CAPÍTULO 1

# Redes de computadoras e Internet

Hoy día, Internet es casi indiscutiblemente el sistema de ingeniería más grande creado por la mano del hombre, con cientos de millones de computadoras conectadas, enlaces de comunicaciones y switches; con miles de millones de usuarios que se conectan a través de computadoras portátiles, tabletas y teléfonos inteligentes; y con una amplia variedad de nuevas "cosas" conectadas a Internet, incluyendo consolas de juegos, sistemas de vigilancia, relojes, gafas, termostatos, básculas y vehículos. Dado que Internet es una red tan enorme e incluye tantos componentes distintos y tiene tantos usos, ¿es posible tener la esperanza de comprender cómo funciona? ¿Existen unos principios y una estructura básicos que puedan proporcionar los fundamentos para comprender un sistema tan asombrosamente complejo y grande? Y, en caso afirmativo, ¿es posible que pueda resultar interesante y divertido aprender acerca de las redes de computadoras? Afortunadamente, la respuesta a todas estas preguntas es un rotundo ¡SÍ! De hecho, el objetivo de este libro es proporcionar al lector una introducción moderna al dinámico campo de las redes de computadoras, exponiendo los principios y los conocimientos prácticos que necesitará para entender no solo las redes actuales, sino también las del futuro.

En el primer capítulo se hace una amplia introducción al mundo de las redes de computadoras y de Internet. Nuestro objetivo es proporcionar una visión general y establecer el contexto para el resto del libro, con el fin de poder ver el bosque a través de los árboles. En este capítulo de introducción se abordan muchos de los fundamentos, así como muchos de los componentes que forman una red de computadoras, siempre sin perder de vista la panorámica general.

Vamos a estructurar esta introducción a las redes de computadoras de la siguiente forma: después de exponer algunos conceptos y términos fundamentales, examinaremos los componentes hardware y software esenciales que forman una red de computadoras. Comenzaremos por la frontera de la red y echaremos un vistazo a los sistemas terminales y aplicaciones que se ejecutan en la red. A continuación, exploraremos el núcleo de una red de computadoras, examinando los enlaces y los switches que transportan los datos, así como las redes de acceso y los medios físicos que conectan los sistemas terminales con el núcleo de la red. Aprenderemos que Internet es una red de redes y cómo estas redes se conectan entre sí.

Una vez completada la introducción sobre la frontera y el núcleo de una red de computadoras, en la segunda mitad del capítulo adoptaremos un punto de vista más amplio y abstracto. Examinaremos los retardos, las pérdidas y la tasa de transferencia de datos en una red de computadoras y proporcionaremos modelos cuantitativos simples para los retardos y tasas de transferencia de terminal a terminal: modelos que tienen en cuenta los retardos de transmisión, de propagación y de cola. A continuación, presentaremos algunos de los principios básicos sobre las arquitecturas de red: en concreto, las capas de protocolos y los modelos de servicios. También veremos que las redes son vulnerables a muchos tipos distintos de ataques; revisaremos algunos de estos ataques y veremos cómo es posible conseguir que las redes sean más seguras. Por último, concluiremos el capítulo con una breve historia de las redes de comunicaciones.

## 1.1 ¿Qué es Internet?

En este libro, vamos a emplear la red pública Internet, una red de computadoras específica, como nuestro principal vehículo para explicar las redes de computadoras y sus protocolos. Pero, ¿qué *es* Internet? Hay dos formas de responder a esta pregunta. La primera de ellas es describiendo las tuercas y tornillos que forman la red, es decir, los componentes hardware y software básicos que forman Internet. La segunda es describiéndola en términos de la infraestructura de red que proporciona servicios a aplicaciones distribuidas. Comenzaremos por la descripción de los componentes esenciales, utilizando la Figura 1.1 para ilustrar la exposición.

### 1.1.1 Descripción de los componentes esenciales

Internet es una red de computadoras que interconecta miles de millones de dispositivos informáticos a lo largo de todo el mundo. No hace demasiado tiempo, estos dispositivos eran fundamentalmente computadoras PC de escritorio tradicionales, estaciones de trabajo Linux y los llamados servidores, que almacenan y transmiten información tal como páginas web y mensajes de correo electrónico. Sin embargo, cada vez se conectan a Internet más "cosas" (dispositivos) no tradicionales, como computadoras portátiles, teléfonos inteligentes, tabletas, televisiones, consolas de juegos, termostatos, sistemas domésticos de alarma, electrodomésticos, relojes, gafas, vehículos, sistemas de control de tráfico y otros. De hecho, el término *red de computadoras* está comenzando a sonar algo obsoleto, a causa de la gran cantidad de dispositivos no tradicionales que se conectan a Internet. En la jerga de Internet, todos estos dispositivos se denominan **hosts** o **sistemas terminales**. Según algunas estimaciones, en 2015 había unos 5.000 millones de dispositivos conectados a Internet, y esa cifra alcanzará los 25.000 millones en 2020 [Gartner 2014]. Se estima que en 2015 había más de 3.200 millones de usuarios de Internet en todo el mundo, aproximadamente el 40% de la población mundial [ITU 2015].

Los sistemas terminales se conectan entre sí mediante una red de **enlaces de comunicaciones** y **conmutadores de paquetes**. En la Sección 1.2 veremos que existen muchos tipos de enlaces de comunicaciones, los cuales están compuestos por diferentes tipos de medios físicos, entre los que se incluyen el cable coaxial, el hilo de cobre, la fibra óptica y el espectro de radio. Los distintos enlaces pueden transmitir los datos a distintas velocidades y la **velocidad de transmisión** de un enlace se mide en bits/segundo. Cuando un sistema terminal tiene datos que enviar a otro sistema terminal, el emisor segmenta los datos y añade bytes de cabecera a cada segmento. Los paquetes de información resultantes, conocidos como **paquetes** en la jerga informática, se envían entonces a través de la red hasta el sistema terminal receptor, donde vuelven a ser ensamblados para obtener los datos originales.

Un conmutador de paquetes toma un paquete que llega a través de uno de sus enlaces de comunicaciones de entrada y lo reenvía a través de a uno de sus enlaces de comunicaciones de salida. Los conmutadores de paquetes se suministran en muchas formas y modelos, pero los dos

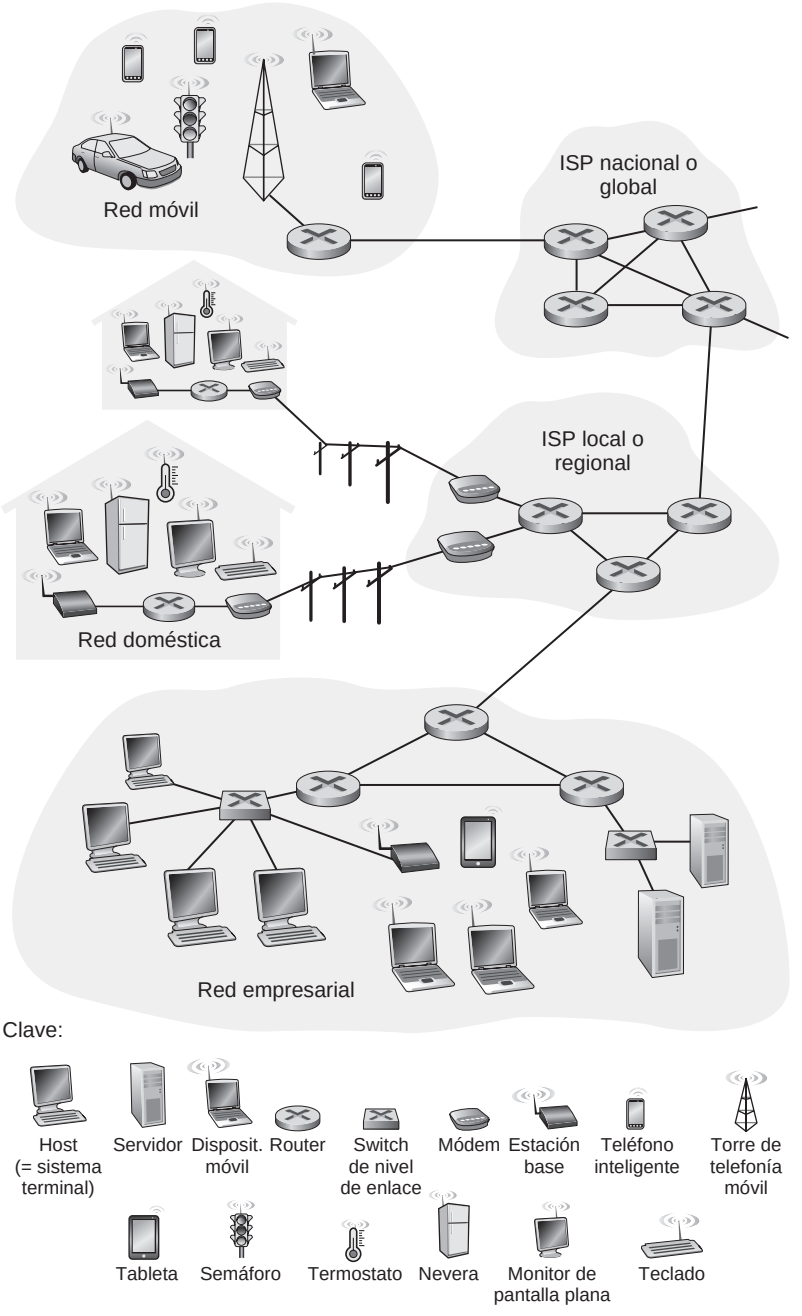



**Figura 1.1** ♦ Algunos de los componentes de Internet.

tipos más utilizados actualmente en Internet son los **routers** y los **switches de la capa de enlace**. Ambos tipos de conmutadores reenvían paquetes hacia sus destinos finales. Los switches de la capa de enlace normalmente se emplean en las redes de acceso, mientras que los routers suelen utilizarse en el núcleo de la red. La secuencia de enlaces de comunicaciones y conmutadores de paquetes que atraviesa un paquete desde el sistema terminal emisor hasta el sistema terminal receptor, se conoce con el nombre de **ruta** a través de la red. Según las predicciones de Cisco, el tráfico IP global anual sobrepasará el umbral del zettabyte ($10^{21}$ bytes) a finales de 2016 y alcanzará los 2 zettabytes por año en 2019 [Cisco VNI 2015].

Las redes de conmutación de paquetes (que transportan paquetes) son similares en muchos aspectos a las redes de transporte formadas por autopistas, carreteras e intersecciones (que transportan vehículos). Por ejemplo, imagine que una fábrica necesita trasladar un enorme cargamento a un cierto almacén de destino que se encuentra a miles de kilómetros. En la fábrica, el cargamento se reparte y se carga en una flota de camiones. Cada camión hace el viaje hasta el almacén de destino de forma independiente a través de la red de autopistas, carreteras e intersecciones. En el almacén de destino, la carga de cada camión se descarga y se agrupa con el resto del cargamento correspondiente al mismo envío. Así, en cierto sentido, los paquetes son como los camiones, los enlaces de comunicaciones como las autopistas y carreteras, los dispositivos de conmutación de paquetes como las intersecciones y los sistemas terminales son como los edificios (la fábrica y el almacén). Al igual que un camión sigue una ruta a través de la red de transporte por carretera, un paquete sigue una ruta a través de una red de computadoras.

Los sistemas terminales acceden a Internet a través de los **ISP (*Internet Service Provider*, Proveedor de servicios de Internet)**, incluyendo los ISP residenciales, como son las compañías telefónicas o de cable locales; los ISP corporativos; los ISP universitarios; los ISP que proporcionan acceso inalámbrico (WiFi) en aeropuertos, hoteles, cafeterías y otros lugares públicos; y los ISP de datos móviles, que proporcionan acceso móvil a nuestros teléfonos inteligentes y otros dispositivos. Cada ISP es en sí mismo una red de conmutadores de paquetes y enlaces de comunicaciones. Los ISP proporcionan una amplia variedad de tipos de acceso a red a los sistemas terminales, entre los que se incluyen el acceso de banda ancha residencial, mediante módem por cable o DSL; el acceso LAN (*Local Area Network*, Red de área local) de alta velocidad y el acceso inalámbrico para dispositivos móviles. Los ISP también proporcionan acceso a Internet a los proveedores de contenido, conectando sitios web y servidores de vídeo directamente a Internet. El objetivo de Internet no es otro que conectar los sistemas terminales entre sí, por lo que los ISP que proporcionan el acceso a los sistemas terminales también tienen que estar interconectados entre ellos. Estos ISP de nivel inferior se interconectan a través de ISP de nivel superior nacionales e internacionales, como Level 3 Communications, AT&T, Sprint y NTT. Un ISP de nivel superior está compuesto por routers de alta velocidad interconectados a través de enlaces de fibra óptica de alta velocidad. La red de cada ISP, sea de nivel inferior o superior, se administra de forma independiente, ejecuta el protocolo IP (véase más adelante) y se ajusta a determinados convenios de denominación y de asignación de direcciones. En la Sección 1.3 examinaremos más detalladamente los ISP y sus interconexiones.

Los sistemas terminales, los conmutadores de paquetes y otros dispositivos de Internet ejecutan **protocolos** que controlan el envío y la recepción de información dentro de Internet. El protocolo **TCP c*Transmission Control Protocol*, Protocolo de control de transmisión)** y el protocolo **IP** (***Internet Protocol*, Protocolo Internet)** son dos de los protocolos más importantes de Internet. El protocolo IP especifica el formato de los paquetes que se envían y reciben entre los routers y los sistemas terminales. Los principales protocolos de Internet se conocen colectivamente como protocolos **TCP/IP**. En este capítulo de introducción comenzaremos a estudiar los protocolos, pero esto solo es el principio, ya que gran parte del libro se dedica a los protocolos empleados por las redes de computadoras.

Debido a la importancia de los protocolos en Internet, es importante que todo el mundo esté de acuerdo en qué hacen todos y cada uno de ellos, para que la gente pueda crear sistemas y productos capaces de interoperar. Aquí es donde entran en juego los estándares. Los **estándares de Internet** son desarrollados por el IETF (Internet Engineering Task Force, Grupo de trabajo de ingeniería de Internet) [IETF 2016]. Los documentos asociados a estos estándares IETF se conocen como documentos **RFC** (***Request For Comments*, Solicitud de comentarios)**. Los RFC nacieron como solicitudes de comentarios de carácter general (de ahí su nombre) para solucionar los problemas de diseño de la red y de los protocolos a los que se enfrentó el precursor de Internet [Allman 2011]. El contenido de estos documentos suele ser bastante técnico y detallado. Definen protocolos tales como TCP, IP, HTTP (para la Web) y SMTP (para el correo electrónico). Actualmente, existen más de 7.000 documentos RFC. Existen también otros organismos dedicados a especificar estándares para componentes de red, especialmente para los enlaces de red. El comité de estándares IEEE 802 LAN/

MAN [IEEE 802 2016], por ejemplo, especifica los estándares para redes Ethernet y comunicación WiFi inalámbrica.

