

la línea de estado el texto 304 *Not Modified* (no modificado), lo que indica a la caché que puede reenviar la copia del objeto que tiene en caché al navegador que lo haya solicitado.

Aquí terminamos nuestra exposición acerca de HTTP, el primer protocolo de Internet (un protocolo de la capa de aplicación) que hemos estudiado en detalle. Hemos examinado el formato de los mensajes HTTP y las acciones realizadas por el servidor y el cliente web según se envían y reciben estos mensajes. También hemos visto algo acerca de la infraestructura de las aplicaciones web, incluyendo las cachés, las cookies y las bases de datos back-end, elementos todos ellos que están ligados de alguna manera con el protocolo HTTP.

## 2.3 Correo electrónico en Internet

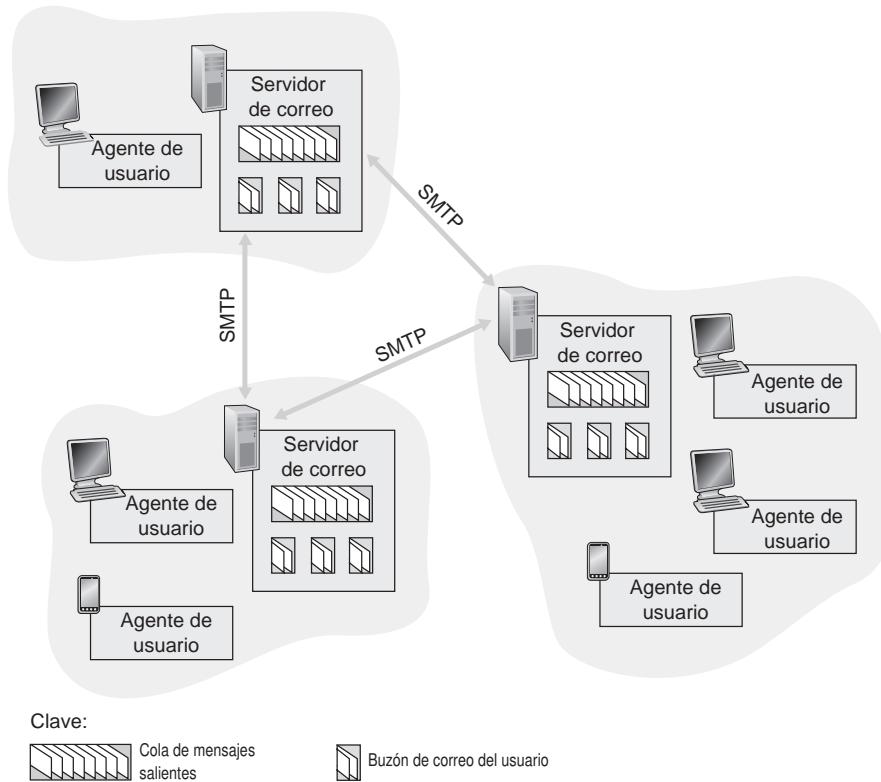
El correo electrónico ha existido desde que Internet viera la luz. Era la aplicación más popular cuando Internet estaba en sus comienzos [Segaller 1998] y a lo largo de los años se ha ido convirtiendo en una aplicación cada vez más elaborada y potente. En la actualidad continúa siendo una de las aplicaciones más importantes y utilizadas de Internet.

Al igual que el servicio ordinario de correo postal, el correo electrónico es un medio de comunicación asíncrono (las personas envían y leen los mensajes cuando les conviene, sin tener que coordinarse con las agendas de otras personas). En contraste con el correo postal, el correo electrónico es rápido, fácil de distribuir y barato. Las aplicaciones modernas de correo electrónico disponen de muchas características potentes. Mediante las listas de correo, se pueden enviar mensajes de correo electrónico y correo basura (spam) a miles de destinatarios a un mismo tiempo. A menudo, los mensajes de correo electrónico incluyen adjuntos, hipervínculos, texto en formato HTML y fotografías.

En esta sección vamos a examinar los protocolos de la capa de aplicación que forman la base del correo electrónico en Internet. Pero antes de sumergirnos en una explicación detallada acerca de estos protocolos, vamos a proporcionar una visión de conjunto del sistema de correo de Internet y sus componentes fundamentales.

La Figura 2.14 muestra una visión general del sistema de correo de Internet. A partir de este diagrama, vemos que existen tres componentes principales: **agentes de usuario**, **servidores de correo** y el **Protocolo simple de transferencia de correo (SMTP, Simple Mail Transfer Protocol)**. A continuación vamos a describir cada uno de estos componentes en el contexto de un emisor, Alicia, que envía un mensaje de correo electrónico a un destinatario, Benito. Los agentes de usuario permiten a los usuarios leer, responder, reenviar, guardar y componer mensajes. Microsoft Outlook, y Apple Mail son ejemplos de agentes de usuario para correo electrónico. Cuando Alicia termina de componer su mensaje, su agente de usuario envía el mensaje a su servidor de correo, donde el mensaje es colocado en la cola de mensajes salientes del servidor de correo. Cuando Benito quiere leer un mensaje, su agente de usuario recupera el mensaje de su buzón, que se encuentra en el servidor de correo.

Los servidores de correo forman el núcleo de la infraestructura del correo electrónico. Cada destinatario, como por ejemplo Benito, tiene un **buzón de correo** ubicado en uno de los servidores de correo. El buzón de Benito gestiona y mantiene los mensajes que le han sido enviados. Un mensaje típico inicia su viaje en el agente de usuario del emisor, viaja hasta el servidor de correo del emisor y luego hasta el servidor de correo del destinatario, donde es depositado en el buzón del mismo. Cuando Benito quiere acceder a los mensajes contenidos en su buzón, el servidor de correo que lo contiene autentica a Benito (mediante el nombre de usuario y la contraseña). El servidor de correo de Alicia también tiene que ocuparse de los fallos que se producen en el servidor de correo de Benito. Si el servidor de Alicia no puede enviar el mensaje de correo al servidor de Benito, entonces el servidor de Alicia mantiene el mensaje en una **cola de mensajes** e intenta enviarlo más tarde. Normalmente, los reintentos de envío se realizan más o menos cada 30 minutos; si después de varios días no se ha conseguido, el servidor elimina el mensaje y se lo notifica al emisor (Alicia) mediante un mensaje de correo electrónico.



**Figura 2.14** ♦ Esquema general del sistema de correo electrónico de Internet.

SMTP es el principal protocolo de la capa de aplicación para el correo electrónico por Internet. Utiliza el servicio de transferencia de datos fiable de TCP para transferir el correo desde el servidor de correo del emisor al servidor de correo del destinatario. Al igual que la mayoría de los protocolos de la capa de aplicación, SMTP tiene dos lados: el lado del cliente, que se ejecuta en el servidor de correo del emisor, y el lado del servidor, que se ejecuta en el servidor de correo del destinatario. Tanto el lado del cliente como el del servidor de SMTP se ejecutan en todos los servidores de correo. Cuando un servidor de correo envía mensajes de correo a otros servidores de correo, actúa como un cliente SMTP. Cuando un servidor de correo recibe correo de otros servidores, actúa como un servidor SMTP.

### 2.3.1 SMTP

El protocolo SMTP, que está definido en el documento RFC 5321, es el corazón del correo electrónico por Internet. Como hemos mencionado anteriormente, SMTP transfiere mensajes desde los servidores de correo de los emisores a los servidores de correo de los destinatarios. SMTP es mucho más antiguo que HTTP. (El RFC original que se ocupa de SMTP data de 1982 y SMTP ya existía bastante tiempo antes.) Aunque SMTP tienen muchas cualidades maravillosas, como prueba su presencia en Internet, es una tecnología heredada que utiliza algunas funcionalidades arcaicas. Por ejemplo, restringe el cuerpo (no sólo las cabeceras) de todos los mensajes a formato ASCII de 7 bits. Esta restricción tenía sentido a principios de la década de 1980, cuando la capacidad de transmisión era escasa y nadie enviaba mensajes con adjuntos o imágenes de gran tamaño, o archivos de audio o vídeo. Pero actualmente, en la era multimedia, la restricción del formato ASCII de 7 bits causa muchos problemas: requiere que los datos binarios multimedia se codifiquen a

ASCII antes de ser transmitidos a través de SMTP y requiere que el correspondiente mensaje ASCII sea decodificado de vuelta a binario una vez realizado el transporte SMTP. Recuerde, como hemos visto en la Sección 2.2, que HTTP no precisa que los datos multimedia sean codificados a ASCII antes de ser transferidos.

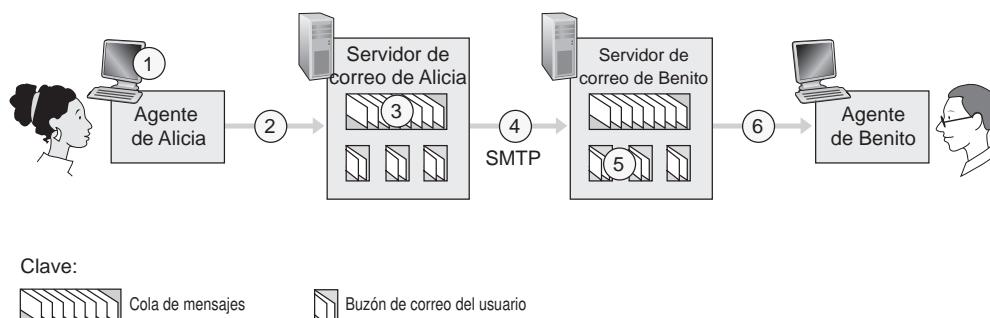
Para ilustrar el funcionamiento básico de SMTP vamos a recurrir a un escenario ya conocido. Suponga que Alicia desea enviar a Benito un sencillo mensaje ASCII.

1. Alicia invoca a su agente de usuario para correo electrónico, proporciona la dirección de correo electrónico de Benito (por ejemplo, benito@unaescuela.edu), compone un mensaje e indica al agente de usuario que lo envíe.
2. El agente de usuario de Alicia envía el mensaje al servidor de correo de ella, donde es colocado en una cola de mensajes.
3. El lado del cliente de SMTP, que se ejecuta en el servidor de correo de Alicia, ve el mensaje en la cola de mensajes. Abre una conexión TCP con un servidor SMTP, que se ejecuta en el servidor de correo de Benito.
4. Después de la fase de negociación inicial de SMTP, el cliente SMTP envía el mensaje de Alicia a través de la conexión TCP.
5. En el servidor de correo de Benito, el lado del servidor de SMTP recibe el mensaje. El servidor de correo de Benito coloca entonces el mensaje en el buzón de Benito.
6. Benito invoca a su agente de usuario para leer el mensaje cuando le apetezca.

En la Figura 2.15 se resume este escenario.

Es importante observar que normalmente SMTP no utiliza servidores de correo intermedios para enviar correo, incluso cuando los dos servidores de correo se encuentran en extremos opuestos del mundo. Si el servidor de Alicia está en Hong Kong y el de Benito está en St. Louis, la conexión TCP será una conexión directa entre los servidores de Hong Kong y St. Louis. En particular, si el servidor de correo de Benito está fuera de servicio, el servidor de Alicia conservará el mensaje y lo intentará de nuevo (el mensaje no se deja en un servidor de correo intermedio).

Veamos en detalle cómo transfiere SMTP un mensaje desde un servidor de correo emisor a un servidor de correo receptor. Comprobaremos que el protocolo SMTP presenta muchas similitudes con los protocolos empleados por las personas para las interacciones cara a cara. En primer lugar, el cliente SMTP (que se ejecuta en el host servidor de correo emisor) establece una conexión TCP con el puerto 25 del servidor SMTP (que se ejecuta en el host servidor de correo receptor). Si el servidor no está operativo, el cliente lo intentará más tarde. Una vez que se ha establecido la conexión, el servidor y el cliente llevan a cabo el proceso de negociación de la capa de aplicación (al igual que las personas, que antes de intercambiar información se presentan, los clientes y servidores SMTP se presentan a sí mismos antes de transferir la información). Durante esta fase de negociación SMTP, el cliente SMTP especifica la dirección de correo electrónico del emisor (la persona que ha generado el mensaje) y la dirección de correo electrónico del destinatario. Una vez que el cliente y el servidor



**Figura 2.15** ♦ Alicia envía un mensaje a Benito.

SMTP se han presentado a sí mismos, el cliente envía el mensaje. SMTP cuenta con el servicio de transferencia de datos fiable de TCP para transferir el mensaje al servidor sin errores. El cliente repite entonces este proceso a través de la misma conexión TCP si tiene que enviar otros mensajes al servidor; en caso contrario, indica a TCP que cierre la conexión.

Veamos ahora una transcripción de ejemplo de los mensajes intercambiados entre un cliente SMTP (C) y un servidor SMTP (S). El nombre de host del cliente es `crepes.fr` y el nombre de host del servidor es `hamburger.edu`. Las líneas de texto ASCII precedidas por C: son exactamente las líneas que el cliente envía a su socket TCP y las líneas de texto ASCII precedidas por S: son las líneas que el servidor envía a su socket TCP. La siguiente transcripción comienza tan pronto como se establece la conexión TCP.

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alicia@crepes.fr>
S: 250 alicia@crepes.fr ... Sender ok
C: RCPT TO: <benito@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: ¿Te gusta el ketchup?
C: ¿Y los pepinillos en vinagre?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

En el ejemplo anterior, el cliente envía un mensaje (“¿Te gusta el ketchup? ¿Y los pepinillos en vinagre?”) desde el servidor de correo `crepes.fr` al servidor de correo `hamburger.edu`. Como parte del diálogo, el cliente ejecuta cinco comandos: HELO (una abreviatura de HELLO), MAIL FROM, RCPT TO, DATA y QUIT. Estos comandos se explican por sí mismos. El cliente también envía una línea que consta únicamente de un punto, que indica el final del mensaje para el servidor. (En la jerga ASCII, cada mensaje termina con CRLF. CRLF, donde CR y LF se corresponden con el retorno de carro y el salto de línea, respectivamente.) El servidor responde a cada comando, teniendo cada una de las respuestas un código de respuesta y una explicación (opcional) en inglés. Hemos mencionado que SMTP utiliza conexiones persistentes: si el servidor de correo emisor tiene varios mensajes que enviar al mismo servidor de correo receptor, puede enviar todos los mensajes a través de la misma conexión TCP. Para cada mensaje, el cliente inicia el proceso con un nuevo comando MAIL FROM: `crepes.fr`, designa el final del mensaje con un único punto y ejecuta el comando QUIT sólo después de que todos los mensajes hayan sido enviados.

Es muy recomendable que utilice Telnet para establecer un diálogo directo con un servidor SMTP. Para ello, ejecute:

```
telnet nombreServidor 25
```

donde `nombreServidor` es el nombre de un servidor de correo local. Al hacer esto, simplemente está estableciendo una conexión TCP entre su host local y el servidor de correo. Después de escribir esta línea, debería recibir inmediatamente la respuesta 220 del servidor. A continuación, ejecute los comandos SMTP HELO, MAIL FROM, RCPT TO, DATA, CRLF. CRLF y QUIT en los instantes apropiados. También es extremadamente recomendable que realice la Tarea de programación 2 incluida al final del capítulo. En esta tarea, tendrá que crear un agente de usuario simple que implemente el lado del cliente de SMTP. Esto le permitirá enviar un mensaje de correo electrónico a un destinatario arbitrario a través de un servidor de correo local.

### 2.3.2 Comparación con HTTP

Vamos ahora a comparar brevemente SMTP con HTTP. Ambos protocolos se emplean para transferir archivos de un host a otro: HTTP transfiere archivos (también denominados objetos) desde un servidor web a un cliente web (normalmente, un navegador); SMTP transfiere archivos (es decir, mensajes de correo electrónico) desde un servidor de correo a otro servidor de correo. Para transferir los archivos, tanto HTTP persistente como SMTP emplean conexiones persistentes. Por tanto, ambos protocolos tienen características comunes. Sin embargo, también presentan algunas diferencias. En primer lugar, HTTP es principalmente un **protocolo pull** (protocolo de extracción): alguien carga la información en un servidor web y los usuarios utilizan HTTP para extraer la información del servidor cuando desean. En concreto, la máquina que desea recibir el archivo inicia la conexión TCP. Por el contrario, SMTP es fundamentalmente un **protocolo push** (protocolo de inserción): el servidor de correo emisor introduce el archivo en el servidor de correo receptor. En concreto, la máquina que desea enviar el archivo inicia la conexión TCP.

Una segunda diferencia, a la que hemos aludido anteriormente, es que SMTP requiere que cada mensaje, incluyendo el cuerpo de cada mensaje, esté en el formato ASCII de 7 bits. Si el mensaje contiene caracteres que no corresponden a dicho formato (como por ejemplo, caracteres acentuados) o contiene datos binarios (como por ejemplo un archivo de imagen), entonces el mensaje tiene que ser codificado en ASCII de 7 bits. Los datos HTTP no imponen esta restricción.

Una tercera diferencia importante tiene que ver con cómo se maneja un documento que conste de texto e imágenes (junto con posiblemente otros tipos multimedia). Como hemos visto en la Sección 2.2, HTTP encapsula cada objeto en su propio mensaje de respuesta HTTP. El correo Internet incluye todos los objetos del mensaje en un mismo mensaje.

### 2.3.3 Formatos de los mensajes de correo

Cuando Alicia escribe una carta por correo ordinario a Benito, puede incluir todo tipo de información adicional en la parte superior de la carta, como la dirección de Benito, su propia dirección de respuesta y la fecha. De forma similar, cuando una persona envía un mensaje de correo electrónico a otra, una cabecera que contiene la información administrativa antecede al cuerpo del mensaje. Esta información se incluye en una serie de líneas de cabecera, que están definidas en el documento RFC 5322. Las líneas de cabecera y el cuerpo del mensaje se separan mediante una línea en blanco (es decir, mediante CRLF). RFC 5322 especifica el formato exacto de las líneas de cabecera, así como sus interpretaciones semánticas. Como con HTTP, cada línea de cabecera contiene texto legible, que consta de una palabra clave seguida de dos puntos y de un valor. Algunas de las palabras clave son obligatorias y otras son opcionales. Una cabecera tiene que estar formada por una línea de cabecera `From:` y una línea de cabecera `To:`; también puede incluir una línea `Subject:`, así como otras líneas de cabeceraopcionales. Es importante destacar que estas líneas de cabecera son *diferentes* de los comandos SMTP que hemos estudiado en la Sección 2.3.1 (incluso aunque contengan algunas palabras comunes como “*from*” y “*to*”). Los comandos vistos en esa sección forman parte del protocolo de negociación de SMTP; las líneas de cabecera examinadas en esta sección forman parte del propio mensaje de correo.

Una cabecera de mensaje típica sería como la siguiente:

```
From: alicia@crepes.fr
To: benito@hamburger.edu
Subject: Búsqueda del significado de la vida.
```

Después del mensaje de cabecera se incluye una línea en blanco y, a continuación, el cuerpo del mensaje (en ASCII). Debería utilizar Telnet para enviar un mensaje a un servidor de correo que contenga varias líneas de cabecera, incluyendo la línea de cabecera del asunto `Subject:`. Para ello, ejecute el comando telnet NombreServidor 25, como se ha explicado en la Sección 2.3.1.

### 2.3.4 Protocolos de acceso para correo electrónico

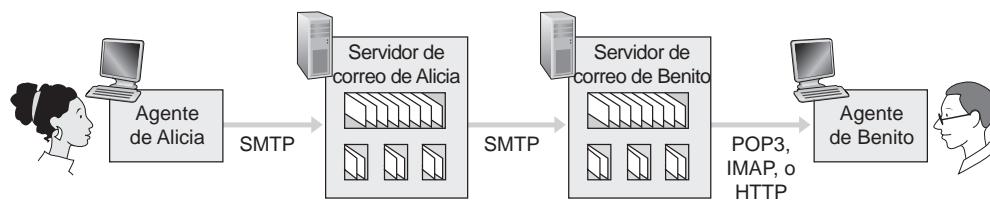
Una vez que SMTP envía el mensaje del servidor de correo de Alicia al servidor de correo de Benito, el mensaje se coloca en el buzón de este último. A lo largo de esta exposición, hemos supuesto tácitamente que Benito lee su correo registrándose en el host servidor y utilizando después un lector de correo que se ejecuta en dicho host. Hasta principios de la década de 1990 esta era la forma habitual de hacer las cosas. Pero, actualmente, el acceso al correo electrónico utiliza una arquitectura cliente-servidor; el usuario típico lee el correo electrónico con un cliente que se ejecuta en el sistema terminal del usuario, por ejemplo, en un PC de la oficina, en un portátil o en un smartphone. Ejecutando un cliente de correo en un PC local, los usuarios disponen de un rico conjunto de funcionalidades, entre las que se incluye la posibilidad de visualizar documentos adjuntos y mensajes multimedia.

Puesto que Benito (el destinatario) ejecuta su agente de usuario en su PC local, es natural que considere también incluir un servidor de correo en su PC local. De esta forma, el servidor de correo de Alicia dialogará directamente con el PC de Benito. Sin embargo, hay un problema con este método. Recuerde que un servidor de correo gestiona buzones de correo y ejecuta los lados del cliente y del servidor de SMTP. Si el servidor de correo de Benito residiera en su PC local, entonces el PC de Benito tendría que estar siempre encendido y conectado a Internet para recibir los nuevos correos que pudieran llegar en cualquier momento. Pero esto es impráctico para muchos usuarios de Internet. En su lugar, un usuario típico ejecuta un agente de usuario en el PC local pero accede a su buzón de correo almacenado en un servidor de correo compartido que siempre está encendido. Este servidor es compartido con otros usuarios y, normalmente, mantenido por el ISP del usuario (por ejemplo, una universidad o una empresa).

Ahora vamos a ver qué ruta sigue un mensaje de correo electrónico que Alicia envía a Benito. Acabamos de estudiar que en algún punto a lo largo de la ruta el mensaje de correo electrónico tiene que ser depositado en el servidor de correo de Benito. Esto se podría conseguir fácilmente haciendo que el agente de usuario de Alicia envíe el mensaje directamente al servidor de correo de Benito. Y esto se podría hacer utilizando SMTP (de hecho, SMTP ha sido diseñado para llevar el correo electrónico de un host a otro. Sin embargo, normalmente el agente de usuario del emisor no se comunica directamente con el servidor de correo del destinatario. En su lugar, como se muestra en la Figura 2.16, el agente de usuario de Alicia utiliza SMTP para introducir el mensaje de correo en su propio servidor de correo, y a continuación el servidor de correo de Alicia utiliza SMTP (como un cliente SMTP) para pasar el mensaje al servidor de correo de Benito. ¿Por qué este procedimiento en dos pasos? Fundamentalmente, porque sin la retransmisión a través del servidor de correo de Alicia, el agente de usuario de esta no tiene ninguna forma de acceder a un servidor de correo de destino inalcanzable. Al hacer que Alicia deposite primero el mensaje en su propio servidor de correo, este puede intentar una y otra vez enviar el mensaje al servidor de correo de Benito, por ejemplo, cada 30 minutos, hasta que el servidor de Benito esté de nuevo operativo. (Y si el servidor de correo de Alicia no funciona, entonces ella tiene el recurso de quejarse al administrador del sistema!). El RFC que se ocupa de SMTP define cómo se pueden utilizar los comandos SMTP para transmitir un mensaje a través de varios servidores SMTP.

¡Pero todavía falta una pieza del puzzle! ¿Cómo un destinatario como Benito, que ejecuta un agente de usuario en su PC local, obtiene sus mensajes, que se encuentran en un servidor de correo de su ISP? Tenga en cuenta que el agente de usuario de Benito no puede utilizar SMTP para obtener los mensajes porque es una operación de extracción (pull) mientras que SMTP es un protocolo push (de inserción). Así, el puzzle se completa añadiendo un protocolo especial de acceso al correo que permita transferir los mensajes del servidor de correo de Benito a su PC local. Actualmente existen varios protocolos de acceso a correo electrónico populares, entre los que se incluyen el **Protocolo de oficina de correos versión 3 (POP3, Post Office Protocol—Version 3)**, el **Protocolo de acceso de correo de Internet (IMAP, Internet Mail Access Protocol)** y HTTP.

La Figura 2.16 proporciona un resumen de los protocolos que se utilizan para el correo de Internet: SMTP se emplea para transferir correo desde el servidor de correo del emisor al servidor



**Figura 2.16** ◆ Protocolos de correo electrónico y entidades que los utilizan.

de correo del destinatario; SMTP también se utiliza para transferir correo desde el agente de usuario del emisor al servidor de correo del mismo. Para transferir los mensajes de correo almacenados en el servidor de correo del destinatario al agente de usuario del mismo se emplea un protocolo de acceso a correo, como POP3.

