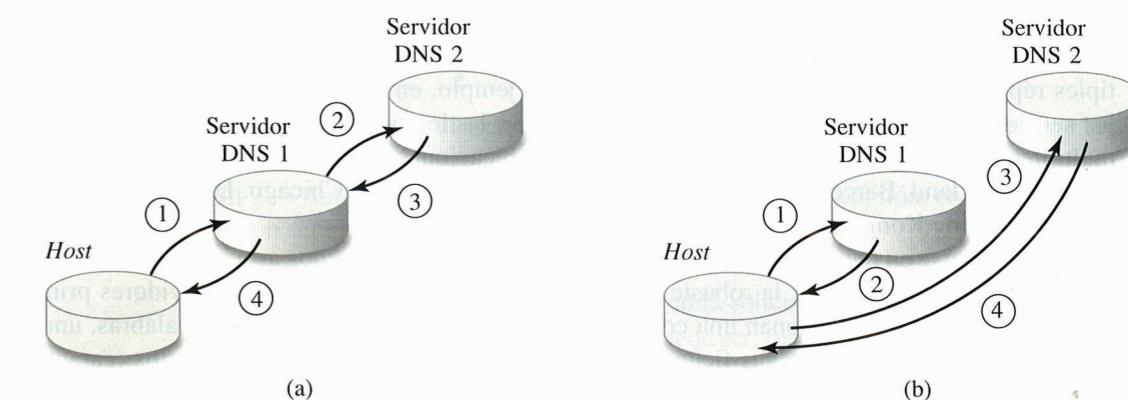


Figura 11.12. Resolución de nombres recursiva (a) e iterativa (b).

una memoria *cache* se llaman «no autorizadas»). A partir de las definiciones anteriores es claro que la autoridad asociada a la zona raíz debería conocer la resolución de todos los posibles nombres de dominio. Para ganar en escalabilidad, se define el concepto de *delegación de autoridad*. Una autoridad para una zona dada puede liberarse de la obligación de conocer y gestionar parte de su zona cediendo esta obligación a un servidor delegado que se responsabilizará de esta obligación una vez que la autoridad haya sido delegada. En este sentido, es fácil imaginar que el servidor autoridad asociado al raíz tiene delegada la autoridad por cuestiones de escalabilidad a las correspondientes autoridades delegadas de primer nivel, como por ejemplo `.es`. De esta forma, la autoridad asociada a la raíz se libera de la obligación de conocer y gestionar la zona `.es` de nuestro ejemplo. La autoridad se puede ir delegando en tantos niveles como se estime necesario. Por ejemplo, la autoridad asociada a la zona `.es` delega la autoridad de `.ugr.es` y así se libera de la gestión de todo el espacio de nombres por debajo de `.ugr` (y así sucesivamente).

Por tanto, las resoluciones (recursivas o iterativas) cuya respuestas no sean conocidas por el primer servidor de nombres, como hemos comentado previamente, son redirigidas al raíz. Este, tras realizar un análisis sintáctico del nombre de dominio en cuestión, obtiene el nombre de primer nivel, por ejemplo `.es`. A partir de ese momento, si tienen delegada la autoridad para esa zona y no tiene en *cache* la resolución, trasladará «hacia abajo» en el árbol la resolución hacia la autoridad responsable de la zona delegada (en nuestro ejemplo `.es`). Este procedimiento se repite «hacia abajo» tantos niveles como hagan falta hasta que, o bien se encuentra solución proveniente de la memoria *cache* de un servidor, o bien se llega (al nivel de profundidad que sea necesario) a la autoridad responsable de la zona correspondiente, la cual, si no ha delegado en sucesivos niveles la autoridad, tiene la obligación de conocer la respuesta asociada a la resolución de nombres planteada.

A la vista del procedimiento, es natural plantearse objeciones en cuanto a la escalabilidad del sistema, toda vez que cualquier consulta que sea desconocida por el servidor de nombres configurado al efecto, según hemos explicado, ha de pasar necesariamente (de forma recursiva o iterativa) por el servidor raíz. Aunque el servidor raíz haya delegado toda su autoridad a los correspondientes servidores de las zonas de primer nivel, sigue siendo un posible cuello de botella. La solución a esta circunstancia pasa por entender que el servidor raíz, lejos de ser un único servidor, consiste en un conjunto de instancias distribuidas por todo el globo. En concreto, hay 13 instancias del servidor asociada a la zona raíz (identificadas por las letras de la A a la M). Su ubicación geográfica —normalmente

están asociados a puntos neutros o NAPs, ver Apartado 8.3— y otros detalles adicionales se pueden consultar en <http://www.root-servers.org>. Es más, cada una de estas instancias puede tener múltiples réplicas igualmente distribuidas. A modo de ejemplo, en Madrid se encuentran instancias de los servidores raíz («root servers») correspondientes a las letras C, F y J, las cuales a su vez tienen múltiples réplicas (por ejemplo, la instancia F tiene ubicaciones en 55 sitios, tales como Ámsterdam, Atlanta, Auckland, Barcelona, Pekín, Buenos Aires, Cairo, Caracas, Chicago, Dar es Salaam, Dubái, Frankfurt, Hong Kong, etc.).

Ha de tenerse en cuenta que, por tanto, cada zona debe tener al menos asociado un servidor autoridad. Además, para mejorar la robustez del servicio, en cada zona se definen servidores primarios (es decir, aquellos que gestionan una copia maestra de la base de datos; en otras palabras, una base de datos obtenida a partir de sus ficheros locales) frente a los servidores secundarios (caracterizados por obtener la base de datos por transferencia, mediante el protocolo DNS que explicaremos más adelante).

Antes de comentar detalles sintácticos y de procedimiento del protocolo DNS, el cual es el responsable de formular las consultas («queries» a la base de datos del espacio de nombres de dominio) y devolver las respuestas, es conveniente conocer la definición de la base de datos en cuestión. En realidad, todo nombre de dominio tiene asociado al menos un registro de la base de datos. Los registros se denominan RR («Resource Record») y están definidos como una tupla de 5 campos:

- *Nombre del dominio*: nombre del dominio al que se refiere el RR.
- *Tiempo de vida*: tiempo de validez de un registro (para la *cache*).
- *Clase*: en Internet siempre IN.
- *Tipo*: tipo de registro. Algunos de los más relevantes son:
 - SOA «Start of Authority», registro con la autoridad de la zona.
 - NS «Name Server», registro que contiene un servidor de nombres. Permite la delegación de autoridad.
 - A «Address», registro que define una dirección IP.
 - MX «Mail eXchange», registro que define uno o varios servidores de correo electrónico.
 - CNAME «Canonical Name», registro que define el nombre canónico de un nombre de dominio.
 - HINFO «Hardware Info», información del tipo de máquina y sistema operativo.
 - TXT «TeXT», información del dominio.
- *Valor*: Contenido que depende del campo tipo.

El formato de los mensajes intercambiados entre un cliente, o *resolver DNS*, y un servidor de este servicio es el indicado en la Figura 11.13. Los campos son los siguientes:

- *Identificación*: campo de 16 bits que identifica cada mensaje DNS. Permite llevar a cabo la correspondencia entre solicitudes y respuestas.
- *Parámetro*: campo de 16 bits cuyo significado es el siguiente:
 - Bit 0 → Tipo de operación, la cual puede ser *solicitud* (bit a valor 0) o *respuesta* (bit a valor 1).
 - Bits 1-4 → **Tipo de solicitud**, aceptándose los valores 0, para indicar una conversión estándar nombre→dirección IP; 1, para indicar una conversión inversa en la que, a partir de una respuesta, se trata de generar la pregunta que la originó; o 2, correspondiente a la solicitud de estado del servidor.

- Bit 5 → Válido en mensajes de respuesta, este bit se encuentra activado si esta es autorizada; es decir, si el servidor es la autoridad encargada de la resolución de nombres del dominio.
- Bit 6 → Activado si el mensaje está truncado debido a una longitud superior a la permitida por la ruta.
- Bit 7 → Activado en el mensaje de solicitud si se desea recursión; en caso contrario, la resolución será iterativa.
- Bit 8 → Activado en el mensaje de respuesta si la recursión está disponible.
- Bits 9-11 → Reservados, a valor 0 tanto en la solicitud como en la respuesta.
- Bits 12-15 → **Tipo de respuesta**, la cual puede ser: resolución llevada a cabo con éxito (valor 0), error en el formato de la solicitud (valor 1), fallo en el servidor (valor 2), el nombre no existe (valor 3), tipo de solicitud no implementada (valor 4) o solicitud rechazada (valor 5).

— *Nsolicitudes*: número de entradas en la sección *Ssolicitud* del mensaje.
 — *Nrespuestas*: número de entradas en la sección *Srespuesta* del mensaje.
 — *Nautoridades*: número de entradas en la sección *Sautoridad* del mensaje.
 — *Nadicionales*: número de entradas en la sección *Sadicional* del mensaje.
 — *Ssolicitud*: conjunto de octetos a través del cual se lleva a cabo una consulta. Como se indica en la Figura 11.13(b), esta sección consta de los siguientes campos:

- *Solicitud*: campo de longitud variable donde se especifica el nombre de dominio por el que se pregunta (por ejemplo, el nombre de un *host*) como una secuencia de etiquetas. Cada etiqueta consta de un octeto de longitud seguido de dicho número de bytes.
- *Tipo*: indicando el tipo de solicitud, este campo puede tomar valores comprendidos entre 1 y 16. Así, por ejemplo, *tipo*=1 (denominado A) indica que se trata de una consulta respecto de un *host*, *tipo*=2 (NS) hace referencia a un servidor de nombres autorizado para el dominio, *tipo*=5 (CNAME) al nombre canónico de un *alias*, *tipo*=6 (SOA) al comienzo de una zona de autoridad, *tipo*=12 (PTR) a una resolución inversa dirección IP→nombre, *tipo*=15 (MX) a un servidor de correo electrónico, etc.
- *Clase*: tipo de direcciones de que se trata. Los valores aceptados son 1, para Internet (IN); 2, para CSNET (CS); etc.

— *Srespuesta*: conjunto de octetos a través del cual se da respuesta a una consulta DNS. Como se muestra en la Figura 11.13(c), los campos de la sección *Srespuesta* son los siguientes:

- *Recurso*: nombre de dominio al que se refiere esta sección.
- *Tipo*: como en la sección *Ssolicitud*.
- *Clase*: como en la sección *Ssolicitud*.
- *Tiempo*: campo de 16 bits que indica el tiempo máximo de validez de esta resolución. Transcurrido dicho tiempo, la entrada en la *cache* debería ser eliminada.
- *Longitud*: número de octetos de que consta el campo *Datos*.
- *Datos*: conjunto de octetos donde se especifica la resolución como tal; por ejemplo, la dirección IP del *host* cuyo nombre se indicó en la solicitud.
- *Sautoridad*: sección que contiene datos que describen servidores autorizados. El formato es el mismo que el de la sección *Srespuesta*.
- *Sadicional*: sección donde se pueden incluir datos adicionales complementarios a las otras secciones. El formato es el mismo que el indicado para la sección *Srespuesta*.