### 1.1.2 Descripción de los servicios

Hasta el momento hemos identificado muchos de los componentes que forman Internet, pero también podemos describir Internet desde un punto de vista completamente diferente, en concreto como *una infraestructura que proporciona servicios a las aplicaciones*. Además de aplicaciones tradicionales como el correo electrónico o la navegación web, las aplicaciones Internet abarcan aplicaciones para tabletas y teléfonos móviles inteligentes, incluyendo la mensajería Internet, mapas con información de tráfico en tiempo real, reproducción de música desde la nube, reproducción de películas y programas de televisión a través de Internet, redes sociales en línea, videoconferencia, juegos multipersona y sistemas de recomendación basados en la ubicación. Se dice que estas aplicaciones son **aplicaciones distribuidas**, porque implican a varios sistemas terminales que intercambian datos entre sí. Es importante saber que las aplicaciones de Internet se ejecutan en los sistemas terminales, no en los conmutadores de paquetes que forman el núcleo de la red. Aunque los dispositivos de conmutación de paquetes facilitan el intercambio de datos entre sistemas terminales, no se preocupan de la aplicación que esté actuando como origen o destino de los datos.

Vamos a ahondar un poco más en lo que queremos decir al hablar de una infraestructura que proporciona servicios a las aplicaciones. Para ello, supongamos que tenemos una idea nueva y excitante acerca de una aplicación distribuida de Internet, que puede beneficiar enormemente a la humanidad o que simplemente puede hacernos ricos y famosos. ¿Cómo podríamos transformar esa idea en una aplicación real de Internet? Puesto que las aplicaciones se ejecutan en los sistemas terminales, tendremos que escribir programas software que se ejecuten en dichos sistemas. Por ejemplo, podríamos escribir programas en Java, C o Python. Ahora bien, dado que estamos desarrollando una aplicación Internet distribuida, los programas que se ejecuten en los distintos sistemas terminales tendrán que enviarse datos entre sí. Y aquí es cuando llegamos al meollo de la cuestión, que nos lleva a la forma alternativa de describir Internet como una plataforma para aplicaciones. ¿Cómo hace un programa que se ejecuta en un sistema terminal para ordenar a Internet que entregue datos a otro programa que se ejecuta en otro sistema terminal?

Los sistemas terminales conectados a Internet proporcionan una **interfaz de sockets** que especifica cómo un programa software, que se ejecuta en un sistema terminal, pide a la infraestructura de Internet que suministre datos a un programa de destino específico que se está ejecutando en otro sistema terminal. Esta interfaz de sockets de Internet es un conjunto de reglas que el programa que transmite los datos debe cumplir, para que Internet pueda entregar esos datos al programa de destino. En el Capítulo 2 se aborda en detalle la interfaz de sockets de Internet. Por el momento, veamos una sencilla analogía, una que emplearemos con frecuencia a lo largo de este libro. Supongamos que Alicia desea enviar una carta a Benito utilizando el servicio postal. Por supuesto, Alicia no puede escribir la carta (los datos) y lanzarla por la ventana. En lugar de ello, el servicio postal exige que Alicia introduzca la carta en un sobre, escriba el nombre completo de Benito, su dirección y código postal en el sobre, lo cierre y pegue un sello en la esquina superior derecha del sobre. Por último, tendrá que introducir el sobre en un buzón oficial del servicio postal. Por tanto, el servicio postal de correos tiene su propia "interfaz de servicio postal", es decir, su propio conjunto de reglas que Alicia debe seguir para que el servicio postal entregue su carta a Benito. De forma similar, Internet tiene una interfaz de sockets que el programa que envía los datos debe seguir, para que Internet entregue los datos al programa que debe recibirlos.

Por supuesto, el servicio postal proporciona más de un servicio a sus clientes, como correo urgente, acuse de recibo, correo ordinario y otros muchos. Del mismo modo, Internet proporciona múltiples servicios a sus aplicaciones. Cuando desarrolle una aplicación de Internet, también tendrá que seleccionar para su aplicación uno de los servicios de Internet. En el Capítulo 2 describiremos esos servicios.

Acabamos de proporcionar dos descripciones de Internet: una en términos de sus componentes hardware y software, y otra como infraestructura que proporciona servicios a aplicaciones distribuidas. Pero es posible que el lector no tenga claro todavía qué es Internet. ¿Qué son la conmutación de paquetes y TCP/IP? ¿Qué son los routers? ¿Qué tipos de enlaces de comunicaciones existen en Internet? ¿Qué es una aplicación distribuida? ¿Cómo puede conectarse a Internet un termostato o una báscula? Si se siente un poco abrumado ahora por todas estas preguntas, no se preocupe: el propósito de este libro es presentarle tanto los componentes esenciales de Internet, como los principios que regulan cómo y por qué funciona. En las siguientes secciones y capítulos explicaremos todos estos términos y daremos respuesta a estas cuestiones.

## 1.1.3 ¿Qué es un protocolo?

Ahora que ya hemos visto por encima qué es Internet, vamos a ocuparnos de otro término importante en el mundo de las redes de computadoras: *protocolo*. ¿Qué es un protocolo? ¿Qué hace un protocolo?

### Analogía humana

Probablemente, sea más sencillo comprender el concepto de protocolo de red considerando en primer lugar algunas analogías humanas, ya que las personas utilizamos protocolos casi constantemente. Piense en lo que hace cuando necesita preguntar a alguien qué hora es. En la Figura 1.2 se muestra cómo se lleva a cabo un intercambio de este tipo. El protocolo entre personas (o las buenas maneras, al menos) dicta que para iniciar un proceso de comunicación con alguien lo primero es saludar (el primer "Hola" mostrado en la Figura 1.2). La respuesta típica a este saludo será también "Hola". Implícitamente, el saludo cordial de respuesta se toma como una indicación de que se puede continuar

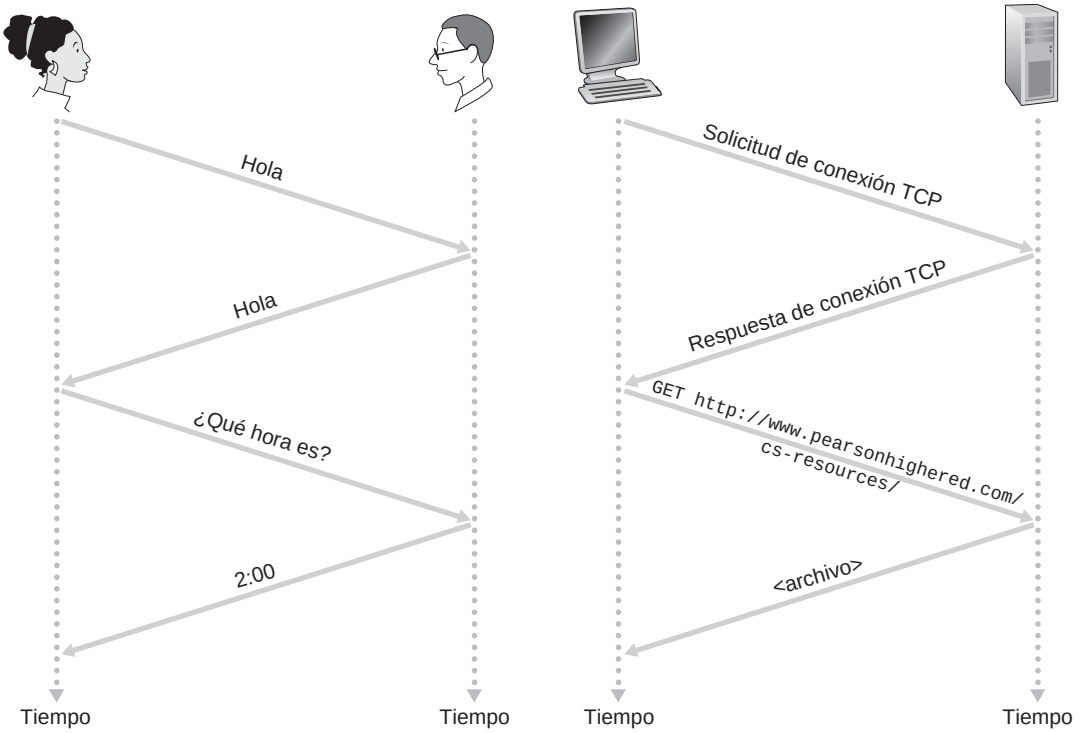



**Figura 1.2** ♦ Un protocolo humano y un protocolo de red.

con el proceso de comunicación y preguntar la hora. Una respuesta diferente al "Hola" inicial (como por ejemplo, "¡No me moleste!" o "No hablo su idioma", o cualquier respuesta impublicable), podría indicar una falta de disposición o una incapacidad para comunicarse. En este caso, el protocolo de las relaciones entre personas establece que no debe preguntarse la hora. En ocasiones, no se obtiene ninguna respuesta, en cuyo caso normalmente renunciamos a preguntar a esa persona la hora que es. Observe que, en el protocolo entre personas, *existen mensajes específicos que enviamos y acciones específicas que tomamos como respuesta a los mensajes de contestación recibidos o a otros sucesos* (como, por ejemplo, no recibir una respuesta en un periodo de tiempo determinado). Evidentemente, los mensajes transmitidos y recibidos, y las acciones tomadas al enviar o recibir estos mensajes o al producirse otros sucesos, desempeñan un papel principal en el protocolo humano. Si las personas adoptan protocolos diferentes (por ejemplo, si una persona guarda las formas pero la otra no lo hace, o si uno comprende el concepto de tiempo y el otro no), los protocolos no interoperarán y no podrá llevarse a cabo ninguna tarea útil. Esta misma idea también es aplicable a las redes: hacen falta dos (o más) entidades comunicándose y ejecutando el mismo protocolo para poder llevar a cabo una tarea.

Consideremos ahora una segunda analogía humana. Suponga que está asistiendo a una clase en la universidad (¡por ejemplo, sobre redes!). El profesor está hablando acerca de los protocolos y usted se siente confuso. El profesor detiene su explicación y dice: "¿Alguna pregunta?" (un mensaje que se transmite a todos los estudiantes que no estén dormidos, y que todos ellos reciben). Usted levanta la mano (transmitiendo un mensaje implícito al profesor). El profesor le dirige una sonrisa y le dice "¿Si . . .?" (un mensaje transmitido que le anima a plantear su pregunta, ya que los profesores *adoran* que les planteen cuestiones) y, a continuación, usted hace la pregunta (es decir, transmite su mensaje al profesor). El profesor escucha la pregunta (recibe su mensaje) y le responde (le transmite una respuesta). De nuevo, vemos que la transmisión y la recepción de mensajes y el conjunto de acciones convencionales tomadas cuando se envían y reciben estos mensajes, constituyen el núcleo de este protocolo de pregunta-respuesta.

### Protocolos de red

Un protocolo de red es similar a un protocolo humano, excepto en que las entidades que intercambian mensajes y llevan a cabo las acciones son los componentes hardware o software de cierto dispositivo (por ejemplo, una computadora, un teléfono inteligente, una tableta, un router u otro dispositivo de red). Cualquier actividad de Internet que implique dos o más entidades remotas que se comunican está gobernada por un protocolo. Por ejemplo, los protocolos implementados por hardware en las tarjetas de interfaz de red de dos computadoras conectadas físicamente controlan el flujo de bits a través del "cable" conectado entre las dos tarjetas de interfaz de red; los protocolos de control de congestión de los sistemas terminales controlan la velocidad a la que se transmiten los paquetes entre el emisor y el receptor; los protocolos de los routers determinan la ruta que seguirá un paquete desde el origen al destino. Los protocolos se ejecutan por todas partes en Internet y, en consecuencia, gran parte de este libro está dedicada a los protocolos de redes de computadoras.

Como ejemplo de un protocolo de red con el que probablemente estará familiarizado, vamos a ver lo que ocurre cuando se hace una solicitud a un servidor web, es decir, cuando usted escribe el URL de una página web en un navegador. Este escenario se ilustra en la mitad derecha de la Figura 1.2. En primer lugar, su computadora enviará un mensaje de solicitud de conexión al servidor web y esperará una respuesta. El servidor web recibirá su mensaje de solicitud de conexión y le devolverá un mensaje de respuesta de conexión. Sabiendo ahora que es posible solicitar el documento web, su computadora envía el nombre de la página web que desea extraer del servidor web, mediante un mensaje GET. Por último, el servidor web envía la página web (archivo) a su computadora.

Basándonos en los ejemplos anteriores de protocolos humanos y de red, el intercambio de mensajes y las acciones tomadas cuando se envían y reciben estos mensajes constituyen los elementos claves para la definición de un protocolo:

*Un **protocolo** define el formato y el orden de los mensajes intercambiados entre dos o más entidades que se comunican, así como las acciones tomadas al producirse la transmisión y/o recepción de un mensaje u otro suceso.*

Internet, y las redes de computadoras en general, hacen un uso extensivo de los protocolos. Se utilizan diferentes protocolos para llevar a cabo las distintas tareas de comunicación. A medida que avance en el libro, verá que algunos protocolos son simples y directos, mientras que otros son complejos e intelectualmente profundos. Dominar el campo de las redes de computadoras es equivalente a entender el qué, el por qué y el cómo de los protocolos de red.

## 1.2 La frontera de la red

En la sección anterior hemos presentado una introducción de carácter general sobre Internet y los protocolos de red. Ahora vamos a profundizar un poco más en los componentes de una red de computadoras (y de Internet, en concreto). Comenzaremos la sección en la frontera de una red y nos fijaremos en los componentes con los que estamos más familiarizados, es decir, las computadoras, los teléfonos inteligentes y otros dispositivos que utilizamos a diario. En la siguiente sección nos desplazaremos desde la frontera de la red hasta el núcleo de la misma y examinaremos los procesos de conmutación y enrutamiento que tienen lugar en las redes.

Recuerde de la sección anterior que en la jerga de las redes informáticas, las computadoras y el resto de los dispositivos conectados a Internet a menudo se designan como sistemas terminales, porque se sitúan en la frontera de Internet, como se muestra en la Figura 1.3. Entre los sistemas terminales de Internet se incluyen las computadoras de escritorio (por ejemplo, PCs de escritorio, computadoras Mac y equipos Linux), servidores (por ejemplo, servidores web y de correo electrónico) y dispositivos móviles (por ejemplo, computadoras portátiles, teléfonos inteligentes y tabletas). Además, cada vez se conectan más “cosas” no tradicionales a Internet como sistemas terminales (véase el recuadro Historia).

Los sistemas terminales también se conocen como *hosts* (huéspedes), ya que albergan (es decir, ejecutan) programas de aplicación tales como navegadores web, servidores web, programas cliente de correo electrónico o servidores de correo electrónico. A lo largo de este libro utilizaremos indistintamente los términos host y sistema terminal; es decir, *host = sistema terminal*. En ocasiones, los hosts se clasifican en dos categorías: **clientes** y **servidores**. De un modo informal, podríamos decir que los clientes suelen ser las computadoras de escritorio y portátiles, los teléfonos inteligentes, etc., mientras que los servidores suelen ser equipos más potentes que almacenan y distribuyen páginas web, flujos de vídeo, correo electrónico, etc. Hoy en día, la mayoría de los servidores desde los que recibimos resultados de búsqueda, correo electrónico, páginas web y vídeos, residen en grandes **centros de datos**. Por ejemplo, Google tiene entre 50 y 100 centros de datos, incluyendo unos 15 grandes centros, cada uno con más de 100.000 servidores.

### 1.2.1 Redes de acceso

Una vez vistas las aplicaciones y los sistemas terminales existentes en la “frontera de la red”, podemos pasar a ver las redes de acceso – la red que conecta físicamente un sistema terminal con el primer router (conocido también como “router de frontera”) de la ruta existente entre el sistema terminal y cualquier otro sistema terminal distante. La Figura 1.4 muestra varios tipos de redes de acceso, resaltadas mediante líneas gruesas y oscuras, junto con el entorno (doméstico, empresarial y acceso móvil inalámbrico de área extensa) en el que se utilizan.