### POP3

POP3 es un protocolo de acceso a correo extremadamente simple. Está definido en [RFC 1939], que es un documento corto y bastante claro. Dado que el protocolo es tan simple, su funcionalidad es bastante limitada. POP3 se inicia cuando el agente de usuario (el cliente) abre una conexión TCP en el puerto 110 al servidor de correo (el servidor). Una vez establecida la conexión TCP, POP3 pasa a través de tres fases: autorización, transacción y actualización. Durante la primera fase, la autorización, el agente de usuario envía un nombre de usuario y una contraseña (en texto legible) para autenticar al usuario. Durante la segunda fase, la de transacción, el agente de usuario recupera los mensajes; también durante esta fase, el agente de usuario puede marcar los mensajes para borrado, eliminar las marcas de borrado y obtener estadísticas de correo. La tercera fase, la actualización, tiene lugar después que el cliente haya ejecutado el comando `quit`, terminando la sesión POP3; en este instante, el servidor de correo borra los mensajes que han sido marcados para borrado.

En una transacción POP3, el agente de usuario ejecuta comandos y el servidor devuelve para cada comando una respuesta. Existen dos posibles respuestas: `+OK` (seguida en ocasiones por una serie de datos servidor-cliente), utilizada por el servidor para indicar que el comando anterior era correcto; y `-ERR`, utilizada por el servidor para indicar que había algún error en el comando anterior.

La fase de autorización tiene dos comandos principales: `user <nombreasuario>` y `pass <contraseña>`. Para ilustrar estos dos comandos, le sugerimos que establezca una conexión Telnet directamente en un servidor POP3, utilizando el puerto 110, y ejecute estos comandos. Suponga que `mailServer` es el nombre de su servidor de correo. Verá algo similar a lo siguiente:

```
telnet mailServer 110
+OK POP3 server ready
user benito
+OK
pass hambre
+OK user successfully logged on
```

Si escribe mal un comando, el servidor POP3 le responderá con un mensaje `-ERR`.

Abordemos ahora la fase de transacción. Un agente de usuario que utilice POP3 suele ser configurado (por el usuario) para “descargar y borrar” o para “descargar y guardar”. La secuencia de comandos que ejecute un agente de usuario POP3 dependerá de en cuál de estos dos modos esté operando. En el modo descargar y borrar, el agente de usuario ejecutará los comandos `list`, `retry` y `dele`. Por ejemplo, suponga que el usuario tiene dos mensajes en su buzón de correo. En el diálogo que proporcionamos a continuación, `C:` (que quiere decir cliente) es el agente de usuario y `S:` (que quiere decir servidor) es el servidor de correo. La transacción será similar a lo siguiente:

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: (bla bla ...
S: .....
S: .....bla)
S: .
C: dele 1
C: retr 2
S: (bla bla ...
S: .....
S: .....bla)
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

En primer lugar, el agente de usuario pide al servidor de correo que le informe del tamaño de cada uno de los mensajes almacenados. A continuación, el agente de usuario recupera y borra cada uno de los mensajes del servidor. Observe que después de la fase de autorización, el agente de usuario solo ha utilizado cuatro comandos: `list`, `retr`, `dele` y `quit`. La sintaxis de estos comandos está definida en el documento RFC 1939. Después de procesar el comando `quit`, el servidor POP3 entra en la fase de actualización y borra los mensajes 1 y 2 del buzón.

Un problema en este modo de descarga y borrado es que el destinatario, Benito, puede ser algo nómada y puede desear poder acceder a sus mensajes de correo desde varias máquinas; por ejemplo, desde el PC de la oficina, el PC de su casa y el portátil. El modo de descarga y borrado reparte los mensajes de correo de Benito entre estas tres máquinas; en particular, si Benito lee un mensaje en el PC de la oficina, no podrá volver a leerlo en el portátil en su casa por la noche. En el modo descargar y guardar, el agente de usuario deja los mensajes en el servidor de correo después de que se hayan descargado. En este caso, Benito podrá volver a leer los mensajes desde distintas máquinas; puede acceder a un mensaje en el trabajo y luego también en su casa si lo desea.

Durante una sesión POP3 entre un agente de usuario y el servidor de correo, el servidor POP3 mantiene cierta información de estado; en concreto, mantiene la relación de los mensajes de usuario que han sido marcados para ser borrados. Sin embargo, el servidor POP3 no conserva la información de estado de una sesión POP3 a otra. Esta falta de memoria del estado entre sesiones simplifica enormemente la implementación de un servidor POP3.

## IMAP

Con el acceso POP3, una vez que Benito ha descargado sus mensajes en la máquina local, puede crear carpetas de correo y mover los mensajes descargados a las carpetas. A continuación, puede borrar los mensajes, pasarllos a carpetas y realizar búsquedas de mensajes (por nombre del emisor o asunto). Pero este paradigma, es decir, las carpetas y los mensajes guardados en la máquina local, plantea un problema para el usuario nómada, que preferirá mantener una jerarquía de carpetas en un servidor remoto al que pueda acceder desde cualquier computadora. Esto no es posible con POP3 (el protocolo POP3 no proporciona ningún medio al usuario para crear carpetas remotas y asignar mensajes a las mismas).

Para resolver este y otros problemas se inventó el protocolo IMAP, definido en [RFC 3501]. Al igual que POP3, IMAP es un protocolo de acceso a correo. Ofrece muchas más funcionalidades que POP3, pero también es significativamente más complejo (y, por tanto, las implementaciones del lado del cliente y del lado del servidor son bastante más complejas).

Un servidor IMAP asociará cada mensaje con una carpeta; cuando un mensaje llega al servidor, se asocia con la carpeta INBOX (Bandeja de entrada) del destinatario, el cual puede entonces pasar el mensaje a una nueva carpeta creada por el usuario, leer el mensaje, borrarlo, etc. El protocolo IMAP proporciona comandos que permiten a los usuarios crear carpetas y mover los mensajes de una carpeta a otra. IMAP también proporciona comandos que permiten a los usuarios realizar búsquedas en carpetas remotas para localizar mensajes que cumplen unos determinados criterios. Observe que, a diferencia de POP3, un servidor IMAP mantiene información acerca del estado a lo largo de las sesiones IMAP, como por ejemplo, los nombres de las carpetas y los mensajes asociados con cada una de ellas.

Otra importante característica de IMAP es que dispone de comandos que permiten a un agente de usuario obtener partes componentes de los mensajes. Por ejemplo, un agente de usuario puede obtener solo la cabecera del mensaje o solo una parte de un mensaje MIME de varias partes. Esta característica resulta útil cuando se emplea una conexión con un ancho de banda pequeño (por ejemplo, un enlace de módem de baja velocidad) entre el agente de usuario y su servidor de correo. Con una conexión de ancho de banda pequeño, puede que el usuario no quiera descargar todos los mensajes guardados en su buzón y que desee saltarse los mensajes largos que pueda haber; por ejemplo un archivo de audio o de vídeo.

### Correo electrónico web

Actualmente, cada vez más usuarios envían y acceden a su correo electrónico a través de sus navegadores web. Hotmail introdujo a mediados de la década de 1990 el acceso basado en la Web; actualmente, también ofrece este servicio Yahoo, Google, así como casi todas las principales universidades y empresas. Con este servicio, el agente de usuario es un navegador web corriente y el usuario se comunica con su buzón remoto a través de HTTP. Cuando un destinatario, como Benito, desea acceder a un mensaje de su buzón, este es enviado desde el servidor de correo de Benito al navegador del mismo utilizando el protocolo HTTP en lugar de los protocolos POP3 o IMAP. Cuando un emisor, como Alicia, desea enviar un mensaje de correo electrónico, este es transmitido desde su navegador a su servidor de correo a través de HTTP en lugar de mediante SMTP. Sin embargo, el servidor de correo de Alicia, continúa enviando mensajes a, y recibiendo mensajes de, otros servidores de correo que emplean SMTP.

## 2.4 DNS: el servicio de directorio de Internet

Las personas podemos ser identificadas de muchas maneras. Por ejemplo, podemos ser identificadas por el nombre que aparece en nuestro certificado de nacimiento, por nuestro número de la seguridad social o por el número del carnet de conducir. Aunque cada uno de estos datos se puede utilizar para identificar a las personas, dentro de un determinado contexto un identificador puede ser más apropiado que otro. Por ejemplo, las computadoras del IRS (la terrible agencia tributaria de Estados Unidos) prefieren utilizar los números de la seguridad social de longitud fija que el nombre que aparece en el certificado de nacimiento. Por otro lado, las personas prefieren emplear el nombre que figura en los certificados de nacimiento, más fáciles de recordar, a los números de la seguridad social (puede imaginar que alguien le dijera “Hola. Mi nombre es 132-67-9875. Este es mi marido, 178-87-1146.”)

No solo las personas podemos ser identificadas de diversas formas, los hosts de Internet también. Un identificador para los hosts es el **nombre de host**. Los nombres de host, como por ejemplo [www.facebook.com](http://www.facebook.com), [www.google.com](http://www.google.com), [gaia.cs.umass.edu](http://gaia.cs.umass.edu), son mnemónicos y son, por tanto, entendidos por las personas. Sin embargo, los nombres de host proporcionan poca o ninguna información acerca de la ubicación del host dentro de Internet. (Un nombre de host como [www.eurecom.fr](http://eurecom.fr), que termina con el código de país, .fr, nos informa de que es probable que el host

se encuentre en Francia, pero esto no dice mucho más.) Además, puesto que los nombres de host pueden constar de caracteres alfanuméricos de longitud variable, podrían ser difíciles de procesar por los routers. Por estas razones, los hosts también se identifican mediante **direcciones IP**.

En el Capítulo 4 veremos en detalle las direcciones IP, pero le resultará útil leer ahora una breve exposición sobre las mismas. Una dirección IP consta de cuatro bytes y sigue una rígida estructura jerárquica. El aspecto de una dirección IP es, por ejemplo, 121.7.106.83, donde cada punto separa uno de los bytes expresados en notación decimal con valores comprendidos entre 0 y 255. Una dirección IP es jerárquica porque al explorar la dirección de izquierda a derecha, se obtiene información cada vez más específica acerca del lugar en el que está ubicado el host dentro de Internet (es decir, en qué red de la red de redes). De forma similar, cuando examinamos una dirección postal de abajo a arriba, obtenemos cada vez información más específica acerca del lugar definido por la dirección.

### 2.4.1 Servicios proporcionados por DNS

Acabamos de ver que hay dos formas de identificar un host, mediante un nombre de host y mediante una dirección IP. Las personas prefieren utilizar como identificador el nombre de host, mientras que los routers prefieren emplear las direcciones IP de longitud fija y que siguen una estructura jerárquica. Para reconciliar estas preferencias necesitamos un servicio de directorio que traduzca los nombres de host en direcciones IP. Esta es la tarea principal que lleva a cabo el **Sistema de nombres de dominio (DNS, Domain Name System)** de Internet. DNS es (1) una base de datos distribuida implementada en una jerarquía de servidores DNS y (2) un protocolo de la capa de aplicación que permite a los hosts consultar la base de datos distribuida. Los servidores DNS suelen ser máquinas UNIX que ejecutan el software BIND (*Berkeley Internet Name Domain, Dominio de nombres de Internet de Berkeley*) [BIND 2016]. El protocolo DNS se ejecuta sobre UDP y utiliza el puerto 53.

Otros protocolos de la capa de aplicación, como HTTP y SMTP, emplean habitualmente DNS para traducir los nombres de host suministrados por el usuario en direcciones IP. Por ejemplo, veamos qué ocurre cuando un navegador (es decir, un cliente HTTP), que se ejecuta en un determinado host de usuario, solicita el URL `www.unaescuela.edu/index.html`. Para que el host del usuario pueda enviar un mensaje de solicitud HTTP al servidor web `www.unaescuela.edu`, el host del usuario debe obtener en primer lugar la dirección IP de `www.unaescuela.edu`. Esto se hace del siguiente modo:

1. La propia máquina cliente ejecuta el lado del cliente de la aplicación DNS.
2. El navegador extrae el nombre de host, `www.unaescuela.edu`, del URL y pasa el nombre de host al lado del cliente de la aplicación DNS.
3. El cliente DNS envía una consulta que contiene el nombre de host a un servidor DNS.
4. El cliente DNS recibe finalmente una respuesta, que incluye la dirección IP correspondiente al nombre del host.
5. Una vez que el navegador recibe la dirección IP del servidor DNS, puede iniciar una conexión TCP con el proceso servidor HTTP localizado en el puerto 80 en esa dirección IP.

Podemos ver a partir de este ejemplo que DNS añade un retardo adicional, en ocasiones, sustancial, a las aplicaciones de Internet que le utilizan. Afortunadamente, como veremos más adelante, la dirección IP deseada a menudo está almacenada en caché en un servidor DNS “próximo”, lo que ayuda a reducir el tráfico de red DNS, así como el retardo medio del servicio DNS.

DNS proporciona algunos otros servicios importantes además de la traducción de los nombres de host en direcciones IP:

- **Alias de host.** Un host con un nombre complicado puede tener uno o más alias. Por ejemplo, un nombre de host como `relay1.west-coast.enterprise.com` podría tener, digamos, dos alias como `enterprise.com` y `www.enterprise.com`. En este caso, el nombre de host

`relay1.west-coast.enterprise.com` se dice que es el **nombre de host canónico**. Los alias de nombres de host, cuando existen, normalmente son más mnemónicos que los nombres canónicos. Una aplicación puede invocar DNS para obtener el nombre de host canónico para un determinado alias, así como la dirección IP del host.

- **Alias del servidor de correo.** Por razones obvias, es enormemente deseable que las direcciones de correo electrónico sean mnemónicas. Por ejemplo, si Benito tiene una cuenta de Yahoo Mail, su dirección de correo puede ser tan simple como `benito@yahoo.mail`. Sin embargo, el nombre de host del servidor de correo de Yahoo es más complicado y mucho menos mnemónico que simplemente `yahoo.com` (por ejemplo, el nombre canónico podría ser algo parecido a `relay1.west-coast.yahoo.com`). Una aplicación de correo puede invocar al servicio DNS para obtener el nombre de host canónico para un determinado alias, así como la dirección IP del host. De hecho, el registro MX (véase más adelante) permite al servidor de correo y al servidor web de una empresa tener nombres de host (con alias) iguales; por ejemplo, tanto el servidor web como el servidor de correo de una empresa se pueden llamar `enterprise.com`.
- **Distribución de carga.** DNS también se emplea para realizar la distribución de carga entre servidores replicados, como los servidores web replicados. Los sitios con una gran carga de trabajo, como `cnn.com`, están replicados en varios servidores, ejecutándose cada servidor en un sistema terminal distinto y teniendo cada uno una dirección IP diferente. Para los servidores web replicados hay asociado un *conjunto* de direcciones IP con un único nombre de host canónico. La base de datos DNS contiene este conjunto de direcciones IP. Cuando los clientes realizan una consulta DNS sobre un nombre asignado a un conjunto de direcciones, el servidor responde con el conjunto completo de direcciones IP, pero rota el orden de las direcciones en cada respuesta. Dado que normalmente un cliente envía su mensaje de solicitud HTTP a la dirección IP que aparece en primer lugar dentro del conjunto, la rotación DNS distribuye el tráfico entre los servidores replicados. La rotación DNS también se emplea para el correo electrónico de modo que múltiples servidores de correo pueden tener el mismo alias. Además, las empresas de distribución de contenido como Akamai han utilizado DNS de formas muy sofisticadas [Dilley 2002] para proporcionar la distribución de contenido web (véase la Sección 2.6.3).

DNS está especificado en los documentos RFC 1034 y RFC 1035, y actualizado en varios RFC adicionales. Es un sistema complejo y aquí solo hemos visto los aspectos básicos de su funcionamiento. El lector interesado puede consultar estos RFC y el libro de Abitz y Liu [Abitz 1993]; también puede consultar el documento retrospectivo [Mockapetris 1988], que proporciona una descripción sobre qué es DNS y el por qué de DNS; asimismo puede ver [Mockapetris 2005].

### 2.4.2 Cómo funciona DNS

Ahora vamos a presentar a alto nivel cómo funciona DNS. Nos centraremos en el servicio de traducción nombre de host-dirección IP.

Suponga que una determinada aplicación (como por ejemplo un navegador web o un lector de correo), que se ejecuta en el host de un usuario, necesita traducir un nombre de host en una dirección IP. La aplicación invocará al lado del cliente de DNS, especificando el nombre de host del que necesita la correspondiente traducción. (En muchos sistemas basados en UNIX, `gethostbyname()` es la llamada a función que una aplicación utiliza para llevar a cabo la traducción.) Entonces, la aplicación DNS en el host del usuario entra en funcionamiento, enviando un mensaje de consulta a la red. Todos los mensajes de consulta y de respuesta DNS se envían dentro de datagramas UDP al puerto 53. Transcurrido un cierto retardo, del orden de milisegundos a segundos, el servicio DNS del host del usuario recibe un mensaje de respuesta DNS que proporciona la traducción deseada, la cual se pasa entonces a la aplicación que lo ha invocado. Por tanto, desde la perspectiva de dicha aplicación que se ejecuta en el host del usuario, DNS es una caja negra que proporciona un servicio de traducción simple y directo. Pero, de hecho, la caja negra que implementa el servicio es compleja,



## EN LA PRÁCTICA

### DNS: FUNCIONES CRÍTICAS DE RED MEDIANTE EL PARADIGMA CLIENTE-SERVIDOR

Como HTTP, FTP y SMTP, el protocolo DNS es un protocolo de la capa de aplicación puesto que (1) se ejecuta entre sistemas terminales que están en comunicación utilizando el paradigma cliente-servidor y (2) se basa en un protocolo de transporte subyacente extremo a extremo para transferir los mensajes DNS entre los sistemas terminales en comunicación. Sin embargo, desde otro punto de vista, la función de DNS es bastante diferente a la de las aplicaciones web de transferencia de archivos y de correo electrónico. A diferencia de estas aplicaciones, DNS no es una aplicación con la que el usuario interactúe directamente; en su lugar, DNS lleva a cabo una de las funciones básicas en Internet: traducir los nombres de hosts en sus direcciones IP subyacentes para las aplicaciones de usuario y otras aplicaciones software de Internet. Hemos mencionado en la Sección 1.2 que gran parte de la complejidad de la arquitectura de Internet se encuentra en las “fronteras” de la red. DNS, que implementa el importante proceso de traducción de nombres en direcciones, utilizando clientes y servidores ubicados en la frontera de la red, es otro ejemplo más de dicha filosofía de diseño.

constando de un gran número de servidores DNS distribuidos por todo el mundo, así como de un protocolo de la capa de aplicación que especifica cómo los servidores DNS y los hosts que realizan las consultas se comunican.

Un diseño simple de DNS podría ser un servidor DNS que contuviera todas las correspondencias entre nombres y direcciones. En este diseño centralizado, los clientes simplemente dirigirían todas las consultas a un mismo servidor DNS y este respondería directamente a las consultas de los clientes. Aunque la simplicidad de este diseño es atractiva, es inapropiado para la red Internet de hoy día a causa de la enorme (y creciente) cantidad de hosts. Entre los problemas con un diseño centralizado podemos citar los siguientes:

- **Un único punto de fallo.** Si el servidor DNS falla, entonces ¡también falla toda la red Internet!
- **Volumen de tráfico.** Un único servidor DNS tendría que gestionar todas las consultas DNS (de todas las solicitudes HTTP y mensajes de correo electrónico generados en cientos de millones de hosts).
- **Base de datos centralizada distante.** Un único servidor DNS no puede “estar cerca” de todos los clientes que efectúan consultas. Si colocamos ese único servidor DNS en la ciudad de Nueva York, entonces todas las consultas desde Australia deberían viajar hasta el otro extremo del globo, quizás a través de enlaces lentos y congestionados. Esto podría llevar por tanto a retardos significativos.
- **Mantenimiento.** Ese único servidor DNS tendría que mantener registros de todos los hosts de Internet. No solo sería una enorme base de datos centralizada, sino que tendría que ser actualizada con frecuencia con el fin de incluir todos los hosts nuevos.

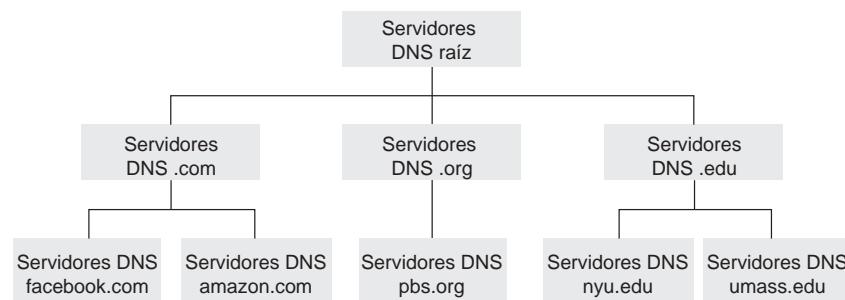
En resumen, una base de datos centralizada almacenada en un único servidor DNS simplemente *no podría escalarse*. En consecuencia, el sistema DNS utiliza un diseño distribuido. De hecho, DNS es un estupendo ejemplo de cómo se puede implementar una base de datos distribuida en Internet.

### Una base de datos jerárquica y distribuida

Para abordar el problema del escalado, DNS utiliza un gran número de servidores, organizados de forma jerárquica y distribuidos alrededor de todo el mundo. Ningún servidor DNS dispone de todas las correspondencias de todos los hosts de Internet. En su lugar, las correspondencias están repartidas por los servidores DNS. En una primera aproximación, podemos decir que existen tres clases de

servidores DNS: los servidores DNS raíz, los servidores DNS de dominio de nivel superior (TLD, *Top-Level Domain*) y los servidores DNS autoritativos, organizados en una jerarquía como la mostrada en la Figura 2.17. Con el fin de comprender cómo estas tres clases de servidores interactúan, suponga que un cliente DNS desea determinar la dirección IP correspondiente al nombre de host `www.amazon.com`. En una primera aproximación tienen lugar los siguientes sucesos: primero, el cliente contacta con uno de los servidores raíz, el cual devuelve las direcciones IP para los servidores TLD del dominio de nivel superior `.com`. A continuación, el cliente contacta con uno de estos servidores TLD, que devuelve la dirección IP de un servidor autoritativo para `amazon.com`. Por último, el cliente contacta con uno de los servidores autoritativos de `amazon.com`, el cual devuelve la dirección IP correspondiente al nombre de host `www.amazon.com`. Enseguida examinaremos este proceso de búsqueda DNS con más detalle. Pero en primer lugar, echemos un vistazo a estas tres clases de servidores DNS:

- **Servidores DNS raíz.** Existen unos 400 servidores de nombres raíz distribuidos por todo el mundo. La Figura 2.18 muestra los países que disponen de estos servidores, con los países que tienen más de diez servidores sombreados en gris oscuro. Trece organizaciones diferentes gestionan estos servidores de nombres raíz. Puede encontrar una lista completa de los servidores de nombres raíz, junto con las organizaciones que los gestionan y sus direcciones IP en [Root Servers 2016]. Los servidores de nombres raíz proporcionan las direcciones IP de los servidores TLD.
- **Servidores de dominio de nivel superior (TLD).** Para todos los dominios de nivel superior, como son `.com`, `.org`, `.net`, `.edu` y `.gov`, y todos los dominios de nivel superior correspondientes a los distintos países, como por ejemplo, `.uk`, `.fr`, `.ca` y `.jp`, existe un servidor TLD (o agrupación de servidores). La empresa Verisign Global Registry Services mantiene los servidores TLD para el dominio de nivel superior `.com` y la empresa Educause mantiene los servidores TLD para el dominio de nivel superior `.edu`. La infraestructura de red que da soporte a un TLD puede ser muy grande y compleja; consulte [Osterweil 2012] para ver una introducción de la red de Verisign. Consulte [TLD list 2016] para ver una lista de todos los dominios de nivel superior. Los servidores TLD proporcionan las direcciones IP para los servidores DNS autoritativos.
- **Servidores DNS autoritativos.** Todas las organizaciones que tienen hosts accesibles públicamente (como son los servidores web y los servidores de correo) a través de Internet deben proporcionar registros DNS accesibles públicamente que establezcan la correspondencia entre los nombres de dichos hosts y sus direcciones IP. Un servidor DNS autoritativo de una organización alberga estos registros DNS. Una organización puede elegir implementar su propio servidor DNS autoritativo para almacenar estos registros; alternativamente, la organización puede pagar por tener esos registros almacenados en un servidor DNS autoritativo de algún proveedor de servicios. La mayoría de las universidades y de las empresas de gran tamaño implementan y mantienen sus propios servidores DNS autoritativos principal y secundario (*backup*).