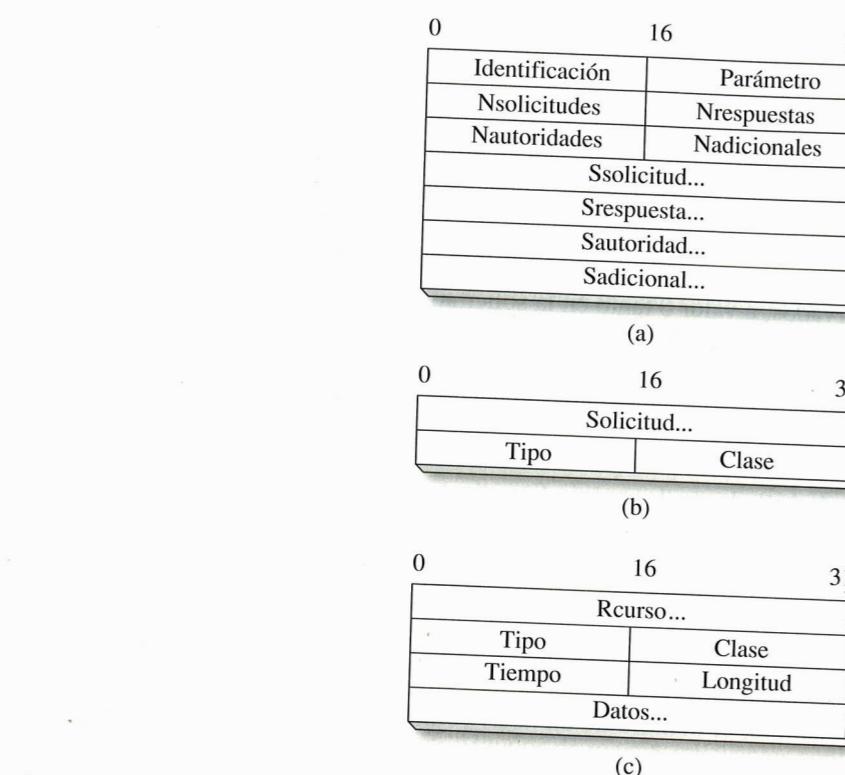


Figura 11.13. Formato de los mensajes DNS (a) y secciones Ssolicitud (b) y Srespuesta (c) de los mismos.

11.4. Acceso remoto

Uno de los servicios TCP/IP básicos es el acceso remoto, mediante el cual se posibilita el acceso a hosts remotos con el fin de usar sus recursos de disco, memoria, etc. como si fuesen locales. Inicialmente, coincidiendo con los primeros despliegues de Internet se especificó TELNET (Figura 11.14), y durante décadas ha sido ampliamente usado, si bien motivados fundamentalmente por sus vulnerabilidades de seguridad (ver Capítulo 12), desde un tiempo a esta parte no se recomienda su uso, siendo SSH la alternativa recomendada. En este apartado, se comentan brevemente las características más relevantes de estas dos aplicaciones.

11.4.1. TELNET

A pesar de no estar recomendado, por su interés académico y porque en ocasiones se usa como herramienta para configurar inicialmente equipos, en este apartado estudiamos el servicio TELNET («TELe-communication NETwork»). El servidor TELNET se implementa sobre el puerto reservado 23, es concurrente y orientado a conexión; es decir, se implementa sobre TCP. Especificado inicialmente en el RFC 854 (STD 8) y actualizado por el RFC 5198, las características del servicio TELNET son:

1. Es transparente en el sentido de que el usuario es como si estuviese trabajando directamente sobre el host remoto.
2. Es un servicio identificado, requiriendo como paso previo que el usuario se identifique ante el servidor introduciendo un nombre de usuario y una palabra de paso («login/password»).

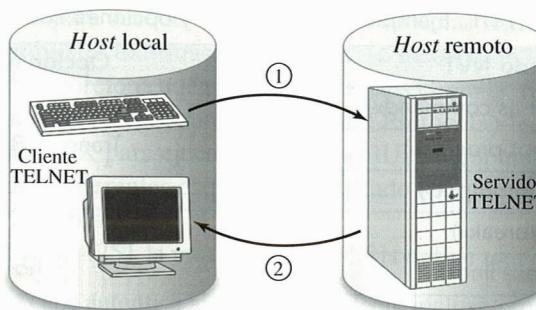


Figura 11.14. Servicio TELNET.

3. Define un «lenguaje» estándar intermedio llamado *terminal virtual de red* (NVT, «Network Virtual Terminal») que permite la interconexión de sistemas heterogéneos (Figura 11.15).
4. Incluye un mecanismo para la negociación de opciones entre el cliente y el servidor.
5. Ambos extremos pueden iniciar la negociación, por lo que se dice que la conexión es simétrica.

El código definido mediante NVT es ASCII estándar de 7 bits. Así, ambos extremos deberán traducir a este los datos a enviar y convertir desde este a su representación local los recibidos. El código USASCII del NVT consta de un conjunto de 95 caracteres imprimibles más 33 de control (retorno de carro, nueva línea, desplazamiento del cursor a izquierda y derecha, etc.). Adicionalmente a estos caracteres, el octavo bit en NVT permite la gestión de la conexión. Dicha gestión de conexión se lleva a cabo a través de *secuencias de escape*, en las que todo carácter de control aparece precedido por el carácter IAC («Interpret As Command»), de código decimal 255. Algunos de los caracteres de control más destacables son los siguientes (Tabla 11.1):

- IP (244): señal «interrupt process», que interrumpe el programa («Control-C»).
- AO (245): señal «abort output», que vacía el *buffer* de salida.
- BRK (243): señal de atención.
- EL (248): carácter para el borrado de línea completa.
- AYT (246): test de accesibilidad del servidor.
- DM (242): carácter de marca de datos («data mark») que, en conjunción con una notificación TCP urgente, permite la sincronización del flujo de datos.
- WILL (251): solicitud de uso de opción.
- WON'T (252): solicitud de no uso de opción.
- DO (253): respuesta positiva al uso de opción.
- DON'T (254): respuesta negativa al uso de opción.

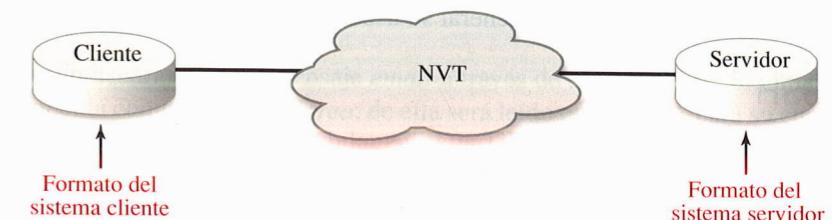


Figura 11.15. Comunicación TELNET mediante NVT.

Tabla 11.1. Ejemplos de comandos y opciones TELNET

Comando NVT	Opción NVT
IAC («interpret as command»)	Transmit-Binary
IP («interrupt process»)	Echo
AO («abort output»)	Terminal-Type
BRK («break»)	Linemode
EL («erase line»)	
AYT («are you there»)	
DM («data mark»)	
WILL	
WON'T	
DO	
DON'T	

La negociación de opciones entre el cliente y el servidor es simétrica, dado que cualquiera de las dos partes puede iniciarla. El proceso de negociación es el siguiente:

- a) El solicitante envía la secuencia IAC WILL X para indicar que desea hacer uso de la opción X. Si se usa el comando WON'T en lugar de WILL, se indica que no se hará uso de la opción especificada.
- b) El receptor contestará afirmativamente a la solicitud mediante la secuencia IAC DO X. En caso de no estar de acuerdo, enviará la secuencia IAC DON'T X.

Como se indica en la Tabla 11.1, algunas opciones NVT son las siguientes:

- *Transmit-binary* (código 0): cambio a transmisión de datos de 8 bits (ver RFC 856). Cuando se usan datos de 8 bits en la transmisión y el carácter IAC aparece como dato en el flujo de información, el emisor lo duplica y envía la secuencia IAC IAC¹.
- *Echo* (código 1): permite a un extremo replicar lo que recibe (ver RFC 857).
- *Terminal-Type* (código 24): permite fijar el tipo de terminal considerado (ver RFC 1091).
- *Linemode* (código 34): envío de líneas completas en lugar de caracteres individuales (ver RFC 1184).

Un último comentario acerca de TELNET es que si bien este servicio es identificado, la palabra de paso («password») introducida por el usuario en el host cliente se transfiere en texto llano («plain-text») hacia el servidor para su comprobación. Esto significa que una persona ajena a la comunicación puede capturarla y usarla con posterioridad de forma fraudulenta para suplantar la identidad del usuario. Además de lo anterior, en las diferentes implementaciones del protocolo se han encontrado otras vulnerabilidades, por lo que como norma general su uso no está recomendado.

11.4.2. SSH

Para solventar las debilidades de TELNET en la actualidad se usa el protocolo SSH («Secure SHell»), el cual, implementado sobre el puerto 22, proporciona autenticación y privacidad en las comunicaciones desarrolladas (para más detalles sobre estos y otros aspectos de seguridad véase el Capítulo 12).

¹ Ver técnicas de delimitación basadas en relleno de caracteres y de bits descritas en el Capítulo 3.

SSH es un protocolo para acceder a un «shell» (o consola) remota de forma segura. Para ello no solo se cifra el intercambio de información, sino que se incorporan los procedimientos más recientes de autenticación. SSH proporciona además otros servicios como la transferencia segura de ficheros — con el protocolo asociado SFTP («Secure File Transfer Protocol»)—, la redirección de puertos y túneles de sesiones de ventanas X para ejecutar entornos GUI («Graphical User Interface») en servidores remotos. SSH puede usar diferentes esquemas de cifrado (como por ejemplo AES, 3DES, etc., también explicados en el Capítulo 12).

De acuerdo con el RFC 4251 la arquitectura de SSH define tres capas:

- Transporte: esta capa autentica al servidor (es decir, incluye procedimientos que garantizan que la identidad del servidor es la indicada), se garantiza la confidencialidad de la información intercambiada y su integridad. Además, esta capa es la responsable del intercambio de claves de cifrado y de configurar el cifrado a usar, los algoritmos de compresión y de integridad (ver Apartado 12.3.1).
- Autenticación de usuario: en esta capa se autentica al cliente, para ello se prevén una serie de métodos de autenticación, que van desde los más sencillos basados en palabra de paso («password»), a otros más complejos como Kerberos 5 (Figura 12.30).
- Conexión: una conexión SSH puede incluir múltiples canales lógicos (transmitiendo datos en las dos direcciones), incluyendo además canales de señalización (denominados «channel requests») para enviar información de control (a través de los cuales enviar códigos de control o parámetros de configuración) y túneles.

Para autenticar a los hosts, SSH define los registros de recursos de DNS (ver Apartado 11.3) tipo SSHFP, los cuales contienen su clave pública. Para obtener más detalles sobre SSH, su especificación, funcionamiento y uso, consultar la bibliografía recomendada.

11.5. Servicio de correo electrónico

Uno de los servicios en Internet más relevantes desde sus inicios es el de correo electrónico o *email* (del inglés «electronic mail»). Emulando al correo postal, esta aplicación permite el envío de mensajes (electrónicos) entre usuarios finales. Como se muestra en la Figura 11.16, en un sistema *email* se identifican las siguientes entidades:

- *Agente de usuario* (MUA, «Mail User Agent»): es la interfaz entre el sistema de correo y el usuario, permitiendo a este último la lectura y escritura de mensajes electrónicos. Denominados también *clientes de mensajería*, algunos MUA comerciales existentes son *Mozilla Thunderbird*, en sistemas Linux, y *Outlook*, en sistemas Windows.
- *Agente de transporte* (MTA, «Mail Transport Agent»): encargado de transmitir y recibir el correo electrónico interno y externo al sistema, este módulo también se conoce como *estafeta de correo*. Como ejemplo de MTA de amplio uso podemos mencionar *sendmail*, de uso principal en entornos Unix.