#### Acceso doméstico: DSL, cable, FTTH, acceso telefónico y satélite

En 2014, más del 78% de las viviendas en los países desarrollados disponían de acceso a Internet, siendo los países líderes Corea, Países Bajos, Finlandia y Suecia, donde disponen de acceso a

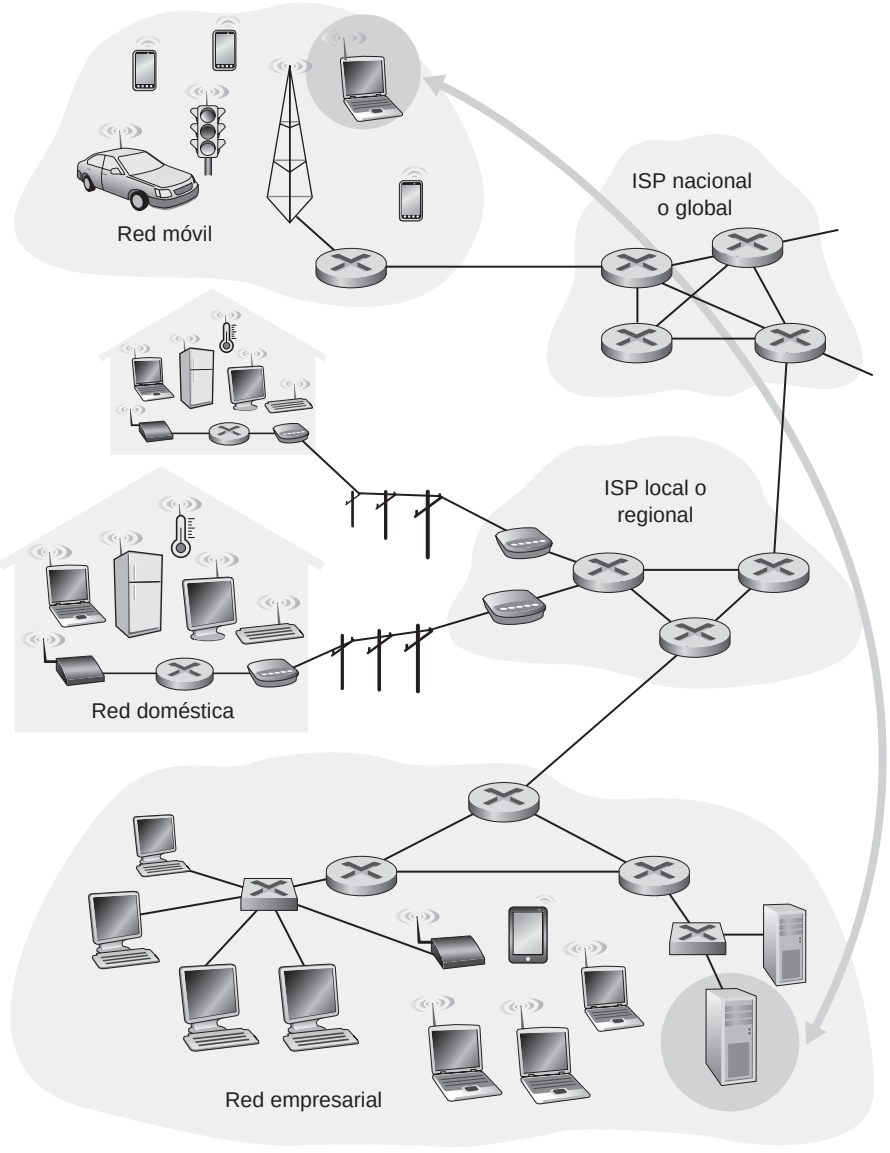



**Figura 1.3** ♦ Interacción de los sistemas terminales.

Internet más del 80% de las viviendas, casi todas ellas a través de conexión de banda ancha de alta velocidad [ITU 2015]. Dado este uso generalizado de redes de acceso domésticas, comencemos nuestro repaso de las redes de acceso analizando cómo se conectan las viviendas a Internet.

Hoy en día, los dos tipos de acceso residencial de banda ancha predominantes son las líneas **DSL (*Digital Subscriber Line,* Línea de abonado digital)** y el cable. Por regla general, los domicilios particulares contratan el servicio DSL de acceso a Internet con la misma empresa telefónica local (telco) que les proporciona el acceso telefónico local fijo. Por tanto, cuando se utiliza el acceso mediante DSL, la compañía telefónica del cliente también actúa como ISP. Como se muestra en la Figura 1.5, el módem DSL de cada cliente utiliza la línea telefónica existente (hilo de cobre de par trenzado, del que hablaremos en la Sección 1.2.2) para intercambiar datos con un multiplexor de acceso DSL (DSLAM), ubicado en la central de la compañía telefónica local. El módem DSL de la vivienda toma los datos digitales y los traduce a tonos de alta frecuencia para su transmisión a través

## HISTORIA

### LA INTERNET DE LAS COSAS

¿Se imagina un mundo en el que prácticamente todo esté conectado a Internet por vía inalámbrica? ¿Un mundo en el que la mayoría de las personas, vehículos, bicicletas, gafas, relojes, juguetes, equipos hospitalarios, sensores domóticos, aulas, sistemas de videovigilancia, sensores atmosféricos, productos expuestos en los comercios y mascotas estén conectados? Puede que este mundo de la Internet de las Cosas (IoT) esté ya a la vuelta de la esquina.

Según algunas estimaciones, en 2015 había ya 5.000 millones de cosas conectadas a Internet, y ese número podría alcanzar los 25.000 millones en 2020 [Gartner 2014]. Entre esas cosas se incluyen nuestros teléfonos inteligentes, que ya nos siguen a todas partes en nuestros domicilios, oficinas y vehículos, informando de nuestra geolocalización y nuestros datos de uso a nuestro ISP y a las aplicaciones de Internet. Pero además de nuestros teléfonos inteligentes, ya están disponibles como productos una amplia variedad de "cosas" no tradicionales. Por ejemplo, existen complementos conectados a Internet (*wearables*), incluyendo relojes (de Apple y muchos otros) y gafas. Las gafas conectadas a Internet pueden, por ejemplo, cargar en la nube todo lo que vemos, permitiéndonos compartir nuestras experiencias visuales en tiempo real con personas de todo el mundo. Ya hay disponibles cosas conectadas a Internet para viviendas inteligentes, incluyendo termostatos conectados a Internet que podemos controlar de manera remota desde nuestro teléfono inteligente; o básculas conectadas a Internet que nos permiten revisar gráficamente, desde nuestro teléfono inteligente, el progreso de nuestra dieta. Existen juguetes conectados a Internet, como por ejemplo muñecas que reconocen e interpretan las palabras de un niño y responden apropiadamente.

La Internet de las Cosas ofrece ventajas potencialmente revolucionarias a los usuarios. Pero al mismo tiempo, también existen enormes riesgos de seguridad y confidencialidad. Por ejemplo, los atacantes podrían ser capaces, vía Internet, de acceder a los dispositivos de la Internet de las Cosas, o a los servidores que recopilan datos de esos dispositivos. O por ejemplo, un atacante podría acceder a una muñeca conectada a Internet y hablar directamente con un niño. O podría entrar en una base de datos que almacene información personal sobre nuestra salud y nuestras actividades, recopilada a partir de los dispositivos que vestimos. Estos problemas de seguridad y confidencialidad podrían socavar la confianza del consumidor que se necesita para que las tecnologías desarrollen todo su potencial, y podrían hacer que la adopción de las mismas fuera menos generalizada [FTC 2015].

de los hilos telefónicos hasta la central; las señales analógicas de una gran cantidad de esas viviendas son vueltas a traducir a datos digitales en el DSLAM.

La línea telefónica residencial transporta simultáneamente los datos y las señales telefónicas tradicionales, las cuales se codifican a frecuencias distintas:

- Un canal de descarga de alta velocidad, en la banda de 50 kHz a 1 MHz.
- Un canal de carga de velocidad media, en la banda de 4 kHz a 50 kHz.
- Un canal telefónico ordinario bidireccional, en la banda de 0 kHz a 4 kHz.

Este método hace que el único enlace DSL existente se comporte como tres enlaces separados, de manera que una llamada de teléfono y una conexión a Internet pueden compartir el enlace DSL a un mismo tiempo. (En la Sección 1.3.1 describiremos esta técnica de multiplexación por división de frecuencia.) En el lado del cliente, las señales que llegan al domicilio son separadas en señales de datos y telefónicas mediante un circuito separador (splitter), que reenvía la señal de datos al módem DSL. En el lado de la compañía telefónica, en la central, el multiplexor DSLAM separa las señales de datos y de telefonía y envía los datos a Internet. Cientos o incluso miles de viviendas pueden estar conectadas a un mismo DSLAM [Dischinger 2007].

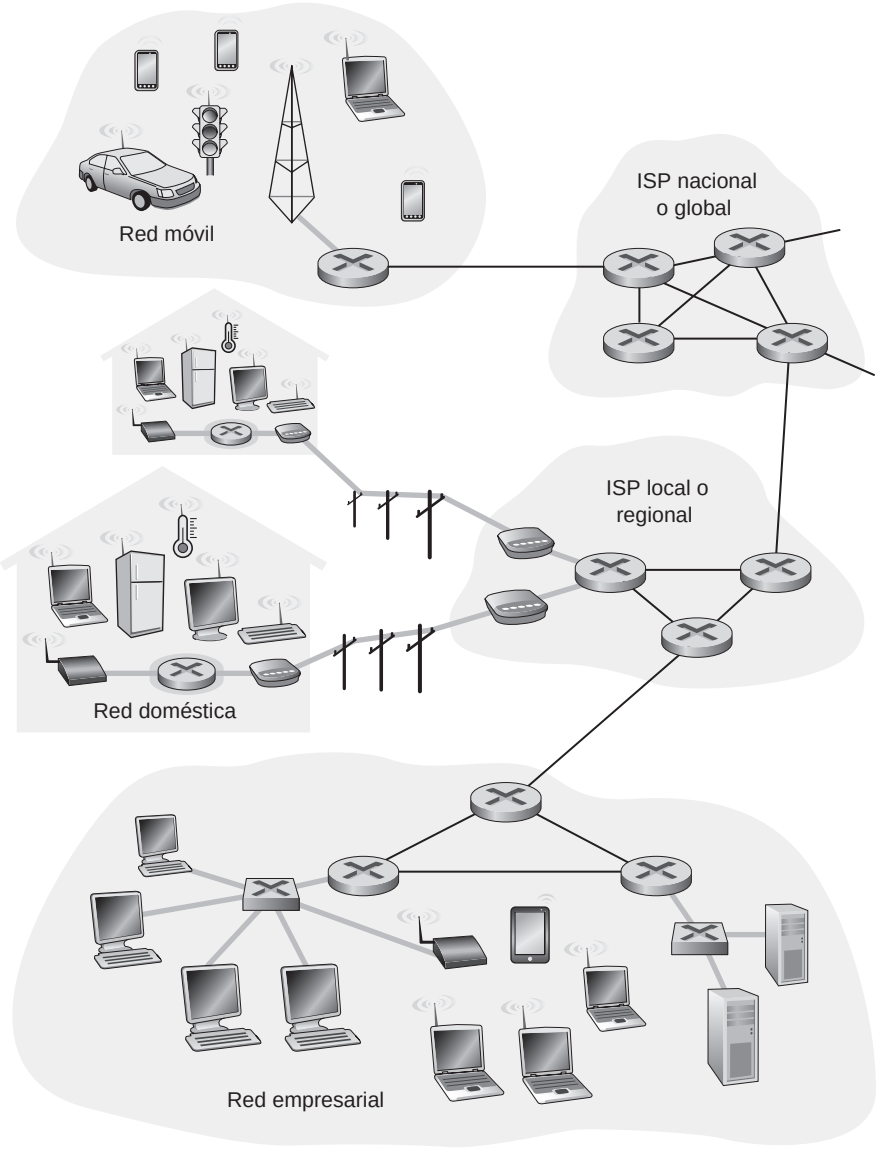



**Figura 1.4** ♦ Redes de acceso.

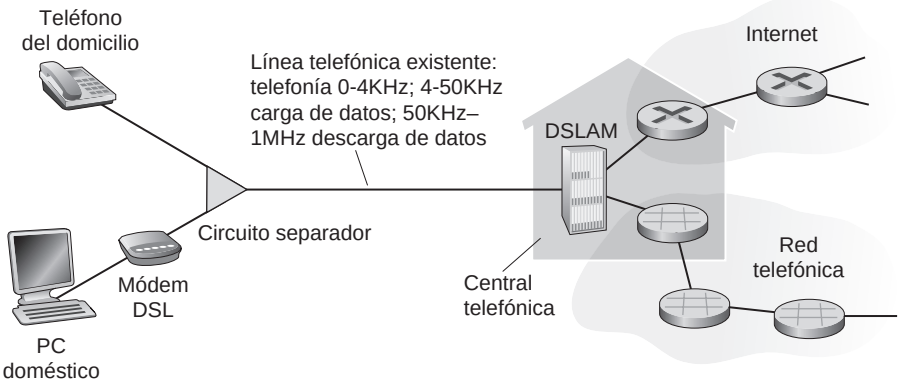



**Figura 1.5** ♦ Acceso a Internet mediante DSL.

Los estándares DSL definen múltiples velocidades de transmisión, incluyendo 12 Mbps de bajada (descarga) y 1,8 Mbps de subida (carga) [ITU 1999], y 55 Mbps de bajada y 15 Mbps de subida [ITU 2006]. Puesto que las velocidades de subida y de bajada son diferentes, se dice que este tipo de acceso es asimétrico. Las velocidades de transmisión reales de subida y de bajada que se logren obtener pueden ser inferiores a las anteriormente indicadas, ya que el proveedor del servicio DSL puede limitar deliberadamente la velocidad de una vivienda cuando ofrece un servicio por niveles (diferentes velocidades disponibles a diferentes precios). La velocidad máxima está limitada también por la distancia existente entre la vivienda y la central telefónica, por el calibre de la línea de par trenzado y por el grado de interferencia eléctrica. Los ingenieros han diseñado expresamente los sistemas DSL para distancias cortas entre el domicilio y la central; generalmente, si el domicilio no se encuentra en un radio de entre 8 y 16 kilómetros de la central, el usuario deberá recurrir a una forma alternativa de acceso a Internet.

Mientras que la tecnología DSL emplea la infraestructura local existente de la compañía telefónica, el **acceso por cable a Internet** utiliza la infraestructura de televisión por cable existente. Las viviendas obtienen el acceso por cable a Internet de la misma compañía que les proporciona la televisión por cable. Como se ilustra en la Figura 1.6, se usa fibra óptica para conectar el terminal de cabecera del cable a una serie de nodos de área situados en el vecindario, a partir de los cuales se utiliza cable coaxial tradicional para llegar a todos los domicilios. Cada nodo de área suele dar soporte a entre 500 y 5.000 viviendas. Puesto que en este sistema se emplea tanto cable coaxial como fibra, a menudo se denomina sistema HFC (*Hybrid Fiber Coax*, Híbrido de fibra y coaxial).