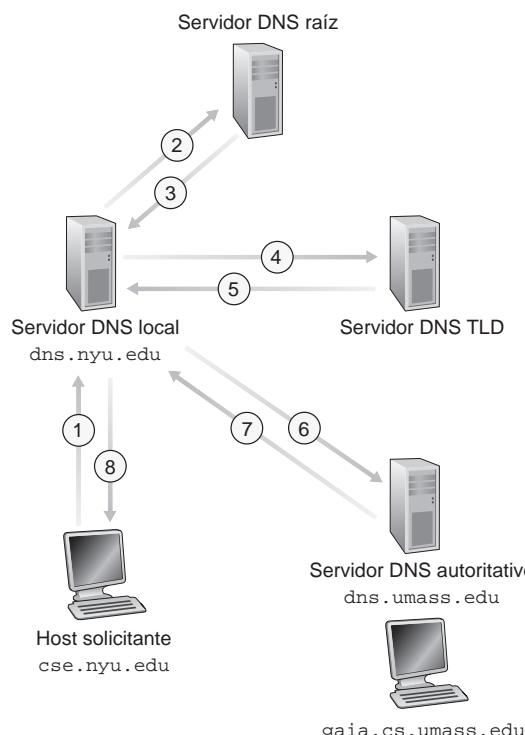
**Figura 2.17** ♦ Parte de la jerarquía de los servidores DNS.



**Figura 2.18** ♦ Servidores DNS raíz en 2016.

Todos los servidores DNS raíz, TLD y autoritativos pertenecen a la jerarquía de servidores DNS, como se muestra en la Figura 2.17. Existe otro tipo importante de servidor DNS conocido como **servidor DNS local**. Un servidor DNS local no pertenece estrictamente a la jerarquía de servidores, pero no obstante es importante dentro de la arquitectura DNS. Cada ISP (como por ejemplo un ISP residencial o un ISP institucional) tiene un servidor DNS local (también denominado servidor de nombres predeterminado). Cuando un host se conecta a un ISP, este proporciona al host las direcciones IP de uno o más de sus servidores DNS locales (normalmente a través de DHCP, lo que veremos en el Capítulo 4). Usted puede determinar fácilmente la dirección IP de su servidor DNS local accediendo a las ventanas de estado de la red en Windows o UNIX. Generalmente, un servidor DNS local de un host se encuentra “cerca” de ese host. En el caso de un ISP institucional, el servidor DNS local puede encontrarse en la misma LAN que el host; en el caso de un ISP residencial, estará separado del host no más de unos pocos routers. Cuando un host realiza una consulta DNS, esta se envía al servidor DNS local, que actúa como proxy, reenviando la consulta a la jerarquía de servidores DNS, como veremos más detalladamente a continuación.

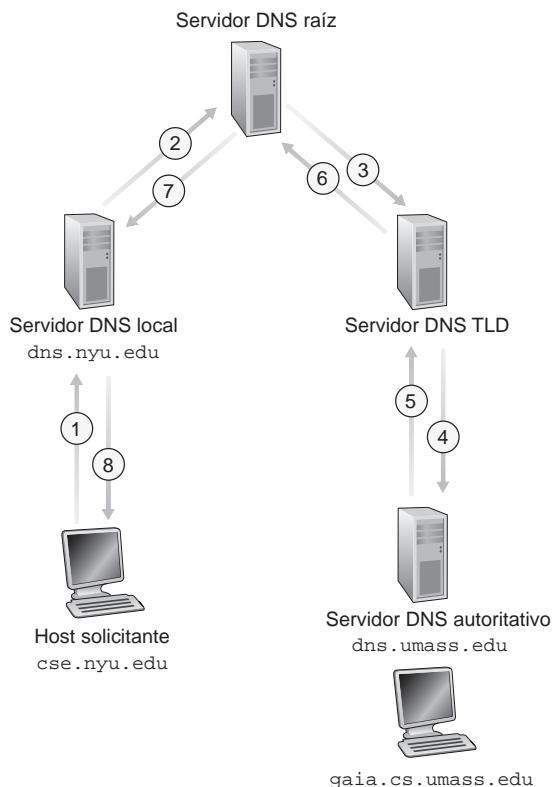
Veamos un ejemplo sencillo. Suponga que el host `cse.nyu.edu` desea conocer la dirección de `gaia.cs.umass.edu`. Suponga también que el servidor DNS local de la Universidad de Nueva York para `cse.nyu.edu` se denomina `dns.nyu.edu` y que un servidor DNS autoritativo para `gaia.cs.umass.edu` se llama `dns.umass.edu`. Como se muestra en la Figura 2.19, el host `cse.nyu.edu` envía en primer lugar un mensaje de consulta DNS a su servidor DNS local, `dns.nyu.edu`. El mensaje de consulta contiene el nombre de host que debe ser traducido, `gaia.cs.umass.edu`. El servidor DNS local reenvía la consulta a un servidor DNS raíz, el cual toma el sufijo `edu` y devuelve al servidor DNS local una lista de las direcciones IP de los servidores TLD responsables del dominio `edu`. El servidor DNS local reenvía a continuación el mensaje de consulta a uno de estos servidores TLD. El servidor TLD toma nota del sufijo `umass.edu` y responde con la dirección IP del servidor DNS autoritativo correspondiente a la Universidad de Massachusetts, es decir, `dns.umass.edu`. Por último, el servidor DNS local reenvía la consulta directamente a `dns.umass.edu`, que responde con la dirección IP de `gaia.cs.umass.edu`. Observe que en este ejemplo, para obtener la dirección correspondiente a un nombre de host, se han enviado ocho mensajes DNS: ¡cuatro mensajes de consulta y cuatro mensajes de respuesta! Pronto veremos cómo el almacenamiento en caché DNS reduce este tráfico de consultas.



**Figura 2.19** ◆ Interacción de los distintos servidores DNS.

En el ejemplo anterior suponímos que el servidor TLD conoce el servidor DNS autoritativo correspondiente al nombre de host. En general esto no es así. En lugar de ello, el servidor TLD puede saber únicamente de un servidor DNS intermedio, el cual a su vez sabe cuál es el servidor DNS autoritativo para el nombre de host. Por ejemplo, suponga de nuevo que la Universidad de Massachusetts tiene un servidor DNS para la universidad, denominado dns.umass.edu. Suponga también que cada uno de los departamentos de la Universidad de Massachusetts tiene su propio servidor DNS y que cada servidor DNS departamental es un servidor autoritativo para todos los hosts del departamento. En este caso, cuando el servidor DNS intermedio dns.umass.edu recibe una consulta para un host cuyo nombre termina en cs.umass.edu, devuelve a dns.nyu.edu la dirección IP de dns.cs.umass.edu, que es autoritativo para todos los nombres de host que terminan con cs.umass.edu. El servidor DNS local dns.nyu.edu envía entonces la consulta al servidor DNS autoritativo, que devuelve la correspondencia deseada al servidor DNS local, el cual a su vez devuelve la correspondencia al host que ha hecho la solicitud. En este caso, ¡se han enviado un total de 10 mensajes!

El ejemplo mostrado en la Figura 2.19 utiliza tanto **consultas recursivas** como **consultas iterativas**. La consulta enviada desde cse.nyu.edu a dns.nyu.edu es una consulta recursiva, ya que la consulta solicita a dns.nyu.edu que obtenga por sí mismo la correspondencia. Pero las tres consultas siguientes son iterativas puesto que todas las respuestas son devueltas directamente a dns.nyu.edu. En teoría, cualquier consulta DNS puede ser iterativa o recursiva. Por ejemplo, la Figura 2.20 muestra una cadena de consultas DNS para las que todas las consultas son recursivas. En la práctica, las consultas normalmente siguen el patrón mostrado en la Figura 2.19: la consulta procedente del host que hace la solicitud al servidor DNS local es recursiva y las restantes consultas son iterativas.



**Figura 2.20** ♦ Consultas recursivas en DNS.

### Almacenamiento en caché DNS

Hasta el momento hemos ignorado el **almacenamiento en caché DNS**, una funcionalidad extremadamente importante del sistema DNS. En realidad, DNS explota exhaustivamente el almacenamiento en caché para mejorar el comportamiento de los retardos y reducir el número de mensajes DNS que van de un lado a otro por Internet. La idea que se esconde detrás del almacenamiento en caché DNS es muy simple. En una cadena de consultas, cuando un servidor DNS recibe una respuesta DNS (que contiene, por ejemplo, una correspondencia entre un nombre de host y una dirección IP), puede almacenar esta información en su memoria local. Por ejemplo, en la Figura 2.19, cada vez que el servidor DNS local `dns.nyu.edu` recibe una respuesta de algún servidor DNS, puede almacenar en caché cualquier información contenida en esa respuesta. Si una relación nombre de host/dirección IP se almacena en la caché de un servidor DNS y llegan otras consultas a ese mismo servidor DNS preguntando por el mismo nombre de host, el servidor DNS puede proporcionar la dirección IP deseada, incluso aunque no sea autoritativo para el nombre de host. Dado que los hosts y las correspondencias entre nombres de host y direcciones IP no son permanentes, los servidores DNS descartan la información almacenada en caché pasado un cierto periodo de tiempo (normalmente, unos dos días).

Por ejemplo, suponga que un host `apricot.nyu.edu` consulta a `dns.nyu.edu` solicitándole la dirección IP correspondiente al nombre de host `cnn.com`. Suponga también que unas pocas horas más tarde, otro host de la Universidad de Nueva York, digamos `kiwi.nyu.edu`, también hace una consulta a `dns.nyu.edu` especificando el mismo nombre de host. Gracias al almacenamiento en caché, el servidor DNS local podrá devolver de forma inmediata la dirección IP de `cnn.com` al segundo host que la ha solicitado sin tener que consultar a ningún

otro servidor DNS. Un servidor DNS local también puede almacenar en caché las direcciones IP de los servidores TLD, permitiendo así a los servidores DNS locales saltarse a los servidores DNS raíz en una cadena de consultas. De hecho, gracias al almacenamiento en caché, todas las consultas DNS, excepto una pequeña fracción de las mismas, se saltan a los servidores raíz.

### 2.4.3 Registros y mensajes DNS

Los servidores DNS que implementan conjuntamente la base de datos distribuida DNS almacenan los **registros de recursos (RR)**, incluyendo los que proporcionan las correspondencias entre nombre de host y dirección IP. Cada mensaje de respuesta DNS transporta uno o más registros de recursos. En esta y en la subsección siguiente, proporcionamos una breve introducción a los recursos y mensajes DNS; puede encontrar información detallada en [Abitz 1993] o en los RFC sobre DNS [RFC 1034; RFC 1035].

Un registro de recurso está formado por los siguientes cuatro campos:

(Nombre, Valor, Tipo, TTL)

TTL es el tiempo de vida del registro de recurso; determina cuándo un recurso debería ser eliminado de una caché. En los registros de ejemplo dados a continuación, hemos ignorado el campo TTL. El significado de Nombre y Valor depende del campo Tipo:

- Si Tipo=A, entonces Nombre es un nombre de host y Valor es la dirección IP correspondiente a dicho nombre. Por tanto, un registro de tipo A proporciona la correspondencia estándar nombre de host-dirección IP. Por ejemplo, (relay1.bar.foo.com, 145.37.93.126, A) es un registro de tipo A.
- Si Tipo = NS, entonces Nombre es un dominio (como foo.com) y Valor es el nombre de host de un servidor DNS autoritativo que sabe cómo obtener las direcciones IP de los hosts del dominio. Este registro se utiliza para enrutar las consultas DNS a lo largo de la cadena de consultas. Por ejemplo, (foo.com, dns.foo.com, NS) es un registro de tipo NS.
- Si Tipo = CNAME, entonces Valor es un nombre de host canónico correspondiente al alias especificado por Nombre. Este registro puede proporcionar a los hosts que hacen consultas el nombre canónico correspondiente a un nombre de host. Por ejemplo, (foo.com, relay1.bar.foo.com) es un registro CNAME.
- Si Tipo = MX, entonces Valor es el nombre canónico de un servidor de correo que tiene un alias dado por Nombre. Por ejemplo, (foo.com, mail.bar.foo.com, MX) es un registro MX. Los registros MX permiten a los nombres de host de los servidores de correo tener alias simples. Observe que utilizando el registro MX, una empresa puede tener el mismo alias para su servidor de correo y para uno de sus otros servidores (como por ejemplo, el servidor web). Para obtener el nombre canónico del servidor de correo, un cliente DNS consultaría un registro MX y para conocer el nombre canónico del otro servidor, consultaría el registro CNAME.

Si un servidor DNS es autoritativo para un determinado nombre de host, entonces el servidor DNS contendrá un registro de tipo A para el nombre de host. (Incluso aunque el servidor DNS no sea autoritativo, puede contener un registro de tipo A en su caché.) Si un servidor no es autoritativo para un nombre de host, entonces el servidor contendrá un registro de tipo NS para el dominio que incluye el nombre de host; también contendrá un registro de tipo A que proporcione la dirección IP del servidor DNS en el campo Valor del registro NS. Por ejemplo, suponga un servidor TLD edu que no es autoritativo para el host gaia.cs.umass.edu. Por tanto, este servidor contendrá un registro para un dominio que incluye el host gaia.cs.umass.edu, por ejemplo, (umass.edu, dns.umass.edu, NS). El servidor TLD edu también contendrá un registro de tipo A, que establece la correspondencia entre el servidor DNS dns.umass.edu y una dirección IP, como en (dns.umass.edu, 128.119.40.111, A).

## Mensajes DNS

Anteriormente en esta sección hemos hecho referencia a los mensajes de respuesta y consultas DNS. Únicamente existen estas dos clases de mensajes DNS. Además, tanto los mensajes de respuesta como de consulta utilizan el mismo formato, que se muestra en la Figura 2.21. La semántica en los distintos campos de un mensaje DNS es la siguiente:

- Los primeros 12 bytes constituyen la *sección de cabecera*, la cual contiene una serie de campos. El primero de estos campos es un número de 16 bits que identifica la consulta. Este identificador se copia en el mensaje de respuesta a la consulta, lo que permite al cliente establecer las correspondencias correctas entre las respuestas recibidas y las consultas enviadas. En el campo Indicadores se incluyen una serie de indicadores. Un indicador consulta/respondida de 1 bit informa de si el mensaje es una consulta (0) o una respuesta (1). Un indicador autoritativo de 1 bit se activa en un mensaje de respuesta cuando un servidor DNS es un servidor autoritativo para un nombre solicitado. El indicador recursión-deseada, también de 1 bit, se activa cuando un cliente (host o servidor DNS) desea que el servidor DNS realice una recursión cuando no disponga del registro. En un mensaje de respuesta, el campo de recursión-disponible de 1 bit se activa si el servidor DNS soporta la recursión. En la cabecera también se incluyen cuatro campos “número de”, que indican el número de apariciones de los cuatro tipos de secciones de datos que siguen a la cabecera.
- La *sección cuestiones* contiene información acerca de la consulta que se va a realizar. Esta sección incluye (1) un campo de nombre que contiene el nombre que se va a consultar y (2) un campo de tipo que especifica el tipo de cuestión que se plantea acerca del nombre; por ejemplo, la dirección del host asociada con un nombre (tipo A) o el servidor de correo para un nombre (tipo MX).
- En una respuesta de un servidor DNS, la *sección respuestas* contiene los registros del recurso para el nombre que fue consultado originalmente. Recuerde que en cada registro de recurso existe un parámetro **Tipo** (por ejemplo, A, NS, CNAME y MX), un parámetro **Valor** y el parámetro **TTL**. Una respuesta puede devolver varios registros de recursos, ya que un nombre de host puede tener asociadas varias direcciones IP (por ejemplo, para los servidores web replicados, como hemos visto anteriormente en esta sección).
- La *sección autoridad* contiene registros de otros servidores autoritativos.
- La *sección información adicional* contiene otros registros útiles. Por ejemplo, el campo de respuesta en un mensaje de respuesta a una consulta MX contiene un registro de recurso que

Identificación	Indicadores	
Nº de cuestiones	Nº de RR de respuesta	12 bytes
Nº de RR de autoridad	Nº de RR adicionales	
Cuestiones (número variable de cuestiones)		Campos de nombre y tipo para una consulta
Respuestas (número variable de registros de recursos)		Registros RR en respuesta a la consulta
Autoridad (número variable de registros de recursos)		Registros para servidores autoritativos
Información adicional (número variable de registros de recursos)		Información de “ayuda” adicional que se puede utilizar

**Figura 2.21** ♦ Formato de los mensajes DNS.

proporciona el nombre de host canónico de un servidor de correo. Esta sección de información adicional contiene un registro de tipo A que proporciona la dirección IP para el nombre de host canónico del servidor de correo.

¿Le gustaría enviar un mensaje de consulta DNS directamente desde el host en el que está trabajando a algún servidor DNS? Esto podría hacerse fácilmente con el **programa nslookup**, que está disponible en la mayoría de las plataformas Windows y UNIX. Por ejemplo, en un host Windows basta con abrir la aplicación Símbolo del sistema (*prompt*) e invocar al programa nslookup escribiendo simplemente “nslookup”. A continuación, podrá enviar una consulta DNS a cualquier servidor DNS (raíz, TLD o autoritativo). Después de recibir el mensaje de respuesta del servidor DNS, nslookup mostrará los registros incluidos en la respuesta (en un formato legible para las personas). Como alternativa a la ejecución de nslookup desde su propio host, puede visitar uno de los muchos sitios web que permiten utilizar nslookup de forma remota (basta con escribir “nslookup” en un motor de búsqueda para acceder a uno de estos sitios). La práctica de laboratorio DNS Wireshark disponible al final de este capítulo le permitirá explorar DNS en detalle.

### Inserción de registros en la base de datos DNS

La exposición anterior se ha centrado en cómo se recuperan los registros de la base de datos DNS. Es posible que ahora se esté preguntando cómo se introducen los registros en dicha base de datos. Veamos cómo se hace esto en el contexto de un ejemplo concreto. Suponga que hemos creado una nueva empresa llamada Network Utopia. Lo primero que seguramente deseará hacer es registrar el nombre de dominio `networkutopia.com` en un registro. Un **registrador** es una entidad comercial que verifica la unicidad del nombre de dominio, lo añade a la base de datos DNS (como veremos más adelante) y percibe unas pequeñas tasas por sus servicios. Antes de 1999, un único registrador, Network Solutions, tenía el monopolio sobre el registro de nombres para los dominios `.com`, `.net` y `.org`. Pero actualmente, existen muchas entidades registradoras que compiten por los clientes. La ICANN (*Internet Corporation for Assigned Names and Numbers*, Corporación de Internet para nombres y números asignados) acredita a las distintas entidades registradoras. En la dirección <http://www.internic.net> puede encontrar una lista completa de estas entidades.

Al registrar el nombre de dominio `networkutopia.com` en alguna entidad registradora, también tendrá que proporcionarle los nombres y direcciones IP de sus servidores DNS autoritativos principal y secundario. Suponga que en nuestro ejemplo estos datos son: `dns1.networkutopia.com`, `dns2.networkutopia.com`, `212.212.212.1` y `212.212.212.2`. Para cada uno de estos dos servidores DNS autoritativos, el registrador se asegura de que se introduzca un registro de tipo NS y un registro de tipo A en los servidores TLD `.com`. Específicamente, para el servidor autoritativo principal de `networkutopia.com`, la entidad registradora deberá insertar los siguientes dos registros de recursos en el sistema DNS:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

También tendrá que asegurarse de que el registro de recurso de tipo A para su servidor web `www.networkutopia.com` y el registro de recurso de tipo MX para su servidor de correo `mail.networkutopia.com` se han introducido en sus servidores DNS autoritativos. (Hasta hace poco, los contenidos de los servidores DNS se configuraban estáticamente; por ejemplo, a partir de un archivo de configuración creado por un administrador del sistema. Sin embargo, recientemente se ha añadido una opción de actualización (UPDATE) al protocolo DNS que permite añadir o borrar dinámicamente los datos de la base de datos mediante mensajes DNS. En los documentos [RFC 2136] y [RFC 3007] se especifican las actualizaciones dinámicas de DNS.)

Una vez que haya completado estos pasos, los usuarios podrán visitar su sitio web y enviar correos electrónicos a los empleados de su empresa. Vamos a terminar esta exposición sobre DNS verificando que esta afirmación es cierta. Esta verificación también le ayudará a consolidar lo que

## SEGURIDAD

### VULNERABILIDADES DNS

Hemos visto que DNS es un componente fundamental de la infraestructura de Internet y que muchos servicios importantes, entre los que incluyen las aplicaciones web y de correo electrónico, no podrían funcionar sin él. Por tanto, lo natural es preguntarse: ¿Cómo puede ser atacado DNS? ¿Es DNS un blanco fácil, que espera a ser puesto fuera de servicio, arrastrando con él a la mayoría de las aplicaciones de Internet?

El primer tipo de ataque que nos viene a la mente es un ataque DDoS por de ancho de banda (véase la Sección 1.6) contra los servidores DNS. Por ejemplo, un atacante podría intentar enviar a cada uno de los servidores DNS raíz una gran cantidad de paquetes, tantos que la mayor parte de las consultas DNS legítimas nunca fueran contestadas. Un ataque DDoS a gran escala contra servidores DNS raíz tuvo lugar realmente el 21 de octubre de 2002. En ese ataque, los atacantes utilizaron una botnet para enviar enormes cargas de mensajes ping ICMP a las direcciones IP de 13 servidores DNS raíz. (Los mensajes ICMP se estudian en la Sección 5.6. Por el momento, nos basta con saber que los paquetes ICMP son tipos especiales de datagramas IP.) Afortunadamente, este ataque a gran escala causó unos daños mínimos, sin tener apenas impacto sobre la experiencia en Internet de los usuarios. Los atacantes dirigieron con éxito gran cantidad de paquetes a los servidores raíz, pero muchos de estos servidores estaban protegidos mediante mecanismos de filtrado de paquetes configurados para bloquear siempre todos los mensajes ping ICMP dirigidos a los mismos. Estos servidores protegidos estaban resguardados y funcionaron normalmente. Además, la mayoría de los servidores DNS locales tenían almacenadas en caché las direcciones IP de los servidores de dominio de nivel superior, permitiendo el procesamiento de consultas ignorando normalmente a los servidores DNS raíz.

Un ataque DDoS potencialmente más efectivo contra servidores DNS consistiría en enviar una gran cantidad de consultas DNS a los servidores de dominio de alto nivel; por ejemplo, a todos aquellos servidores que administren el dominio .com. Sería bastante complicado filtrar las consultas DNS dirigidas a servidores DNS; y los servidores de dominio de nivel superior no pueden ser puenteados tan fácilmente como los servidores raíz. Pero la severidad de un ataque así podría ser parcialmente mitigada por el almacenamiento en caché de los servidores DNS locales.

El sistema DNS también podría ser atacado, en teoría, de otras maneras. En un ataque por intermediación (*man-in-the-middle*), el atacante intercepta las consultas procedentes de los hosts y devuelve respuestas falsas. En el ataque por envenenamiento DNS, el atacante envía respuestas falsas a un servidor DNS, engañándole y haciendo que almacene en su caché registros falsos. Cualquiera de estos ataques podría utilizarse, por ejemplo, para redirigir a un usuario web inadvertido al sitio web del atacante. Sin embargo, estos ataques son difíciles de implementar, ya que requieren interceptar los paquetes o engañar a los servidores [Skoudis 2006].

En resumen, DNS ha demostrado ser sorprendentemente robusto frente a los ataques. Hasta el momento, no ha habido ningún ataque que haya conseguido interrumpir con éxito el servicio DNS.

ha aprendido sobre DNS. Suponga que Alicia se encuentra en Australia y desea ver la página web [www.networkutopia.com](http://www.networkutopia.com). Como hemos explicado anteriormente, su host enviará en primer lugar una consulta DNS a su servidor DNS local, el cual a su vez se pondrá en contacto con un servidor TLD .com. (El servidor DNS local también tendrá que comunicarse con un servidor DNS raíz si no tiene en caché la dirección de un servidor TLD .com.) El servidor TLD contendrá los registros de recursos de tipo NS y de tipo A enumerados anteriormente, ya que la entidad registradora los habrá almacenado en todos los servidores TLD .com. El servidor TLD .com envía entonces una respuesta al servidor DNS local de Alicia, conteniendo dicha respuesta los dos registros de recursos. A continuación, el servidor DNS local transmite una consulta DNS a 212.212.212.1, solicitando

el registro de tipo A correspondiente a [www.networkutopia.com](http://www.networkutopia.com). Este registro proporciona la dirección IP del servidor web deseado, por ejemplo, 212.212.71.4, que el servidor DNS local pasa al host de Alicia. En este momento, el navegador de Alicia puede iniciar una conexión TCP con el host 212.212.71.4 y enviar una solicitud HTTP a través de la misma. Como ha podido ver, son muchas las cosas que suceden entre bastidores cuando navegamos por la Web.