Cuando un usuario crea un mensaje *email* a través de su MUA, este lo almacena en una zona del sistema conocida como *spool de correo*; de ella será leído por el MTA para su envío (en segundo plano o «background») hacia el destino. Por su parte, cuando el MTA recibe un *email*, lo almacena en una zona del sistema denominada *buzón* o *destinatario de correo*; a ella deberá acceder el usuario a través del MUA para su lectura.

Un destinatario de correo se indica a través de lo que se conoce como *dirección de correo electrónico*, la cual se especifica generalmente en la forma *nombre@dominio*, donde se utiliza el carácter

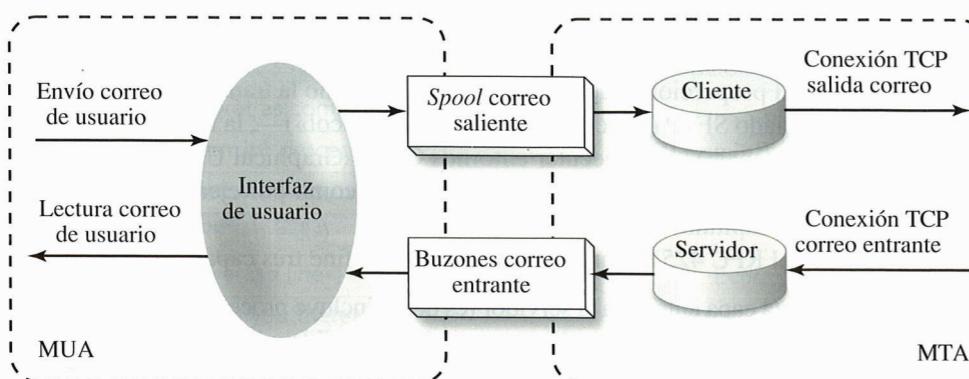


Figura 11.16. Esquema conceptual de un sistema de correo electrónico.

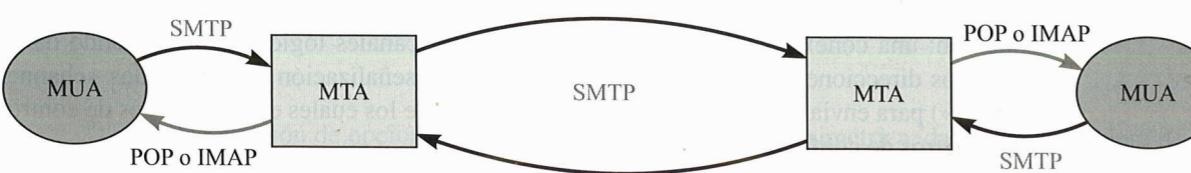


Figura 11.17. Entidades y protocolos involucrados en el servicio e-mail.

@ («arroba») como separador entre el nombre del buzón como tal y el del *host* o dominio en el que este se encuentra (ver registros RR tipo MX de DNS en el Apartado 11.3). Un ejemplo de dirección de *email* es *juanjose1975@gmail.com*, donde *juanjose1975* es el nombre del buzón y *gmail.com* el nombre del dominio de la MTA.

En el servicio de correo se identifican dos acciones claramente diferenciadas que involucran a dos (o más) protocolos. Por un lado, se encuentra el procedimiento de envío de correo que puede ser ejecutado tanto en el MUA como en la MTA (Figura 11.17). En este caso el protocolo especificado es el SMTP, «Simple Mail Transfer Protocol»; y el procedimiento de consulta o lectura del buzón de correo entrante. En este último caso hay dos protocolos especificados, POP e IMAP (Figura 11.17) que pueden usarse de forma indistinta. Además de los anteriores, WebMail es una alternativa (en realidad, puede considerarse un recubrimiento) que simplifica la administración, ya que reduce el número de puertos abiertos necesarios, además de proporcionar un interfaz universal con independencia de dónde se ejecute el MUA. WebMail consiste en usar el protocolo HTTP (estudiado más adelante en el Apartado 11.6) como contenedor de los mensajes correspondientes a POP, IMAP o SMTP.

El servicio de correo electrónico se especifica formalmente en dos partes: un formato concreto de generación de los mensajes y un protocolo de transmisión de los mismos. A continuación se describen ambas cuestiones.

11.5.1. Formato de los mensajes email

El formato aceptado para los mensajes de correo electrónico se especificó inicialmente en el RFC 822, y posteriormente en 2822 y 5322. Todo mensaje *email* consta de dos partes:

- Cuerpo («mail body»): datos del mensaje propiamente dicho.

— Datos administrativos: conjunto de información necesaria para el envío y la gestión del mensaje. Dentro de los datos administrativos, o cabecera, encontramos dos grupos:

- *De transporte* («mail envelope»), tales como el destinatario y el remitente, necesarios para el envío correcto del mensaje.
- *Otros datos*, tales como la fecha, de carácter complementario.

Cada uno de los campos que componen los datos administrativos de un mensaje *email* se identifica con una secuencia de caracteres ASCII seguidos del carácter «:». Entre los campos de cabecera más usuales podemos mencionar los siguientes:

- From: → Campo donde se indica el remitente del mensaje.
- To: → Campo donde se especifica el destinatario o destinatarios del mensaje.
- Cc: → Del inglés «carbon copy», este campo indica el destinatario o destinatarios a los que se envía una copia del mensaje.
- Subject: → Descripción del contenido del mensaje.
- Keywords: → Palabras clave seleccionadas por el usuario para resumir el mensaje.
- Message-ID: → Identificador único del mensaje.
- Date: → Fecha de envío del mensaje.
- In-Reply-To: → Identificador del mensaje al que se refiere el actual.
- Return-Path: → Ruta de regreso al remitente.

11.5.2. Protocolo simple de transferencia de correo electrónico (SMTP)

El protocolo para el envío de los mensajes de correo electrónico es SMTP («Simple Mail Transfer Protocol»), —Figura 11.17—, el cual se especificó inicialmente en el RFC 821 y posteriormente en los RFC 2821 y 5321, a las que han seguido algunas actualizaciones como los RFC 5335 y 5336. Dos son las principales características a reseñar de SMTP:

1. Tal como se indica en la Figura 11.16, SMTP usa el protocolo TCP. Además, el puerto estándar reservado para el servicio es el 25.
2. A pesar de la consideración ocasional de *pasarelas de correo* («mail gateway»), dispositivos intermediarios a los que se recurre en ocasiones para el almacenamiento temporal y retransmisión de los mensajes, la transmisión de los mensajes *email* se suele realizar mediante una conexión directa entre los MTA origen y destino involucrados (Figura 11.17).

Como otros protocolos de aplicación, SMTP es orientado a texto. Es decir, el cliente formula solicitudes (o comandos) hacia el servidor, una vez establecida la conexión TCP correspondiente entre ambos; y el servidor devuelve al cliente el resultado de la ejecución de la solicitud previa. Los comandos o solicitudes SMTP más usuales son los siguientes (ver RFC 821):

- HELLO (HELO): Identificación del *host* cliente.
- MAIL: Especificación del remitente del mensaje.
- RECIPIENT (RCPT): Especificación del (buzón) destinatario del mensaje.
- DATA: Datos del mensaje (cuerpo y cabecera).
- QUIT: Solicitud de cierre de la conexión.
- HELP: Comando de ayuda.
- EXPAND (EXPN): Expande una lista de correo.
- VRFY: Permite verificar la existencia de un buzón en el *host* servidor.
- NOOP: Comando de «no operación», usado para solicitar una respuesta de eco.
- RESET (RSET): Reinicio de la transacción actual.

En el RFC 2821 se especificó una extensión de SMTP conocida como ESMTP («Extended SMTP»), de la cual hemos de destacar:

- Se identifica por usar el comando EHLO, en lugar HELO.
- Especificación por parte del servidor, en respuesta al comando EHLO enviado por el cliente, de la lista o conjunto de extensiones aceptadas.
- Inclusión de parámetros adicionales para los comandos MAIL y RCPT.
- Sustitución de algunos comandos como DATA para transmisiones no ASCII (ver Apartado 11.4.3).

En la Figura 11.18 se muestra un ejemplo de servicio *email* mediante la ejecución de TELNET al puerto 25 del servidor de correo goliat.ugr.es. En ella hemos de señalar que, las respuestas proporcionadas por el servidor SMTP son del tipo *XYZ texto*, siendo *XYZ* un código numérico que identifica la respuesta del servidor, mientras que *texto* es una breve explicación ASCII de la respuesta para su comprensión por parte del usuario.

```
#> telnet goliat.ugr.es 25
Trying 150.214.20.3...
Connected to goliath.ugr.es.
Escape character is '^>'.
220 goliat.ugr.es ESMTP Sendmail 8.9.3/8.9.1 (IRIS 3.0); Wed, 5 Mar 2003 12:17:19
+0100 (MET)
ehlo fobos.ugr.es
250-goliat.ugr.es Hello fobos.ugr.es [150.214.190.83], pleased to meet you
250-8BITMIME
250-SIZE 8000000
250-DSN
250-ONEX
250-ETRN
250-XUSR
250-HELP
help
214-This is Sendmail version 8.9.3
214-Topics:
214- HELO EHLO MAIL RCPT DATA
214- RSET NOOP QUIT HELP VRFY
214- EXPN VERB ETRN DSN
214-For more info use "HELP <topic>".
214-To report bugs in the implementation contact Sun Microsystems
214-Technical Support.
214-For local information send email to Postmaster at your site.
214 End of HELP info
mail from: usuario@fobos.ugr.es
250 usuario@fobos.ugr.es... Sender ok
rcpt to: pgteodor@ugr.es
250 pgteodor@ugr.es... Recipient ok
data
354 Enter mail, end with "." on a line by itself
Subject: Resumen del mensaje
Cc: usuario2@dominio
Cuerpo del mensaje ...
.
250 MAA21341 Message accepted for delivery
quit
221 goliat.ugr.es closing connection
Connection closed by foreign host.
#> -
```

Figura 11.18. Ejemplo de envío de correo electrónico haciendo uso directo de los comandos SMTP. En negrita se indican los comandos introducidos desde el host cliente.

11.5.3. Extensiones MIME

El protocolo SMTP original presenta algunas limitaciones, entre las que hemos de destacar dos:

1. Solo permite el envío de mensajes ASCII.
2. La longitud de las líneas de datos tiene un límite máximo de 1.000 caracteres US-ASCII.

Queda patente, pues, el hecho de que con SMTP tal y como se ha descrito no sería posible la transmisión de mensajes multimedia en los que pueden aparecer imágenes, audio, gráficos, etc. Para resolver esta situación, los RFC 2045 a 2049 (y sus correspondientes actualizaciones) constituyen lo que se denomina *extensiones multipropósito de correo Internet* (MIME, «Multipurpose Internet Mail Extensions»), en las cuales se redefine el formato de los mensajes de correo electrónico de manera que:

1. El cuerpo y cabeceras de los mensajes puedan estar formados por caracteres no ASCII.
2. Se acepte un conjunto extensible de formatos para los mensajes.
3. Los mensajes puedan estar constituidos por varias partes independientes y, posiblemente, con formatos distintos.