El acceso por cable a Internet requiere el uso de un módem especial, que se conoce como módem por cable. Al igual que un módem DSL, normalmente el módem por cable es un dispositivo externo que se conecta a un PC de la vivienda a través de un puerto Ethernet (en el Capítulo 6 veremos más detalles acerca de Ethernet). En el terminal de cabecera del cable, un sistema CMTS (*Cable Modem Termination System*, Sistema de terminación del módem de cable) cumple una función similar al DSLAM de la red DSL —transformar a formato digital la señal analógica enviada por los modems de cable de numerosas viviendas situadas aguas abajo. Los modems por cable dividen la red HFC en dos canales: un canal de descarga y un canal de carga. Al igual que en el caso de DSL, el acceso suele ser asimétrico, teniendo el canal de descarga asignada normalmente una velocidad de transmisión mayor que el canal de carga. El estándar DOCSIS 2.0 define velocidades de descarga de hasta 42,8 Mbps y velocidades de carga de hasta 30,7 Mbps. Como en el caso de las redes DSL, la velocidad máxima alcanzable puede no llegar a obtenerse, debido a las deficiencias del medio o a que las velocidades de datos contratadas sean más bajas.

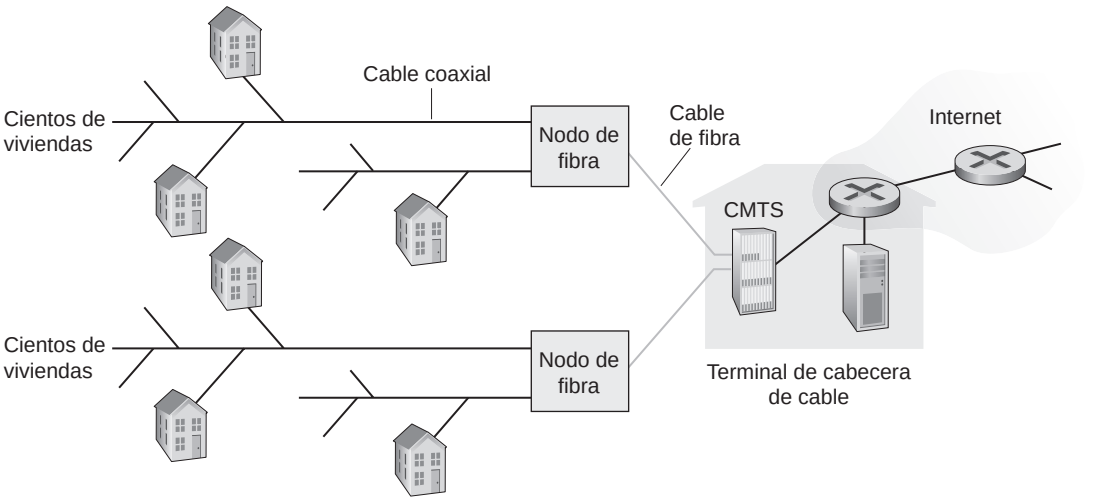



**Figura 1.6** ♦ Red de acceso híbrida de fibra óptica y cable coaxial.

Una característica importante del acceso a Internet por cable es que se trata de un medio de difusión compartido. Es decir, cada uno de los paquetes enviados por el extremo de cabecera viaja a través de cada enlace hasta cada vivienda y los paquetes enviados desde las viviendas viajan a través del canal de subida hasta el extremo de cabecera. Así, si varios usuarios descargan simultáneamente un archivo de vídeo a través del canal de descarga, la velocidad real a la que cada usuario recibe su archivo de vídeo será significativamente menor que la velocidad agregada de descarga por cable. Por el contrario, si solo hay unos pocos usuarios activos que están navegando por la Web, cada uno de ellos recibirá las páginas web a la velocidad de descarga máxima del cable, ya que los usuarios rara vez solicitarán sus páginas web exactamente al mismo tiempo. Puesto que el canal de subida también es compartido, se necesita un protocolo distribuido de acceso múltiple para coordinar las transmisiones y evitar las colisiones (veremos este problema de las colisiones con más detalle en el Capítulo 6).

Aunque las redes DSL y de cable representan en la actualidad más del 85% del acceso de banda ancha residencial en Estados Unidos, una tecnología emergente que proporciona velocidades aún más altas es la tecnología **FTTH (*Fiber-To-The-Home,* Fibra hasta el hogar)** [FTTH Council 2016]. Como su nombre sugiere, el concepto en que se basa FTTH es muy simple: proporcionar una ruta de fibra óptica directa hasta la vivienda desde la central telefónica. Muchos países —incluyendo los Emiratos Árabes Unidos, Corea del Sur, Hong Kong, Japón, Singapur, Taiwan, Lituania y Suecia— tienen hoy tasas de penetración en el mercado residencial superiores al 30% [FTTH Council 2016].

Existen varias tecnologías que compiten por la distribución a través de fibra óptica desde las centrales a los hogares. La red de distribución óptica más simple se denomina fibra directa, en la que existe una fibra que sale de la central hasta cada domicilio. Sin embargo, lo más habitual es que cada fibra saliente de la central sea compartida por muchas viviendas y esta no se divida en fibras individuales específicas del cliente hasta llegar a un punto muy próximo a las viviendas. Hay disponibles dos arquitecturas de distribución de fibra óptica que llevan a cabo esta separación: las redes ópticas activas (AON, *Active Optical Network*) y las redes ópticas pasivas (PON, *Passive Optical Network*). Las redes AON son fundamentalmente redes Ethernet conmutadas, de las cuales hablaremos en el Capítulo 6.

Aquí vamos a ver brevemente las redes ópticas pasivas, que se utilizan en el servicio FIOS de Verizon. La Figura 1.7 muestra un sistema FTTH que utiliza la arquitectura de distribución PON. Cada vivienda dispone de una terminación de red óptica (ONT, *Optical Network Terminator*), que se conecta a un distribuidor del vecindario mediante un cable de fibra óptica dedicado. El distribuidor combina una cierta cantidad de viviendas (normalmente menos de 100) en un único cable de fibra óptica compartido, que se conecta a una terminación de línea óptica (OLT, *Optical Line Terminator*) en la central de la compañía telefónica. La OLT, que realiza la conversión de señales ópticas en

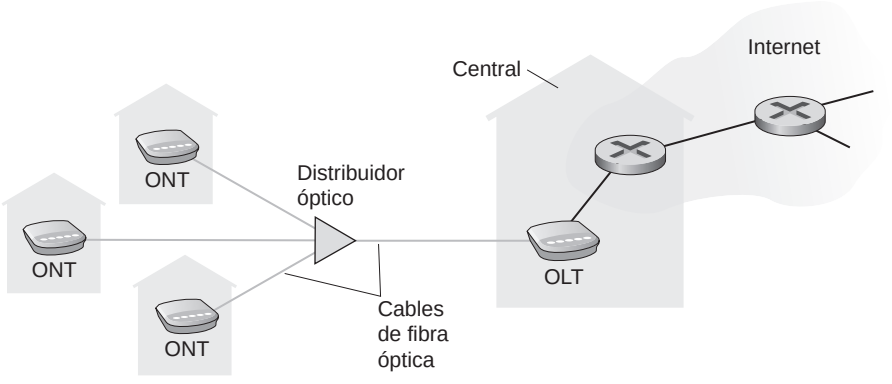



**Figura 1.7** ♦ Acceso a Internet mediante FTTH.

eléctricas, se conecta a Internet a través de un router de la compañía telefónica. En los domicilios, los usuarios conectan su router doméstico (normalmente un router inalámbrico) con la ONT y acceden a Internet a través de este router. En la arquitectura PON, todos los paquetes enviados desde la OLT al distribuidor se replican en este distribuidor (de forma similar a un terminal de cabecera de cable).

En teoría, la tecnología FTTH puede proporcionar velocidades de acceso a Internet del orden de los gigabits por segundo. Sin embargo, la mayoría de los ISP de FTTH ofrecen diferentes velocidades, siendo lógicamente las tarifas más caras cuanto mayor es la velocidad. La velocidad media de descarga para los clientes de FTTH en Estados Unidos era de aproximadamente 20 Mbps en 2011 (comparada con los 13 Mbps de las redes de acceso por cable y los menos de 5 Mbps de DSL) [FTTH Council 2011b].

Hay otras dos tecnologías de redes de acceso que también se usan para proporcionar acceso a Internet a las viviendas. En aquellos lugares donde no estén disponibles las tecnologías DSL, FTTH y de cable (por ejemplo, en ciertos entornos rurales), puede utilizarse un enlace vía satélite para conectar a Internet un domicilio a velocidades superiores a 1 Mbps; dos de los proveedores de acceso vía satélite son StarBand y HughesNet. El acceso telefónico a través de líneas telefónicas tradicionales se basa en el mismo modelo que DSL – un módem doméstico se conecta a través de la línea telefónica a un módem situado en las instalaciones del ISP. Comparado con DSL y otras redes de acceso de banda ancha, el acceso telefónico es insoportablemente lento, de solo 56 kbps.

### Acceso empresarial (y doméstico): Ethernet y WiFi

En los campus universitarios y corporativos, y cada vez más en entornos domésticos, se utiliza una red de área local (LAN, *Local Area Network*) para conectar un sistema terminal al router de frontera. Aunque existen muchos tipos de tecnologías LAN, Ethernet es con mucho la tecnología de acceso predominante en las redes corporativas, universitarias y domésticas. Como se ilustra en la Figura 1.8, los usuarios de Ethernet utilizan cable de cobre de par trenzado para conectarse a un switch Ethernet, una tecnología que se verá en detalle en el Capítulo 6. A su vez, el switch Ethernet (o una red de tales switches) se conecta a Internet. Con acceso Ethernet, normalmente los usuarios disponen de velocidades de acceso de 100 Mbps o 1 Gbps al switch Ethernet, mientras que los servidores pueden disponer de acceso de 1 Gbps o incluso 10 Gbps.

Sin embargo, cada vez es más habitual que los usuarios accedan a Internet a través de conexiones inalámbricas desde computadoras portátiles, teléfonos inteligentes, tabletas y otras "cosas" (véase el recuadro anterior "La Internet de las cosas"). En un entorno de LAN inalámbrica, los usuarios inalámbricos transmiten/reciben paquetes hacia/desde un punto de acceso que está conectado a la red empresarial (probablemente utilizando Ethernet cableada), que a su vez se conecta a la red Internet cableada. Habitualmente, los usuarios de una LAN inalámbrica deben encontrarse a unas

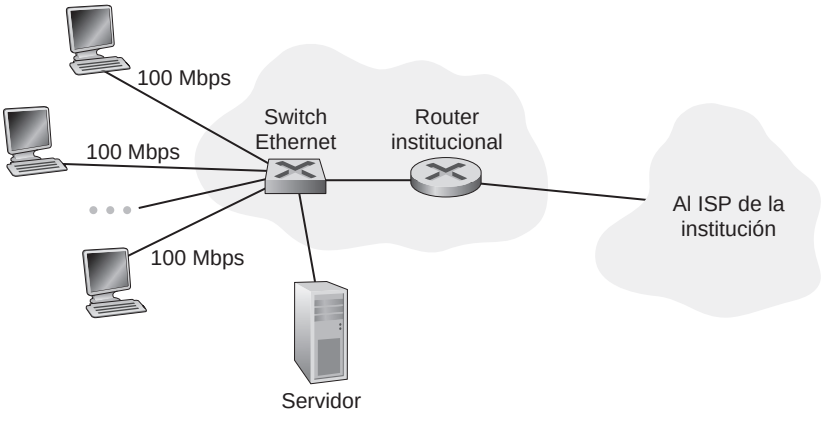



**Figura 1.8** ♦ Acceso a Internet utilizando tecnología Ethernet.

pocas decenas de metros del punto de acceso. Actualmente, el acceso mediante LAN inalámbrica basada en la tecnología IEEE 802.11, más coloquialmente conocido como WiFi, podemos encontrarlo por todas partes: universidades, oficinas, cafés, aeropuertos, domicilios e incluso en los aviones. En muchas ciudades, alguien puede estar parado en la esquina de una calle y encontrarse dentro del alcance de diez o veinte estaciones base (para ver un mapa global navegable de estaciones base 802.11 descubiertas y registradas en un sitio web por personas que disfrutan haciendo este tipo de cosas, consulte [wigle.net 2009]). Como se explica detalladamente en el Capítulo 7, hoy en día la tecnología 802.11 proporciona una velocidad de transmisión compartida de hasta 100 Mbps o superior.

Aunque las redes de acceso Ethernet y WiFi se implantaron inicialmente en entornos empresariales (corporativos, universitarios), en los últimos tiempos se han convertido en componentes relativamente comunes de las redes domésticas. Muchas viviendas combinan acceso residencial de banda ancha (es decir, modems por cable o DSL) con estas baratas tecnologías de LAN inalámbrica para crear redes domésticas potentes [Edwards 2011]. La Figura 1.9 muestra el esquema de una red doméstica típica. Esta red doméstica está formada por un portátil con función de itinerancia (*roaming*) y un PC de sobremesa; una estación base (el punto de acceso inalámbrico), que se comunica con el portátil y otros dispositivos inalámbricos de la vivienda; un módem por cable, que proporciona el acceso de banda ancha a Internet; y un router, que interconecta la estación base y el PC de sobremesa con el módem por cable. Esta red permite a los miembros de la familia disponer de acceso de banda ancha a Internet, pudiendo uno de ellos acceder mientras se mueve de la cocina a los dormitorios y al jardín.

### Acceso inalámbrico de área extensa: 3G y LTE

Cada vez se utilizan más dispositivos como los iPhone o los basados en Android para enviar mensajes, compartir fotos en redes sociales, ver películas y oír música a través de la red mientras nos desplazamos de un sitio a otro. Estos dispositivos emplean la misma infraestructura inalámbrica de la telefonía móvil para enviar/recibir paquetes a través de una estación base operada por el proveedor de telefonía móvil. A diferencia de WiFi, un usuario solo necesita estar a unas pocas decenas de kilómetros (en vez de unas pocas decenas de metros) de la estación base.

Las empresas de telecomunicaciones han hecho grandes inversiones en lo que se conoce como redes inalámbricas de tercera generación (3G), que proporcionan acceso inalámbrico a Internet mediante una red de área extensa de conmutación de paquetes, a velocidades por encima de 1 Mbps. Pero ya se están implantando tecnologías de acceso de área extensa con velocidades aun más altas: una cuarta generación (4G) de redes inalámbricas de área extensa. LTE (*Long-Term Evolution*, Evolución a largo plazo, un claro candidato al Premio Anual al Peor Acrónimo del Año) tiene sus raíces en la tecnología 3G y puede conseguir velocidades superiores a 10 Mbps. En las redes LTE comerciales se han medido velocidades de descarga de varias decenas de Mbps. En el Capítulo 7 hablaremos de los principios básicos de las redes inalámbricas y la movilidad, así como de las tecnologías WiFi, 3G y LTE (¡y otras cosas!).

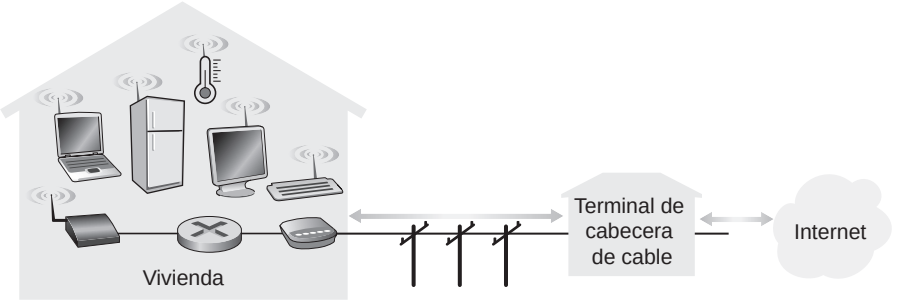



**Figura 1.9** ♦ Una red doméstica típica.

## 1.2.2 Medios físicos

En la subsección anterior hemos proporcionado una panorámica de algunas de las tecnologías de acceso a red más importantes disponibles para Internet. Según hemos ido describiendo estas tecnologías, hemos indicado los medios físicos utilizados. Por ejemplo, hemos dicho que la tecnología HFC emplea una combinación de cable de fibra óptica y de cable coaxial. También hemos señalado que Ethernet y DSL utilizan cable de cobre. Y dijimos que las redes para acceso móvil usan el espectro de radio. En esta subsección vamos a hacer una breve introducción a estos y otros medios de transmisión que se emplean habitualmente en Internet.