## 2.5 Distribución de archivos P2P

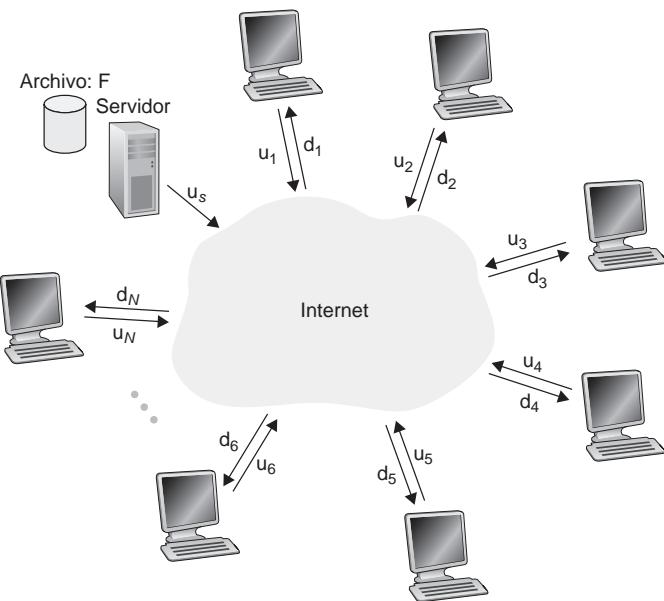
Las aplicaciones descritas en este capítulo hasta el momento, incluyendo las aplicaciones web, de correo electrónico y DNS, emplean todas ellas arquitecturas cliente-servidor que dependen en gran medida de que exista una infraestructura de servidores siempre activos. Recuerde de la Sección 2.1.1 que con una arquitectura P2P no se depende más que mínimamente (o no se depende en absoluto) de que exista esa infraestructura de servidores siempre activos. En su lugar, una serie de parejas de hosts conectados de forma intermitente, denominados pares o *peers*, se comunican directamente entre sí. Los pares no son propiedad de un proveedor de servicios, sino que son computadoras de escritorio o portátiles controlados por los usuarios.

En esta sección vamos a ver una aplicación P2P muy corriente, la distribución de un archivo de gran tamaño desde un único servidor a muchos otros hosts (denominados pares). El archivo podría ser una nueva versión del sistema operativo Linux, un parche software para una aplicación o un sistema operativo existentes, un archivo de audio MP3 o un archivo de vídeo MPEG. En la distribución de archivos cliente-servidor, el servidor debe enviar una copia del archivo a cada uno de los pares, provocando una enorme sobrecarga en el servidor y consumiendo una gran cantidad de su ancho de banda. En la distribución de archivos P2P, cada par puede redistribuir cualquier parte del archivo que ha recibido a cualesquier otros pares, ayudando de este modo al servidor a llevar a cabo el proceso de distribución. En 2016, el protocolo de distribución de archivos P2P más popular es BitTorrent. Originalmente, fue desarrollado por Bram Cohen, pero ahora existen muchos clientes BitTorrent independientes distintos que cumplen con el protocolo BitTorrent, al igual que existen diversos navegadores web que cumplen el protocolo HTTP. En esta subsección examinaremos en primer lugar la característica de auto-escalabilidad de las arquitecturas P2P en el contexto de la distribución de archivos. Después describiremos en detalle BitTorrent, destacando sus funcionalidades y características más importantes.

### Escalabilidad de las arquitecturas P2P

Con el fin de comparar las arquitecturas cliente-servidor con las arquitecturas P2P y para ilustrar la auto-escalabilidad inherente de P2P, ahora vamos a considerar un modelo cuantitativo simple para la distribución de un archivo a un conjunto fijo de pares en ambos tipos de arquitectura. Como se muestra en la Figura 2.22, el servidor y los pares están conectados a Internet mediante enlaces de acceso. Sea  $u_s$  la velocidad de carga del enlace de acceso del servidor,  $u_i$  la velocidad de carga del enlace de acceso del par  $i$  y  $d_i$  la velocidad de descarga del enlace de acceso del par  $i$ . Sea  $F$  el tamaño en bits del archivo que se va a distribuir y  $N$  el número de pares que desean obtener una copia del archivo. El **tiempo de distribución** es el tiempo que tardan los  $N$  pares en obtener una copia del archivo. En el análisis del tiempo de distribución que proporcionamos a continuación, para ambas arquitecturas, cliente-servidor y P2P, hemos hecho una simplificación (pero generalmente precisa [Akella 2003]): suponer que el núcleo de Internet tiene el ancho de banda suficiente, lo que implica que todos los cuellos de botella se encuentran en el acceso a red. También hemos supuesto que el servidor y los clientes no están participando en ninguna otra aplicación de red, de modo que los anchos de banda para carga y descarga están dedicados completamente a distribuir el archivo.

En primer lugar, vamos a determinar el tiempo de distribución para la arquitectura cliente-servidor, el cual denotaremos como  $D_{cs}$ . En esta arquitectura, ninguno de los pares ayudan a distribuir el archivo. Tenemos que hacer las dos observaciones siguientes:



**Figura 2.22** ♦ Problema de distribución de un archivo.

- El servidor debe transmitir una copia del archivo a cada uno de los  $N$  pares. Por tanto, el servidor tiene que transmitir  $NF$  bits. Puesto que la velocidad de carga del servidor es  $u_s$ , el tiempo para distribuir el archivo tiene que ser como mínimo  $NF/u_s$ .
- Sea  $d_{\min}$  la velocidad de descarga del par cuya velocidad de descarga sea menor; es decir,  $d_{\min} = \min\{d_1, d_2, \dots, d_N\}$ . El par con la menor velocidad de descarga no puede obtener los  $F$  bits del archivo en menos de  $F/d_{\min}$  segundos. Por tanto, el tiempo mínimo de distribución es, al menos igual a  $F/d_{\min}$ .

Teniendo en cuenta estas dos observaciones, se obtiene:

$$D_{cs} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{\min}}\right\}$$

Esto proporciona un límite inferior al tiempo de distribución mínimo para la arquitectura cliente-servidor. En los problemas de repaso se le pedirá que demuestre que el servidor puede planificar sus transmisiones de manera que el límite inferior sea alcanzado realmente. Por tanto, tomemos este límite inferior como el tiempo de distribución real, es decir,

$$D_{cs} = \max\left\{\frac{NF}{u_s}, \frac{F}{d_{\min}}\right\} \quad (2.1)$$

A partir de la Ecuación 2.1, se ve que para  $N$  lo suficientemente grande, el tiempo de distribución en una arquitectura cliente-servidor está dada por  $NF/u_s$ . Por tanto, el tiempo de distribución aumenta linealmente con el número de pares  $N$ . Así, por ejemplo, si el número de pares se multiplica por mil en una semana, pasando de mil a un millón, el tiempo necesario para distribuir el archivo a todos los pares se verá multiplicado por 1.000.

Hagamos ahora un análisis similar para la arquitectura P2P, donde cada par puede ayudar al servidor a distribuir el archivo. En particular, cuando un par recibe datos del archivo, puede utilizar su propia capacidad de carga para redistribuir los datos a otros pares. Calcular el tiempo

de distribución para la arquitectura P2P es algo más complicado que para la arquitectura cliente-servidor, ya que el tiempo de distribución depende de cómo cada par implicado distribuya partes del archivo a los demás pares. No obstante, puede obtenerse una expresión simple que permite calcular el tiempo mínimo de distribución [Kumar 2006]. Para este fin, debemos tener en cuenta las siguientes observaciones:

- Al comenzar el proceso de distribución, el archivo solo lo tiene el servidor. Para que este archivo llegue a la comunidad de pares, el servidor tiene que enviar cada bit del archivo al menos una vez por su enlace de acceso. Por tanto, el tiempo mínimo de distribución es, como mínimo,  $F/u_s$ . (A diferencia de lo que ocurre en el esquema cliente-servidor, un bit enviado por el servidor puede no tener que ser enviado de nuevo por el mismo, ya que los pares pueden redistribuirlo entre ellos.)
- Al igual que en la arquitectura cliente-servidor, el par con la menor velocidad de descarga no puede obtener los  $F$  bits del archivo en menos de  $F/d_{\min}$  segundos. Por tanto, el tiempo mínimo de distribución es al menos igual a  $F/d_{\min}$ .
- Por último, observe que la capacidad total de carga del sistema como un todo es igual a la velocidad de carga del servidor más las velocidades de carga de cada par, es decir,  $u_{total} = u_s + u_1 + \dots + u_N$ . El sistema tiene que suministrar (cargar)  $F$  bits en cada uno de los  $N$  peers, suministrando en total  $NF$  bits. Esto no se puede hacer a una velocidad mayor que  $u_{total}$ ; por tanto, el tiempo mínimo de distribución es también mayor o igual que  $NF/(u_s + u_1 + \dots + u_N)$ .

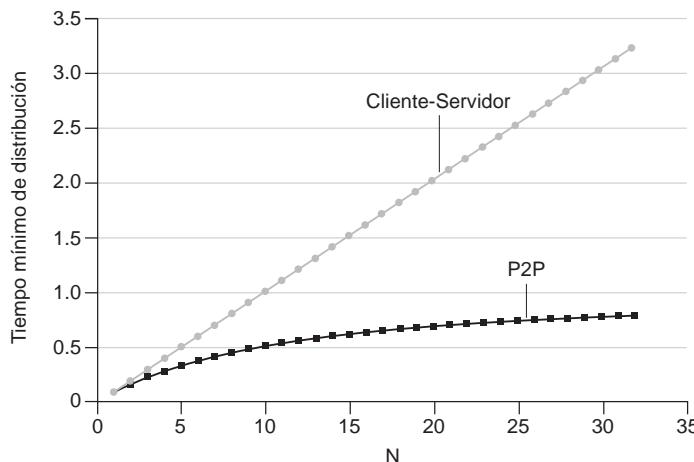
Teniendo en cuenta estas tres observaciones, obtenemos el tiempo mínimo de distribución para la arquitectura P2P,  $D_{P2P}$ :

$$D_{P2P} \geq \max \left\{ \frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^N u_i} \right\} \quad (2.2)$$

La Ecuación 2.2 proporciona un límite inferior para el tiempo mínimo de distribución en una arquitectura P2P. Si suponemos que cada peer puede redistribuir un bit tan pronto como lo recibe, entonces existe un esquema de redistribución que permite alcanzar este límite inferior [Kumar 2006]. (Demostraremos un caso especial de este resultado en los problemas de repaso.) En realidad, cuando se redistribuyen fragmentos del archivo en lugar de bits individuales, la Ecuación 2.2 sirve como una buena aproximación del tiempo mínimo de distribución real. Por tanto, vamos a tomar el límite inferior dado por la Ecuación 2.2 como el tiempo mínimo de distribución real, es decir,

$$D_{P2P} = \max \left\{ \frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^N u_i} \right\} \quad (2.3)$$

La Figura 2.23 compara el tiempo mínimo de distribución de las arquitecturas cliente-servidor y P2P, suponiendo que todos los pares tienen la misma velocidad de carga  $u$ . En la figura, hemos establecido que  $F/u = 1$  hora,  $u_s = 10u$  y  $d_{\min} \geq u_s$ . Por tanto, un par puede transmitir el archivo completo en una hora, la velocidad de transmisión del servidor es 10 veces la velocidad de carga del par y (para simplificar) las velocidades de descarga de los pares son lo suficientemente grandes como para no tener ningún efecto. A partir de la Figura 2.23, podemos ver que para la arquitectura cliente-servidor el tiempo de distribución aumenta linealmente y sin límite a medida que el número de pares aumenta. Sin embargo, en una arquitectura P2P, el tiempo mínimo de distribución no solo siempre es menor que el tiempo de distribución en la arquitectura cliente-servidor; también es menor que una hora para *cualquier* número  $N$  de pares. Por tanto, las aplicaciones que emplean arquitectura P2P pueden auto-escalarse. Esta escalabilidad es una consecuencia directa de que los pares actúan a la vez como redistribuidores y consumidores de bits.



**Figura 2.23** ♦ Tiempo de distribución para las arquitecturas P2P y cliente-servidor.

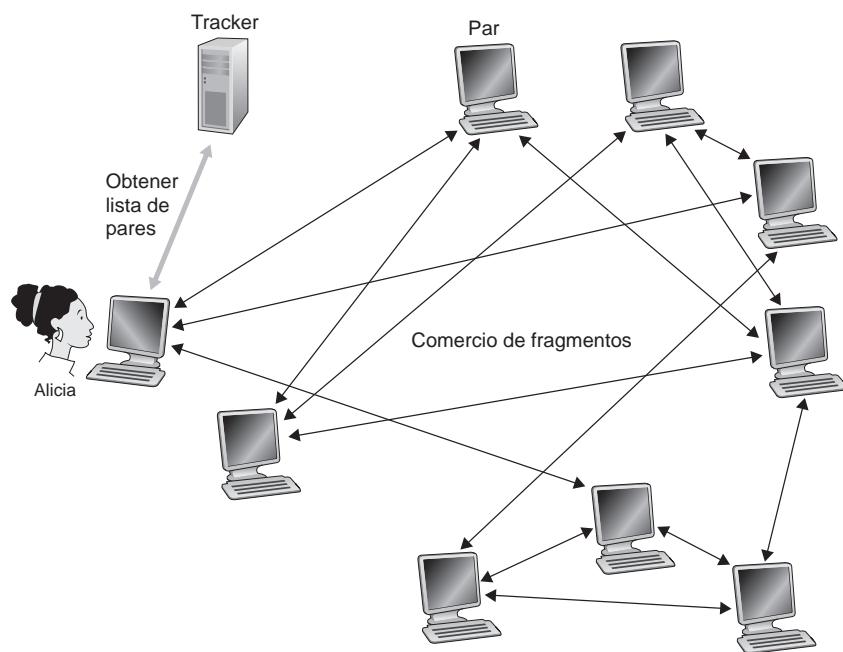
### BitTorrent

BitTorrent es un popular protocolo P2P para la distribución de archivos [Chao 2011]. En la jerga de BitTorrent, la colección de todos los pares que participan en la distribución de un determinado archivo se conoce como *torrent* (torrente). Los peers de un torrente descargan *fragmentos* del mismo tamaño del archivo de uno a otro, siendo el tamaño típico de un fragmento de 256 KBytes. Cuando un par se une por primera vez a un torrente, no tiene fragmentos del archivo. A lo largo del tiempo va acumulando cada vez más fragmentos. A la vez que descarga fragmentos, actualiza fragmentos en otros pares. Una vez que un par ha adquirido el archivo completo, puede (egoístamente) abandonar el torrente, o (de forma altruista) permanecer en el mismo y continuar suministrando fragmentos a otros pares. Además, cualquier par puede abandonar el torrente en cualquier instante con solo un subconjunto de fragmentos, y volver a unirse más tarde.

Veamos ahora más de cerca cómo opera BitTorrent. Puesto que BitTorrent es un sistema y protocolo bastante complejo, sólo vamos a describir sus mecanismos más importantes, vamos a dejar al margen algunos detalles con el fin de poder ver claramente cómo funciona. Cada torrente tiene un nodo de infraestructura denominado *tracker* (rastreador). Cuando un par se une a un torrente, se registra mediante el *tracker* y, periódicamente, informa de que todavía se encuentra en el torrente. De esta manera, el *tracker* sigue la pista a los pares que están participando en el torrente. Un determinado torrente puede tener en un instante dado un número de pares participantes tan bajo como diez o tan alto como mil.

Como se muestra en la Figura 2.24, cuando un nuevo par, Alicia, se une al torrente, el *tracker* selecciona aleatoriamente un subconjunto de pares (digamos por ejemplo 50, con el fin de concretar) del conjunto de peers participantes y envía las direcciones IP de estos 50 peers a Alicia. Teniendo en su poder esta lista de pares, Alicia intenta establecer conexiones TCP concurrentes con todos los pares incluidos en dicha lista. Denominaremos a todos los pares con los que Alicia consigue establecer con éxito una conexión TCP “pares vecinos”. (En la Figura 2.24 vemos que Alicia solo tiene tres pares vecinos. Normalmente, podría tener muchos más.) A medida que pasa el tiempo, algunos de estos pares pueden abandonar la conexión y otros (aparte de los 50 iniciales) pueden intentar establecer conexiones TCP con Alicia. Por tanto, los pares vecinos de un determinado par irán variando con el tiempo.

En un determinado instante de tiempo, cada par tendrá un subconjunto de fragmentos del archivo, disponiendo los distintos pares de subconjuntos diferentes. Periódicamente, Alicia preguntará a cada uno de sus vecinos (a través de las conexiones TCP) por la lista de fragmentos de la que disponen.



**Figura 2.24** ♦ Distribución de archivos con BitTorrent.

Si Alicia tiene  $L$  vecinos diferentes obtendrá  $L$  listas de fragmentos. Con esta información, Alicia solicitará (a través de las conexiones TCP) los fragmentos que ella no tiene.

De esta manera, en un determinado instante, Alicia tendrá un subconjunto de fragmentos y sabrá qué fragmentos tienen sus vecinos. Con esta información, Alicia tendrá que tomar dos importantes decisiones. En primer lugar, qué fragmentos debe solicitar primero a sus vecinos. Y en segundo lugar, a cuáles de sus vecinos debe enviar ella los fragmentos solicitados. Para decidir qué fragmentos solicitar, Alicia utiliza una técnica conocida como **primero el menos común**. La idea es determinar, de entre los fragmentos que ella no tiene, los fragmentos menos comunes entre sus vecinos (es decir, los fragmentos de los que existe el menor número de copias repetidas repartidas entre los vecinos) y solicitar entonces en primer lugar esos fragmentos menos comunes. De esta manera, dichos fragmentos se redistribuirán más rápidamente, consiguiendo que el número de copias de cada fragmento sea aproximadamente igual dentro del torrente.

Para determinar a qué solicitudes debe ella responder, BitTorrent utiliza un algoritmo de intercambio inteligente. La idea básica es que Alicia dé prioridad a los vecinos que actualmente están suministrando sus datos *a mayor velocidad*. Específicamente, para cada uno de los vecinos, Alicia mide de forma continua la velocidad a la que recibe bits y determina cuáles son los cuatro pares que le envían bits a mayor velocidad. Entonces, ella, de forma recíproca, envía fragmentos a esos mismos cuatro pares. Cada 10 segundos, vuelve a calcular las velocidades y, posiblemente, tendrá que modificar el conjunto formado por los cuatro pares. En la jerga de BitTorrent, se dice que estos cuatro pares están **no filtrados (unchoked)**. Además, y lo que es más importante, cada 30 segundos Alicia elige de forma aleatoria un vecino adicional y le envía fragmentos. Supongamos que el par elegido aleatoriamente es el de Benito. En la jerga de BitTorrent, se dice que Benito está **no filtrado de forma optimista**. Dado que Alicia está enviando datos a Benito, ella puede convertirse en uno de los cuatro suministradores principales de Benito, en cuyo caso Benito comenzaría a enviar datos a Alicia. Si la velocidad a la que Benito envía datos a Alicia es lo suficientemente alta, Benito podría entonces, a su vez, convertirse en uno de los cuatro suministradores principales de Alicia. En otras palabras, cada 30 segundos Alicia elegirá aleatoriamente un nuevo socio de intercambio e iniciará

las transacciones con él. Si los dos pares están satisfechos con el intercambio, se incluirán en sus respectivas listas de los cuatro principales y continuarán realizando intercambios hasta que uno de los pares encuentre un socio mejor. El efecto es que los pares capaces de suministrar datos a velocidades compatibles tienden a emparejarse. La selección aleatoria de vecinos también permite a los nuevos pares obtener fragmentos, con el fin de tener algo que intercambiar. Todos los demás pares vecinos excepto estos cinco (los cuatro pares “principales” más el par de prueba) están “filtrados”, es decir, no reciben fragmentos de Alicia. BitTorrent dispone de una serie de interesantes mecanismos que no vamos a ver aquí, entre los que se incluyen la gestión de piezas (minifragmentos), el procesamiento en cadena, la selección aleatoria del primer fragmento, el modo *endgame* y el *anti-snubbing* [Cohen 2003].

El mecanismo de incentivos para intercambio que acabamos de describir a menudo se denomina *tit-for-tat* (toma y daca, una estrategia de la teoría de juegos) [Cohen 2003]. Se ha demostrado que este esquema de incentivos puede soslayarse maliciosamente [Liogkas 2006; Locher 2006; Piatek 2007]. No obstante, el ecosistema BitTorrent ha tenido un éxito bárbaro, con millones de pares simultáneos compartiendo activamente archivos en cientos de miles de torrentes. Si BitTorrent se hubiera diseñado sin la estrategia *tit-for-tat* (o una variante), y aunque todo el resto de características fueran las mismas, es posible que BitTorrent no existiera actualmente, ya que la mayor parte de los usuarios hubieran pretendido aprovecharse de los demás [Saroiu 2002].

Vamos a terminar esta exposición sobre P2P mencionando brevemente otra aplicación de P2P, las tablas hash distribuidas (DHT, *Distributed Hast Table*). Una tabla hash distribuida es una base de datos simple, diatribuyéndose los registros de la base de datos a través de los pares de un sistema P2P. Las DHT han sido ampliamente implementadas (por ejemplo, en BitTorrent) y han sido objeto de exhaustivas investigaciones. Proporcionamos una introducción a las mismas en una nota de vídeo disponible en el sitio web de acompañamiento.



Nota de video

Introducción a las tablas hash distribuidas

## 2.6 Flujos de vídeo y redes de distribución de contenido

La distribución de flujos pregrabados de vídeo representa ya la mayor parte del tráfico en los ISP residenciales de Norteamérica. En concreto, sólo los servicios de Netflix y YouTube consumieron un asombroso 37% y 16%, respectivamente, del tráfico de los ISP residenciales en 2015 [Sandvine 2015]. En esta sección proporcionaremos un resumen del modo en que se implementan en la Internet de hoy en día los servicios más populares de flujos de vídeo. Como veremos, se implementan utilizando protocolos de nivel de aplicación y servidores que funcionan, en cierto modo, como una caché. En el Capítulo 9, dedicado a las redes multimedia, examinaremos más en detalle el tema del vídeo por Internet, así como otros servicios Internet de carácter multimedia.

### 2.6.1 Vídeo por Internet

En las aplicaciones con flujos de vídeo almacenado, el medio subyacente es un vídeo pregrabado, como por ejemplo una película, un programa de televisión, un evento deportivo en diferido o un vídeo pregrabado generado por el usuario (como los que se pueden ver comúnmente en YouTube). Estos vídeos pregrabados se almacenan en servidores, y los usuarios envían solicitudes a los servidores para ver los vídeos *a la carta*. Muchas empresas de Internet proporcionan hoy en día flujos de vídeo, como por ejemplo Netflix, YouTube (Google), Amazon y Youku.

Pero antes de entrar a analizar los flujos de vídeo, conviene primero echar un rápido vistazo al propio medio subyacente: el vídeo. Un vídeo es una secuencia de imágenes, que normalmente se visualizan a velocidad constante, por ejemplo de 24 o 30 fotogramas por segundo. Una imagen codificada digitalmente y no comprimida, consta de una matriz de píxeles, codificándose cada píxel mediante una serie de bits que representan la luminancia y el color. Una característica importante del vídeo es que se puede comprimir, sacrificando algo de calidad de imagen a cambio de reducir la tasa

de transmisión de bits. Los algoritmos comerciales de compresión existentes hoy en día permiten comprimir un vídeo prácticamente a cualquier tasa de bits que se desee. Por supuesto, cuanto mayor sea la tasa de bits, mayor será la calidad de la imagen y mejor será la experiencia global del usuario, en lo que a visualización se refiere.

Desde la perspectiva de las redes, quizás la característica más destacable del vídeo sea su alta tasa de bits. El vídeo comprimido para Internet suele requerir entre 100 kbps (para vídeo de baja calidad) y más de 3 Mbps (para flujos de películas de alta resolución); los flujos de vídeo en formato 4K prevén una tasa de bits superior a 10 Mbps. Esto se traduce en una enorme cantidad de tráfico y de almacenamiento, particularmente para vídeo de alta gama. Por ejemplo, un único vídeo a 2 Mbps con una duración de 67 minutos consumirá 1 gigabyte de almacenamiento y de tráfico. La medida de rendimiento más importante para los flujos de vídeo es, con mucho, la tasa de transferencia media de extremo a extremo. Para poder garantizar una reproducción continua, la red debe proporcionar a la aplicación de flujos de vídeo una tasa de transferencia media que sea igual o superior a la tasa de bits del vídeo comprimido.

También podemos usar la compresión para crear múltiples versiones del mismo vídeo, cada una con un nivel diferente de calidad. Por ejemplo, podemos usar la compresión para crear tres versiones del mismo vídeo, con tasas de 300 kbps, 1 Mbps y 3 Mbps. Los usuarios pueden entonces decidir qué versión quieren ver, en función de su ancho de banda actualmente disponible. Los usuarios con conexiones Internet de alta velocidad podrían seleccionar la versión a 3 Mbps, mientras que los usuarios que vayan a ver el vídeo a través de un teléfono inteligente con tecnología 3G podrían seleccionar la versión a 300 kbps.