Algunas de las cabeceras *email* adicionales definidas por MIME son las siguientes:

- MIME-Version: → Número de la versión de MIME.
- Content-Description: → Cadena de texto que describe el contenido del mensaje.
- Content-Id: → Descriptor de mensaje análogo a Message-Id en SMTP.
- Content-Type: → Formato de los datos contenidos en el mensaje.
- Content-Transfer-Encoding: → Codificación utilizada.

Tabla 11.2. Tipos y subtipos aceptados para la cabecera Content-Type de MIME

Tipo	Subtipo
text	plain
	richtext
image	gif
	jpeg
audio	basic
	mpeg
video	octet-stream
	postscript
application	rfc822
	partial
message	external-body
	mixed
multipart	alternative
	parallel
x-nuevotipo	--

A continuación se comentan brevemente las dos últimas cabeceras mencionadas: Content-Type y Content-Transfer-Encoding. La primera ellas tiene el siguiente formato:

Content-Type: tipo/subtipo [; parámetro_1; parámetro_2; ...]

donde cada *parámetro_i* es opcional y de formato *atributo_i=valor_i*. En la Tabla 11.2 se indican distintos valores aceptados para las variables *tipo* y *subtipo*. Estos son los siguientes:

- **text**: mensaje tipo texto. Los subtipos aceptados son: plain, para texto sin formato, y richtext, para texto enriquecido. Algunos parámetros que pueden especificarse para este tipo de formato son name=*nombre*, que permite especificar un nombre de fichero, y charset=*conjunto*, que indica el conjunto de caracteres utilizado (por ejemplo, us-ascii, iso-8859-1, etc.).
- **image**: el mensaje es una imagen, la cual puede ser GIF o JPEG; es decir, *subtipo=gif* o *subtipo/jpeg*.
- **audio**: mensaje de audio para el que solo se acepta el subtipo basic, con el que se indica que las muestras son mono, PCM y la frecuencia de muestreo 8000 Hz.
- **video**: animación en la que se acepta el subtipo mpeg, indicativo de la codificación de los datos mediante el estándar de este nombre.
- **application**: los datos no se ajustan a ninguna de las categorías anteriores, estando procesados por algún tipo de aplicación. Dentro de los subtipos aceptados en este tipo encontramos octet-stream, para indicar una secuencia de bytes no interpretada, y postscript, correspondiente a un documento con este formato.
- **message**: tipo usado para indicar el encapsulado de un mensaje dentro de otro. En particular, el subtipo rfc822 hace referencia a un mensaje SMTP que sigue el RCF 822 ya estudiado. El subtipo partial significa que el mensaje ha sido dividido para su transmisión, mientras que external-body indica que el mensaje debe obtenerse de la red.
- **multipart**: a través de este tipo se indica que el mensaje está compuesto por varios conjuntos de datos. Los subtipos aceptados son:
 - **mixed**: las partes son independientes y necesitan ser tratadas en un orden particular.
 - **alternative**: cada parte es una versión distinta de la misma información.
 - **parallel**: todas las partes pueden tratarse simultáneamente si es posible.
 - **digest**: usado para enviar conjuntos de mensajes que siguen el estándar RFC 822.

Un parámetro básico en un mensaje multipart es boundary=*secuencia*, el cual especifica la secuencia patrón de caracteres utilizada para separar las distintas partes que componen el mensaje.

— **x-nuevotipo**: donde *nuevotipo* se refiere a un tipo específico distinto a los estándares ya mencionados. Por supuesto, los subtipos aceptados dependerán del tipo concreto.

La cabecera Content-Transfer-Encoding puede tomar los siguientes valores, correspondientes todos ellos a distintos mecanismos de codificación de datos contemplados en MIME:

- **7bit/8bit/binary**: indica que no se ha realizado codificación alguna.
- **quoted-printable**: todos los caracteres del mensaje se representan mediante caracteres ASCII imprimibles. Cuando el carácter a codificar tiene un valor decimal comprendido entre 33 y 126 (ambos incluidos y excluyendo el valor 61), el dato codificado será el mismo. En cambio, si el carácter tiene otro valor el resultado codificado estará formado por el carácter '=' seguido por el valor hexadecimal del carácter. Por ejemplo, CRLF se codificará como =0D=0A.

Tabla 11.3. Conjunto de caracteres imprimibles base64, además de '='

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

— **base64**: esquema de codificación en el que se usa un subconjunto de US-ASCII de 65 caracteres imprimibles (64 más '=') —ver Tabla 11.3—, de modo que cada grupo de 24 bits del mensaje de entrada se codifica como una cadena de 4 caracteres imprimibles (6 bits por carácter).

Cada línea del mensaje codificado tiene un máximo de 76 caracteres, utilizándose el carácter '=' para codificar la última línea de la siguiente forma:

- Si quedan exactamente 24 bits, la salida estará formada por 4 caracteres y no se utilizará al carácter '='.
- Si quedan 8 bits, se añaden a la derecha 16 bits a valor 0, de forma que la salida estará formada por los dos caracteres que correspondan más dos caracteres '='.
- Si quedan 16 bits, se añaden a la derecha 8 bits a valor 0 y se genera una salida constituida por los tres caracteres correspondientes más un carácter '='.

— **x-nuevotipo**: permite de forma fácil la consideración de tipos particulares distintos de los estándares anteriores.

11.5.4. Protocolos de correo de entrega final

Las funciones propias de un MTA o agente de transporte de correo electrónico son:

— Por lo que se refiere al correo local:

1. redireccionar los mensajes si fuese preciso,
2. añadir el mensaje al *spool* de correo del destinatario y
3. devolver los mensajes que no lleguen a su destino, junto con un mensaje de error (*bouncing*).

— Por su parte, para el correo remoto o externo, el MTA debe, además, determinar el camino a seguir por el mensaje.

Es fácil concluir, pues, que la implementación de una estafeta de correo o MTA en cada uno de los *hosts* de una red resulta imprácticable por el alto consumo de recursos que ello supondría. Además, implicaría un fuerte acoplamiento temporal, es decir los dos *hosts* deberían estar disponibles simultáneamente. Para resolver este problema aparecen los servicios de correo de entrega final, a través de los cuales se permite a un usuario contactar con una estafeta de correo remota a fin de acceder a sus mensajes de *email*.

Los protocolos de correo de entrega final más extendidos son:

- POP («Post Office Protocol»): inicialmente era una especificación denominada «*download and delete*», es decir, tras la descarga del correo electrónico correspondiente a un usuario el *host* local lo elimina del servidor.
- IMAP («Internet Message Access Protocol»): más avanzado y complejo que POP, IMAP posibilita la gestión «virtual» del buzón correspondiente a un usuario como si esta se realizase directamente desde el *host* servidor. En este sentido, a diferencia de como sucede en POP, IMAP no implica el «volcado» del buzón al *host* cliente.

A continuación se describen los protocolos de correo comentados, ambos basados en TCP.

POP

Especificado inicialmente en el RFC 918, la especificación actual corresponde al RFC 1939, al que le ha seguido alguna actualización. La versión actual de POP es la 3, denominándose al protocolo POP3. Una vez establecida la conexión TCP entre el cliente y el servidor correspondientes, los estados por los que pasa una conexión POP3 son:

1. *Autorización*: corresponde al proceso de identificación del cliente.
2. *Transacción*: estado en el que, a través de solicitudes por parte del cliente, se accede al correo electrónico del usuario especificado en el proceso anterior.
3. *Actualización*: tras la ejecución del comando QUIT por parte del cliente, el servidor elimina los recursos adquiridos durante el estado anterior y cierra la conexión TCP.

Los comandos (ASCII) POP3, emitidos por el usuario cliente, son los siguientes (los dos primeros se utilizan en el estado de autorización):

- USER: Identificación del usuario cliente.
- PASS: Palabra de paso para la autenticación del usuario.
- QUIT: Finalizar servicio.
- LIST: Listado de mensaje/s en el buzón.
- RETR: Obtención de un mensaje.
- DELE: Borrado de un mensaje.
- RSET: Reinicialización del servicio.
- STAT: Estadística de mensajes.
- NOOP: No operación.

Por su parte, las respuestas proporcionadas por el servidor POP3 son de la forma +OK *texto* ó -ERR' *texto*, donde *texto* es una secuencia de caracteres ASCII explicativa del resultado, exitoso o no, respectivamente, de la ejecución del comando en cuestión solicitado.

En los RFC 1957 y 2449 podemos encontrar extensiones a POP3. Estas consisten en la definición de comandos adicionales como AUTH, utilizado para la selección de un sistema de autenticación del cliente, y CAPA, a través del cual se especifican las capacidades extra soportadas (códigos de respuesta extendidos, diferentes mecanismos de autenticación aceptados, etc.)

```
#> telnet goliat.ugr.es 110
+OK QPOP (version 2.4) at goliat starting.
AUTH KERBEROS_V4
-ERR This command is not supported yet
USER usuario
+OK Password required for usuario.
PASS clave
+OK usuario has 2 messages (320 octets).
STAT
+OK 2 320
LIST
+OK 2 messages (320 octets)
1 120
2 200
.
RETR 3
-ERR Message 3 does not exist.
RETR 2
+OK 120 octets
<mensaje1> ...
.
DELE 2
+OK Message 2 has been deleted.
QUIT
+OK Pop server at goliat signing off.
#>
```

Figura 11.19. Ejemplo de servicio POP3. En negrita se indican los comandos introducidos desde el host cliente.

En la Figura 11.19 se muestra un ejemplo del servicio POP3 desarrollado en base a TELNET sobre el puerto propio del servicio POP, el 110.

IMAP

POP era inicialmente un protocolo «*download and delete*»; es decir, el correo del usuario se descarga al *host* cliente y se elimina del servidor. En cambio, IMAP permite manipular los mensajes en el servidor de forma equivalente a como se haría localmente en él, manteniéndolos en el buzón remoto tras el cierre del servicio.

La versión actual de IMAP es la 4, conociéndose al protocolo como IMAP4. Detallado inicialmente en el RFC 1730, su última versión se describe en el RFC 3501. Un servicio IMAP se desarrolla en cuatro estados (Figura 11.20):

- *No-autenticado* (NA): se ha establecido la conexión pero el usuario no se ha autenticado.
- *Autenticado* (A): el usuario se ha autenticado y debe seleccionar un buzón de correo.
- *Seleccionado* (S): el usuario ha seleccionado adecuadamente un buzón de correo.
- *Desconexión* (D): la sesión finaliza, cerrando el servidor la conexión.

Los comandos que puede utilizar un cliente IMAP en este servicio se hacen preceder por un identificador o *tag* alfanumérico (por ejemplo, a001) y pueden agruparse como sigue:

- Comunes a todos los estados:
 - CAPABILITY: determina las capacidades del servidor.
 - NOOP: comando de «no operación».
 - LOGOUT: finalizar servicio.

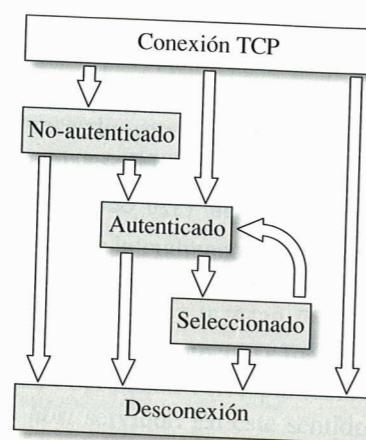


Figura 11.20. Estados y transiciones en una conexión IMAP.