Para definir lo que se entiende por medio físico, reflexionemos sobre la breve vida de un bit. Imagine un bit que viaja desde un sistema terminal atravesando una serie de enlaces y routers hasta otro sistema terminal. ¡Este pobre bit se desplaza de un lado a otro y es transmitido un montón de veces! En primer lugar, el sistema terminal de origen transmite el bit y poco tiempo después el primer router de la serie recibe dicho bit; el primer router transmite entonces el bit y poco después lo recibe el segundo router, y así sucesivamente. Por tanto, nuestro bit, al viajar desde el origen hasta el destino, atraviesa una serie de parejas de transmisores y receptores. En cada par transmisor-receptor, el bit se envía mediante ondas electromagnéticas o pulsos ópticos a lo largo de un **medio físico**. Este medio físico puede tener muchas formas y no tiene por qué ser del mismo tipo para cada par transmisor-receptor existente a lo largo de la ruta. Entre los ejemplos de medios físicos se incluyen el cable de cobre de par trenzado, el cable coaxial, el cable de fibra óptica multimodo, el espectro de radio terrestre y el espectro de radio por satélite. Los medios físicos se pueden clasificar dentro de dos categorías: **medios guiados** y **medios no guiados**. En los medios guiados, las ondas se canalizan a través de un medio sólido, como por ejemplo un cable de fibra óptica, un cable de cobre de par trenzado o un cable coaxial. En los medios no guiados, las ondas se propagan por la atmósfera y el espacio exterior, tal como ocurre en las redes LAN inalámbricas o en un canal de satélite digital.

Pero antes de abordar las características de los distintos tipos de medios, veamos algunos detalles acerca de los costes. El coste real de un enlace físico (cable de cobre, de fibra óptica, etc.) suele ser relativamente pequeño cuando se compara con los restantes costes de la red. En particular, el coste de mano de obra asociado con la instalación del enlace físico puede ser varios órdenes de magnitud mayor que el coste del material. Por ello, muchos constructores instalan cables de par trenzado, de fibra óptica y coaxiales en todas las habitaciones de los edificios. Incluso aunque inicialmente solo se utilice uno de los medios, existen muchas posibilidades de que se emplee algún otro medio físico en un futuro próximo y, por tanto, se ahorre dinero al no tener que tirar cables adicionales.

### Cable de cobre de par trenzado

El medio de transmisión guiado más barato y más comúnmente utilizado es el cable de cobre de par trenzado. Se ha utilizado durante más de un siglo en las redes telefónicas. De hecho, más del 99 por ciento de las conexiones cableadas utilizan cable de cobre de par trenzado entre el propio teléfono y el conmutador telefónico local. La mayoría de nosotros hemos visto cable de par trenzado en nuestros hogares (¡o en los de nuestros padres o abuelos!) y entornos de trabajo. Este cable consta de dos hilos de cobre aislados, de un milímetro de espesor cada uno de ellos, que siguen un patrón regular en espiral. Los hilos se trenzan para reducir las interferencias eléctricas procedentes de pares similares próximos. Normalmente, se meten una serie de pares dentro de un mismo cable, envolviendo todos los pares en una pantalla protectora. Cada par de hilos constituye un único enlace de comunicaciones. El **par trenzado no apantallado (UTP, *Unshielded Twisted Pair*)** se utiliza habitualmente para redes de computadoras confinadas dentro de un edificio, es decir, para redes LAN. La velocidad de transmisión de datos de las LAN actuales que emplean cables de par trenzado varía entre 10 Mbps y 10 Gbps. Las velocidades de transmisión de datos que se pueden alcanzar dependen del espesor del cable y de la distancia existente entre el transmisor y el receptor.

Cuando surgió la tecnología de la fibra óptica en la década de 1980, muchas personas despreciaron el cable de par trenzado a causa de sus relativamente bajas velocidades de transmisión. Algunos pensaron incluso que la fibra óptica desplazaría por completo al cable de par trenzado.

Pero el cable de par trenzado no se dio por vencido tan fácilmente. La tecnología moderna de par trenzado, como por ejemplo los cables de categoría 6a, pueden alcanzar velocidades de datos de 10 Gbps para distancias de hasta 100 metros. Al final, los cables de par trenzado se han establecido como la solución dominante para las redes LAN de alta velocidad.

Como hemos mencionado anteriormente, los cables de par trenzado también suelen utilizarse para el acceso a Internet de tipo residencial. Hemos dicho que los modems de acceso telefónico permiten proporcionar acceso a velocidades de hasta 56 kbps utilizando cables de par trenzado. También hemos comentado que la tecnología DSL (Digital Subscriber Line) ha permitido a los usuarios residenciales acceder a Internet a velocidades de decenas de Mbps empleando cables de par trenzado (siempre y cuando los usuarios vivan en las proximidades de la central de comunicaciones del ISP).

### Cable coaxial

Al igual que el par trenzado, el cable coaxial consta de dos conductores de cobre, pero dispuestos de forma concéntrica, en lugar de en paralelo. Con esta construcción y un aislamiento y apantallamiento especiales, el cable coaxial puede proporcionar altas velocidades de transmisión de datos. El cable coaxial es bastante común en los sistemas de televisión por cable. Como ya hemos visto, recientemente los sistemas de televisión por cable han comenzado a incorporar modems por cable con el fin de proporcionar a los usuarios residenciales acceso a Internet a velocidades de decenas de Mbps. En la televisión por cable y en el acceso a Internet por cable, el transmisor desplaza la señal digital a una banda de frecuencia específica y la señal analógica resultante se envía desde el transmisor a uno o más receptores. El cable coaxial puede utilizarse como un **medio compartido** guiado; es decir, una serie de sistemas terminales pueden estar conectados directamente al cable, recibiendo todos ellos lo que envíen los otros sistemas terminales.

### Fibra óptica

La fibra óptica es un medio flexible y de muy pequeño espesor que conduce pulsos de luz, representando cada pulso un bit. Un único cable de fibra óptica puede soportar velocidades de bit tremendamente altas, de hasta decenas o incluso centenares de gigabits por segundo. La fibra óptica es inmune a las interferencias electromagnéticas, presenta una atenuación de la señal muy baja para distancias de hasta 100 kilómetros y es muy difícil que alguien pueda llevar a cabo un "pinchazo" en una de estas líneas. Estas características hacen de la fibra óptica el medio de transmisión guiado a larga distancia preferido, especialmente para los enlaces transoceánicos. Muchas de las redes telefónicas para larga distancia de Estados Unidos y otros países utilizan hoy día exclusivamente fibra óptica. La fibra óptica también es el medio predominante en las redes troncales de Internet. Sin embargo, el alto coste de los dispositivos ópticos —como transmisores, receptores y switches— está entorpeciendo su implantación para el transporte a corta distancia, como por ejemplo en el caso de una LAN o en el domicilio en una red de acceso residencial. Las velocidades de enlace estándar para portadora óptica (OC, *Optical Carrier*) van desde 51,8 Mbps a 39,8 Gbps; suele hacerse referencia a estas especificaciones mediante la designación OC-*n*, donde la velocidad del enlace es igual a $n \times$ 51,8 Mbps. Entre los estándares en uso actuales se encuentran: OC-1, OC-3, OC-12, OC-24, OC-48, OC-96, OC-192, OC-768. Consulte [Mukherjee 2006, Ramaswami 2010] para ver más información acerca de diversos aspectos de las redes ópticas.

### Canales de radio terrestres

Los canales de radio transportan señales dentro del espectro electromagnético. Constituyen un medio atractivo, porque no requieren la instalación de cables físicos, pueden atravesar las paredes, proporcionan conectividad a los usuarios móviles y potencialmente pueden transportar una señal a grandes distancias. Las características de un canal de radio dependen de forma significativa

del entorno de propagación y de la distancia a la que la señal tenga que ser transportada. Las consideraciones ambientales determinan la pérdida del camino y la atenuación por sombra (que disminuyen la intensidad de la señal a medida que recorre una distancia y rodea/atraviesa los objetos que obstruyen su camino), la atenuación multicamino (debido a la reflexión de la señal en los objetos que interfieren) y las interferencias (debidas a otras transmisiones y a señales electromagnéticas en general).

Los canales de radio terrestre pueden clasificarse en tres amplios grupos: aquéllos que operan a distancia muy corta (por ejemplo, de uno o dos metros), los que operan en áreas locales, normalmente con un alcance de entre diez y unos cientos de metros y los que operan en un área extensa, con alcances de decenas de kilómetros. Los dispositivos personales, como los cascos inalámbricos, teclados y ciertos dispositivos médicos, operan a distancias cortas; las tecnologías LAN inalámbricas descritas en la Sección 1.2.1 emplean canales de radio de área local y las tecnologías de acceso celulares utilizan canales de radio de área extensa. En el Capítulo 7 se estudian en detalle los canales de radio.

#### Canales de radio vía satélite

Un satélite de comunicaciones enlaza dos o más transmisores/receptores de microondas situados en la superficie, que se conocen como estaciones terrestres. El satélite recibe las transmisiones en una banda de frecuencia, regenera la señal utilizando un repetidor (véase más adelante) y transmite la señal a otra frecuencia. En este tipo de comunicaciones se emplean dos tipos de satélites: los **satélites geoestacionarios** y los **satélites de la órbita baja terrestre (LEO, *Low-Earth Orbiting*)** [Wiki Satellite 2016].

Los satélites geoestacionarios están permanentemente situados sobre el mismo punto de la Tierra. Esta presencia estacionaria se consigue poniendo el satélite en órbita a una distancia de 36.000 kilómetros por encima de la superficie terrestre. Esta enorme distancia de ida y vuelta entre la estación terrestre y el satélite introduce un significativo retardo de propagación de la señal de 280 milisegundos. No obstante, los enlaces vía satélite, que pueden operar a velocidades de cientos de Mbps, a menudo se emplean en áreas en las que no hay disponible acceso a Internet mediante DSL o cable.

Los satélites LEO se colocan mucho más cerca de la Tierra y no se encuentran permanentemente sobre un mismo punto de la superficie, sino que giran alrededor de la Tierra (al igual que lo hace la Luna) y pueden comunicarse entre sí, así como con las estaciones terrestres. Para poder proporcionar una cobertura continua a un área, es preciso poner en órbita muchos satélites. Actualmente se están desarrollando muchos sistemas de comunicaciones de baja altitud. La tecnología de satélites LEO podrá utilizarse, en algún momento del futuro, para acceder a Internet.

## 1.3 El núcleo de la red

Una vez que hemos examinado la frontera de Internet, vamos a adentrarnos en el núcleo de la red, la malla de conmutadores de paquetes y enlaces que interconectan los sistemas terminales de Internet. En la Figura 1.10 se ha resaltado el núcleo de la red con líneas más gruesas.

### 1.3.1 Conmutación de paquetes

En una aplicación de red, los sistemas terminales intercambian **mensajes** unos con otros. Los mensajes pueden contener cualquier cosa que el diseñador de la aplicación desee. Los mensajes pueden realizar una función de control (por ejemplo, los mensajes de saludo “Hola” del ejemplo anterior sobre establecimiento de la comunicación, en la Figura 1.2) o pueden contener datos, como por ejemplo un mensaje de correo electrónico, una imagen JPEG o un archivo de audio MP3. Para enviar un mensaje desde un sistema terminal de origen hasta un sistema terminal de destino, el

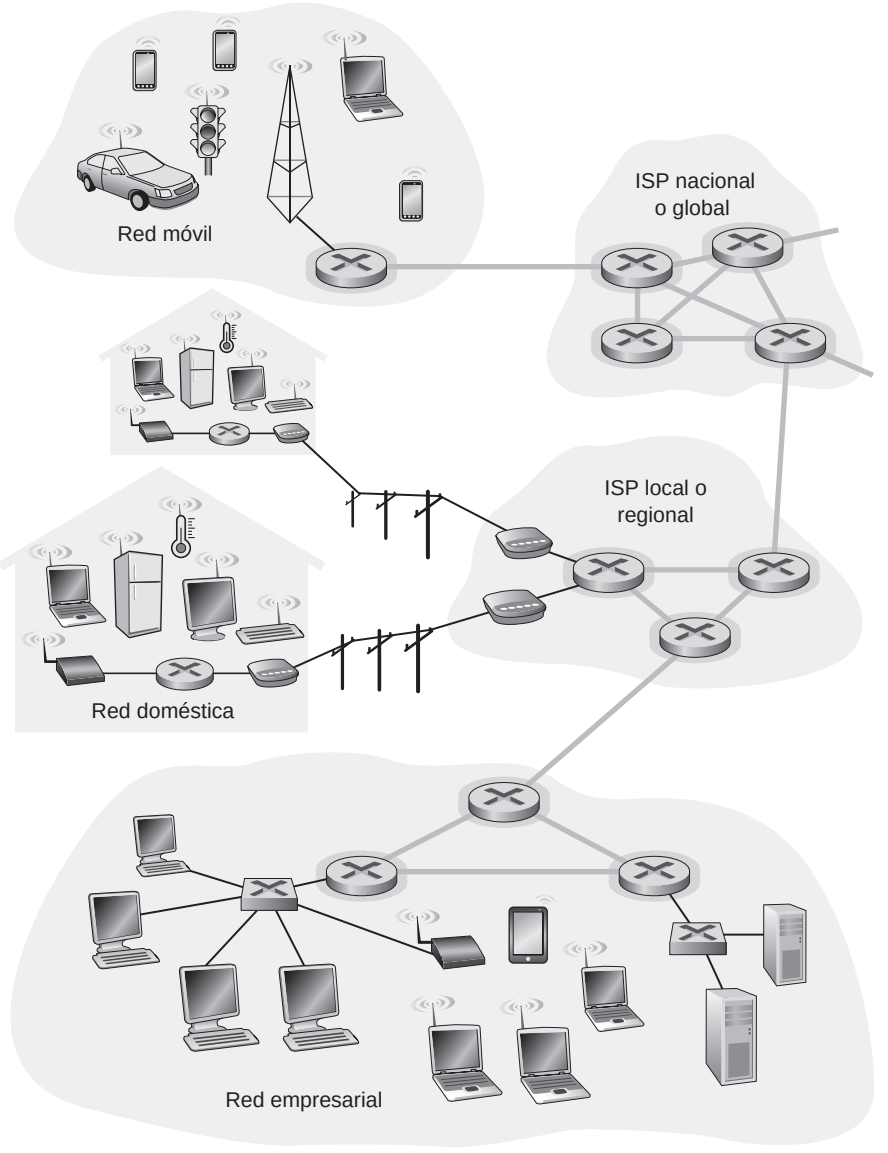



**Figura 1.10** ♦ El núcleo de la red.

origen divide los mensajes largos en fragmentos de datos más pequeños que se conocen como **paquetes**. Entre el origen y el destino, cada uno de estos paquetes viaja a través de enlaces de comunicaciones y de **conmutadores de paquetes** (de los que existen dos tipos predominantes: los **routers** y los **switches de la capa de enlace**). Los paquetes se transmiten a través de cada enlace de comunicaciones a una velocidad igual a la velocidad de transmisión *máxima* del enlace. Por tanto, si un sistema terminal de origen o un conmutador de paquetes están enviando un paquete de $L$ bits a través de un enlace cuya velocidad de transmisión es $R$ bits/s, entonces el tiempo necesario para transmitir el paquete es igual a $L/R$ segundos.