### 2.6.2 Flujos de vídeo HTTP y tecnología DASH

En los flujos multimedia HTTP, el vídeo simplemente se almacena en un servidor HTTP como un archivo normal, con un URL específico. Cuando un usuario quiere ver el vídeo, el cliente establece una conexión TCP con el servidor y emite una solicitud GET HTTP para dicho URL. El servidor envía entonces el archivo de vídeo dentro de un mensaje de respuesta HTTP, con la máxima velocidad que permitan los protocolos de red subyacentes y las condiciones existentes de tráfico. En el lado del cliente, los bytes se acumulan en un buffer de la aplicación cliente. En cuanto el número de bytes en el buffer sobrepasa un umbral predeterminado, la aplicación cliente comienza la reproducción: para ser concretos, la aplicación de flujos de vídeo extrae periódicamente fotogramas de vídeo del buffer de la aplicación cliente, descomprime los fotogramas y los muestra en la pantalla del usuario. De ese modo, la aplicación de flujos de vídeo muestra el vídeo al mismo tiempo que recibe y almacena en el buffer otros fotogramas, correspondientes a secuencias posteriores del vídeo.

Aunque los flujos HTTP, tal como se describen en el párrafo anterior, se han implantado ampliamente en la práctica (por ejemplo por parte de YouTube, desde su nacimiento), presentan una carencia fundamental: todos los clientes reciben la misma versión codificada del vídeo, a pesar de las grandes variaciones existentes en cuanto al ancho de banda disponible para cada cliente, e incluso en cuanto al ancho de banda disponible para un cliente concreto a lo largo del tiempo. Esto condujo al desarrollo de un nuevo tipo de flujos de vídeo basados en HTTP, una tecnología a la que se suele denominar **DASH, (Dynamic Adaptive Streaming over HTTP, Flujos dinámicos adaptativos sobre HTTP)**. En DASH, el vídeo se codifica en varias versiones diferentes, teniendo cada versión una tasa de bits distinta y, por tanto, un nivel de calidad diferente. El cliente solicita dinámicamente segmentos de vídeo de unos pocos segundos de duración. Cuando el ancho de banda disponible es grande, el cliente selecciona de forma natural los segmentos de una versión de alta tasa de bits; y cuando el ancho de banda disponible es pequeño, selecciona de forma natural los segmentos de una versión con menor tasa de bits. El cliente selecciona los segmentos de uno en uno mediante mensajes de solicitud GET HTTP [Akhshabi 2011].

DASH permite que los clientes con diferentes tasas de acceso a Internet reciban flujos de vídeo con tasas de codificación diferentes. Los clientes con conexiones 3G a baja velocidad pueden recibir una versión con baja tasa de bits (y baja calidad), mientras que los clientes con conexiones de fibra

pueden recibir una versión de alta calidad. DASH también permite a un cliente adaptarse al ancho de banda disponible, si el ancho de banda extremo a extremo varía durante la sesión. Esta característica es particularmente importante para los usuarios móviles, que suelen experimentar fluctuaciones del ancho de banda disponible a medida que se desplazan con respecto a las estaciones base.

Con DASH, cada versión del vídeo se almacena en un servidor HTTP, teniendo cada versión un URL distinto. El servidor HTTP dispone también de un **archivo de manifiesto**, que indica el URL de cada versión, junto con su correspondiente tasa de bits. El cliente solicita primero el archivo de manifiesto y determina cuáles son las diferentes versiones disponibles. Después solicita los segmentos de uno en uno, especificando un URL y un rango de bytes mediante un mensaje de solicitud GET HTTP para cada segmento. Mientras está descargando los segmentos, el cliente también mide el ancho de banda de recepción y ejecuta un algoritmo de determinación de la tasa de bits, para seleccionar el segmento que debe solicitar a continuación. Naturalmente, si el cliente tiene una gran cantidad de vídeo almacenado en el buffer y si el ancho de banda de recepción medido es grande, seleccionará un segmento correspondiente a una versión con alta tasa de bits. Igualmente, si tiene poca cantidad de vídeo almacenado en el buffer y el ancho de banda de recepción medido es pequeño, seleccionará un segmento correspondiente a una versión con baja tasa de bits. DASH permite, de ese modo, que el cliente cambie libremente entre distintos niveles de calidad.

### 2.6.3 Redes de distribución de contenido

Actualmente, muchas empresas de vídeo a través de Internet están distribuyendo a la carta flujos de vídeo de múltiples Mbps a millones de usuarios diariamente. YouTube, por ejemplo, con una librería de cientos de millones de vídeos, distribuye a diario centenares de millones de flujos de vídeo a usuarios repartidos por todo el mundo. Enviar todo este tráfico a ubicaciones de todo el mundo, al mismo tiempo que se proporciona una reproducción continua y una alta interactividad, constituye claramente un auténtico desafío.

Para una empresa de vídeo a través de Internet, quizás la solución más directa para proporcionar un servicio de flujos de vídeo sea construir un único centro de datos masivo, almacenar todos los vídeos en ese centro de datos y enviar los flujos de vídeo directamente desde el centro de datos a clientes repartidos por todo el mundo. Pero esta solución tiene tres problemas principales. En primer lugar, si el cliente está lejos del centro de datos, los paquetes que viajan desde el servidor al cliente atravesarán muchos enlaces de comunicaciones y probablemente atravesen muchos ISP, estando algunos de los ISP posiblemente ubicados en continentes distintos. Si uno de esos enlaces proporciona una tasa de transferencia inferior a la velocidad a la que se consume el vídeo, la tasa de transferencia extremo a extremo también será inferior a la tasa de consumo, lo que provocará molestas congelaciones de la imagen de cara al usuario. (Recuerde del Capítulo 1 que la tasa de transferencia extremo a extremo de un flujo de datos está determinada por la tasa de transferencia del enlace que actúa como cuello de botella.) La probabilidad de que esto suceda se incrementa a medida que aumenta el número de enlaces que componen la ruta extremo a extremo. Una segunda desventaja es que un vídeo muy popular será probablemente enviado muchas veces a través de los mismos enlaces de comunicaciones. Esto no solo hace que se desperdicie ancho de banda de la red, sino que la propia empresa de vídeo a través de Internet estará pagando a su ISP proveedor (conectado al centro de datos) por enviar los *mismos* bytes una y otra vez a Internet. Un tercer problema de esta solución es que un único centro de datos representa un punto único de fallo, si se caen el centro de datos o sus enlaces de conexión con Internet, la empresa no podrá distribuir *ningún* flujo de vídeo.

Para poder afrontar el desafío de distribuir cantidades masivas de datos de vídeo a usuarios dispersos por todo el mundo, casi todas las principales empresas de flujos de vídeo utilizan **redes de distribución de contenido (CDN, Content Distribution Network)**. Una CDN gestiona servidores situados en múltiples ubicaciones geográficamente distribuidas, almacena copias de los vídeos (y de otros tipos de contenido web, como documentos, imágenes y audio) en sus servidores y trata de dirigir cada solicitud de usuario a una ubicación de la CDN que proporcione la mejor experiencia de usuario posible. La CDN puede ser una **CDN privada**, es decir, propiedad del

propio proveedor de contenido; por ejemplo, la CDN de Google distribuye vídeos de YouTube y otros tipos de contenido. Alternativamente, la CDN puede ser una **CDN comercial** que distribuya contenido por cuenta de múltiples proveedores de contenido; Akamai, Limelight y Level-3, por ejemplo, operan redes CDN comerciales. Un resumen muy legible de las redes CDN modernas es [Leighton 2009; Nygren 2010].

Las redes CDN adoptan normalmente una de dos posibles filosofías de colocación de los servidores [Huang 2008]:

- **Introducción profunda.** Una de las filosofías, de la que Akamai fue pionera, consiste en *introducirse en profundidad* en las redes de acceso de los proveedores de servicios Internet (ISP), implantando clústeres de servidores en proveedores ISP de acceso por todo el mundo. (Las redes de acceso se describen en la Sección 1.3.) Akamai ha adoptado esta solución con clústeres de servidores en aproximadamente 1.700 ubicaciones. El objetivo es acercarse a los usuarios finales, mejorando así el retardo percibido por el usuario y la tasa de transferencia por el procedimiento de reducir el número de enlaces y de routers existentes entre el usuario final y el servidor CDN del que recibe el contenido. Debido a este diseño altamente distribuido, la tarea de mantener y gestionar los clústeres se convierte en todo un desafío.
- **Atraer a los ISP.** Una segunda filosofía de diseño, adoptada por Limelight y muchos otras empresas de redes CDN, consiste en *atraer a los ISP* construyendo grandes clústeres en un número más pequeño (por ejemplo, unas decenas) de lugares. En lugar de introducirse en los ISP de acceso, estas CDN suelen colocar sus clústeres en puntos de intercambio Internet (IXP, *Internet Exchange Point*, véase la Sección 1.3). Comparada con la filosofía de diseño basada en la introducción profunda, el diseño basado en atraer a los ISP suele tener menores costes de mantenimiento y gestión, posiblemente a cambio de un mayor retardo y una menor tasa de transferencia para los usuarios finales.

Una vez implantados los clústeres, la CDN replica el contenido entre todos ellos. La red CDN puede no siempre almacenar una copia de cada vídeo en cada clúster, ya que algunos vídeos solo se visualizan en raras ocasiones o solo son populares en ciertos países. De hecho, muchas CDN no copian activamente los vídeos en sus clústeres, sino que usan una estrategia simple: si un cliente solicita un vídeo de un clúster que no lo tiene almacenado, el clúster extrae el vídeo (de un repositorio central o de otro clúster) y almacena una copia localmente, al mismo tiempo que envía el flujo de vídeo al cliente. De forma similar a lo que ocurre con las cachés web (véase la Sección 2.2.5), cuando el espacio de almacenamiento del clúster se llena, el clúster elimina los vídeos que no son solicitados frecuentemente.

### Funcionamiento de una red CDN

Habiendo identificado las dos soluciones principales de implantación de una CDN, profundicemos en el modo en que una de estas redes opera. Cuando se indica al navegador del host de un usuario que extraiga un vídeo concreto (identificado mediante un URL), la CDN debe interceptar la solicitud para (1) determinar un clúster de servidores de la CDN que resulte adecuado para ese cliente en ese preciso instante y (2) redirigir la solicitud del cliente a un servidor situado en dicho clúster. En breve veremos cómo puede la CDN determinar un clúster adecuado. Pero antes, examinemos la mecánica del proceso de interceptación y redirección de una solicitud.

La mayoría de las CDN aprovechan DNS para interceptar y redirigir las solicitudes; un análisis interesante de esa utilización de DNS es [Vixie 2009]. Consideremos un ejemplo simple para ilustrar el modo en que DNS suele participar. Suponga que un proveedor de contenido, NetCinema, utiliza a una empresa proveedora de servicios CDN, KingCDN, para distribuir sus vídeos a sus clientes. En las páginas web de NetCinema, cada uno de los vídeos tiene asignado un URL que incluye la cadena “video” y un identificador único del propio vídeo; por ejemplo, a *Transformers 7* se le podría asignar <http://video.netcinema.com/6Y7B23V>. Entonces se sucederán seis pasos, como se muestra en la Figura 2.25:

## CASO DE ESTUDIO

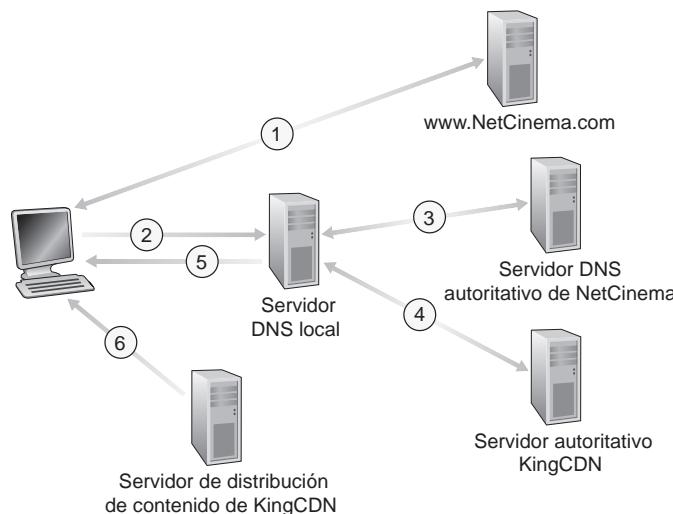
### INFRAESTRUCTURA DE RED DE GOOGLE

Para dar soporte a su amplia variedad de servicios en la nube —incluyendo las búsquedas, Gmail, calendarios, vídeos YouTube, mapas, documentos y redes sociales—, Google ha implantado una amplia red privada y una infraestructura CDN. La infraestructura CDN de Google tiene tres niveles de clústeres de servidores:

- Catorce “mega-centros de datos” (ocho en Norteamérica, cuatro en Europa y dos en Asia [Google Locations 2016]), teniendo cada centro de datos del orden de 100.000 servidores. Estos mega-centros de datos se encargan de servir contenido dinámico (y a menudo personalizado), incluyendo resultados de búsqueda y mensajes Gmail.
- Unos 50 clústeres en IXP dispersos por todo el mundo, consistiendo cada clúster en unos 100-500 servidores [Adhikari 2011a]. Estos clústeres son responsables de servir contenido estático, incluyendo vídeos de YouTube [Adhikari 2011a].
- Varios cientos de clústeres de “introducción profunda”, ubicados dentro de proveedores ISP de acceso. En este caso, los clústeres suelen estar compuestos por decenas de servidores, situados en un mismo bastidor. Estos servidores de introducción profunda se encargan de la división TCP (véase la Sección 3.7) y de servir contenido estático [Chen 2011], incluyendo las partes estáticas de las páginas web donde se insertan los resultados de búsqueda.

Todos estos centros de datos y clústeres de servidores están conectados en red mediante la propia red privada de Google. Cuando un usuario hace una búsqueda, a menudo la búsqueda se envía primero a través del ISP local hasta una caché cercana de introducción profunda, de donde se extrae el contenido estático; mientras se proporciona el contenido estático al cliente, la caché cercana reenvía también la consulta a través de la red privada de Google hasta uno de los mega-centros de datos, de donde se extraen los resultados de búsqueda personalizados. Para un vídeo de YouTube, el propio vídeo puede provenir de una de las cachés en los IXP, mientras que parte de la página web que rodea al vídeo puede provenir de la caché cercana de introducción profunda y los anuncios que rodean al vídeo vienen de los centros de datos. Resumiendo: salvo por lo que se refiere al ISP local, los servicios en la nube de Google son proporcionados, en buena medida, por una infraestructura de red que es independiente de la Internet pública.

1. El usuario visita la página web en NetCinema.
2. Cuando el usuario hace clic sobre el vínculo <http://video.netcinema.com/6Y7B23V>, el host del usuario envía una solicitud DNS preguntando por video.netcinema.com.
3. El servidor DNS local del usuario (al que llamaremos LDNS) retransmite la solicitud DNS a un servidor DNS autoritativo de NetCinema, que observa la cadena “video” en el nombre de host video.netcinema.com. Para “transferir” la consulta DNS a KingCDN, lo que hace el servidor DNS autoritativo de NetCinema es, en vez de devolver una dirección IP, enviar al LDNS un nombre de host perteneciente al dominio de KingCDN, como por ejemplo a1105. kingcdn.com.
4. A partir de ese punto, la consulta DNS entra en la infraestructura DNS privada de KingCDN. El LDNS del usuario envía entonces una segunda consulta, preguntando ahora por a1105. kingcdn.com, y el sistema DNS de KingCDN termina por devolver al LDNS las direcciones IP de un servidor de contenido de KingCDN. Es por tanto aquí, dentro del sistema DNS de KingCDN, donde se especifica el servidor CDN desde el cual recibirá el cliente su contenido.
5. El LDNS reenvía al host del usuario la dirección IP del nodo CDN encargado de servir el contenido.



**Figura 2.25** ◆ DNS redirige una solicitud de usuario hacia un servidor CDN.

6. Una vez que el cliente recibe la dirección IP de un servidor de contenido de KingCDN, establece una conexión TCP directa con el servidor situado en dicha dirección IP y transmite una solicitud GET HTTP para el vídeo deseado. Si se utiliza DASH, el servidor enviará primero al cliente un archivo de manifiesto con una lista de URL, uno para cada versión del vídeo, y el cliente seleccionará dinámicamente segmentos de las distintas versiones.

### Estrategias de selección de clústeres

Uno de los fundamentos de cualquier implantación de una red CDN es la **estrategia de selección de clústeres**, es decir, el mecanismo para dirigir a los clientes dinámicamente hacia un clúster de servidores o un centro de datos pertenecientes a la CDN. Como acabamos de ver, la CDN determina la dirección IP del servidor LDNS del cliente a partir de la búsqueda DNS realizada por el cliente. Después de determinar esta dirección IP, la red CDN necesita seleccionar un clúster apropiado, dependiendo de dicha dirección. Las redes CDN emplean, generalmente, estrategias propietarias de selección de clústeres. Vamos a ver brevemente algunas soluciones, cada una de las cuales tiene sus ventajas y sus desventajas.

Una estrategia sencilla consiste en asignar al cliente al clúster **geográficamente más próximo**. Usando bases de datos comerciales de geolocalización (como Quova [Quova 2016] y MaxMind [MaxMind 2016]), la dirección IP de cada LDNS se hace corresponder con una ubicación geográfica. Cuando se recibe una solicitud DNS de un LDNS concreto, la CDN selecciona el clúster geográficamente más próximo, es decir, el clúster situado a menos kilómetros “a vuelo de pájaro” del LDNS. Una solución de este estilo puede funcionar razonablemente bien para una gran parte de los clientes [Agarwal 2009]. Sin embargo, para algunos clientes la solución puede proporcionar un mal rendimiento, porque el clúster geográficamente más cercano no es necesariamente el clúster más próximo en términos de la longitud o el número de saltos de la ruta de red. Además, un problema inherente a todas las soluciones basadas en DNS es que algunos usuarios finales están configurados para usar servidores LDNS remotos [Shaikh 2001; Mao 2002], en cuyo caso la ubicación del LDNS puede estar lejos de la del cliente. Además, esta estrategia tan simple no tiene en cuenta la variación a lo largo del tiempo del retardo y del ancho de banda disponible de las rutas en Internet, asignando siempre el mismo clúster a cada cliente concreto.

Para determinar el mejor clúster para un cliente basándose en las condiciones *actuales* de tráfico, las redes CDN pueden, alternativamente, realizar periódicamente **medidas en tiempo real** del retardo y del comportamiento de pérdidas entre sus clústeres y los clientes. Por ejemplo, una CDN puede hacer que todos sus clústeres envíen periódicamente mensajes de sondeo (por ejemplo, mensajes ping o consultas DNS) a todos los LDNS de todo el mundo. Una desventaja de esta técnica es que muchos LDNS están configurados para no responder a tales mensajes de sondeo.

### 2.6.4 Casos de estudio: Netflix, YouTube y Kankan

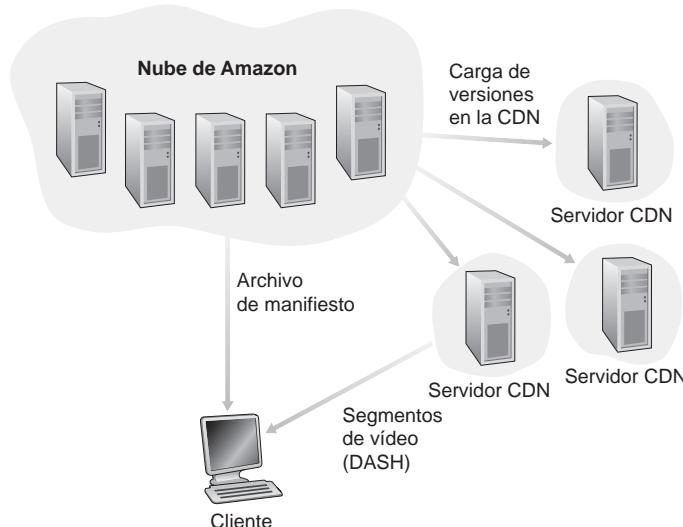
Terminamos nuestra exposición sobre los flujos de vídeo almacenados echando un vistazo a tres implantaciones a gran escala de enorme éxito: Netflix, YouTube y Kankan. Veremos que cada uno de estos sistemas adopta una solución muy diferente, aunque todos ellos emplean muchos de los principios subyacentes expuestos en esta sección.

#### Netflix

Netflix, que generó el 37% del tráfico de bajada en los ISP residenciales de Norteamérica en 2015, se ha convertido en el principal proveedor de servicios para películas y series de TV en línea en los Estados Unidos [Sandvine 2015]. Como vamos a ver, la distribución de vídeo de Netflix tiene dos componentes principales: la nube de Amazon y su propia infraestructura CDN privada.

Netflix dispone de un sitio web que se encarga de gestionar numerosas funciones, incluyendo los registros e inicio de sesión de los usuarios, la facturación, el catálogo de películas que puede hojearse o en el que se pueden realizar búsquedas y un sistema de recomendación de películas. Como se muestra en la Figura 2.26, este sitio web (y sus bases de datos *back-end* asociadas) se ejecuta enteramente en servidores de Amazon, dentro de la nube de Amazon. Además, la nube de Amazon se encarga de las siguientes funciones críticas:

- **Ingesta de contenidos.** Antes de que Netflix pueda distribuir una película a sus usuarios, debe primero realizar la ingesta y procesar la película. Netflix recibe versiones maestras de estudio de las películas y las carga en hosts situados en la nube de Amazon.



**Figura 2.26** ♦ Plataforma de flujos de vídeo de Netflix.

- **Procesamiento del contenido.** Las máquinas de la nube de Amazon generan muchos formatos distintos para cada película, adecuados para una amplia variedad de clientes de reproducción de vídeo que se ejecutan en computadoras de sobremesa, teléfonos inteligentes y consolas de juegos conectadas a televisiones. Para cada uno de estos formatos se crea una versión diferente, con múltiples tasas de bits, lo que permite un envío adaptativo de los flujos multimedia a través de HTTP, usando DASH.
- **Carga de las versiones en su CDN.** Una vez generadas todas las versiones de una película, los hosts de la nube de Amazon cargan esas versiones en la CDN de Netflix.

Cuando Netflix inauguró su servicio de distribución de flujos de vídeo en 2007, empleaba tres empresas proveedoras de servicios de red CDN para distribuir su contenido de vídeo. Desde entonces, Netflix ha creado su propia CDN privada, desde la cual distribuye ahora todos sus flujos de vídeo. (Sin embargo, Netflix sigue usando Akamai para distribuir sus páginas web.) Para crear su propia CDN, Netflix ha instalado bastidores de servidores tanto en IXP como dentro de los propios ISP residenciales. Netflix dispone actualmente de bastidores de servidores en más de 50 ubicaciones de IXP; en [Netflix Open Connect 2016] puede ver una lista actualizada de IXP que albergan bastidores de Netflix. También hay centenares de ubicaciones de ISP que albergan bastidores de Netflix; véase también [Netflix Open Connect 2016], donde Netflix proporciona a los potenciales ISP asociados instrucciones sobre cómo instalar un bastidor Netflix (gratuito) para sus redes. Cada servidor del bastidor dispone de varios puertos Ethernet a 10 Gbps y de más de 100 terabytes de almacenamiento. El número de servidores de un bastidor es variable: las instalaciones de los IXP suelen tener decenas de servidores y contienen toda la biblioteca de flujos de vídeo de Netflix, incluyendo las múltiples versiones de los vídeos que hacen falta para soportar DASH; los ISP locales pueden disponer de un único servidor y almacenar solo los vídeos más populares. Netflix no utiliza un sistema de caché que se rellena bajo demanda (*pull-caching*) para cargar el contenido en sus servidores CDN ubicados en los IXP e ISP. En lugar de ello, Netflix realiza la distribución cargando activamente los vídeos en sus servidores CDN durante las horas menor tráfico. Para aquellas ubicaciones que no pueden almacenar la biblioteca completa, Netflix carga solo los vídeos más populares, que se determinan diariamente. El diseño de CDN de Netflix se describe con un cierto grado de detalle en los vídeos de Youtube [Netflix Video 1] y [Netflix Video 2].

Habiendo descrito los componentes de la arquitectura Netflix, examinemos más detalladamente la interacción entre el cliente y los distintos servidores implicados en la distribución de las películas. Como hemos dicho anteriormente, las páginas web a través de las que se explora la videoteca de Netflix se sirven desde servidores situados en la nube de Amazon. Cuando un usuario selecciona una película para reproducirla, el software de Netflix, ejecutándose en la nube de Amazon, determina primero cuáles de sus servidores CDN disponen de una copia de la película. A continuación, el software determina cuál de entre los servidores que disponen de la película es el “mejor” para esa solicitud del cliente. Si el cliente está utilizando un ISP residencial que tiene instalado un bastidor de servidores de la CDN de Netflix, y si ese bastidor dispone de una copia de la película solicitada, entonces suele seleccionarse un servidor de ese bastidor. Si no, lo que se suele seleccionar es un servidor de algún IXP cercano.