— Propios del estado NA:

- LOGIN: comando para la identificación del usuario a través de la especificación de una palabra de paso.
- AUTHENTICATE: a diferencia de LOGIN, este comando permite establecer el mecanismo de autenticación a considerar.

— En el estado A existen, entre otros, los siguientes comandos:

- SELECT: selecciona un buzón.
- CREATE: permite la creación de un buzón.
- DELETE: elimina un buzón.
- LIST: devuelve un conjunto de nombres disponibles para el usuario.
- APPEND: añade un mensaje al buzón.
- UN/SUBSCRIBE: des/activa un buzón.

— En el estado *seleccionado* (S) se consideran, entre otros, los siguientes comandos:

- CHECK: comprobación de buzón.
- CLOSE: retorno al estado A.
- SEARCH: búsqueda de cadena de texto en los mensajes del buzón.
- FETCH: obtiene datos asociados a un mensaje del buzón.
- STORE: modifica datos asociados a un mensaje del buzón.
- COPY: copia de mensaje en buzón.

— También se aceptan comandos experimentales con el formato Xcomando.

Una vez ejecutado un comando solicitado por el cliente IMAP4, la respuesta proporcionada por el servidor puede ser de tres tipos:

- *De estado*, a través de la cual se indica el resultado de la ejecución del comando. Las posibles respuestas en este caso son: OK, BAD, NO, PREAUTH y BYE.
- *De datos*, proporcionada para comandos como CAPABILITY, LIST o SEARCH.
- *De solicitud de continuación de comando*, mediante la que el servidor indica al cliente la continuación de un comando previo.

En los RFC 2060 y 2061 se detalla la revisión 1 de IMAP4 (IMAP4rev1), que se corrige en el RFC 3501, en la que se incluyen varios comandos entre ellos, STATUS, utilizado para la obtención del estado de un buzón, y la respuesta al mismo proporcionada por el servidor; y STARTTLS, que inicia una negociación TLS (ver Apartado 12.3.2) para la transferencia segura de información. Como hemos procedido para POP3, en la Figura 11.21 se muestra un ejemplo de servicio IMAP4 a través de la ejecución del servicio TELNET al puerto estándar de IMAP, el 143.

Como conclusión al estudio de POP e IMAP, diremos que ambos protocolos presentan las siguientes características comunes:

1. Son abiertos, definidos en los RFC.
2. Implican la existencia de un servidor compartido por los usuarios.
3. Posibilitan la accesibilidad del servicio de correo desde diversas plataformas de cliente y desde cualquier punto en la red.
4. Ambos soportan operación *offline*.
5. Existen implementaciones cliente/servidor fuentes disponibles para distintas plataformas.
6. No precisan gateways de correo SMTP.

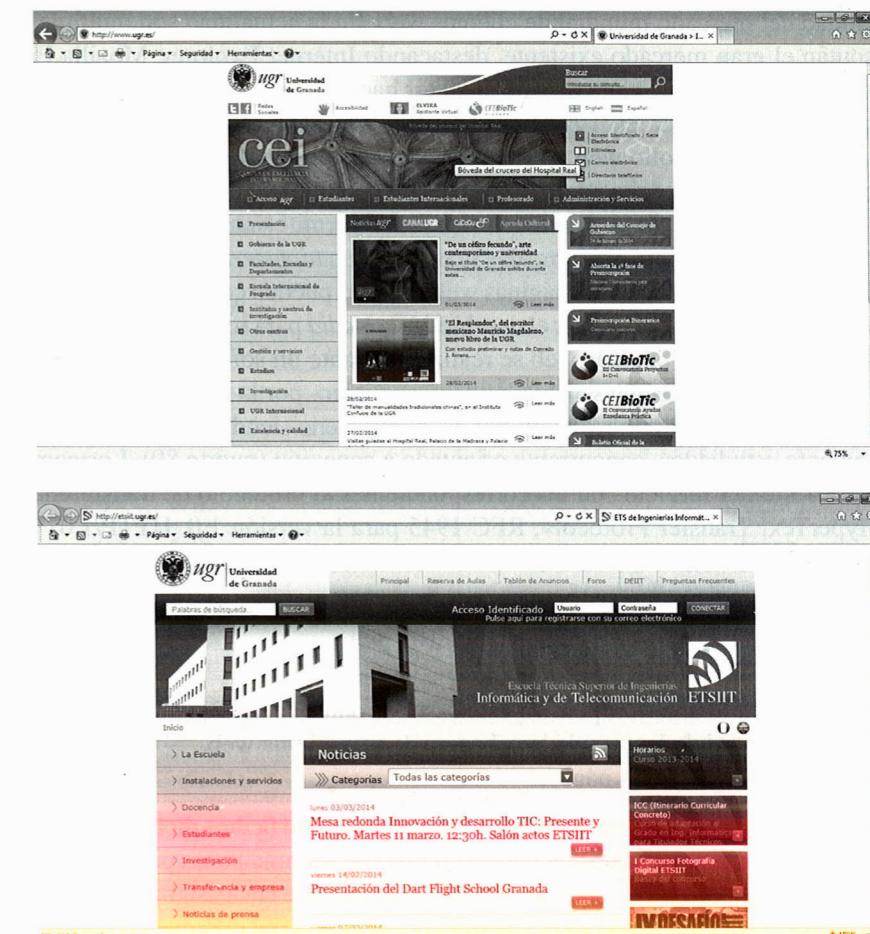


Figura 11.21. Páginas Web la ETSI de Ingenierías Informática y Telecomunicación y de la Universidad de Granada.

POP es más sencillo y fácil de implementar que IMAP. Por su parte, IMAP también presenta ventajas sobre POP: (a) permite una gestión de buzones y mensajes más completa, (b) la conexión cliente-servidor es *online*, aunque también se posibilita que esta sea *offline*, (c) permite asociar un estado (borrado, leído, contestado, etc.) a cada mensaje, y (d) permite hacer lecturas parciales (de la cabecera) de los mensajes.

11.6. WWW y transferencia de hipertexto: servicio HTTP

Tim Berners-Lee, investigador del CERN («Centre Européen pour la Recherche Nucléaire»), concibió en 1990 el desarrollo de un servicio para permitir a los científicos compartir información de una manera cómoda a través de la existencia de enlaces entre documentos distribuidos Internet. Este sistema es lo que ha venido en denominarse como WWW («World Wide Web», la *telaraña mundial*) o, simplemente, web.

Tras el primer prototipo de tipo texto aparecido en 1991, en 1993 salió a la luz *Mosaic*, el primer sistema Web con interfaz gráfica. La enorme repercusión que provocó este novedoso sistema fue el germen de la creación en 1995 de la empresa Netscape Communications Corp., creadora del software con el mismo nombre que, junto a *Internet Explorer*, de Microsoft, acaparó durante bastante tiempo la casi totalidad del mercado mundial de clientes web. En la actualidad son múltiples los navegadores que se disputan el gran mercado existente, destacando Internet Explorer, Mozilla Firefox y Chrome. También puede resultar interesante de cara al lector hacer referencia al consorcio WWW (<http://www.w3.org>), organización creada en 1994 por el CERN y el MIT («Massachusetts Institute of Technology») y dedicada al desarrollo web y estandarización de protocolos relacionados con este servicio.

Desde el punto de vista del usuario, la web es un conjunto de documentos, llamados *páginas*, que contienen enlaces a otras páginas, gracias a los cuales se permite «navegar» entre ellas mediante un simple ‘click’ de ratón. Las páginas así formadas se denominan *hipertexto* y a los enlaces en ellas existentes *hiperenlaces*. Además de enlaces, las páginas web actuales pueden contener gráficos, audio, vídeo, aplicaciones, etc.; por ello, se suele utilizar el nombre *hipermedia* para referirse a ellas (véase Figura 11.22).

La visualización de las páginas web se realiza mediante un software cliente denominado *navegador* (del inglés «browser»), gracias al cual podemos «viajar» por la telaraña global de información que constituye la WWW. Por su parte, el servidor es, como viene siendo habitual en los servicios Internet hasta el momento estudiados, concurrente orientado a conexión (puerto 80). Los comandos involucrados en el desarrollo del servicio son los correspondientes al protocolo de transferencia de hipertextos HTTP («HyperText Transfer Protocol», RFC 1945 para la versión 1.0, HTTP/1.0, y RFC 2616 para la versión 1.1, HTTP/1.1), a los cuales, en este contexto, se les denomina *métodos*. Entre ellos podemos mencionar los siguientes:

- GET: solicitud de lectura de una página web dada.
- PUT: solicitud de envío o escritura de una página web.
- POST: similar al anterior, pero en este caso los datos se añaden a la página indicada.
- HEAD: solicitud de lectura de la cabecera de una página web.
- DELETE: borrado de una página.

Una cuestión fundamental acerca del servicio web es, como hemos dicho, la existencia de enlaces a otras páginas. La cuestión que se nos plantea es: ¿cómo se refieren las páginas de modo que no existan problemas en su localización y acceso? La solución consiste en la asignación de un URL («Uniform Resource Locator»), ya presentado en el Capítulo 8 y el cual no es más que un nombre cuya función es identificar un recurso de forma fácil. El formato de un URL es *método_acceso:localización_recurso*, definiéndose inicialmente en los RFC 1738 y 1808 los siguientes esquemas de acceso:

- [http://servidor\[:puerto\]/página](http://servidor[:puerto]/página)
- [ftp://servidor\[:puerto\]/camino/fichero](ftp://servidor[:puerto]/camino/fichero)
- [telnet://servidor\[:puerto\]](telnet://servidor[:puerto])
- <mailto:usuario@dominio>

donde el formato cursiva significa «valor a especificar» y [] denota el carácter opcional del número de puerto a considerar, tomando este parámetro por defecto los valores predeterminados en cada caso: 80 para HTTP, 23 para TELNET, 25 para SMTP, 21 para FTP, etc.

Otros conceptos importantes relacionados con el servicio web son los de URI («Universal Resource Identifier») y URN («Uniform Resource Name») —ver RFC 1630, 1737 y 2141—. El primero es una generalización de los URL, definiendo una forma de referenciar un recurso en cualquier espacio de nombres registrado, no solo en los actualmente conocidos. Por su parte, un URN hace referencia a un identificador de recursos genérico frente al específico que supone el uso de URL, lo que permite separar el nombre de un recurso dado de su localización concreta. La especificación más reciente para los URI se encuentra en el RFC 3986.

¿Cómo se crea una página web? Aunque esta cuestión está fuera del interés de este texto, a continuación se presenta un breve comentario al respecto. La herramienta básica utilizada para ello es HTML («HyperText Markup Language»), el cual no es un lenguaje de programación tradicional, sino más bien un conjunto de etiquetas o marcas (*tags*) a través de las cuales se especifica el formato de visualización de un documento. En la Tabla 11.4 se enuncian algunas de las marcas HTML más usuales, la práctica totalidad de las cuales son de la forma `<marca> ... </marca>`; de esta forma, la marca en cuestión actúa sobre los datos existentes entre los delimitadores de principio y de fin.