### Transmisión de almacenamiento y reenvío

La mayoría de los conmutadores de paquetes aplican el método de **transmisión de almacenamiento y reenvío** en las entradas de los enlaces. Transmisión de almacenamiento y reenvío

significa que el conmutador de paquetes tiene que recibir el paquete completo antes de poder comenzar a transmitir el primer bit del paquete al enlace de salida. Para analizar con más detalle la transmisión de almacenamiento y reenvío, consideremos una red simple, compuesta de dos sistemas terminales conectados mediante un único router, como se muestra en la Figura 1.11. Normalmente, un router tendrá muchos enlaces entrantes, porque su trabajo consiste en conmutar un paquete entrante hacia un enlace de salida; en este ejemplo simple, el router tiene encomendada la sencilla tarea de transferir paquetes desde un enlace (de entrada) hacia el único otro enlace conectado. En este ejemplo, el origen dispone de tres paquetes para enviar hacia el destino, cada uno de ellos compuesto por L bits. En el instante temporal mostrado en la Figura 1.11, el origen ha transmitido ya parte del paquete 1, y el inicio del paquete 1 ha llegado ya al router. Puesto que el router emplea el mecanismo de almacenamiento y reenvío, en este instante de tiempo el router no puede enviar los bits que ha recibido; en lugar de ello, debe primero guardar (es decir, "almacenar") los bits del paquete. Solo después de que el router haya recibido *todos* los bits del paquete, podrá empezar a transmitir (es decir, "reenviar") el paquete a través del enlace de salida. Para comprender mejor la transmisión de almacenamiento y reenvío, calculemos ahora la cantidad de tiempo que transcurre desde el momento en que el origen comienza a enviar el paquete, hasta que el destino termina de recibirlo. (Aquí vamos a ignorar el retardo de propagación —el tiempo que los bits necesitan para viajar a través del cable a una velocidad cercana a la de la luz—, del que hablaremos en la Sección 1.4.) El origen comienza a transmitir en el instante 0; en el instante $L/R$ segundos, el origen habrá transmitido todo el paquete, el cual habrá sido recibido y almacenado en el router (puesto que no hay retardo de propagación). En ese mismo instante $L/R$ segundos, como el router ha recibido ya todo el paquete, puede comenzar a transmitirlo hacia el destino a través del enlace de salida; en el instante $2L/R$, el router habrá transmitido todo el paquete y este habrá sido recibido en su totalidad por el destino. Por tanto, el retardo total es de $2L/R$. Si, en lugar de comportarse así, el router reenviara los bits a medida que van llegando (sin esperar a recibir primero todo el paquete), entonces el retardo total sería $L/R$, puesto que los bits no se verían retenidos en el router. Pero, como veremos en la Sección 1.4, los routers necesitan recibir, almacenar y *procesar* el paquete completo antes de reenviarlo.

Calculemos ahora la cantidad de tiempo que transcurre desde el momento en que el origen empieza a enviar el primer paquete, hasta que el destino ha recibido los tres paquetes. Al igual que antes, en el instante $L/R$ el router empieza a reenviar el primer paquete. Pero también en el instante $L/R$, el origen comenzará a enviar el segundo paquete, puesto que acaba de terminar de enviar el primer paquete completo. Por tanto, en el instante $2L/R$ el destino ha terminado de recibir el primer paquete y el router ha recibido el segundo. De forma similar, en el instante $3L/R$ el destino ha terminado de recibir los dos primeros paquetes y el router ha recibido el tercero. Finalmente, en el instante $4L/R$ el destino habrá recibido los tres paquetes.

Consideremos ahora el caso general consistente en enviar un paquete desde el origen al destino a través de una ruta compuesta por $N$ enlaces, cada uno de ellos de velocidad $R$ (y en la que, por tanto, hay $N$-1 routers entre el origen y el destino). Aplicando la misma lógica anterior, vemos que el retardo extremo a extremo es:

$$d_{extremo-extremo} = N\frac{L}{R} \tag{1.1}$$

Pruebe a intentar determinar cuál sería el retardo para $P$ paquetes enviados a través de una serie de $N$ enlaces.

### Retardos de cola y pérdida de paquetes

Cada conmutador de paquetes tiene varios enlaces conectados a él y para cada enlace conectado, el conmutador de paquetes dispone de un **buffer de salida** (también denominado **cola de salida**), que almacena los paquetes que el router enviará a través de dicho enlace. El buffer de salida desempeña

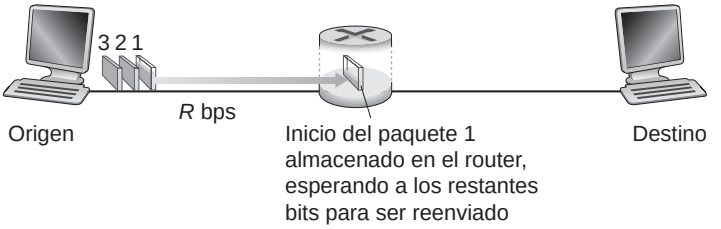



**Figura 1.11** ♦ Conmutación de paquetes con almacenamiento y reenvío.

un papel clave en la conmutación de paquetes. Si un paquete entrante tiene que ser transmitido a través de un enlace, pero se encuentra con que el enlace está ocupado transmitiendo otro paquete, el paquete entrante tendrá que esperar en el buffer de salida. Por tanto, además de los retardos de almacenamiento y reenvío, los paquetes se ven afectados por los **retardos de cola** del buffer de salida. Estos retardos son variables y dependen del nivel de congestión de la red. Puesto que la cantidad de espacio en el buffer es finita, un paquete entrante puede encontrarse con que el buffer está completamente lleno con otros paquetes que esperan a ser transmitidos. En este caso, se producirá una **pérdida de paquetes**: el paquete que acaba de llegar o uno que ya se encuentre en la cola será descartado.

La Figura 1.12 ilustra una red de conmutación de paquetes simple. Al igual que en la Figura 1.11, los paquetes se han representado mediante bloques tridimensionales. El ancho de un bloque representa el número de bits que contiene el paquete. En esta figura, todos los paquetes tienen el mismo ancho y, por tanto, la misma longitud. Suponga ahora que los hosts A y B están enviando paquetes al host E. En primer lugar, los hosts A y B envían sus paquetes a través de los enlaces Ethernet a 100 Mbps hasta el primer router. A continuación, este dirige los paquetes al enlace de 15 Mbps. Si, durante un corto intervalo de tiempo, la velocidad de llegada de los paquetes al router (medida en bits por segundo) excede los 15 Mbps, se producirá congestión en el router, a medida que los paquetes se ponen en cola en el buffer del enlace de salida, antes de poder ser transmitidos a través de él. Por ejemplo, si tanto el Host A como el Host B envían simultáneamente una ráfaga de cinco paquetes seguidos, entonces la mayoría de esos paquetes necesitarán esperar un cierto tiempo en la cola. Esta situación es, de hecho, completamente análoga a muchas situaciones de la vida cotidiana —por ejemplo, cuando nos vemos obligados a esperar en la cola de un cajero o en la del peaje de una autopista. En la Sección 1.4 examinaremos más detalladamente el retardo de cola.

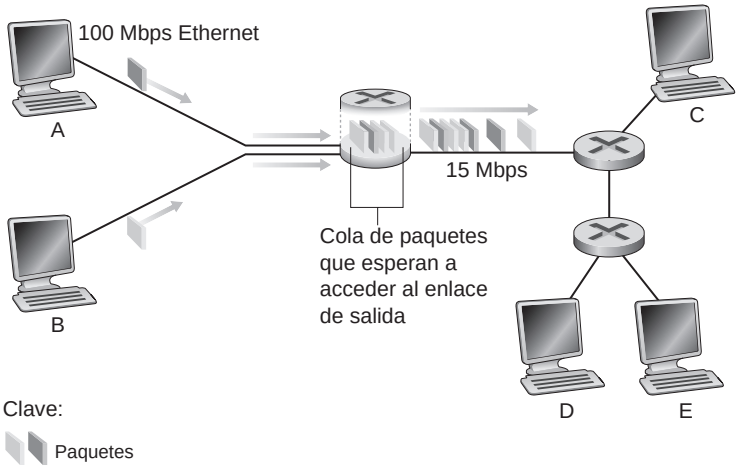



**Figura 1.12** ♦ Conmutación de paquetes.

### Tablas de reenvío y protocolos de enrutamiento

Anteriormente hemos dicho que un router toma un paquete entrante a través de uno de sus enlaces de comunicaciones y reenvía dicho paquete a través de otro de sus enlaces de comunicaciones. ¿Pero cómo determina el router cuál es el enlace a través del cual deberá reenviar el paquete? En la práctica, el reenvío de paquetes se lleva a cabo de distinta manera en los diferentes tipos de redes de computadoras. En este capítulo, vamos a describir brevemente el método empleado por Internet.

En Internet, cada sistema terminal tiene asignada una dirección, denominada dirección IP. Cuando un sistema terminal de origen quiere enviar un paquete a un sistema terminal de destino, el origen incluye la dirección IP de destino en la cabecera del paquete. Al igual que las direcciones postales, esta dirección tiene una estructura jerárquica. Cuando un paquete llega a un router de la red, el router examina una parte de la dirección de destino del paquete y lo reenvía a un router adyacente. Más específicamente, cada router dispone de una **tabla de reenvío** que asigna las direcciones de destino (o una parte de las mismas) a los enlaces salientes de ese router. Cuando llega un paquete a un router, este examina la dirección y busca en su tabla de reenvío esa dirección de destino, para determinar el enlace de salida apropiado. A continuación, el router dirige el paquete a ese enlace de salida.

El proceso de enrutamiento terminal a terminal es análogo al que sigue el conductor de un automóvil que no utiliza un mapa, sino que prefiere preguntar cómo llegar hasta una determinada dirección. Por ejemplo, suponga que Juan sale de Filadelfia y tiene que llegar al 156 de la calle Lakeside Drive en Orlando, Florida. Lo primero que hace Juan es dirigirse a la estación de servicio más próxima y preguntar cómo llegar a su destino. El empleado se queda con el nombre del estado, Florida, y le dice que debe tomar la autopista interestatal I-95 Sur y que existe una entrada a la misma nada más salir de la estación de servicio. También le dice a Juan que una vez que haya entrado en Florida, pregunte a alguien cómo llegar a su destino. Así, Juan toma la I-95 Sur hasta Jacksonville, Florida, lugar donde vuelve a preguntar en otra estación de servicio. El dependiente extrae de la dirección la información que hace referencia a Orlando y le dice que debe continuar por la I-95 hasta Daytona Beach y que luego pregunte. En otra estación de servicio de Daytona Beach, el empleado extrae de nuevo la información referente a Orlando y le dice que tomando la I-4 llegará directamente a Orlando. Juan toma la I-4 y la abandona en la salida correspondiente a Orlando. De nuevo se detiene en otra gasolinera y esta vez el dependiente extrae la parte de la información de la dirección referente a Lakeside Drive, y le indica la carretera que debe seguir para llegar allí. Una vez que Juan se encuentra en Lakeside Drive, pregunta a un niño que va en bicicleta cómo llegar a su destino. El niño extrae el dato 156 de la dirección y le señala una casa. Por fin, Juan ha llegado a su destino. En esta analogía, los dependientes de las estaciones de servicio y el niño de la bicicleta son los routers.

Acabamos de ver que un router utiliza la dirección de destino de un paquete para indexar una tabla de reenvío y determinar el enlace de salida apropiado. Pero esta afirmación nos lleva a la siguiente pregunta: ¿cómo se definen las tablas de reenvío? ¿Se configuran manualmente en cada router o Internet utiliza un procedimiento más automatizado? Estas cuestiones se abordan en detalle en el Capítulo 5, pero para ir abriendo boca, diremos que Internet dispone de una serie de **protocolos de enrutamiento** especiales que se utilizan para definir automáticamente las tablas de reenvío. Por ejemplo, un protocolo de enrutamiento puede determinar la ruta más corta desde cada router hasta cada destino y usar esas rutas más cortas para configurar las tablas de reenvío en los routers.

¿Cómo podríamos ver la ruta terminal a terminal que siguen los paquetes en Internet? Le invitamos a que se ponga manos a la obra, utilizando el programa Traceroute; simplemente, visite el sitio http://www.traceroute.org, elija un origen en el país que quiera y trace la ruta desde ese origen hasta su computadora. (Para obtener más información acerca de Traceroute, consulte la Sección 1.4.).

### 1.3.2 Conmutación de circuitos

Existen dos métodos fundamentales que permiten transportar los datos a través de una red de enlaces y conmutadores: la **conmutación de circuitos** y la **conmutación de paquetes**. Ya hemos hablado de las redes de conmutación de paquetes en la subsección anterior, así que ahora fijaremos nuestra atención en las redes de conmutación de circuitos.

En las redes de conmutación de circuitos, los recursos necesarios a lo largo de una ruta (buffers, velocidad de transmisión del enlace) que permiten establecer la comunicación entre los sistemas terminales están *reservados* durante el tiempo que dura la sesión de comunicación entre dichos sistemas terminales. En las redes de conmutación de paquetes, estos recursos no están reservados; los mensajes de una sesión utilizan los recursos bajo petición y, en consecuencia, pueden tener que esperar (es decir, ponerse en cola) para poder acceder a un enlace de comunicaciones. Veamos una sencilla analogía. Piense en dos restaurantes, en uno de los cuales es necesario hacer reserva, mientras que en el otro no se necesita hacer reserva ni tampoco las admiten. Para comer en el restaurante que precisa reserva, tenemos que molestarnos en llamar por teléfono antes de salir de casa, pero al llegar allí, en principio, podremos sentarnos y pedir nuestro menú al camarero de manera inmediata. En el restaurante que no admite reservas, no tenemos que molestarnos en reservar mesa, pero al llegar allí, es posible que tengamos que esperar a que haya una mesa disponible, antes de poder sentarnos.

Las redes telefónicas tradicionales son ejemplos de redes de conmutación de circuitos. Considere lo que ocurre cuando una persona desea enviar información (de voz o faxsímil) a otra a través de una red telefónica. Antes de que el emisor pueda transmitir la información, la red debe establecer una conexión entre el emisor y el receptor. Se trata de una conexión *de buena fe* en la que los conmutadores existentes en la ruta entre el emisor y el receptor mantienen el estado de la conexión para dicha comunicación. En la jerga del campo de la telefonía, esta conexión se denomina **circuito**. Cuando la red establece el circuito, también reserva una velocidad de transmisión constante en los enlaces de la red (que representa una fracción de la capacidad de transmisión de cada enlace) para el tiempo que dure la conexión. Dado que ha reservado una determinada velocidad de transmisión para esta conexión emisor-receptor, el emisor puede transferir los datos al receptor a la velocidad constante *garantizada*.