Una vez que Netflix ha determinado el servidor CDN que tiene que distribuir el contenido, envía al cliente la dirección IP de ese servidor concreto, junto con un archivo de manifiesto, que contiene los URL de las diferentes versiones de la película solicitada. A continuación, el cliente y ese servidor CDN interactúan directamente, usando una versión propietaria de DASH. Específicamente, como se describe en la Sección 2.6.2, el cliente usa la cabecera de rango de bytes de los mensajes de solicitud GET HTTP para solicitar segmentos de las diferentes versiones de la película. Netflix usa segmentos de una duración aproximada de cuatro segundos [Adhikari 2012]. Mientras se descargan los segmentos, el cliente mide la tasa de transferencia de recepción y ejecuta un algoritmo de determinación de la velocidad para identificar la calidad del segmento que debe solicitar a continuación.

Netflix utiliza muchos de los principios básicos que hemos expuesto anteriormente en esta sección, incluyendo los flujos adaptativos y la distribución a través de una CDN. Sin embargo, como Netflix emplea su propia CDN privada, que solo distribuye vídeo (y no páginas web), Netflix ha sido capaz de simplificar y adaptar su diseño de CDN. En particular, Netflix no necesita usar la redirección DNS, explicada en la Sección 2.6.3, para conectar un cliente concreto con un servidor CDN; en lugar de ello, el software de Netflix (que se ejecuta en la nube de Amazon) instruye directamente al cliente para que utilice un servidor CDN concreto. Además, la CDN de Netflix carga el contenido de la caché en sus servidores en momentos planificados (*push-caching*), durante las horas de menor tráfico, en lugar de cargarlo dinámicamente a medida que se producen fallos de localización en caché.

## YouTube

Con 300 horas de vídeo cargadas en YouTube cada minuto y varios miles de millones de reproducciones diarias de vídeo [YouTube 2016], YouTube es, sin lugar a dudas, el mayor sitio de compartición de vídeos del mundo. YouTube comenzó a prestar servicio en abril de 2005 y fue adquirido por Google en noviembre de 2006. Aunque el diseño y los protocolos de Google/YouTube son propietarios, podemos hacernos una idea básica de cómo opera YouTube gracias a diversos trabajos de medida independientes [Zink 2009; Torres 2011; Adhikari 2011a]. Al igual que Netflix, YouTube hace un amplio uso de la tecnología CDN para distribuir sus vídeos [Torres 2011]. De forma similar a Netflix, Google utiliza su propia CDN privada para distribuir los vídeos de YouTube y ha instalado clústeres de servidores en muchos cientos de ubicaciones IXP e ISP distintas. Google distribuye los vídeos de YouTube desde estas ubicaciones y directamente desde sus inmensos centros de datos [Adhikari 2011a]. A diferencia de Netflix, sin embargo, Google emplea *pull-caching* y un mecanismo de redirección DNS, como se describe en la Sección 2.6.3. La mayoría de las veces, la estrategia de selección de clústeres de Google dirige al cliente hacia el clúster que tenga un menor RTT entre el clúster y el cliente; sin embargo, para equilibrar la carga entre los clústeres, en ocasiones se dirige al cliente (a través de DNS) a un clúster más distante [Torres 2011].

YouTube emplea flujos HTTP, ofreciendo a menudo un pequeño número de versiones distintas de cada vídeo, cada una con diferente tasa de bits y, correspondientemente, un diferente nivel de calidad. YouTube no utiliza flujos adaptativos (como DASH), sino que exige al cliente que seleccione una versión de forma manual. Para ahorrar ancho de banda y recursos de servidor, que se desperdiciarían en caso de que el usuario efectúe un reposicionamiento o termine la reproducción anticipadamente, YouTube emplea la solicitud de rango de bytes de HTTP para limitar el flujo de datos transmitidos, después de precargar una cierta cantidad predeterminada de vídeo.

Cada día se cargan en YouTube varios millones de vídeos. No solo se distribuyen a través de HTTP los flujos de vídeo de YouTube de los servidores a los clientes, sino que los usuarios que cargan vídeos en YouTube desde el cliente hacia el servidor también los cargan a través de HTTP. YouTube procesa cada vídeo que recibe, convirtiéndolo a formato de vídeo de YouTube y creando múltiples versiones con diferentes tasas de bits. Este procesamiento se realiza enteramente en los centros de datos de Google (véase el caso de estudio sobre la infraestructura de red de Google en la Sección 2.6.3).

## Kankan

Acabamos de ver cómo una serie de servidores dedicados, operados por redes CDN privadas, se encargan de distribuir a los clientes los vídeos de Netflix y YouTube. Ambas empresas tienen que pagar no solo por el hardware del servidor, sino también por el ancho de banda que los servidores usan para distribuir los vídeos. Dada la escala de estos servicios y la cantidad de ancho de banda que consumen, ese tipo de implantación de una CDN puede ser costoso.

Concluiremos esta sección describiendo un enfoque completamente distinto para la provisión a gran escala de vídeos a la carta a través de Internet - un enfoque que permite al proveedor del servicio reducir significativamente sus costes de infraestructura y de ancho de banda. Como el lector estará suponiendo, este enfoque utiliza distribución P2P en lugar de (o además de) distribución cliente-servidor. Desde 2011, Kankan (cuyo propietario y operador es Xunlei) ha estado implantando con gran éxito su sistema P2P de distribución de vídeo, que cuenta con decenas de millones de usuarios cada mes [Zhang 2015].

A alto nivel, los flujos de vídeo P2P son muy similares a la descarga de archivos con BitTorrent. Cuando uno de los participantes quiere ver un vídeo, contacta con un *tracker* para descubrir otros homólogos en el sistema que dispongan de una copia de ese vídeo. El homólogo solicitante pide entonces segmentos de vídeo en paralelo a todos los homólogos que dispongan de él. Sin embargo, a diferencia de lo que sucede con la descarga en BitTorrent, las solicitudes se realizan preferentemente para segmentos que haya que reproducir en un futuro próximo, con el fin de garantizar una reproducción continua [Dhungel 2012].

Recientemente, Kankan ha efectuado la migración a un sistema de flujos de vídeo híbrido CDN-P2P [Zhang 2015]. Específicamente, Kankan tiene ahora implantados unos pocos cientos de servidores en China y carga de forma activa contenido de vídeo en esos servidores. Esta CDN de Kankan juega un papel principal durante la etapa inicial de transmisión de los flujos de vídeo. En la mayoría de los casos, el cliente solicita el principio del contenido a los servidores de la CDN y en paralelo pide contenido a los homólogos. Cuando el tráfico P2P total es suficiente para la reproducción de vídeo, el cliente deja de descargar de la CDN y descarga sólo de los homólogos. Pero si el tráfico de descarga de flujos de vídeo P2P pasa a ser insuficiente, el cliente restablece las conexiones con la CDN y vuelve al modo de flujos de vídeo híbrido CDN-P2P. De esta manera, Kankan puede garantizar retardos iniciales de arranque cortos, al mismo tiempo que minimiza la utilización de ancho de banda y de una costosa infraestructura de servidores.

## 2.7 Programación de sockets: creación de aplicaciones de red

Ahora que hemos examinado una serie de importantes aplicaciones de red, vamos a ver cómo se escriben en la práctica los programas de aplicaciones de redes. Recuerde de la Sección 2.1 que muchas aplicaciones de red están compuestas por una pareja de programas (un programa cliente y un programa servidor) que residen en dos sistemas terminales distintos. Cuando se ejecutan estos dos programas, se crean un proceso cliente y un proceso servidor, y estos dos procesos se comunican entre sí leyendo y escribiendo en sockets. Cuando se crea una aplicación de red, la tarea principal del desarrollador es escribir el código para los programas cliente y servidor.

Existen dos tipos de aplicaciones de red. Uno de ellos es una implementación de un estándar de protocolo definido en, por ejemplo, un RFC o algún otro documento relativo a estándares. Para este tipo de implementaciones, los programas cliente y servidor deben adaptarse a las reglas dictadas por ese RFC. Por ejemplo, el programa cliente podría ser una implementación del lado del cliente del protocolo HTTP, descrito en la Sección 2.2 y definido explícitamente en el documento RFC 2616; de forma similar, el programa servidor podría ser una implementación del protocolo de servidor HTTP, que también está definido explícitamente en el documento RFC 2616. Si un desarrollador escribe código para el programa cliente y otro desarrollador independiente escribe código para el programa servidor y ambos desarrolladores siguen cuidadosamente las reglas marcadas en el RFC, entonces los dos programas serán capaces de interoperar. Ciertamente, muchas de las aplicaciones de red actuales implican la comunicación entre programas cliente y servidor que han sido creados por desarrolladores independientes (por ejemplo, un navegador Google Chrome comunicándose con un servidor web Apache, o un cliente BitTorrent comunicándose con un tracker BitTorrent).

El otro tipo de aplicación de red son las aplicaciones propietarias. En este caso, el protocolo de la capa de aplicación utilizado por los programas cliente y servidor *no* tiene que cumplir necesariamente ninguna recomendación RFC existente. Un único desarrollador (o un equipo de desarrollo) crea tanto el programa cliente como el programa servidor, y ese desarrollador tiene el control completo sobre aquello que se incluye en el código. Pero como el código no implementa ningún protocolo abierto, otros desarrolladores independientes no podrán desarrollar código que interopere con esa aplicación.

En esta sección vamos a examinar los problemas fundamentales del desarrollo de aplicaciones propietarias cliente-servidor y echaremos un vistazo al código que implementa una aplicación cliente-servidor muy sencilla. Durante la fase de desarrollo, una de las primeras decisiones que el desarrollador debe tomar es si la aplicación se ejecutará sobre TCP o sobre UDP. Recuerde que TCP está orientado a la conexión y proporciona un canal fiable de flujo de bytes a través del cual se transmiten los datos entre los dos sistemas terminales. Por su parte, UDP es un protocolo sin conexión, que envía paquetes de datos independientes de un sistema terminal a otro, sin ningún tipo de garantía acerca de la entrega. Recuerde también que cuando un programa cliente o servidor implementa un protocolo definido por un RFC, debe utilizar el número de puerto bien conocido asociado con el protocolo; asimismo, al desarrollar una aplicación propietaria, el desarrollador debe evitar el uso de dichos números de puerto bien conocidos. (Los números de puerto se han explicado brevemente en la Sección 2.1 y los veremos en detalle en el Capítulo 3).

Vamos a presentar la programación de sockets en UDP y TCP mediante una aplicación UDP simple y una aplicación TCP simple. Mostraremos estas sencillas aplicaciones en Python 3. Podríamos haber escrito el código en Java, C o C++, pero hemos elegido Python fundamentalmente porque Python expone de forma clara los conceptos claves de los sockets. Con Python se usan pocas líneas de código, y cada una de ellas se puede explicar a un programador novato sin dificultad, por lo que no debe preocuparse si no está familiarizado con Python. Podrá seguir fácilmente el código si tiene experiencia en programación en Java, C o C++.

Si está interesado en la programación cliente-servidor con Java, le animamos a que consulte el sitio web de acompañamiento del libro; de hecho, allí podrá encontrar todos los ejemplos de esta sección (y las prácticas de laboratorio asociadas) en Java. Para aquellos lectores que estén interesados en la programación cliente-servidor en C, hay disponibles algunas buenas referencias [Donahoo 2001; Stevens 1997; Frost 1994; Kurose 1996]; los ejemplos en Python que proporcionamos a continuación tienen un estilo y aspecto similares a C.

### 2.7.1 Programación de sockets con UDP

En esta subsección vamos a escribir programas cliente-servidor simples que utilizan UDP. En la siguiente sección, escribiremos programas similares que emplean TCP.

Recuerde de la Sección 2.1 que los procesos que se ejecutan en máquinas diferentes se comunican entre sí enviando mensajes a través de sockets. Dijimos que cada proceso era análogo a una vivienda y que el socket del proceso era análogo a una puerta. La aplicación reside en un lado de la puerta de la vivienda; el protocolo de la capa de transporte reside en el otro lado de la puerta, en el mundo exterior. El desarrollador de la aplicación dispone de control sobre todo lo que está situado en el lado de la capa de aplicación del socket; sin embargo, el control que tiene sobre el lado de la capa de transporte es muy pequeño.

Veamos ahora la interacción existente entre dos procesos que están comunicándose que usan sockets UDP. Si se usa UDP, antes de que un proceso emisor pueda colocar un paquete de datos en la puerta del socket, tiene que asociar en primer lugar una dirección de destino al paquete. Una vez que el paquete atraviesa el socket del emisor, Internet utilizará la dirección de destino para enrutar dicho paquete hacia el socket del proceso receptor, a través de Internet. Cuando el paquete llega al socket de recepción, el proceso receptor recuperará el paquete a través del socket y a continuación inspeccionará el contenido del mismo y tomará las acciones apropiadas.

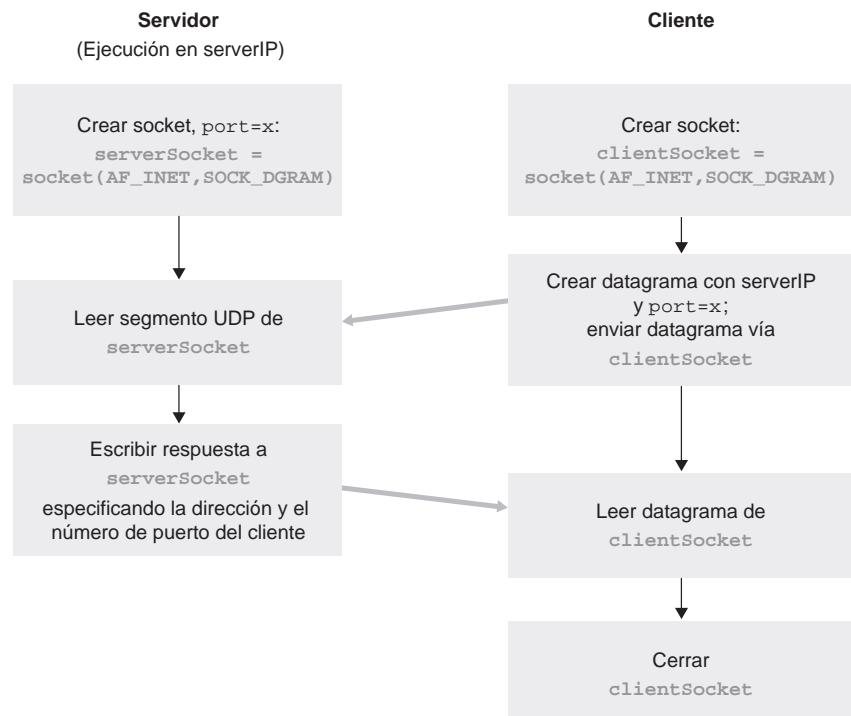
Así que puede que se esté preguntando ahora: ¿qué es lo que se introduce en la dirección de destino asociada al paquete? Como cabría esperar, la dirección IP del host de destino es parte de esa dirección de destino. Al incluir la dirección IP de destino en el paquete, los routers de Internet serán capaces de enrutar el paquete hasta el host de destino. Pero, dado que un host puede estar ejecutando muchos procesos de aplicaciones de red, cada uno de ellos con uno o más sockets, también es necesario identificar el socket concreto dentro del host de destino. Cuando se crea un socket, se le asigna un identificador, al que se denomina **número de puerto**. Por tanto, como cabría esperar, la dirección de destino del paquete también incluye el número de puerto del socket. En resumen, el proceso emisor asocia con el paquete una dirección de destino que está compuesta de la dirección IP del host de destino y del número de puerto del socket de destino. Además, como veremos enseguida, también se asocia al paquete la dirección de origen del emisor —compuesta por la dirección IP del host de origen y por el número de puerto del socket de origen—. Sin embargo, la asociación de la dirección de origen al paquete *no* suele ser realizada por el código de aplicación UDP; en lugar de ello, lo realiza automáticamente el sistema operativo subyacente.

Vamos a utilizar la siguiente aplicación cliente-servidor simple para demostrar cómo programar un socket tanto para UDP como para TCP:

1. El cliente lee una línea de caracteres (datos) de su teclado y envía los datos al servidor.
2. El servidor recibe los datos y convierte los caracteres a mayúsculas.
3. El servidor envía los datos modificados al cliente.
4. El cliente recibe los datos modificados y muestra la línea en su pantalla.

La Figura 2.27 muestra la actividad principal relativa a los sockets del cliente y del servidor que se comunican a través del servicio de transporte UDP.

A continuación proporcionamos la pareja de programas cliente-servidor para una implementación UDP de esta sencilla aplicación. Realizaremos un análisis detallado línea a línea de cada



**Figura 2.27** ♦ Aplicación cliente-servidor usando UDP.

uno de los programas. Comenzaremos con el cliente UDP, que enviará un mensaje del nivel de aplicación simple al servidor. Con el fin de que el servidor sea capaz de recibir y responder al mensaje del cliente, este debe estar listo y ejecutándose; es decir, debe estar ejecutándose como un proceso antes de que cliente envíe su mensaje.

El nombre del programa cliente es `UDPCliente.py` y el nombre del programa servidor es `UDPServidor.py`. Con el fin de poner el énfasis en las cuestiones fundamentales, hemos proporcionado de manera intencionada código que funciona correctamente pero que es mínimo. Un “código realmente bueno” tendría unas pocas más líneas auxiliares, en concreto aquellas destinadas al tratamiento de errores. Para esta aplicación, hemos seleccionado de forma arbitraria el número de puerto de servidor 12000.

### UDPCliente.py

He aquí el código para el lado del cliente de la aplicación:

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = raw_input('Escriba una frase en minúsculas: ')
clientSocket.sendto(message.encode(),(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())
clientSocket.close()
```

Veamos ahora las distintas líneas de código del programa `UDPClient.py`.

```
from socket import *
```

El módulo `socket` constituye la base de todas las comunicaciones de red en Python. Incluyendo esta línea, podemos crear sockets dentro de nuestro programa.

```
serverName = 'hostname'
serverPort = 12000
```

La primera línea define la variable `serverName` como la cadena ‘`hostname`’. Aquí, se proporciona una cadena de caracteres que contiene la dirección IP del servidor (como por ejemplo, “128.138.32.126”) o el nombre de host del servidor (por ejemplo, “`cis.poly.edu`”). Si utilizamos el nombre de host, entonces se llevará a cabo automáticamente una búsqueda DNS para obtener la dirección IP.) La segunda línea asigna el valor 12000 a la variable entera `serverPort`.

```
clientSocket = socket(AF_INET, SOCK_DGRAM)
```

Esta línea crea el socket de cliente denominado `clientSocket`. El primer parámetro indica la familia de direcciones; en particular, `AF_INET` indica que la red subyacente está utilizando IPv4. (No se preocupe en este momento por esto, en el Capítulo 4 abordaremos el tema de IPv4.) El segundo parámetro especifica que el socket es de tipo `SOCK_DGRAM`, lo que significa que se trata de un socket UDP (en lugar de un socket TCP). Observe que no se especifica el número de puerto del socket del cliente al crearlo; en lugar de ello, dejamos al sistema operativo que lo haga por nosotros. Una vez que hemos creado la puerta del proceso del cliente, querremos crear un mensaje para enviarlo a través de la puerta.

```
message = raw_input('Escriba una frase en minúsculas: ')
```

`raw_input()` es una función incorporada de Python. Cuando este comando se ejecuta, se solicita al usuario que se encuentra en el cliente que introduzca un texto en minúsculas con la frase “Escriba

una frase en minúsculas:"". El usuario utiliza entonces su teclado para escribir una línea, la cual se guarda en la variable `message`. Ahora que ya tenemos un socket y un mensaje, desearemos enviar el mensaje a través del socket hacia el host de destino.

```
clientSocket.sendto(message.encode(), (serverName, serverPort))
```

En la línea anterior, en primer lugar convertimos el mensaje de tipo cadena a tipo byte, ya que necesitamos enviar bytes por el socket; esto se hace mediante el método `encode()`. El método `sendto()` asocia la dirección de destino (`serverName, serverPort`) al mensaje y envía el paquete resultante por el socket de proceso, `clientSocket`. (Como hemos mencionado anteriormente, la dirección de origen también se asocia al paquete, aunque esto se realiza de forma automática en lugar de explícitamente a través del código.) ¡Enviar un mensaje de un cliente al servidor a través de un socket UDP es así de sencillo! Una vez enviado el paquete, el cliente espera recibir los datos procedentes del servidor.

```
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
```

Con esta línea, cuando un paquete procedente de Internet llega al socket del cliente, los datos del paquete se colocan en la variable `modifiedMessage` (mensaje modificado) y la dirección de origen del paquete se almacena en la variable `serverAddress`. Esta variable contiene tanto la dirección IP del servidor como el número del puerto del mismo. El programa `UDPClient` realmente no necesita esta información de dirección del servidor, puesto que ya la conoce, pero no obstante esta línea de Python proporciona dicha dirección. El método `recvfrom` también especifica el tamaño de buffer de 2048 como entrada. (Este tamaño de buffer es adecuado para prácticamente todos los propósitos.)

```
print(modifiedMessage.decode())
```

Esta línea muestra el mensaje modificado (`modifiedMessage`) en la pantalla del usuario, después de convertir el mensaje en bytes a un mensaje de tipo cadena, que tiene que ser la línea original que escribió el usuario, pero ahora escrito en letras mayúsculas.

```
clientSocket.close()
```

Esta línea cierra el socket y el proceso termina.

### **UDPServidor.py**

Echemos ahora un vistazo al lado del servidor de la aplicación:

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print("El servidor está listo para recibir")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

Observe que el principio de `UDPServidor` es similar a `UDPClient`. También importa el módulo `socket`, asigna el valor 12000 a la variable entera `serverPort` y crea también un socket de tipo `SOCK_DGRAM` (un socket UDP). La primera línea de código que es significativamente diferente de `UDPClient` es:

```
serverSocket.bind(('', serverPort))
```

La línea anterior asocia (es decir, asigna) el número de puerto 12000 al socket del servidor. Así, en UDPServidor, el código (escrito por el desarrollador de la aplicación) asigna explícitamente un número de puerto al socket. De este modo, cuando alguien envía un paquete al puerto 12000 en la dirección IP del servidor, dicho paquete será dirigido a este socket. A continuación, UDPServidor entra en un bucle `while`; este bucle permite a UDPServidor recibir y procesar paquetes de los clientes de manera indefinida. En el bucle `while`, UDPServidor espera a que llegue un paquete.

```
message, clientAddress = serverSocket.recvfrom(2048)
```

Esta línea de código es similar a la que hemos visto en UDPCliente. Cuando un paquete llega al socket del servidor, los datos del paquete se almacenan en la variable `message` y la dirección de origen del paquete se coloca en la variable `clientAddress`. La variable `clientAddress` contiene tanto la dirección IP del cliente como el número de puerto del cliente. Aquí, UDPServidor *hará uso* de esta información de dirección, puesto que proporciona una dirección de retorno, de forma similar a la dirección del remitente en una carta postal ordinaria. Con esta información de la dirección de origen, el servidor ahora sabe dónde dirigir su respuesta.

```
modifiedMessage = message.decode().upper()
```

Esta línea es la más importante de nuestra sencilla aplicación, ya que toma la línea enviada por el cliente y, después de convertir el mensaje en una cadena, utiliza el método `upper()` para pasarlo a mayúsculas.

```
serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

Esta última línea asocia la dirección del cliente (dirección IP y número de puerto) al mensaje escrito en mayúsculas (después de convertir la cadena a bytes) y envía el paquete resultante al socket del servidor. (Como hemos mencionado anteriormente, la dirección del servidor también se asocia con el paquete, aunque esto se hace de forma automática en lugar de explícitamente mediante el código.) A continuación, se suministrará el paquete a esa dirección de cliente a través de Internet. Una vez que el servidor envía el paquete, permanece en el bucle `while`, esperando la llegada de otro paquete UDP (de cualquier cliente que se esté ejecutando en cualquier host).

Para probar ambos programas, ejecute UDPCliente.py en un host y UDPServidor.py en otro host. Asegúrese de incluir el nombre de host o la dirección apropiados del servidor en UDPCliente.py. A continuación, ejecute UDPServidor.py, el programa del servidor compilado, en el host servidor. De este modo se crea un proceso en el servidor que está a la espera hasta que es contactado por algún cliente. Después, ejecute UDPCliente.py, el programa del cliente compilado, en el cliente. Esto crea un proceso en el cliente. Por último, utilice la aplicación en el cliente, escriba una frase seguida de un retorno de carro.