Tabla 11.4. Ejemplos de etiquetas o tags HTML

Marca	Descripción
<code><HTML> ... </HTML></code>	Principio y fin de la página Web
<code><HEAD> ... </HEAD></code>	Cabecera de la página
<code><TITLE> ... </TITLE></code>	Título de la página (en la cabecera)
<code><BODY> ... </BODY></code>	Cuerpo de visualización de la página Web
<code><Hn> ... </Hn></code>	Nivel n de encabezamiento
<code> ... </code>	Negrita
<code><I> ... </I></code>	Itálica
<code> ... </code>	Lista no numerada de ítems
<code> ... </code>	Lista numerada de ítems
<code>
</code>	Nueva línea
<code><P> ... </P></code>	Delimitación de párrafo
<code><HR></code>	Línea horizontal
<code></code>	Carga de imagen
<code> ... </code>	Hiperenlace
<code><TABLE> ... </TABLE></code>	Tabla
<code><FORM> ... </FORM></code>	Permite la introducción de datos por parte del usuario
<code><FRAME> ... </FRAME></code>	Creación de la página en base a ventanas
<code><APPLET CODE="..."> ... </APPLET></code>	Enlace a un programa Java

A pesar de la existencia de la marca FORM, HTML se suele describir como un lenguaje que define páginas web estáticas; es decir, el contenido de estas es fijo. Indudablemente, este hecho supone una limitación importante por cuanto que impide la interacción usuario cliente→servidor. Para solucionar este problema aparecen herramientas como CGI («Common Gateway Interface»), que permite la ejecución de programas que hacen las veces de interfaz entre información dinámica y la web, o Java, lenguaje portable similar a C++ creado por Sun Microsystems y que permite la ejecución dinámica de programas en el *browser* del cliente. Relacionado con HTML hemos de mencionar el lenguaje XML («eXtensible Markup Language»), que se caracteriza por permitir al usuario definir marcas de forma dinámica y la más reciente especificación HTML 5, en la que se han introducido mejoras para dar mejor soporte a contenidos multimedia, además de mejorar la interoperabilidad entre navegadores e introducir una API para facilitar el desarrollo de aplicaciones dinámicas.

11.7. Aplicaciones multimedia en Internet

Las aplicaciones multimedia tales como videoconferencia, vídeo bajo demanda o voz sobre IP (VoIP, «Voice over IP») son servicios muy demandados y altamente consolidados en la actualidad. Aunque un estudio en detalle acerca de este tipo de aplicaciones precisa mucho más espacio del aquí disponible, no nos gustaría finalizar este tema sin apuntar algunas consideraciones importantes sobre ellas. Las principales características de este tipo de servicios son las siguientes:

1. Elevado consumo de ancho de banda.
2. Son relativamente tolerantes a pérdidas.
3. Necesitan retardos extremo a extremo (latencia) acotados.
4. Necesitan variaciones del retardo también acotadas.
5. En ocasiones usan *multicast*, es decir transmisiones del tipo «uno a muchos».

Por lo que se refiere a la primera de las cuestiones mencionadas, hemos decir que por cuestiones de eficiencia, con independencia de la tecnología de transmisión (fibra óptica, ADSL, etc.) adoptada, es aconsejable recurrir a técnicas de *codificación de la fuente* gracias a las cuales se consiga reducir la cantidad de bits necesarios para representar un conjunto de datos dado. En este sentido, los esquemas de codificación de voz estandarizados más relevantes son, entre otros, G.722, G.723, G.726, AMR, MPEG-1/2/4 audio, Speex, mientras que entre los estándares de codificación de vídeo podemos encontrar los sistemas H.264, HEVC y VP9.

El segundo aspecto apuntado, el relativo a la necesidad del desarrollo de los servicios en tiempo real, resulta especialmente delicado dado el carácter no fiable y no orientado a conexión proporcionado por IP. Para tratar de solventar este problema se han propuesto distintos protocolos cuyo fin último es posibilitar la interactividad de los servicios desarrollados sobre IP. Entre estos protocolos podemos mencionar los siguientes:

1. RTP/RTCP («Real Time Protocol / Real Time Control Protocol», RFC 3550): Proporciona funciones de transporte de datos extremo a extremo en tiempo real.
2. RSVP («Resource reSerVation Protocol», RFC 2205): Protocolo que permite reservar recursos de red para transmisiones de datos tanto *unicast* como *multicast*.
3. RTSP («Real Time Streaming Protocol», RFC 2326): Protocolo para el envío controlado de secuencias de datos con propiedades de tiempo real.
4. SDP («Session Description Protocol», RFC 4566): Protocolo para la descripción de sesiones multimedia.

5. SIP («Session Initiation Protocol», RFC 3261 a 3265): Protocolo de señalización de la capa de aplicación que define la creación, modificación y finalización de sesiones con uno o más participantes.
6. SAP («Session Announcement Protocol», RFC 2974): Anuncio de sesiones multimedia vía *multicast*.

Por último, el envío *multicast* en Internet es un tema de vital importancia por cuanto que el empleo de servicios de transmisión *unicast* (uno a uno) para el tipo de aplicaciones que nos ocupa implicaría un uso no eficiente de recursos. Ahora bien, el problema que se plantea en la transmisión *multicast* es altamente complejo si se desea resolver de un modo óptimo. Dos son los aspectos a considerar acerca de esta cuestión, los cuales han sido ya tratados en el Capítulo 9 del texto:

1. Gestión de los grupos de usuarios suscritos a los distintos servicios, para lo cual se recurre al protocolo IGMP. Adicionalmente a este protocolo podemos mencionar otros relacionados con la asignación y gestión de las direcciones IP de multidifusión, pudiéndose encontrar la arquitectura general considerada en el RFC 6308.
2. Obtención de las rutas de encaminamiento *multicast*. Para ello se suele recurrir a la técnica de árboles de expansión («spanning tree») ya estudiada en el Capítulo 6 del texto. Entre los protocolos utilizados para la definición de los árboles podemos mencionar DVMRP, MOSPF y PIM.

Estos distintos aspectos hacen de este tipo de aplicaciones un problema complejo a la vez que de gran actualidad investigadora. Es por ello que aconsejamos su estudio en cursos de carácter más avanzado y específico que el aquí abordado.

Dentro de las aplicaciones multimedia, por el interés creciente de los usuarios en este tipo de servicios, lo que se traduce en cada vez una mayor cantidad de tráfico generado, en el apartado siguiente se estudia el servicio de vídeo *streaming* más popular: YouTube.

11.7.1. Servicio de vídeo streaming: YouTube

El tráfico de vídeo generado por sitios tales como YouTube, Hulu y Netflix en 2010 alcanzó el 40 % del tráfico total en Internet. De entre todos estos servicios, el tráfico de YouTube es el dominante, alcanzando el puesto número 3 en el ranking de los servidores más visitados.

YouTube es una aplicación que permite a los usuarios ver, subir y compartir vídeos; con ella el usuario no solo consume vídeo sino que adicionalmente puede compartir los contenidos por él generados.

Cuando un usuario solicita un vídeo de YouTube, uno de los múltiples servidores que proporcionan el servicio a través de lo que se conoce como una CDN («Content Delivery Network»), envía al usuario el vídeo con el formato especificado en la solicitud HTTP (transportada sobre TCP) mediante la técnica conocida como *descarga progresiva*. Con esta técnica el reproductor de vídeo en el cliente puede empezar la reproducción del contenido sin necesidad de esperar a recibir el videoclip completo.

Para comprender cómo funciona el servicio, en la Figura 11.23 se muestra un diagrama que explica el funcionamiento general. YouTube asigna a cada vídeo subido por el usuario un identificador único denominado *video id*. Tras ello, el servidor codifica tanto el contenido de audio como el de vídeo con diferentes algoritmos. A continuación, las secuencias de audio y vídeo se encapsulan conjuntamente en un único fichero o contenedor. La combinación del algoritmo de codificación concreto, junto con la resolución del vídeo y el tipo de contenedor empleados, definen lo que se denomina el formato del vídeo. Cada formato de vídeo se identifica con un parámetro denominado *itag*. El proceso de subida

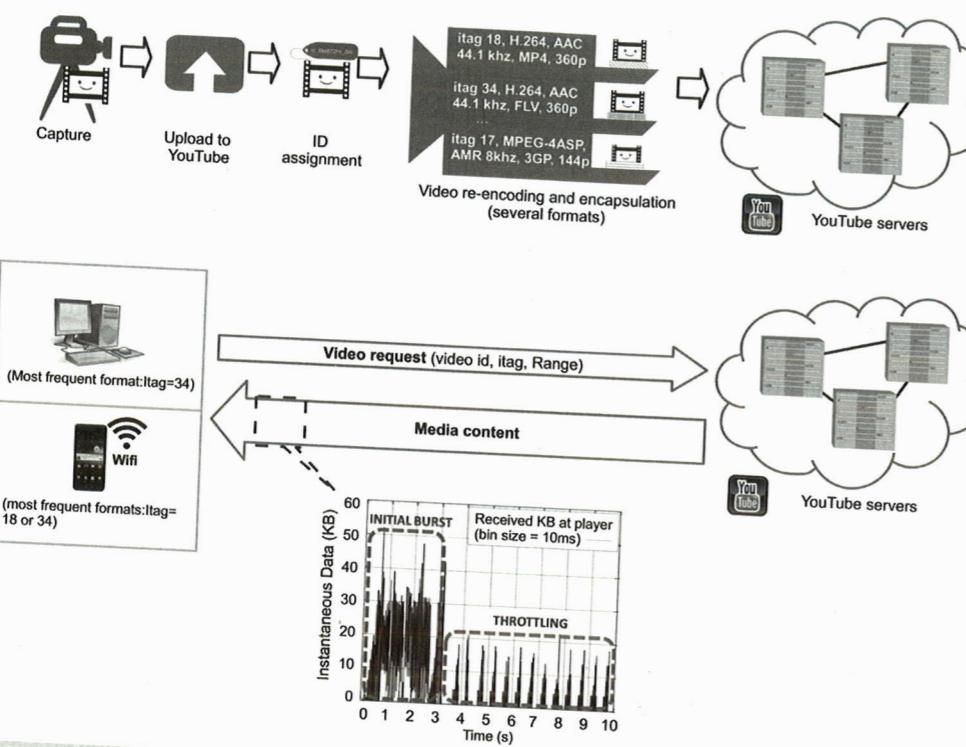


Figura 11.22. Funcionamiento general del servicio de YouTube.