La Figura 1.13 ilustra una red de conmutación de circuitos. En esta red, los cuatro conmutadores de circuitos están interconectados mediante cuatro enlaces. Cada uno de los enlaces tiene cuatro circuitos, por lo que cada enlace puede dar soporte a cuatro conexiones simultáneas. Cada uno de los hosts (por ejemplo, PCs y estaciones de trabajo) está conectado directamente a uno de los conmutadores. Cuando dos hosts desean comunicarse, la red establece una **conexión extremo a extremo** dedicada entre ellos. Por tanto, para que el host A se comunique con el host B, la red tiene que reservar en primer lugar un circuito en cada uno de los dos enlaces. En este ejemplo, la conexión extremo a extremo dedicada usa el segundo circuito en el primer enlace y el cuarto circuito en el segundo. Dado que cada enlace tiene cuatro circuitos, para cada enlace utilizado por la conexión extremo a extremo, esa conexión obtiene un cuarto de la capacidad total de transmisión del enlace mientras dure la conexión. Así, por ejemplo, si cada enlace entre dos conmutadores adyacentes tiene una velocidad de transmisión de 1 Mbps, entonces cada conexión extremo a extremo de conmutación de circuitos obtiene 250 kbps de velocidad de transmisión dedicada.

Por el contrario, veamos qué ocurre cuando un host desea enviar un paquete a otro host a través de una red de conmutación de paquetes, como por ejemplo Internet. Al igual que con la conmutación de circuitos, el paquete se transmite a través de una serie de enlaces de comunicaciones. Pero, a diferencia de la conmutación de circuitos, el paquete se envía a la red sin haber reservado ningún tipo de recurso de enlace. Si uno de los enlaces está congestionado porque otros paquetes tienen que ser transmitidos a través de él al mismo tiempo, entonces nuestro paquete tendrá que esperar en un buffer en el lado del enlace de transmisión correspondiente al emisor y sufrirá un retardo. Internet realiza el máximo esfuerzo para suministrar los paquetes a tiempo, pero no ofrece ningún tipo de garantía.

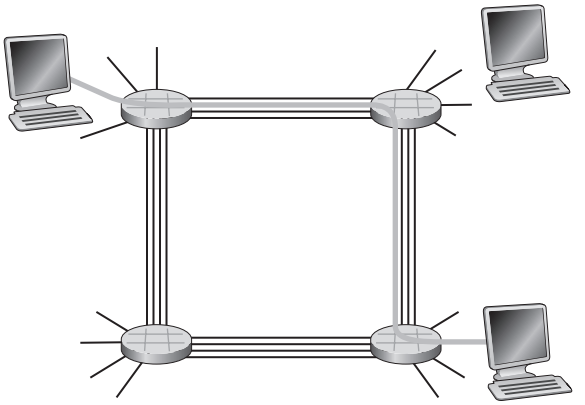


**Figura 1.13** ♦ Una red simple de conmutación de circuitos, compuesta por cuatro conmutadores y cuatro enlaces.

### Multiplexación en redes de conmutación de circuitos

Los circuitos de un enlace se implementan, bien mediante **multiplexación por división de frecuencia (FDM, Frequency-Division Multiplexing)** o bien mediante **multiplexación por división en el tiempo (TDM, *Time-Division Multiplexing*)**. Con FDM, el espectro de frecuencia de un enlace se reparte entre las conexiones establecidas a través del enlace. Específicamente, el enlace dedica una banda de frecuencias a cada conexión durante el tiempo que esta dure. En las redes telefónicas, esta banda de frecuencias normalmente tiene un ancho de 4 kHz (es decir, 4.000 hercios o 4.000 ciclos por segundo). El ancho de esta banda se denomina, como cabría esperar, **ancho de banda**. Las estaciones de radio FM también emplean la multiplexación FDM para compartir el espectro de frecuencias entre 88 MHz y 108 MHz, teniendo cada estación asignada una banda de frecuencias específica.

En un enlace TDM, el tiempo se divide en marcos de duración fija y cada marco se divide en un número fijo de particiones. Cuando la red establece una conexión a través de un enlace, la red dedica una partición de cada marco a dicha conexión. Estas particiones están reservadas para uso exclusivo de dicha conexión, habiendo una partición disponible (en cada marco) para transmitir los datos de esa conexión.

La Figura 1.14 ilustra las multiplexaciones FDM y TDM para un enlace de red específico que da soporte a cuatro circuitos. En el caso de la multiplexación por división de frecuencia, el dominio de frecuencia se segmenta en cuatro bandas, siendo el ancho de banda de cada una de ellas de 4 kHz. En el caso de la multiplexación TDM, el dominio del tiempo se divide en marcos, conteniendo cada uno de ellos cuatro particiones. A cada circuito se le asigna la misma partición dedicada dentro de los marcos, de forma cíclica. En la multiplexación TDM, la velocidad de transmisión de un circuito es igual a la velocidad de marco multiplicada por el número de bits existentes en una partición. Por ejemplo, si el enlace transmite 8.000 marcos por segundo y cada partición consta de 8 bits, entonces la velocidad de transmisión de un circuito es igual a 64 kbps.

Los partidarios de la tecnología de conmutación de paquetes siempre han argumentado que la conmutación de circuitos desperdicia recursos, porque los circuitos dedicados no se usan para nada durante los **periodos de inactividad**. Por ejemplo, cuando una persona deja de hablar durante una llamada telefónica, los recursos de red inactivos (bandas de frecuencia o particiones temporales en los enlaces situados a lo largo de la ruta de la conexión) no pueden ser empleados por otras conexiones en curso. Otro ejemplo de cómo estos recursos pueden ser infrautilizados sería un radiólogo que empleara una red de conmutación de circuitos para acceder remotamente a una serie de radiografías de rayos X. El radiólogo establece una conexión, solicita una imagen, la contempla y luego solicita otra imagen. Los recursos de la red están asignados a la conexión, pero no se

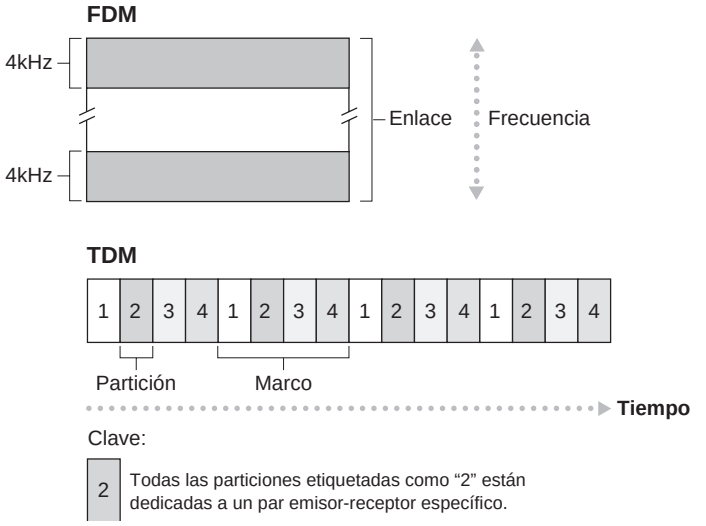



**Figura 1.14** ♦ Con FDM, cada circuito obtiene de forma continua una fracción del ancho de banda. Con TDM, cada circuito dispone de todo el ancho de banda periódicamente durante breves intervalos de tiempo (es decir, durante las particiones).

utilizan (es decir, se desperdician) durante el tiempo que el radiólogo contempla las imágenes. Los partidarios de la conmutación de paquetes también disfrutan apuntando que el establecimiento de circuitos extremo a extremo y la reserva de capacidad de transmisión extremo a extremo son procesos complicados, que requieren el uso de software de señalización complejo para coordinar el funcionamiento de los conmutadores a lo largo de la ruta extremo a extremo.

Antes de terminar con esta exposición acerca de la conmutación de circuitos, vamos a ver un ejemplo numérico que debería arrojar más luz sobre este tema. Consideremos el tiempo que se tarda en enviar un archivo de 640.000 bits desde el host A al host B a través de una red de conmutación de circuitos. Supongamos que todos los enlaces de la red utilizan multiplexación TDM con 24 particiones y tienen una velocidad de bit de 1,536 Mbps. Supongamos también que se tardan 500 milisegundos en establecer el circuito extremo a extremo antes de que el host A pueda comenzar a transmitir el archivo. ¿Cuánto tiempo se tarda en transmitir el archivo? La velocidad de transmisión de cada circuito es (1,536 Mbps)/24 = 64 kbps, por lo que se precisan (640.000 bits)/(64 kbps) = 10 segundos para transmitir el archivo. A estos 10 segundos tenemos que sumarles el tiempo de establecimiento del circuito, lo que da como resultado 10,5 segundos de tiempo total de transmisión del archivo. Observe que el tiempo de transmisión es independiente del número de enlaces: el tiempo de transmisión será 10 segundos independientemente de que el circuito extremo a extremo pase a través de un enlace o de cien enlaces. (El retardo real extremo a extremo también incluye un retardo de propagación; véase la Sección 1.4.).

### Comparación entre la conmutación de paquetes y la conmutación de circuitos

Ahora que ya hemos descrito las tecnologías de conmutación de circuitos y de paquetes, vamos a pasar a compararlas. Los detractores de la tecnología de conmutación de paquetes a menudo han argumentado que esta tecnología no es adecuada para los servicios en tiempo real, como por ejemplo las llamadas telefónicas y las videoconferencias, porque sus retardos extremo a extremo son variables e impredecibles (a causa principalmente de que los retardos de cola de los paquetes son variables e impredecibles). Por otro lado, los partidarios de la conmutación de paquetes argumentan que (1) ofrece una mejor compartición de la capacidad de transmisión existente que

la tecnología de conmutación de circuitos y (2) es más sencilla, más eficiente y menos cara de implementar que la conmutación de circuitos. Puede encontrar una comparación interesante de la conmutación de paquetes y la conmutación de circuitos en [Molinero-Fernández 2002]. En términos generales, las personas a las que no les gusta reservar mesa en un restaurante prefieren la conmutación de paquetes a la conmutación de circuitos.

¿Por qué es más eficiente la conmutación de paquetes? Veamos un ejemplo sencillo. Suponga que varios usuarios comparten un enlace de 1 Mbps. Suponga también que cada usuario alterna entre periodos de actividad (cuando genera datos a una velocidad constante de 100 kbps) y periodos de inactividad (cuando no genera datos). Además, suponga que un usuario solo está activo un 10 por ciento del tiempo (y está inactivo tomando café durante el 90 por ciento del tiempo restante). Con la tecnología de conmutación de circuitos, tienen que *reservarse para cada usuario* 100 kbps en todo momento. Por ejemplo, en una red de conmutación de circuitos con multiplexación TDM, si un marco de un segundo se divide en 10 particiones de 100 ms, entonces cada usuario tendría asignada una partición por marco.

Por tanto, el enlace de conmutación de circuitos solo podrá dar soporte a 10 (= 1 Mbps/100 kbps) usuarios simultáneamente. En el caso de utilizar la conmutación de paquetes, la probabilidad de que un determinado usuario esté activo es 0,1 (es decir, del 10 por ciento). Si hay 35 usuarios, la probabilidad de que 11 o más usuarios estén activos simultáneamente es aproximadamente igual a 0,0004. (El Problema de Repaso R8 indica cómo se obtiene esta probabilidad.) Cuando hay 10 o menos usuarios activos a la vez (lo que ocurre con una probabilidad del 0,9996), la velocidad acumulada de llegada de los datos es menor o igual a 1 Mbps, la velocidad de salida del enlace. Por tanto, cuando el número de usuarios activos es 10 o menos, los paquetes fluyen a través del enlace prácticamente sin retardo, como en el caso de la tecnología de conmutación de circuitos. Cuando hay más de 10 usuarios activos simultáneamente, entonces la velocidad acumulada de llegada de paquetes excede la capacidad de salida del enlace y la cola de salida comenzará a crecer. (Continúa creciendo hasta que la velocidad acumulada de entrada cae por debajo de 1 Mbps, punto en el que la longitud de la cola comenzará a disminuir.) Puesto que la probabilidad de que haya más de 10 usuarios conectados a la vez es muy baja en este ejemplo, la conmutación de paquetes proporciona prácticamente el mismo rendimiento que la conmutación de circuitos, *pero lo hace permitiendo que haya un número de usuarios más de tres veces superior*.

Consideremos ahora otro ejemplo sencillo. Suponga que hay 10 usuarios y que de repente un usuario genera 1.000 paquetes de 1.000 bits, mientras que los usuarios restantes permanecen inactivos y no generan paquetes. Con la tecnología de conmutación de circuitos con multiplexación TDM con 10 particiones por marco y con cada partición formada por 1.000 bits, el usuario activo solo puede emplear su partición por marco para transmitir los datos, mientras que las restantes nueve particiones de cada marco permanecen inactivas. Transcurrirán 10 segundos antes de que el millón de bits de datos del usuario activo hayan sido transmitidos. Sin embargo, con la conmutación de paquetes, el usuario activo puede enviar de forma continuada sus paquetes a la velocidad máxima del enlace de 1 Mbps, ya que no hay ningún otro usuario generando paquetes que tengan que ser multiplexados con los paquetes del usuario activo. En este caso, todos los datos del usuario activo se transmitirán en un segundo.

Los ejemplos anteriores ilustran dos casos en los que el rendimiento de la tecnología de conmutación de paquetes puede resultar superior al de la conmutación de circuitos. También ha quedado patente la diferencia crucial entre las dos formas de compartir la velocidad de transmisión del enlace entre varios flujos de datos. La conmutación de circuitos preasigna el uso del enlace de transmisión independientemente de la demanda, con lo que el tiempo de enlace asignado, pero innecesario, se desperdicia. Por el contrario, la conmutación de paquetes asigna el uso del enlace *bajo demanda*. La capacidad de transmisión del enlace se compartirá paquete a paquete solo entre aquellos usuarios que tengan paquetes que transmitir a través del enlace.

Aunque la conmutación de circuitos y la de paquetes son ambas prevalentes en las actuales redes de telecomunicaciones, se está tendiendo claramente hacia las redes de conmutación de paquetes. Incluso muchas de las redes de telefonía de conmutación de circuitos actuales se están migrando

lentamente a redes de conmutación de paquetes. En particular, las redes telefónicas suelen emplear la conmutación de paquetes en la parte internacional de las llamadas, que es la más cara.

### 1.3.3 Una red de redes

Anteriormente hemos visto que los sistemas terminales (PC de usuario, teléfonos inteligentes, servidores web, servidores de correo electrónico, etc.) se conectan a Internet a través de un ISP de acceso. El ISP de acceso puede proporcionar conectividad cableada o inalámbrica, mediante una amplia variedad de tecnologías de acceso, entre las que se incluyen DSL, cable, FTTH, Wi- Fi y la tecnología celular. Observe que el ISP de acceso no tiene por qué ser una compañía de telecomunicaciones o de cable: puede ser, por ejemplo, una universidad (que proporciona acceso a Internet a los estudiantes, al personal y a las facultades) o una empresa (que proporciona acceso a sus estudiantes, empleados y profesores) o una empresa (que proporciona acceso a su personal). Pero la conexión de los usuarios finales y de los proveedores de contenido a un ISP de acceso es solo una pequeña parte del problema de conectar los miles de millones de sistemas terminales que conforman Internet. Para completar este puzzle, los propios ISP de acceso deben interconectarse. Esto se hace creando una *red de redes* —comprender esta frase es la clave para entender Internet.

A lo largo de los años, la red de redes que es Internet ha evolucionado hasta formar una estructura muy compleja. Buena parte de esta evolución está dictada por razones económicas y políticas nacionales, en vez de por consideraciones de rendimiento. Para comprender la estructura de red actual de Internet, vamos a construir incrementalmente una serie de estructuras de red, siendo cada una de ellas una mejor aproximación a la Internet compleja que tenemos hoy en día. Recuerde que el objetivo fundamental es interconectar los ISP de acceso de modo que todos los sistemas terminales puedan intercambiarse paquetes. Un enfoque simplista consistiría en que cada ISP de acceso se conectara *directamente* con todos los restantes ISP de acceso. Ese diseño en forma de malla resulta, por supuesto, demasiado costoso para los ISP de acceso, ya que requeriría que cada uno de ellos dispusiera de un enlace de comunicación separado con cada uno de los restantes cientos de miles de ISP de acceso existentes en el mundo.