Para crear su propia aplicación cliente-servidor UDP, puede empezar modificando ligeramente los programas de cliente o de servidor. Por ejemplo, en lugar de pasar todas las letras a mayúsculas, el servidor podría contar el número de veces que aparece la letra `s` y devolver dicho número. O puede modificar el cliente de modo que después de recibir la frase en mayúsculas, el usuario pueda continuar enviando más frases al servidor.

## 2.7.2 Programación de sockets con TCP

A diferencia de UDP, TCP es un protocolo orientado a la conexión. Esto significa que antes de que el cliente y el servidor puedan empezar a enviarse datos entre sí, tienen que seguir un proceso de acuerdo en tres fases y establecer una conexión TCP. Un extremo de la conexión TCP se conecta al socket del cliente y el otro extremo se conecta a un socket de servidor. Cuando creamos la conexión TCP, asociamos con ella la dirección del socket de cliente (dirección IP y número de puerto) y la dirección del socket de servidor (dirección IP y número de puerto). Una vez establecida la conexión

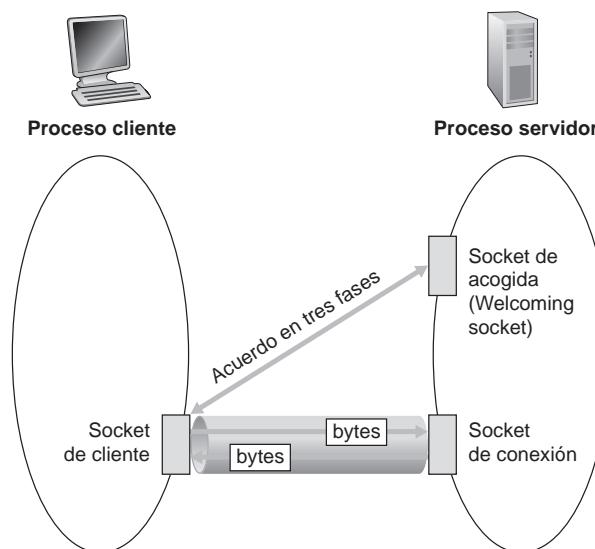
TCP, cuando un lado desea enviar datos al otro lado, basta con colocar los datos en la conexión TCP a través de su socket. Esto es distinto al caso de UDP, en el que el servidor tiene que tener asociada al paquete una dirección de destino antes de colocarlo en el socket.

Ahora, examinemos más en detalle la interacción entre los programas cliente y servidor en TCP. Al cliente le corresponde iniciar el contacto con el servidor. Para que este puede reaccionar al contacto inicial del cliente, tendrá que estar preparado, lo que implica dos cosas. En primer lugar, como en el caso de UDP, el servidor TCP tiene que estar ejecutándose como proceso antes de que el cliente trate de iniciar el contacto. En segundo lugar, el programa servidor debe disponer de algún tipo de puerta especial (o, más precisamente, un socket especial) que acepte algún contacto inicial procedente de un proceso cliente que se esté ejecutando en un host arbitrario. Utilizando nuestra analogía de la vivienda/puerta para un proceso/socket, en ocasiones nos referiremos a este contacto inicial del cliente diciendo que es equivalente a “llamar a la puerta de entrada”.

Con el proceso servidor ejecutándose, el proceso cliente puede iniciar una conexión TCP con el servidor. Esto se hace en el programa cliente creando un socket TCP. Cuando el cliente crea su socket TCP, especifica la dirección del socket de acogida (*wellcoming socket*) en el servidor, es decir, la dirección IP del host servidor y el número de puerto del socket. Una vez creado el socket en el programa cliente, el cliente inicia un proceso de acuerdo en tres fases y establece una conexión TCP con el servidor. El proceso de acuerdo en tres fases, que tiene lugar en la capa de transporte, es completamente transparente para los programas cliente y servidor.

Durante el proceso de acuerdo en tres fases, el proceso cliente llama a la puerta de entrada del proceso servidor. Cuando el servidor “escucha” la llamada, crea una nueva puerta (o de forma más precisa, un *nuevo socket*) que estará dedicado a ese cliente concreto. En el ejemplo que sigue, nuestra puerta de entrada es un objeto socket TCP que denominamos `serverSocket`; el socket que acabamos de crear dedicado al cliente que hace la conexión se denomina `connectionSocket`. Los estudiantes que se topan por primera vez con los sockets TCP confunden en ocasiones el socket de acogida (que es el punto inicial de contacto para todos los clientes que esperan para comunicarse con el servidor) con cada socket de conexión de nueva creación del lado del servidor que se crea posteriormente para comunicarse con cada cliente.

Desde la perspectiva de la aplicación, el socket del cliente y el socket de conexión del servidor están conectados directamente a través de un conducto. Como se muestra en la Figura 2.28, el proceso cliente puede enviar bytes arbitrarios a través de su socket, y TCP garantiza que



**Figura 2.28** ♦ El proceso TCPServidor tiene dos sockets.

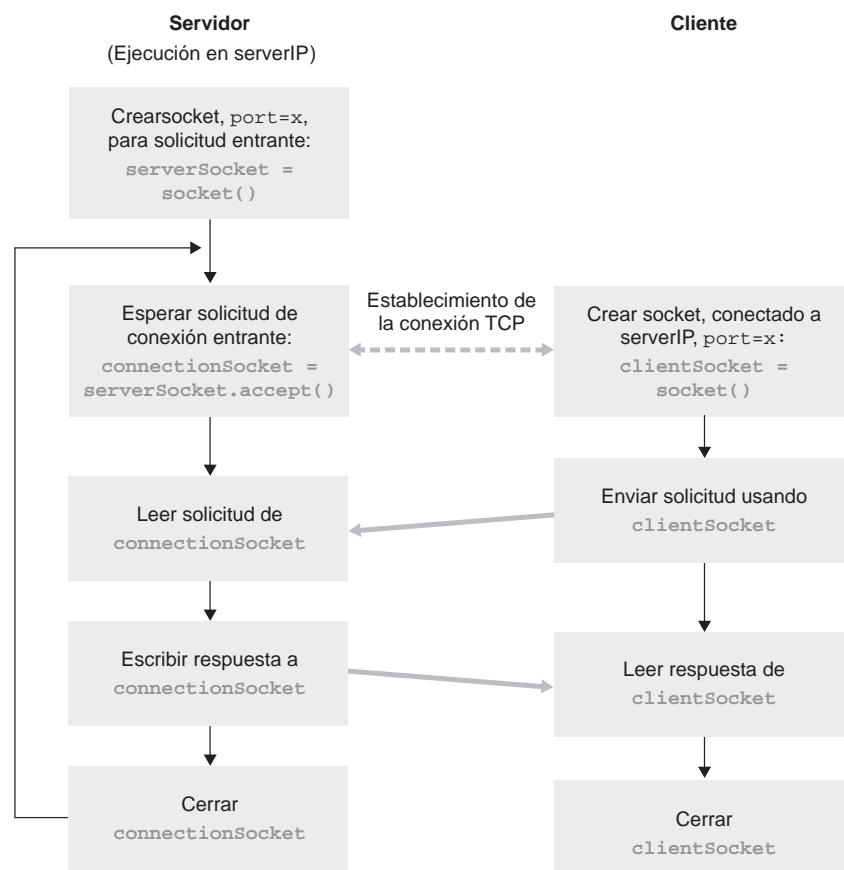
el proceso servidor recibirá (a través del socket de conexión) cada byte en el orden en que ha sido enviado. Por tanto, TCP proporciona un servicio fiable entre los procesos cliente y servidor. Además, al igual que las personas pueden entrar y salir a través de una misma puerta, el proceso cliente no sólo envía bytes a través de su socket, sino que también puede recibirlos; de forma similar, el proceso servidor no sólo puede recibir bytes, sino también enviar bytes a través de su socket de conexión.

Vamos a utilizar la misma aplicación cliente-servidor simple para mostrar la programación de sockets con TCP: el cliente envía una línea de datos al servidor, el servidor pone en mayúsculas esa línea y se la devuelve al cliente. En la Figura 2.29 se ha resaltado la actividad principal relativa al socket del cliente y el servidor que se comunican a través del servicio de transporte de TCP.

### TCPCliente.py

He aquí el código para el lado del cliente de la aplicación:

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
```



**Figura 2.29** ♦ La aplicación cliente-servidor utilizando TCP.

```

sentence = raw_input('Escriba una frase en minúsculas:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print('From Server: ', modifiedSentence.decode())
clientSocket.close()

```

Fíjémonos ahora en las líneas del código que difieren significativamente de la implementación para UDP. La primera de estas líneas es la de creación del socket de cliente.

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

Esta línea crea el socket de cliente, denominado `clientSocket`. De nuevo, el primer parámetro indica que la red subyacente está utilizando IPv4. El segundo parámetro indica que el socket es de tipo `SOCK_STREAM`, lo que significa que se trata de un socket TCP (en lugar de un socket UDP). Observe que de nuevo no especificamos el número de puerto del socket de cliente al crearlo; en lugar de ello, dejamos que sea el sistema operativo el que lo haga por nosotros. La siguiente línea de código es muy diferente de la que hemos visto en `UDPCClient`:

```
clientSocket.connect((serverName, serverPort))
```

Recuerde que antes de que el cliente pueda enviar datos al servidor (o viceversa) empleando un socket TCP, debe establecerse primero una conexión TCP entre el cliente y el servidor. La línea anterior inicia la conexión TCP entre el cliente y el servidor. El parámetro del método `connect()` es la dirección del lado de servidor de la conexión. Después de ejecutarse esta línea, se lleva a cabo el proceso de acuerdo en tres fases y se establece una conexión TCP entre el cliente y el servidor.

```
sentence = raw_input('Escriba una frase en minúsculas:')
```

Como con `UDPCClient`, la línea anterior obtiene una frase del usuario. La cadena `sentence` recopila los caracteres hasta que el usuario termina la línea con un retorno de carro. La siguiente línea de código también es muy diferente a la utilizada en `UDPCClient`:

```
clientSocket.send(sentence.encode())
```

La línea anterior envía la cadena `sentence` a través del socket de cliente y la conexión TCP. Observe que el programa *no* crea explícitamente un paquete y asocia la dirección de destino al paquete, como sucedía en el caso de los sockets UDP. En su lugar, el programa cliente simplemente coloca los bytes de la cadena `sentence` en la conexión TCP. El cliente espera entonces a recibir los bytes procedentes del servidor.

```
modifiedSentence = clientSocket.recv(2048)
```

Cuando llegan los caracteres de servidor, estos se colocan en la cadena `modifiedSentence`. Los caracteres continúan acumulándose en `modifiedSentence` hasta que la línea termina con un carácter de retorno de carro. Después de mostrar la frase en mayúsculas, se cierra el socket de cliente:

```
clientSocket.close()
```

Esta última línea cierra el socket y, por tanto, la conexión TCP entre el cliente y el servidor. Esto hace que TCP en el cliente envíe un mensaje TCP al proceso TCP del servidor (véase la Sección 3.5).

## TCPServidor.py

Veamos ahora el programa del servidor.

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print('El servidor está listo para recibir')
while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.encode())
    connectionSocket.close()
```

Estudiemos ahora las líneas que difieren significativamente en UDPServidor y TCPCliente. Como en el caso de TCPCliente, el servidor crea un socket TCP con:

```
serverSocket=socket(AF_INET,SOCK_STREAM)
```

De forma similar a UDPServidor, asociamos el número de puerto de servidor, `serverPort`, con este socket:

```
serverSocket.bind(('',serverPort))
```

Pero con TCP, `serverSocket` será nuestro socket de acogida. Después de establecer esta puerta de entrada, esperaremos hasta escuchar que algún cliente llama a la puerta:

```
serverSocket.listen(1)
```

Esta línea hace que el servidor esté a la escucha de solicitudes de conexión TCP del cliente. El parámetro especifica el número máximo de conexiones en cola (al menos, 1).

```
connectionSocket, addr = serverSocket.accept()
```

Cuando un cliente llama a esta puerta, el programa invoca el método `accept()` para el `serverSocket`, el cual crea un nuevo socket en el servidor, denominado `connectionSocket`, dedicado a este cliente concreto. El cliente y el servidor completan entonces el acuerdo en tres fases, creando una conexión TCP entre el socket `clientSocket` del cliente y el socket `connectionSocket` del servidor. Con la conexión TCP establecida, el cliente y el servidor ahora pueden enviarse bytes entre sí a través de la misma. Con TCP, no solo está garantizado que todos los bytes enviados desde un lado llegan al otro lado, sino que también queda garantizado que llegarán en orden.

```
connectionSocket.close()
```

En este programa, después de enviar la frase modificada al cliente, se cierra el socket de conexión. Pero puesto que `serverSocket` permanece abierto, otro cliente puede llamar a la puerta y enviar una frase al servidor para su modificación.

Esto completa nuestra exposición acerca de la programación de sockets en TCP. Le animamos a que ejecute los dos programas en dos hosts distintos y a que los modifique para obtener objetivos ligeramente diferentes. Debería comparar la pareja de programas para UDP con los programas para TCP y ver en qué se diferencian. También le aconsejamos que realice los ejercicios sobre programación de sockets descritos al final de los Capítulos 2, 4 y 9. Por último, esperamos que algún día, después de dominar estos y otros programas de sockets más complejos, escriba sus propia aplicación de red popular, se haga muy rico y famoso, y recuerde a los autores de este libro de texto.

## 2.8 Resumen

En este capítulo hemos estudiado los aspectos conceptuales y de implementación de las aplicaciones de red. Hemos podido comprobar la omnipresencia de la arquitectura cliente-servidor adoptada por muchas aplicaciones de Internet y ver su uso en los protocolos HTTP, SMTP, POP3 y DNS. Hemos estudiado estos importantes protocolos del nivel de aplicación y sus correspondientes aplicaciones asociadas (la Web, la transferencia de archivos, el correo electrónico y DNS) con cierto detalle. También hemos aprendido acerca de la arquitectura P2P y cómo se utiliza en muchas aplicaciones. También abordaremos los flujos de vídeo y cómo los modernos sistemas de distribución aprovechan las redes CDN. Hemos examinado cómo puede utilizarse la API de sockets para crear aplicaciones de red. Asimismo, hemos estudiado el uso de los sockets para los servicios de transporte terminal a terminal orientados a la conexión (TCP) y sin conexión (UDP). ¡Hemos completado la primera etapa de nuestro viaje por la arquitectura de red en capas!

Al principio del libro, en la Sección 1.1, hemos proporcionado una definición algo vaga de protocolo: “el formato y el orden de los mensajes intercambiados entre dos o más entidades que se comunican, así como las acciones realizadas en la transmisión y/o recepción de un mensaje u otro evento.” El material facilitado en este capítulo, y en concreto el estudio detallado de los protocolos HTTP, SMTP, POP3 y DNS, aporta a esta definición una gran profundidad. Los protocolos son un concepto clave en las redes y nuestro estudio de los protocolos de aplicación nos ha proporcionado la oportunidad de desarrollar una idea más intuitiva sobre qué son los protocolos.

En la Sección 2.1 hemos descrito los modelos de servicio que ofrecen TCP y UDP a las aplicaciones que los invocan. En las Secciones 2.7 y 2.8 hemos visto en detalle estos modelos de servicio para el desarrollo de aplicaciones sencillas que se ejecutan sobre TCP y UDP. Sin embargo, hemos hablado poco acerca de cómo TCP y UDP proporcionan estos modelos de servicio. Por ejemplo, sabemos que TCP proporciona un servicio de datos fiable, pero todavía no sabemos cómo lo hace. En el siguiente capítulo veremos detenidamente no solo qué servicios proporcionan, sino también el cómo y el por qué del funcionamiento de los protocolos de transporte.

Ahora que ya tenemos algunos conocimientos acerca de la estructura de las aplicaciones de Internet y de los protocolos de la capa de aplicación, estamos preparados para seguir descendiendo por la pila de protocolos y examinar la capa de transporte en el Capítulo 3.

## Problemas y cuestiones de repaso

---

### Capítulo 2 Cuestiones de repaso

#### SECCIÓN 2.1

- R1. Enumere cinco aplicaciones de Internet no propietarias y los protocolos de la capa de aplicación que utilizan.
- R2. ¿Cuál es la diferencia entre la arquitectura de red y la arquitectura de aplicación?
- R3. En una sesión de comunicación entre dos procesos, ¿qué proceso es el cliente y qué proceso es el servidor?
- R4. En una aplicación de compartición de archivos P2P, ¿está de acuerdo con la siguiente afirmación: “No existen los lados de cliente y de servidor en una sesión de comunicación”? ¿Por qué?
- R5. ¿Qué información utiliza un proceso que se ejecuta en un host para identificar a un proceso que se ejecuta en otro host?
- R6. Suponga que desea realizar una transición desde un cliente remoto a un servidor lo más rápidamente posible. ¿Qué utilizaría, UDP o TCP? ¿Por qué?
- R7. Utilizando la Figura 2.4, podemos ver que ninguna de las aplicaciones indicadas en dicha figura presenta a la vez requisitos de temporización y de ausencia de pérdida de datos. ¿Puede

concebir una aplicación que requiera que no haya pérdida de datos y que también sea extremadamente sensible al tiempo?

- R8. Enumere las cuatro clases principales de servicios que puede proporcionar un protocolo de transporte. Para cada una de las clases de servicios, indique si UDP o TCP (o ambos) proporcionan un servicio así.
- R9. Recuerde que TCP puede mejorarse con SSL para proporcionar servicios de seguridad proceso a proceso, incluyendo mecanismos de cifrado. ¿En qué capa opera SSL, en la capa de transporte o en la capa de aplicación? Si el desarrollador de la aplicación desea mejorar TCP con SSL, ¿qué tendrá que hacer?

#### SECCIÓN 2.2-2.5

R10. ¿Qué quiere decir el término protocolo de acuerdo?

R11. ¿Por qué HTTP, SMTP y POP3 se ejecutan sobre TCP en lugar de sobre UDP?

R12. Un sitio de comercio electrónico desea mantener un registro de compras para cada uno de sus clientes. Describa cómo se puede hacer esto utilizando cookies.

R13. Describa cómo el almacenamiento en caché web puede reducir el retardo de recepción de un objeto solicitado. ¿Reducirá este tipo de almacenamiento el retardo de todos los objetos solicitados por el usuario o sólo el de algunos objetos? ¿Por qué?

R14. Establezca una sesión Telnet en un servidor web y envíe un mensaje de solicitud de varias líneas. Incluya en dicho mensaje la línea de cabecera `If-modified-since:` para forzar un mensaje de respuesta con el código de estado `304 Not Modified`.

R15. Enumere algunas aplicaciones de mensajería populares. ¿Utilizan el mismo protocolo que SMS?

R16. Suponga que Alicia, que dispone de una cuenta de correo electrónico web (como por ejemplo Hotmail o gmail), envía un mensaje a Benito, que accede a su correo almacenado en su servidor de correo utilizando POP3. Explique cómo se transmite el mensaje desde el host de Alicia hasta el de Benito. Asegúrese de citar la serie de protocolos de la capa de aplicación que se utilizan para llevar el mensaje de un host al otro.

R17. Imprima la cabecera de un mensaje de correo electrónico que haya recibido recientemente. ¿Cuántas líneas de cabecera `Received:` contiene? Analice cada una de las líneas de cabecera del mensaje.

R18. Desde la perspectiva de un usuario, ¿cuál es la diferencia entre el modo “descargar y borrar” y el modo “descargar y mantener” en POP3?

R19. ¿Pueden el servidor web y el servidor de correo electrónico de una organización tener exactamente el mismo alias para un nombre de host (por ejemplo, `foo.com`)? ¿Cuál sería el tipo especificado en el registro de recurso (RR) que contiene el nombre de host del servidor de correo?

R20. Estudie sus mensajes de correo electrónico recibidos y examine la cabecera de un mensaje enviado desde un usuario con una dirección de correo electrónico `.edu`. ¿Es posible determinar a partir de la cabecera la dirección IP del host desde el que se envió el mensaje? Repita el proceso para un mensaje enviado desde una cuenta de Gmail.

#### SECCIÓN 2.5

R21. En BitTorrent, suponga que Alicia proporciona fragmentos a Benito a intervalos de 30 segundos. ¿Devolverá necesariamente Benito el favor y proporcionará fragmentos a Alicia en el mismo intervalo de tiempo? ¿Por qué?

R22. Suponga que un nuevo par Alicia se une a BitTorrent sin tener en su posesión ningún fragmento. Dado que no posee fragmentos, no puede convertirse en uno de los cuatro principales suministros.

tradores de ninguno de los otros pares, ya que no tiene nada que suministrar. ¿Cómo obtendrá entonces Alicia su primer fragmento?

R23. ¿Qué es una red solapada? ¿Contiene routers? ¿Cuáles son las fronteras en una red solapada?

### SECCIÓN 2.6

R24. Normalmente, las redes CDN adoptan una de dos filosofías diferentes de ubicación de los servidores. Nómbralas y descríbalas brevemente.

R25. Además de las consideraciones relativas a las redes como son los retardos, las pérdidas de paquetes y el ancho de banda, existen otros factores importantes que se deben tener en cuenta a la hora de diseñar una estrategia de selección del servidor CDN. ¿Cuáles son esos factores?

### SECCIÓN 2.7

R26. El servidor UDP descrito en la Sección 2.7 solo necesitaba un socket, mientras que el servidor TCP necesitaba dos. ¿Por qué? Si el servidor TCP tuviera que soportar  $n$  conexiones simultáneas, cada una procedente de un host cliente distinto, ¿cuántos sockets necesitaría el servidor TCP?

R27. En la aplicación cliente-servidor sobre TCP descrita en la Sección 2.7, ¿por qué tiene que ser ejecutado el programa servidor antes que el programa cliente? En la aplicación cliente-servidor sobre UDP, ¿por qué el programa cliente puede ejecutarse antes que el programa servidor?

## Problemas

---

P1. ¿Verdadero o falso?

- Un usuario solicita una página web que consta de texto y tres imágenes. Para obtener esa página, el cliente envía un mensaje de solicitud y recibe cuatro mensajes de respuesta.
- Dos páginas web diferentes (por ejemplo, [www.mit.edu/research.html](http://www.mit.edu/research.html) y [www.mit.edu/students.html](http://www.mit.edu/students.html)) se pueden enviar a través de la misma conexión persistente.
- Con las conexiones no persistentes entre un navegador y un servidor de origen, un único segmento TCP puede transportar dos mensajes de solicitud HTTP distintos.
- La línea de cabecera `Date:` del mensaje de respuesta HTTP indica cuándo el objeto fue modificado por última vez.
- Los mensajes de respuesta HTTP nunca incluyen un cuerpo de mensaje vacío.

P2. SMS, iMessage y WhatsApp son todos ellos sistemas de mensajería en tiempo real para smartphone. Después de llevar a cabo una pequeña investigación en Internet, escriba un párrafo indicando los protocolos que cada uno de estos sistemas emplea. A continuación, escriba un párrafo explicando en qué se diferencian.

P3. Un cliente HTTP desea recuperar un documento web que se encuentra en un URL dado. Inicialmente, la dirección IP del servidor HTTP es desconocida. ¿Qué protocolos de la capa de aplicación y de la capa de transporte además de HTTP son necesarios en este escenario?