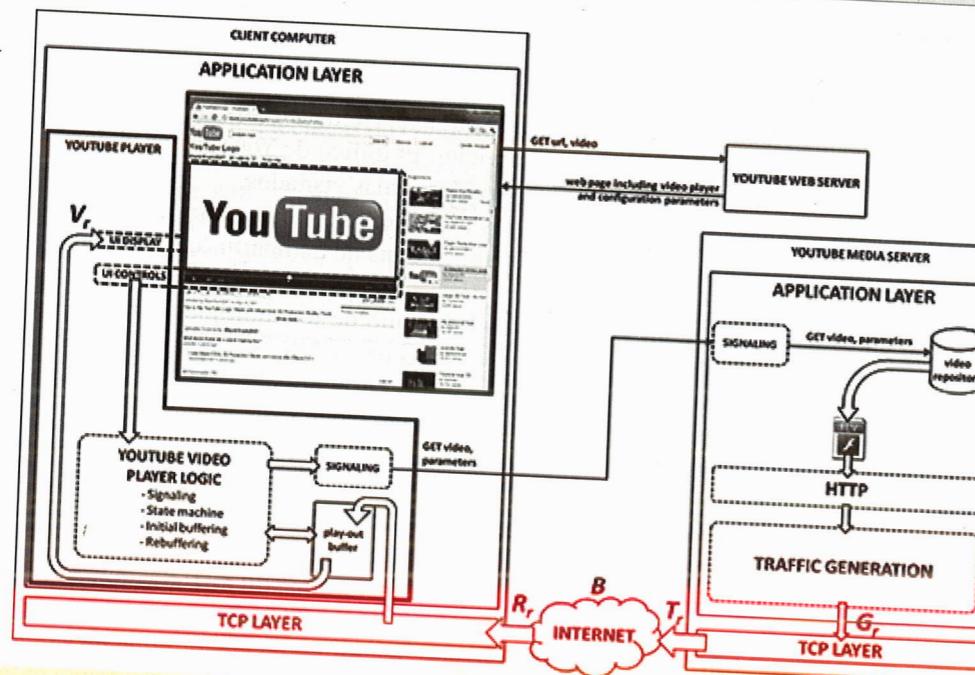


Figura 11.23. Intercambio de datos y señalización entre un cliente y servidor de YouTube.

concluye cuando las versiones diferentes del vídeo clip se almacenan en los correspondientes servidores de YouTube.

El proceso de descarga depende del tipo de dispositivo desde el que se realice la solicitud, así como de la conexión red en particular disponible.

Toda descarga de YouTube comienza con una fase inicial caracterizada por una alta tasa de transferencia, denominada ráfaga inicial o *initial burst*, cuyo objetivo es inyectar una cantidad significativa de datos en el *buffer* del receptor para mitigar futuros episodios de congestión denominados *rebufferings*. En esta fase se envían 40 segundos de vídeo transmitidos al ancho de banda disponible. A continuación le sigue una fase denominada *throttling*, en la que la tasa es igual a 1,25 veces la tasa de codificación del vídeo. De acuerdo con la literatura, es conocido que en el 92 % de los casos para descargas desde un PC el formato corresponde al *tag 34*, formato que corresponde al contenedor de FLV («Flash Video») de Adobe con tasas de codificación variables desde 100 kbps hasta 1 Mbps.

En la Figura 11.24 se muestra un esquema más detallado del servicio, en el que se identifica claramente el papel del cliente y del servidor, así como el intercambio de datos propiamente dicho, además de la señalización.

Cada vídeo almacenado se puede acceder directamente mediante su URL del tipo:

`http://www.youtube.com/watch?v=<video_id>`

o como un objeto embebido en otra página HTML, en este caso usando una URL del tipo:

`http://www.youtube.com/v/<video_id>`

En ambos casos, como se ha comentado, el código HTML contiene un reproductor basado en Adobe Flash que es descargado por el cliente después de que el navegador analice la página.

El reproductor es configurado mediante una serie de parámetros (como el identificador del video clip, el formato del vídeo, y otros), que en el caso de que el videoclip esté embebido en una página web de YouTube se codifican como variables JavaScript; por el contrario, cuando el vídeo está embebido en otra página web, el reproductor obtiene los parámetros mediante una solicitud HTTP del tipo:

`http://www.youtube.com/get_video_info?video_id=<video_id>`

Una vez que el reproductor se ha configurado, este envía una solicitud HTTP al servidor —identificado con una serie de parámetros en el código HTML— y a continuación, como respuesta, la descarga progresiva desde el correspondiente servidor se produce sin más intercambio de señalización posterior. Solo se envía una nueva solicitud si el usuario modifica la posición de la reproducción del vídeo.

La solicitud se envía a la URL del tipo:

`http://<media_server>.youtube.com/videoplayback`

junto con los parámetros de configuración, especificados como variables en la URL. Aunque no están oficialmente documentados, algunos parámetros se han identificado en la bibliografía proporcionada al final del capítulo.

Para la gran mayoría de los videoclips que no son de alta definición, la respuesta HTTP encapsula el vídeo en formato FLV, que incluye una serie de etiquetas para definir las características del contenido encapsulado. Como ejemplo, entre otras etiquetas se define el códec de audio, la frecuencia de muestreo, el tamaño de las muestras y el códec de vídeo. Esta información es esencial para decodificar, reproducir y sincronizar adecuadamente los contenidos.

RESUMEN

Como culminación al desarrollo de la arquitectura de protocolos TCP/IP, este tema ha abordado el estudio de la capa de aplicación. A modo de introducción a todos los servicios Internet descritos en el capítulo se ha presentado el modelo cliente-servidor, comentándose los distintos tipos de servidores existentes y su implementación mediante la interfaz *socket* de la distribución BSD presente en Linux.

El primero de los servicios estudiados ha sido el de nombres de dominio o DNS, a través del cual se facilita la asignación de nombres a direcciones IP para un más cómodo manejo de estas. A continuación se estudia el servicio de acceso remoto. En particular, dado su interés académico e histórico, se explica el protocolo TELNET, el cual ha sido sustituido por el protocolo SSH, del que se proporciona una breve discusión.

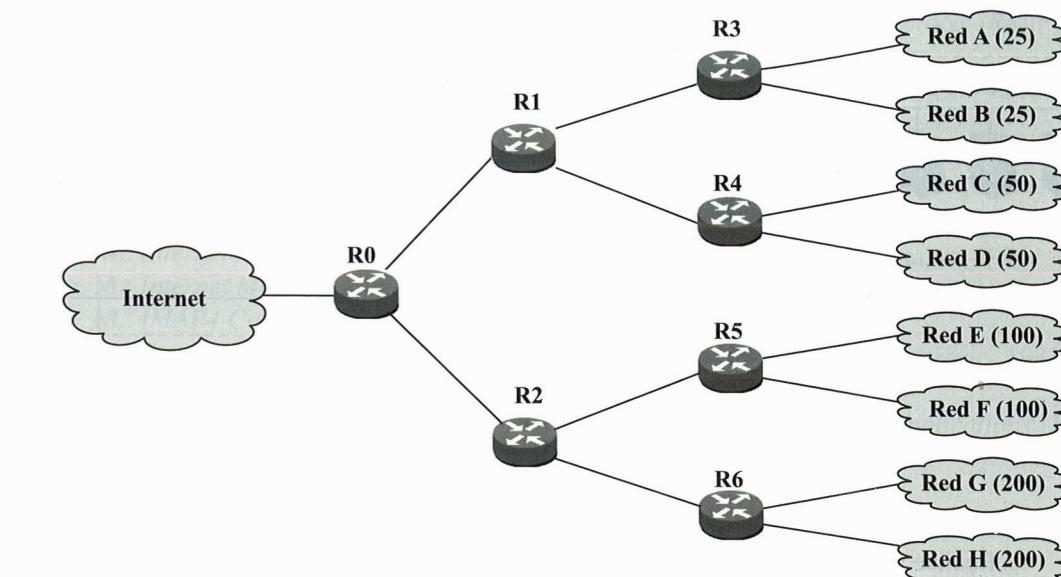
El servicio Internet por antonomasia es el de correo electrónico. Respecto de él se han presentado los conceptos MTA y MUA y se ha detallado el formato de los mensajes *email*, así como el protocolo de transferencia SMTP para su envío y recepción. Adicionalmente, y con objeto de solucionar las limitaciones de SMTP, se han introducido las extensiones MIME. También relacionados con el servicio *email*, se han presentado los protocolos de correo de entrega final POP e IMAP.

El servicio Internet estrella en la actualidad es el web. Este servicio consiste en la navegación del usuario a través de documentos distribuidos a nivel mundial gracias a la existencia de enlaces entre ellos y a la consideración del protocolo HTTP. Respecto de esta aplicación Internet hemos de mencionar que los navegadores o *browsers* comerciales existentes son mucho más que meros clientes HTTP, permitiendo el desarrollo gráfico de otros servicios tales como TELNET, FTP o SMTP, acceso a programas Java, etc.

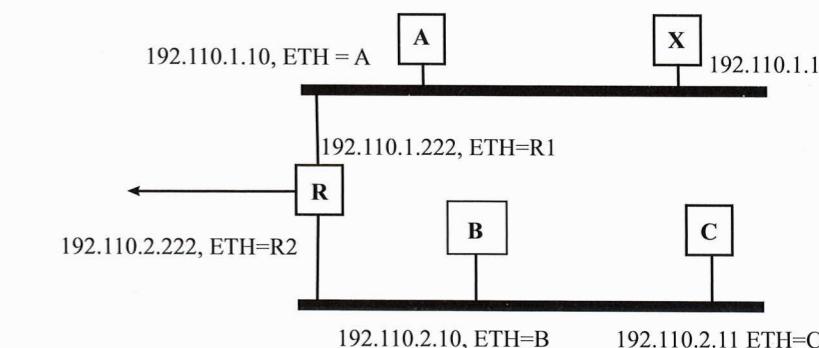
Este capítulo ha concluido con una breve discusión acerca de los distintos aspectos y problemas a abordar en el desarrollo de aplicaciones multimedia, las aplicaciones Internet del futuro. A este respecto se ha mencionado la necesidad de gran ancho de banda y bajo retardo, así como de la gestión de grupos y de árboles de encaminamiento *multicast* para la transmisión de la información. Para finalizar, dentro de este mismo apartado, se incluye una breve explicación del funcionamiento de uno de los servicios más populares en Internet: la descarga progresiva de vídeos de YouTube.

EJERCICIOS

1. Desde un computador se ejecutan tres navegadores diferentes: Internet Explorer, Mozilla Firefox y Google Chrome, y se accede desde los tres a un servidor web en la dirección 147.156.1.4 (el mismo desde los tres). ¿Cuántos *sockets* y conexiones TCP están implicados, tomando en cuenta tanto el lado servidor como el cliente?
2. Enumere las diferencias existentes entre la llamada a la función *connect* para un puerto definido para protocolo TCP y un puerto definido para protocolo UDP.
3. Explique qué mecanismos hacen que el servicio DNS sea escalable.
4. Suponga que R0 en la siguiente figura corresponde con el servidor DNS raíz de un espacio de nombres de dominio ficticio .R0, R1 y R2 son los servidores de los dominios .R1 y .R2, respectivamente, R3 corresponde con el servidor del dominio .R3, etc. Suponiendo un procedimiento de resolución recursiva, describa paso a paso los mensajes DNS intercambiados para enviar un correo desde un MUA situado en la red A a un destinatario cuya MTA estuviera instalada en MTA.R6.R2.