P4. La siguiente cadena de caracteres ASCII ha sido capturada por Wireshark cuando el navegador enviaba un mensaje GET HTTP (es decir, este es el contenido real de un mensaje GET HTTP). Los caracteres `<cr><lf>` representan el retorno de carro y el salto de línea (es decir, la cadena de caracteres en cursiva `<cr>` del texto que sigue a este párrafo representa el carácter de retorno de carro contenido en dicho punto de la cabecera HTTP). Responda a las siguientes cuestiones, indicando en qué parte del siguiente mensaje GET HTTP se encuentra la respuesta.

```
GET /cs453/index.html HTTP/1.1<cr><lf>Host: gai
```

```
a.cs.umass.edu<cr><lf>User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.2) Gecko/20040804 Netscape/7.2 (ax) <cr><lf>Accept: text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/plain;q=0.8, image/png,*/*;q=0.5<cr><lf>Accept-Language: en-us,en;q=0.5<cr><lf>Accept-Encoding: zip,deflate<cr><lf>Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7<cr><lf>Keep-Alive: 300<cr><lf>Connection:keep-alive<cr><lf><cr><lf>
```

- a. ¿Cuál es el URL del documento solicitado por el navegador?
- b. ¿Qué versión de HTTP se está ejecutando en el navegador?
- c. ¿Qué tipo de conexión solicita el navegador, persistente o no persistente?
- d. ¿Cuál es la dirección IP del host en el que se está ejecutando el navegador?
- e. ¿Qué tipo de navegador inicia este mensaje? ¿Por qué es necesario indicar el tipo de navegador en un mensaje de solicitud HTTP?
- P5. El siguiente texto muestra la respuesta devuelta por el servidor al mensaje de solicitud GET HTTP del problema anterior. Responda a las siguientes cuestiones, indicando en qué parte del siguiente mensaje se encuentran las respuestas.

```
HTTP/1.1 200 OK<cr><lf>Date: Tue, 07 Mar 2008  
12:39:45GMT<cr><lf>Server: Apache/2.0.52 (Fedora)  
<cr><lf>Last-Modified: Sat, 10 Dec 2005 18:27:46 GMT<cr><lf>ETag:  
"526c3-f22-a88a4c80"<cr><lf>Accept-  
Ranges: bytes<cr><lf>Content-Length: 3874<cr><lf>  
Keep-Alive: timeout=max=100<cr><lf>Connection:  
Keep-Alive<cr><lf>Content-Type: text/html; charset=  
ISO-8859-1<cr><lf><cr><lf><!DOCTYPE html public "-//  
/w3c//dtd html 4.0 transitional//en"><lf><html><lf> <head><lf>  
<meta http-equiv="Content-Type"  
content="text/html; charset=iso-8859-1"><lf> <meta  
name="GENERATOR" content="Mozilla/4.79 [en] (Windows NT  
5.0; U) Netscape]"><lf> <title>CMPSCI 453 / 591 /  
NTU-ST550ASpring 2005 homepage</title><lf></head><lf>  
<aquí continúa el texto del documento (no mostrado)>
```

- a. ¿Ha podido el servidor encontrar el documento? ¿En qué momento se suministró la respuesta con el documento?
- b. ¿Cuándo fue modificado por última vez el documento?
- c. ¿Cuántos bytes contiene el documento devuelto?
- d. ¿Cuáles son los primeros cinco bytes del documento que se está devolviendo? ¿Ha acordado el servidor emplear una conexión persistente?
- P6. Utilice la especificación HTTP/1.1 (RFC 2616) para responder a las siguientes cuestiones:
- a. Explique el mecanismo de señalización entre el cliente y el servidor para indicar que se está cerrando una conexión persistente. ¿Quién puede señalizar el cierre de la conexión, el cliente, el servidor o ambos?
- b. ¿Qué servicios de cifrado proporciona HTTP?
- c. ¿Puede un cliente abrir tres o más conexiones simultáneas con un determinado servidor?
- d. Un servidor o un cliente pueden cerrar una conexión de transporte entre ellos si uno detecta que la conexión ha estado inactiva durante un cierto tiempo. ¿Es posible que un lado inicie

el cierre de una conexión mientras que el otro lado está transmitiendo datos a través de dicha conexión? Explique su respuesta.

- P7. Suponga que en su navegador hace clic en un vínculo a una página web. La dirección IP correspondiente al URL asociado no está almacenado en la caché de su host local, por lo que es necesario realizar una búsqueda DNS para obtener la dirección IP. Suponga que antes de que su host reciba la dirección IP de DNS se han visitado  $n$  servidores DNS y que los tiempos de ida y vuelta (RTT) de las sucesivas visitas son  $RTT_1, \dots, RTT_n$ . Suponga además que la página web asociada con el vínculo contiene exactamente un objeto, que consta de un pequeño fragmento de texto HTML. Sea  $RTT_0$  el tiempo RTT entre el host local y el servidor que contiene el objeto. Suponiendo un tiempo de transmisión de cero para el objeto, ¿cuánto tiempo transcurre desde que el cliente hace clic en el vínculo hasta que recibe el objeto?
- P8. Continuando con el Problema P7, suponga que el archivo HTML hace referencia a ocho objetos muy pequeños que se encuentran en el mismo servidor. Despreciando los tiempos de transmisión, ¿cuánto tiempo transcurre si se utiliza
- HTTP no persistente sin conexiones TCP en paralelo?
  - HTTP no persistente con el navegador configurado para 5 conexiones paralelo?
  - HTTP persistente?
- P9. En la red institucional conectada a Internet de la Figura 2.12, suponga que el tamaño medio de objeto es de 850.000 bits y que la tasa media de solicitudes de los navegadores de la institución a los servidores de origen es de 16 solicitudes por segundo. Suponga también que el tiempo que se tarda desde que el router en el lado de Internet del enlace de acceso reenvía una solicitud HTTP hasta que recibe la respuesta es, como media, de tres segundos (véase la Sección 2.2.5). Modele el tiempo medio de respuesta total como la suma del retardo medio de acceso (es decir, el retardo desde el router de Internet al router de la institución) y el retardo medio de Internet. Para el retardo medio de acceso, utilice la expresión  $\Delta/(1 - \Delta\beta)$ , donde  $\Delta$  es el tiempo medio requerido para enviar un objeto a través del enlace de acceso y  $\beta$  es la tasa de llegada de los objetos al enlace de acceso.
- Calcule el tiempo medio de respuesta total.
  - Ahora suponga que hay instalada una caché en la LAN institucional. Suponga que la tasa de fallos es de 0,4. Calcule el tiempo de respuesta total.
- P10. Dispone de un enlace corto de 10 metros a través del cual un emisor puede transmitir a una velocidad de 150 bits/segundo en ambos sentidos. Suponga que los paquetes de datos tienen una longitud de 100.000 bits y los paquetes que contienen solo comandos de control (por ejemplo, ACK o de acuerdo) tienen una longitud de 200 bits. Suponga que hay  $N$  conexiones en paralelo y que cada una utiliza  $1/N$  del ancho de banda del enlace. Considere ahora el protocolo HTTP y suponga que cada objeto descargado es de 100 kbits de largo y que el objeto inicialmente descargado contiene 10 objetos referenciados procedentes del mismo emisor. ¿Tiene sentido en este caso realizar descargas en paralelo mediante instancias paralelas de HTTP no persistente? Considere ahora HTTP persistente. ¿Cabe esperar alguna ventaja significativa respecto del caso no persistente? Justifique y explique su respuesta.
- P11. Continuando con el escenario del problema anterior, suponga que Benito comparte el enlace con otros cuatro usuarios. Benito utiliza instancias paralelas de HTTP no persistente y los otros cuatro usuarios utilizan HTTP no persistente sin descargas en paralelo.
- ¿Le ayudan a Benito las conexiones en paralelo a obtener las páginas más rápidamente? ¿Por qué?
  - Si los cinco usuarios abren cinco instancias paralelas de HTTP no persistente, ¿seguirán siendo beneficiosas las conexiones en paralelo de Benito? ¿Por qué?
- P12. Escriba un programa TCP simple para un servidor que acepte líneas de entrada procedentes de un cliente y muestre dichas líneas en la salida estándar del servidor. (Puede realizar esta tarea

modificando el programa TCPServer.py visto en el capítulo.) Compile y ejecute su programa. En cualquier otra máquina que disponga de un navegador web, configure el servidor proxy en el navegador para que apunte al host que está ejecutando su programa servidor; configure también el número de puerto de la forma apropiada. Su navegador deberá ahora enviar sus mensajes de solicitud GET a su servidor y el servidor tendrá que mostrar dichos mensajes en su salida estándar. Utilice esta plataforma para determinar si su navegador genera mensajes GET condicionales para los objetos almacenados localmente en la caché.

- P13. ¿Cuál es la diferencia entre MAIL FROM: en SMTP y From: en el propio mensaje de correo?
- P14. ¿Cómo marca SMTP el final del cuerpo de un mensaje? ¿Cómo lo hace HTTP? ¿Puede HTTP utilizar el mismo método que SMTP para marcar el final del cuerpo de un mensaje? Explique su respuesta.
- P15. Lea el documento RFC 5321 dedicado a SMTP. ¿Qué quiere decir MTA? Considere el siguiente mensaje de correo basura recibido (modificado a partir de un correo basura real). Suponiendo que únicamente el remitente de este mensaje de correo es malicioso y que los demás hosts son honestos, identifique al host malicioso que ha generado este correo basura.

```
From - Fri Nov 07 13:41:30 2008
Return-Path: <tennis5@pp33head.com>
Received: from barmail.cs.umass.edu (barmail.cs.umass.edu
[128.119.240.3]) by cs.umass.edu (8.13.1/8.12.6) for
<hg@cs.umass.edu>; Fri, 7 Nov 2008 13:27:10 -0500
Received: from asusus-4b96 (localhost [127.0.0.1]) by
barmail.cs.umass.edu (Spam Firewall) for <hg@cs.umass.edu>; Fri, 7
Nov 2008 13:27:07 -0500 (EST)
Received: from asusus-4b96 ([58.88.21.177]) by barmail.cs.umass.edu
for <hg@cs.umass.edu>; Fri, 07 Nov 2008 13:27:07 -0500 (EST)
Received: from [58.88.21.177] by inbnd55.exchangedddd.com; Sat, 8
Nov 2008 01:27:07 +0700
From: "Jonny" <tennis5@pp33head.com>
To: <hg@cs.umass.edu>

Subject: Cómo asegurar sus ahorros
```

- P16. Lea el documento RFC 1939 dedicado a POP3. ¿Cuál es el propósito del comando UIDL POP3?
- P17. Imagine que accede a su correo electrónico utilizando POP3.

- a. Suponga que ha configurado su cliente de correo POP para operar en el modo descargar y borrar. Complete la siguiente transacción:

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: bla bla ...
S: .....bla
S: .
?
?
```

- b. Suponga que ha configurado su cliente de correo POP para operar en el modo descargar y guardar. Complete la siguiente transacción:

```

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: bla bla ...
S: .....bla
S: .
?
?
```

- c. Suponga que ha configurado su cliente de correo POP para operar en el modo descargar y guardar. Utilizando su transcripción del apartado (b), suponga que recupera los mensajes 1 y 2, sale de POP, y cinco minutos más tarde vuelve a acceder otra vez a POP para recuperar un nuevo mensaje de correo. Suponga que en ese intervalo de cinco minutos nadie le ha enviado un nuevo mensaje de correo. Proporcione una transcripción de esta segunda sesión de POP.
- P18. a. ¿Qué es una base de datos *whois*?
- Utilice varias bases de datos whois de Internet para obtener los nombres de dos servidores DNS. Indique qué bases de datos whois ha utilizado.
  - Utilice el programa nslookup en su host local para enviar consultas DNS a tres servidores DNS: su servidor DNS local y los dos servidores DNS que haya encontrado en el apartado (b). Intente realizar consultas para obtener registros de recursos de tipo A, NS y MX. Escriba un resumen de sus hallazgos.
  - Utilice el programa nslookup para localizar un servidor web que tenga varias direcciones IP. ¿Tiene el servidor web de su institución (centro de estudios o empresa) varias direcciones IP?
  - Utilice la base de datos whois ARIN para determinar el rango de direcciones IP utilizado en su universidad.
  - Describa cómo un atacante puede utilizar las bases de datos whois y la herramienta nslookup para realizar labores de reconocimiento en una institución antes de lanzar un ataque.
  - Explique por qué las bases de datos whois deben estar disponibles públicamente.
- P19. En este problema empleará la útil herramienta *dig*, disponible en los hosts Unix y Linux para explorar la jerarquía de los servidores DNS. Como se muestra en la Figura 2.19, un servidor DNS que se encuentra en un nivel superior de la jerarquía DNS delega una consulta DNS en un servidor DNS que se encuentra en un nivel más bajo de la jerarquía, devolviendo al cliente DNS el nombre de dicho servidor DNS del nivel más bajo. Lea primero la página de manual dedicada a *dig* y, a continuación, responda a las siguientes preguntas.
- Comenzando por un servidor DNS raíz (uno de los servidores raíz [a-m].root-servers.net), inicie una secuencia de consultas para obtener la dirección IP del servidor web de su departamento, utilizando la herramienta *dig*. Visualice la lista de nombres de los servidores DNS incluidos en la cadena de delegación que ha obtenido como respuesta a su consulta.
  - Repita el apartado (a) para varios sitios web populares, como por ejemplo google.com, yahoo.com o amazon.com.
- P20. Suponga que puede acceder a las cachés de los servidores DNS locales de su departamento. ¿Puede proponer una forma de determinar de manera aproximada los servidores web (situados fuera de su departamento) que son más populares entre los usuarios del departamento? Explique su respuesta.

- P21. Suponga que su departamento tiene un servidor DNS local para todas las computadoras del departamento. Usted es un usuario normal (es decir, no es un administrador de la red o del sistema). ¿Puede determinar de alguna manera si desde alguna computadora de su departamento se ha accedido hace unos pocos segundos a un determinado sitio web externo? Explique su respuesta.
- P22. Desea distribuir un archivo de  $F = 15$  Gbits a  $N$  pares. El servidor tiene una velocidad de carga de  $u_s = 30$  Mbps, y cada par tiene una velocidad de descarga de  $d_i = 2$  Mbps y una velocidad de carga igual a  $u$ . Para  $N = 10, 100$  y  $1.000$ , y  $u = 300$  kbps, 700 kbps y 2 Mbps, prepare una gráfica que proporcione el tiempo mínimo de distribución para cada una de las combinaciones de  $N$  y  $u$ , tanto para una distribución cliente-servidor como para una distribución P2P.
- P23. Desea distribuir un archivo de  $F$  bits a  $N$  pares utilizando una arquitectura cliente-servidor. Suponga un modelo flexible, en el que el servidor puede transmitir simultáneamente a varios pares, transmitiendo a cada par a distintas velocidades, siempre y cuando la velocidad combinada no sea mayor que  $u_s$ .
- Suponga que  $u_s/N \leq d_{\min}$ . Especifique un esquema de distribución que tenga un tiempo de distribución de  $NF/u_s$ .
  - Suponga que  $u_s/N \geq d_{\min}$ . Especifique un esquema de distribución que tenga un tiempo de distribución de  $F/d_{\min}$ .
  - Demuestre que, en general, el tiempo mínimo de distribución está dado por  $\max\{NF/u_s, F/d_{\min}\}$ .
- P24. Desea distribuir un archivo de  $F$  bits a  $N$  pares utilizando una arquitectura P2P. Suponga un modelo flexible. Con el fin de simplificar, suponga que  $d_{\min}$  es muy grande, por lo que el ancho de banda de descarga de los pares no es nunca un cuello de botella.
- Suponga que  $u_s \leq (u_s + u_1 + \dots + u_N)/N$ . Especifique un esquema de distribución que tenga un tiempo de distribución de  $F/u_s$ .
  - Suponga que  $u_s \geq (u_s + u_1 + \dots + u_N)/N$ . Especifique un esquema de distribución que tenga un tiempo de distribución de  $NF/(u_s + u_1 + \dots + u_N)$ .
  - Demuestre que, en general, el tiempo mínimo de distribución está dado por  $\max\{F/u_s, NF/(u_s + u_1 + \dots + u_N)\}$ .
- P25. Considere una red solapada con  $N$  pares activos, disponiendo cada pareja de pares de una conexión TCP activa. Suponga también que las conexiones TCP atraviesan un total de  $M$  routers. ¿Cuántos nodos y fronteras existen en la correspondiente red solapada?
- P26. Suponga que Benito se une a un torrente BitTorrent, pero no desea suministrar datos a otros pares (lo que se denomina “ir por libre”).
- Benito afirma que puede recibir una copia completa del archivo compartido por el conjunto de usuarios. ¿Es correcto lo que dice Benito? ¿Por qué?
  - Benito añade que puede hacer más eficientes sus descargas utilizando varias computadoras (con distintas direcciones IP) del laboratorio de su departamento. ¿Cómo puede hacer esto?
- P27. Considere un sistema DASH para el que hay  $N$  versiones de un vídeo (con  $N$  diferentes velocidades y niveles de calidad) y  $N$  versiones de audio (con  $N$  diferentes velocidades y niveles de calidad). Suponga que queremos permitir que el reproductor seleccione en cualquier instante cualquiera de las  $N$  versiones de vídeo y de las  $N$  versiones de audio.
- Si creamos archivos de modo que el audio esté mezclado con el vídeo, y el servidor envíe solo un flujo multimedia en cualquier momento dado, ¿cuántos archivos necesitará almacenar el servidor (cada uno con un URL distinto)?
  - Si, por el contrario, el servidor envía los flujos de vídeo y de audio por separado y hacemos que el cliente sincronice los flujos, ¿cuántos archivos necesitará almacenar el servidor?

- P28. Instale y compile los programas Python TCPCliente y UDPCliente en un host y TCPServidor y UDPServidor en otro host.
- Suponga que ejecuta TCPCliente antes que TCPServidor. ¿Qué ocurre? ¿Por qué?
  - Suponga que ejecuta UDPCliente antes que UDPServidor. ¿Qué ocurre? ¿Por qué?
  - ¿Qué ocurre si se utilizan diferentes números de puerto para los lados cliente y servidor?
- P29. Suponga que en UDPCliente.py, después de crear el socket, añadimos esta línea:
- ```
clientSocket.bind(('', 5432))
```
- ¿Será necesario modificar el programa UDPServidor.py? ¿Cuáles son los números de puerto para los sockets en UDPCliente y UDPServidor? ¿Cuáles eran antes de realizar este cambio?
- P30. ¿Puede configurar su navegador para abrir múltiples conexiones simultáneas a un sitio web? ¿Cuáles son las ventajas y desventajas de disponer de un gran número de conexiones TCP simultáneas?
- P31. Hemos visto que los sockets TCP de Internet tratan los datos que están siendo enviados como un flujo de bytes, pero que los sockets UDP reconocen las fronteras entre mensajes. Cite una ventaja y una desventaja de la API orientada a bytes, con respecto a la API que reconoce y preserva explícitamente las fronteras entre mensajes definidas por la aplicación.
- P32. ¿Qué es el servidor web Apache? ¿Cuánto cuesta? ¿Cuál es la funcionalidad que tiene en la actualidad? Puede consultar la Wikipedia para responder a esta pregunta.

## Tareas sobre programación de sockets

---

El sitio web de acompañamiento incluye seis tareas de programación de sockets. Las cuatro primeras se resumen a continuación. La quinta tarea utiliza el protocolo ICMP y se resume al final del Capítulo 5. La sexta tarea emplea protocolos multimedia y se resume al final del Capítulo 9. Recomendamos fuertemente a los estudiantes que completen algunas, si no todas, de estas tareas. Los estudiantes pueden encontrar la información completa acerca de estas tareas, así como importantes *snippets* del código Python, en el sitio web [www.pearsonhighered.com/cs-resources](http://www.pearsonhighered.com/cs-resources).

### Tarea 1: servidor web

En esta tarea, tendrá que desarrollar un servidor web simple en Python que sea capaz de procesar una única solicitud. Específicamente, el servidor web deberá (i) crear un socket de conexión al ser contactado por un cliente (navegador); (ii) recibir la solicitud HTTP a través de dicha conexión; (iii) analizar sintácticamente la solicitud para determinar qué archivo concreto se está solicitando; (iv) obtener el archivo solicitado del sistema de archivos del servidor; (v) crear un mensaje de respuesta HTTP consistente en el archivo solicitado, precedido por una serie de líneas de cabecera, y (vi) enviar la respuesta al navegador solicitante a través de la conexión TCP. Si un navegador solicita un archivo que no esté presente en su servidor, el servidor web debe devolver un mensaje de error “404 Not Found”.

En el sitio web de acompañamiento proporcionamos el esqueleto de código para el servidor web. Su tarea consiste en completar el código, ejecutar el servidor y luego probarlo, enviándole solicitudes desde navegadores que se ejecuten en diferentes hosts. Si ejecuta su servidor web en un host que ya tenga otro servidor web funcionando, deberá utilizar para su servidor web un puerto distinto del puerto 80.

### Tarea 2: programa ping UDP

En esta tarea de programación, tendrá que escribir un programa cliente ping en Python. El cliente deberá enviar un simple mensaje ping a un servidor, recibir un mensaje pong correspondiente de

respuesta del servidor y determinar el retardo existente entre el instante en que el cliente envió el mensaje ping y el instante en que recibió el mensaje pong. Este retardo se denomina RTT (*Round Trip Time*, tiempo de ida y vuelta). La funcionalidad determinada por el cliente y el servidor es similar a la que ofrece el programa ping estándar disponible en los sistemas operativos modernos. Sin embargo, los programas ping estándar utilizan el protocolo ICMP (*Internet Control Message Protocol*, protocolo de mensajes de control de Internet), del cual hablaremos en el Capítulo 5. Aquí crearemos un programa ping no estándar (¡pero sencillo!) basado en UDP.

Su programa ping deberá enviar 10 mensajes ping al servidor objetivo a través de UDP. Para cada mensaje, el cliente debe determinar e imprimir el RTT cuando reciba el correspondiente mensaje pong. Como UDP es un protocolo no fiable, puede perderse algún paquete enviado por el cliente o por el servidor. Por esta razón, el cliente no puede quedarse esperando indefinidamente a recibir una respuesta a un mensaje ping. Debe hacer que el cliente espere un máximo de un segundo a recibir una respuesta del servidor; si no se recibe respuesta, el cliente debe asumir que el paquete se perdió e imprimir un mensaje informando de tal hecho.

En esta tarea, le proporcionamos el código completo del servidor (disponible en el sitio web de acompañamiento). Su trabajo consiste en escribir el código del cliente, que será muy similar al código del servidor. Le recomendamos que primero estudie con atención dicho código del servidor. Después podrá escribir el código del cliente, cortando y pegando líneas del código del servidor según sea necesario.

### Tarea 3: cliente de correo

El objetivo de esta tarea de programación es crear un cliente de correo simple que envíe correos electrónicos a cualquier destinatario. Su cliente necesitará establecer una conexión TCP con un servidor de correo (por ejemplo, el servidor de correo de Google), dialogar con el servidor de correo utilizando el protocolo SMTP, enviar un mensaje de correo electrónico a un receptor (por ejemplo, un amigo suyo) a través del servidor de correo y, finalmente, cerrar la conexión TCP con el servidor de correo.

Para esta tarea, el sitio web de acompañamiento proporciona el esqueleto de código para el cliente. Su tarea consiste en completar el código y probar su cliente enviando correos electrónicos a diferentes cuentas de usuario. También puede intentar enviar los correos a través de diferentes servidores (por ejemplo, a través de un servidor de correo de Google y del servidor de correo de su universidad).

### Tarea 4: proxy web multihebra

En esta tarea, tendrá que desarrollar un proxy web. Cuando su proxy reciba de un navegador una solicitud HTTP para un cierto objeto, deberá generar una nueva solicitud HTTP para el mismo objeto y enviarla al servidor de destino. Cuando el proxy reciba desde el servidor de destino la correspondiente respuesta HTTP con el objeto, deberá crear una nueva respuesta HTTP que incluya el objeto y enviarla al cliente. Este proxy deberá ser multihebra, para poder ser capaz de gestionar múltiples solicitudes simultáneamente.

Para esta tarea, el sitio web de acompañamiento proporciona el esqueleto de código del servidor proxy. Su tarea consiste en completar el código y luego probarlo, haciendo que diferentes navegadores soliciten objetos web a través de su proxy.

## Prácticas de laboratorio con Wireshark: HTTP

---

En la práctica de laboratorio 1 nos hemos familiarizado con el husmeador de paquetes (*sniffer*) Wireshark, así que ya estamos preparados para utilizar Wireshark e investigar el funcionamiento de los protocolos. En esta práctica de laboratorio exploraremos varios aspectos del protocolo HTTP: la

interacción básica GET/respuesta, los formatos de los mensajes HTTP, la recuperación de archivos HTML de gran tamaño, la recuperación de archivos HTML con direcciones URL incrustadas, las conexiones persistentes y no persistentes, y la autenticación y la seguridad de HTTP.

Al igual que todas las prácticas de laboratorio con Wireshark, la descripción completa de esta práctica se encuentra en el sitio web del libro, [www.pearsonhighered.com/cs-resources](http://www.pearsonhighered.com/cs-resources).

## Prácticas de laboratorio con Wireshark: DNS

---

En esta práctica de laboratorio echaremos un rápido vistazo al lado del cliente de DNS, el protocolo que traduce los nombres de host Internet en direcciones IP. Como hemos visto en la Sección 2.5, el papel del cliente en el protocolo DNS es relativamente simple: un cliente envía una consulta a su servidor DNS local y recibe una respuesta. Sin embargo, son muchas las cosas que suceden por debajo, invisibles para los clientes DNS, ya que los servidores DNS jerárquicos se comunican entre sí de forma recursiva o iterativa para resolver la consulta DNS del cliente. No obstante, desde el punto de vista del cliente DNS, el protocolo es muy simple: se plantea una consulta al servidor DNS local y dicho servidor devuelve una respuesta. En esta práctica de laboratorio vamos a observar al protocolo DNS en acción.

Al igual que todas las prácticas de laboratorio con Wireshark, la descripción completa de esta práctica de laboratorio está disponible en el sitio web del libro, [www.pearsonhighered.com/cs-resources](http://www.pearsonhighered.com/cs-resources).