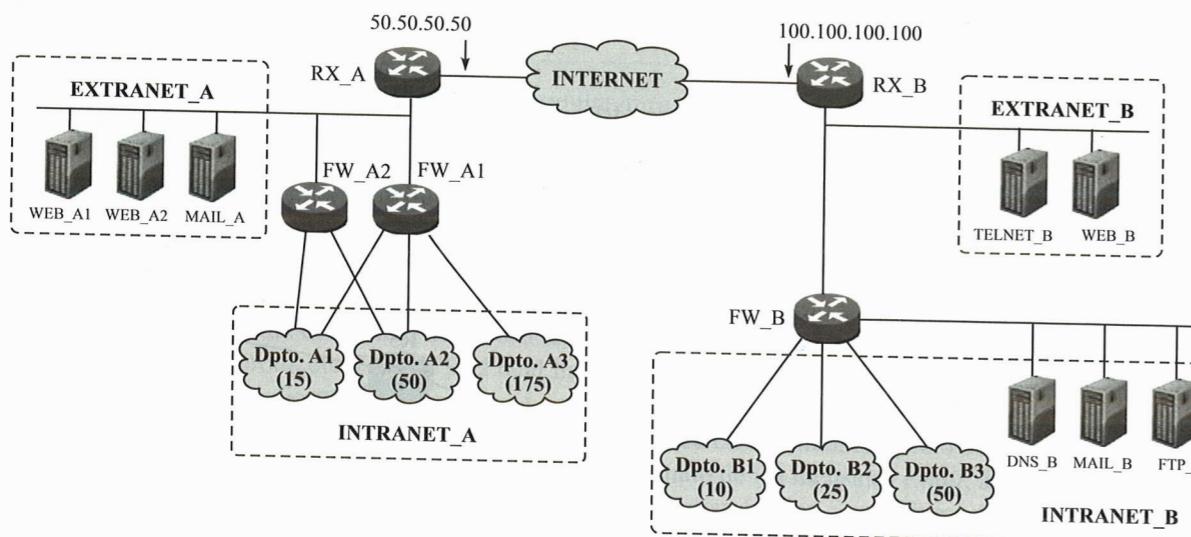
5. Suponga que el host A solicita enviar un correo electrónico a juan@B.com mediante el protocolo SMTP. Suponga que A y B conocen la dirección IP del servidor DNS C (192.110.2.11) y que B aparece en las tablas DNS como MX para su dominio. Rellene la siguiente tabla correspondiente a los distintos mensajes enviados sobre las redes.



	Mensaje aplicación	Flags TCP	Puerto origen	Puerto destino	IP origen	IP destino	ETH origen	ETH destino	Comentarios
1
...

6. La figura siguiente incluye las topologías de red de dos empresas (A y B) conectadas a Internet. Entre paréntesis se incluye el número de hosts de cada departamento. Los equipos de la empresa A utilizan un servidor de nombres con dirección 15.15.15.15, mientras que los equipos de la empresa B utilizan el servidor de nombres ubicado en su intranet.

NOTA: Los equipos de los departamentos A1 y A2 disponen de dos tarjetas de red.



Realice una asignación de las direcciones IP que necesite y suponga que un equipo X del departamento A3 quiere enviar un correo electrónico a un equipo Y del departamento B1. Suponiendo que X e Y utilizan MAIL_A y MAIL_B como servidores de correo respectivamente:

- a) Indique los protocolos (de la capa de aplicación) utilizados entre las diferentes entidades para el envío de este correo electrónico.
 - b) Indique las tramas necesarias para la comunicación entre los servidores de correo. Para estas tramas, incluya: direcciones físicas de origen y destino, direcciones IP origen y destino, puerto origen y destino, protocolo, tipo de mensaje.
7. ¿Qué tienen en común HTTP y SMTP?
 8. ¿Qué tipo de sockets pueden ser atendidos desde un servidor DNS? Con la ayuda de un diagrama de flujo identifique las llamadas al sistema necesarias para ello.
 9. Explique mediante un diagrama de flujo las llamadas al sistema para desarrollar un servidor multiservicio que ofrezca un servicio HTTP y SMTP.
 10. Un alumno desea enviar, desde su cuenta `alumno@micorreo.es`, un correo electrónico a su profesor `profesor@ugr.es`. Indique los elementos por los que pasa el correo electrónico y los protocolos involucrados, desde que se crea el correo hasta que lo lee el destinatario.

BIBLIOGRAFÍA

- Barrett, D. J.; Silverman, R.E.; Byrnes, R.G.: *SSH: The Secure Shell. The Definitive Guide*. Ed. O'Reilly, 2005.
- Berners-Lee, T.: *Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web*. RFC 1630. Junio, 1994.
- Berners-Lee, T.; Masinter, L.; McCahill, M.: *Uniform Resource Locators (URL)*. RFC 1738. Diciembre, 1994.
- Berners-Lee, T; Fielding, R.; Frystyk, H.: *Hypertext Transfer Protocol - HTTP/1.0*. RFC 1945. Mayo, 1996.
- Berners-Lee, T.; Fielding, R.; Masinter, L.: *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. Enero, 2005.
- Borman, D.A.: *Telnet Linemode Option*. RFC 1184. Octubre, 1990.

- Braden, R.; Zhang, L.; Berson, S.; Herzog, S.; Jamin, S.: *Resource reSerVation Protocol (RSVP) - Version 1 Functional Specification*. RFC 2205. Septiembre, 1997.
- Comer, D.E.: *Internetworking with TCP/IP. Volume I: Principles, Protocols and Architecture*. 3^a edición. Prentice Hall, 1995.
- Comer, D.E.; Stevens, D.L.: *Internetworking with TCP/IP. Volume III: Client-Server Programming and Applications*. Prentice Hall, 1996.
- Crispin, M.: *Internet Message Access Protocol - Version 4*. RFC 1730. Diciembre, 1994.
- Crispin, M.: *Internet Message Access Protocol - Version 4rev1*. RFC 2060. Diciembre, 1996.
- Crispin, M.: *Internet Message Access Protocol - Version 4rev1*. RFC 3501. Marzo, 2003.
- Crispin, M.: *IMAP4 Compatibility with IMAP2bis*. RFC 2061. Diciembre, 1996.
- Crocker, D.: Standard for the format of ARPA Internet Text Messages. RFC 822. Agosto, 1982.
- Eastlake, D.: *Domain Name System Security Extensions*. RFC 2535. Marzo, 1999.
- Faltstrom, P.; Hoffman, P.; Costello, A.: *Internationalizing Domain Names in Applications (IDNA)*. RFC 3490. Marzo, 2003.
- Fielding, R.: *Relative Uniform Resource Locators*. RFC 1808. Junio, 1995.
- Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T.: *Hypertext Transfer Protocol, HTTP/1.1*. RFC 2616. Junio, 1999.
- Freed, N.; Borenstein, N.: *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. RFC 2045. Noviembre, 1996.
- Freed, N.; Borenstein, N.: *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. RFC 2046. Noviembre, 1996.
- Freed, N.; Lensin, J.; Postel, J.: *Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures*. RFC 2048. Noviembre, 1996.
- Freed, N.; Borenstein, N.: *Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples*. RFC 2049. Noviembre, 1996.
- Gellens, R.; Newman, C.; Lundblade, L.: *POP3 Extension Mechanism*. RFC 2449. Noviembre, 1998.
- Handley, M.; Jacobson, V.; Perkins, C.: *SDP: Session Description Protocol*. RFC 4566. Julio, 2006.
- Handley, M.; Schulzrinne, H.; Schooler, E.; Rosenberg, J.: *SIP: Session Initiation Protocol*. RFC 2543. Marzo, 1999.
- Handley, M.; Perkins, C.; Whelan, E.: *Session Announcement Protocol*. RFC 2974. Octubre, 2000.
- Horton, M.R.; Adams, R.: *Standard for Interchange of USENET Messages*. RFC 1036. Diciembre, 1987.
- Klensin, J.: *Simple Mail Transfer Protocol*. RFC 2821. Abril, 2001.
- Klensin, J.: *Simple Mail Transfer Protocol*. RFC 5321. Octubre, 2008.
- Moats, R.: *URN Syntax*. RFC 2141. Mayo, 1997.
- Mockapetris, P.V.: *Domain Names - Concepts and Facilities*. RFC 1034. Noviembre, 1987.
- Mockapetris, P.V.: *Domain Names - Implementation and Specification*. RFC 1035. Noviembre, 1987.
- Moore, K.: *MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text*. RFC 2047. Noviembre, 1996.
- Moy, J.: *Multicast Extensions to OSPF*. RFC 1584. Marzo, 1994.
- Myers, J.; Rose, M.: *Post Office Protocol, Version 3*. RFC 1939. Mayo, 1996.
- Nelson, R.: *Some Observations on Implementations of the Post Office Protocol (POP3)*. RFC 1957. Junio, 1996.
- Postel, J.: *Simple Mail Transfer Protocol*. RFC 821. Agosto, 1982.
- Postel, J.; Reynolds, J.K.: *Telnet Protocol Specification*. RFC 854. Mayo, 1983.
- Postel, J.; Reynolds, J.K.: *Telnet Binary Transmission*. RFC 856. Mayo, 1983.
- Postel, J.; Reynolds, J.K.: *Telnet Echo Option*. RFC 857. Mayo, 1983.
- Postel, J.; Reynolds, J.K.: *File Transfer Protocol*. RFC 959. Octubre, 1985.

- Postel, J.: *Domain Name System Structure and Delegation*. RFC 1591. Marzo, 1994.
- Schulzrinne, H.; Casner, S.; Frederick, R.; Jacobson, V.: *RTP: A Transport Protocol for Real-Time Applications*. Audio-Video Transport Working Group. RFC 3550. Julio, 2003.
- Schulzrinne, H.; Rao, A.; Lanphier, R.: *Real Time Streaming Protocol (RTSP)*. RFC 2326. Abril, 1998.
- Sollins, K.; Masinter, L.: *Functional Requirements for Uniform Resource Names*. RFC 1737. Diciembre, 1994.
- Stevens, W.R.: *TCP/IP Illustrated, Vol. 1. The Protocols*. Ed. Addison Wesley, 2000.
- VanBokkelen, J.: *Telnet Terminal-Type Option*. RFC 1091. Febrero, 1989.
- Ylonen, T.; Lonwick, C.: *The Secure Shell (SSH) Protocol Architecture*. RFC 4251. Enero 2006.



GESTIÓN Y SEGURIDAD EN REDES

- 12.1. Introducción
- 12.2. Gestión de redes
- 12.3. Seguridad en redes de computadores

12.1. Introducción

Uno de los aspectos más importantes de una red de computadores es el concerniente a su gestión. El concepto de gestión de red es muy amplio puesto que hace referencia a la supervisión y mantenimiento de todas aquellas facetas de un sistema en red que afectan o pueden afectar a las prestaciones de este en cuanto al número y características de los servicios y recursos ofertados, eficiencia en el uso de los mismos, accesos permitidos, etc. Si bien esto puede resultar excesivamente vasto y complejo, es una tarea perfectamente definida en sus objetivos y primordial para el adecuado funcionamiento de todo sistema en red.

Función cuya responsabilidad principal recae en el administrador del sistema, la gestión de red, según OSI, cubre los siguientes aspectos:

- *Control de la planificación*. El proceso de gestión de una red no se inicia cuando esta ya se encuentra en funcionamiento, sino que parte desde su propia concepción. Resulta fundamental así un adecuado estudio de las necesidades a cubrir y del dimensionado planteado tanto en infraestructura como en servicios; debiéndose tomar ya desde sus orígenes decisiones de importancia que marcarán la vida futura de la red a desplegar.
- *Gestión de la configuración*. Un sistema de red es algo «vivo», dinámico, y así debe ser si, como es previsible, las necesidades, los usuarios, los servicios, etc., varían a lo largo del tiempo. Se entiende, pues, que el proceso de planificación mencionado anteriormente no es algo puntual que se realiza al inicio de la vida del sistema, sino que se extiende a lo largo del tiempo y