Nuestra primera estructura de red, a la que llamaremos *Estructura de Red 1*, interconecta todos los ISP de acceso mediante un *único ISP global de tránsito*. Nuestro (imaginario) ISP global de tránsito sería una red de routers y enlaces de comunicaciones que no solo cubriría todo el planeta, sino que dispondría además de al menos un router cerca de cada uno de los cientos de miles de ISP de acceso. Por supuesto, sería muy costoso para el ISP global construir una red tan extensa. Para ser rentable, tendría por supuesto que cobrar por la conectividad a cada uno de los ISP de acceso, debiendo ese precio depender (aunque no necesariamente de forma directamente proporcional) de la cantidad de tráfico que el ISP de acceso intercambie con el ISP global. Puesto que el ISP de acceso paga al ISP global de tránsito, decimos que el ISP de acceso es un **cliente** y que el ISP global de tránsito es un **proveedor**.

Ahora, si alguna empresa construye y opera de forma rentable un ISP global de tránsito, entonces resulta natural que otras empresas construyan sus propios ISP globales de tránsito y compitan con el ISP global de tránsito original. Esto conduce a la *Estructura de Red 2*, que está formada por los cientos de miles de ISP de acceso y *múltiples* ISP globales de tránsito. Los ISP de acceso prefieren, por supuesto, la Estructura de Red 2 a la Estructura de Red 1, dado que ahora pueden elegir entre los proveedores globales de tránsito competidores, en función de sus precios y de los servicios que ofrezcan. Observe, sin embargo, que los propios ISP globales de tránsito deben interconectarse unos con otros: en caso contrario, los ISP de acceso conectados a uno de los proveedores globales de tránsito no serían capaces de comunicarse con los ISP de acceso conectados a los restantes proveedores globales de tránsito.

La Estructura de red 2 que acabamos de describir es una jerarquía en dos niveles, en la que los proveedores globales de tránsito residen en el nivel superior y los ISP de acceso, en el inferior. Esta estructura presupone que los ISP globales de tránsito no solo son capaces de estar cerca de cada

uno de los ISP de acceso, sino que también encuentran económicamente deseable el hacerlo. En realidad, aunque algunos ISP tienen una impresionante cobertura global y se conectan directamente con muchos ISP de acceso, no hay ningún ISP que tenga presencia en todas y cada una de las ciudades del mundo. En lugar de ello, en cualquier región determinada, puede haber un **ISP regional** al que se conecten los ISP de acceso de esa región. Cada ISP regional se conecta entonces con los **ISP de nivel 1**. Los ISP de nivel 1 son similares a nuestros (imaginarios) ISP globales de tránsito; pero los ISP de nivel 1, que sí que existen, no tienen presencia en todas las ciudades del mundo. Existe aproximadamente una docena de ISP de nivel 1, incluyendo Level 3 Communications, AT&T, Sprint y NTT. Resulta interesante observar que no hay ningún organismo que otorgue oficialmente el estatus de nivel 1; como dice el refrán, si necesitas preguntar si eres miembro de un grupo, probablemente no lo eres.

Volviendo a esta red de redes, no solo hay múltiples ISP competidores de nivel 1, sino que también puede haber múltiples ISP regionales competidores en una determinada región. En dicha jerarquía, cada ISP de acceso paga al ISP regional al que está conectado, y cada ISP regional paga al ISP de nivel 1 con el que se conecta. (Un ISP de acceso también puede conectarse directamente a un ISP de nivel 1, en cuyo caso le paga a él.) Por tanto, hay una relación cliente-proveedor en cada nivel de la jerarquía. Observe que los ISP de nivel 1 no pagan a nadie, ya que se encuentran en lo alto de la jerarquía. Para complicar las cosas todavía más, en algunas regiones puede haber un ISP regional de mayor tamaño (que posiblemente abarque un país completo) al que se conectan los ISP regionales más pequeños de esa región; el ISP regional de mayor tamaño se conecta a su vez a un ISP de nivel 1. Por ejemplo, en China existen ISP de acceso en cada ciudad, que se conectan a ISP provinciales, que a su vez se conectan a ISP nacionales, que finalmente se conectan a ISP de nivel 1 [Tian 2012]. A esta jerarquía multinivel, que sigue siendo tan solo una aproximación burda de la Internet actual, la denominamos *Estructura de red 3*.

Para construir una red que se asemeje más a la Internet de hoy en día, necesitamos añadir a la Estructura de Red 3 jerárquica los puntos de presencia (PoP, *Point of Presence*), la multidomiciliación (*multihoming*), la conexión entre pares (*peering*) y los puntos de intercambio Internet (IXP, Internet eXchange Point). Los PoP existen en todos los niveles de la jerarquía, salvo en el nivel inferior (ISP de acceso). Un **PoP** es, simplemente, un grupo de uno o más routers (en una misma ubicación) de la red del proveedor, a través de los cuales los ISP clientes pueden conectarse al ISP proveedor. Para conectarse al PoP del proveedor, la red del cliente puede arrendar un enlace de alta velocidad a un proveedor de telecomunicaciones cualquiera, con el fin de conectar directamente uno de sus routers a un router del PoP. Cualquier ISP (salvo los ISP de nivel 1) puede optar por la **multidomiciliación**, es decir, por conectarse a dos o más ISP proveedores. Así, por ejemplo, un ISP de acceso puede multidomiciliarse con dos ISP regionales, o con dos ISP regionales y un ISP de nivel 1. De forma similar, un ISP regional puede multidomiciliarse con varios ISP de nivel 1. Cuando un ISP opta por la multidomiciliación, puede continuar enviando y recibiendo paquetes a través de Internet incluso si uno de sus proveedores sufre un fallo.

Como hemos visto, los ISP clientes pagan a su ISP proveedor para obtener una interconectividad global a través de Internet. La cantidad que un ISP cliente paga a un ISP proveedor refleja la cantidad de tráfico que intercambia con el proveedor. Para reducir estos costes, dos ISP próximos, situados en el mismo nivel de la jerarquía, pueden establecer una conexión entre pares, es decir, pueden conectar directamente sus redes, de modo que todo el tráfico que se intercambien pase por la conexión directa, en lugar de a través de intermediarios situados aguas arriba. Cuando dos ISP efectúan una conexión entre pares, suele ser libre de cargo, es decir, ninguno de los dos ISP paga al otro. Como hemos dicho anteriormente, los ISP de nivel 1 también establecen conexiones entre pares, libres de cargo. En [Van der Berg 2008] podrá encontrar una explicación comprensible de las conexiones entre pares y las relaciones cliente-proveedor. Siguiendo estos mismos principios, una empresa puede crear un **punto de intercambio Internet** (IXP, *Internet Exchange Point*), que es un punto de reunión en el que múltiples ISP pueden establecer conexiones entre pares. El IXP se localiza normalmente en su propio edificio, con sus propios conmutadores [Ager 2012]. Hoy en

día, en Internet hay más de 400 IXP [IXP List 2016]. A este ecosistema —compuesto por los ISP de acceso, los ISP regionales, los ISP de nivel 1, los PoP, la multidomiciliación, las conexiones entre pares y los IXP— lo denominamos *Estructura de red 4*.

Con esto llegamos, finalmente, a la *Estructura de red 5*, que describe la Internet de hoy en día. La Estructura de Red 5, ilustrada en la Figura 1.15, se construye sobre la Estructura de red 4, añadiendo las **redes de proveedores de contenido**. Google es, actualmente, uno de los ejemplos punteros de dichas redes de provisión de contenidos. En el momento de escribir estas líneas, se estima que Google dispone de 50–100 centros de datos por Norteamérica, Europa, Asia, Sudamérica y Australia. Algunos de estos centros de datos albergan más de cien mil servidores, mientras que otros son más pequeños, con solo unos centenares de servidores. Todos los centros de datos de Google están interconectados a través de la propia red TCP/IP privada de Google, que abarca todo el planeta pero está separada de la Internet pública. Resulta importante destacar que la red privada de Google solo transporta tráfico hacia/desde los servidores de Google. Como se muestra en la Figura 1.15, la red privada de Google trata de "puentear" los niveles superiores de Internet, estableciendo conexiones entre pares (libres de cargo) con los ISP de nivel inferior, bien conectándose directamente a ellos o bien conectándose con ellos en algún IXP [Labovitz 2010]. Sin embargo, puesto que a muchos ISP de acceso solo se puede llegar pasando por redes de nivel 1, la red de Google también se conecta con los ISP de nivel 1 y paga a esos ISP por el tráfico que intercambia con ellos. Al crear su propia red, un proveedor de contenido no solo reduce sus pagos a los ISP de nivel superior, sino que también tiene un mayor control sobre el modo en que se prestan sus servicios en último término a los usuarios finales. La infraestructura de la red de Google se describe con mayor detalle en la Sección 2.6.

En resumen, la Internet de hoy en día —una red de redes— es compleja, estando compuesta por aproximadamente una docena de ISP de nivel 1 y cientos de miles de ISP de nivel inferior. La cobertura geográfica de los ISP es muy variable, habiendo algunos que abarcan múltiples continentes y océanos, mientras que otros están limitados a pequeñas regiones. Los ISP de nivel inferior se conectan con los de nivel superior y los de nivel superior se interconectan entre sí. Los usuarios y los proveedores de contenido son clientes de los ISP de nivel inferior y estos son clientes de los ISP de nivel superior. En los últimos años, los principales proveedores de contenidos han creado también sus propias redes y se conectan directamente, siempre que pueden, a los ISP de nivel inferior.

## 1.4 Retardos, pérdidas y tasa de transferencia en las redes de conmutación de paquetes

En la Sección 1.1 hemos dicho que Internet puede verse como una infraestructura que proporciona servicios a aplicaciones distribuidas que se ejecutan en sistemas terminales. Idealmente, desearíamos que los servicios de Internet pudieran transportar tantos datos como quisiéramos entre cualesquiera dos sistemas terminales de forma instantánea y sin que tuviera lugar ninguna pérdida de datos. Evidentemente, en la realidad, este objetivo es inalcanzable, ya que necesariamente las redes de computadoras restringen la tasa de transferencia (la cantidad de datos por segundo que pueden transmitir) entre sistemas terminales, introducen retardos entre los sistemas terminales y pueden, de hecho, perder paquetes. Por una parte, es una pena que las leyes de la Física introduzcan retardos y pérdidas, así como que restrinjan las tasas de transferencia, pero, por otra parte, puesto que las redes de computadoras presentan estos problemas, existen muchas cuestiones interesantes relacionadas con cómo abordarlos —¡más que suficientes como para llenar un curso sobre redes de computadoras y para motivar miles de tesis doctorales! En esta sección comenzaremos examinando y cuantificando los retardos, las pérdidas y la tasa de transferencia en las redes de computadoras.

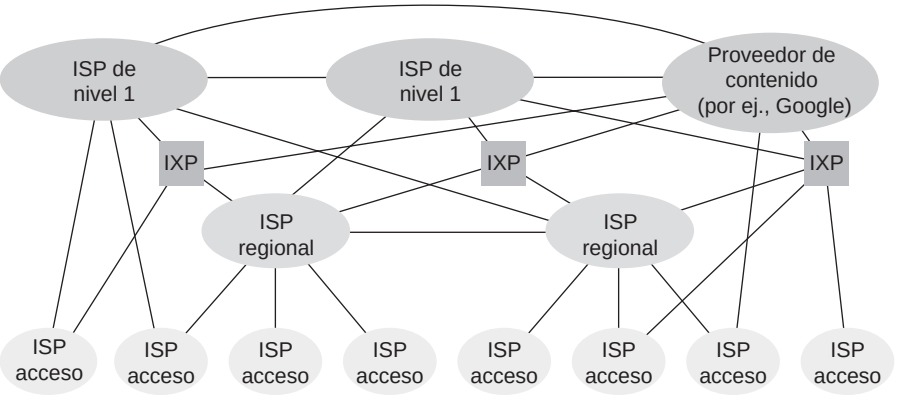



**Figura 1.15** ♦ Interconexión de los ISP.

## 1.4.1 El retardo en las redes de conmutación de paquetes

Recordemos que los paquetes se inician en un host (el origen), atraviesan una serie de routers y terminan su viaje en otro host (el destino). Cuando un paquete viaja de un nodo (host o router) al siguiente nodo (host o router) a lo largo de esa ruta, el paquete sufre varios tipos de retardo *en cada uno* de los nodos de dicha ruta. Los más importantes de estos retardos son: el **retardo de procesamiento nodal**, el **retardo de cola**, el **retardo de transmisión** y el **retardo de propagación**; todos estos retardos se suman para proporcionar el **retardo nodal total**. El rendimiento de muchas aplicaciones de Internet —como las de búsqueda, exploración web, correo electrónico, mapas, mensajería instantánea y voz sobre IP— se ve seriamente afectado por los retardos de red. Para adquirir un conocimiento profundo de la tecnología de conmutación de paquetes y de las redes de computadoras, es preciso comprender la naturaleza e importancia de estos retardos.

### Tipos de retardo

Utilizaremos la Figura 1.16 para explorar estos retardos. Como parte de la ruta extremo a extremo entre el origen y el destino, un paquete se envía desde el nodo situado aguas arriba a través del router A y hasta el router B. Nuestro objetivo es caracterizar el retardo nodal en el router A. Observe que el router A dispone de un enlace de salida que lleva hasta el router B. Este enlace está precedido por una cola (o *buffer*). Cuando el paquete llega al router A procedente del nodo situado aguas arriba, el router A examina la cabecera del paquete para determinar cuál es el enlace de salida apropiado para el paquete y luego dirige dicho paquete a ese enlace. En este ejemplo, el enlace de salida para el paquete es el enlace que lleva hasta el router B. Un paquete puede transmitirse a

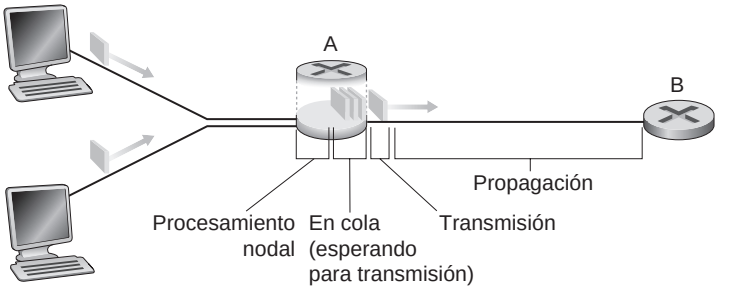



**Figura 1.16** ♦ Retardo nodal en el router A.

través de un enlace solo si actualmente no se está transmitiendo ningún otro paquete a través de él y si no hay otros paquetes que le precedan en la cola; si el enlace está ocupado actualmente o si existen otros paquetes en la cola esperando para ese enlace, entonces el paquete recién llegado tendrá que ponerse a la cola.

### *Retardo de procesamiento*

El tiempo requerido para examinar la cabecera del paquete y determinar dónde hay que enviarlo es parte del **retardo de procesamiento**. El retardo de procesamiento puede también incluir otros factores, como el tiempo necesario para comprobar los errores de nivel de bit del paquete que se hayan producido al transmitir los bits del paquete desde el nodo situado aguas arriba hasta el router A. Los retardos de procesamiento en los routers de alta velocidad suelen ser del orden de los microsegundos o menores. Una vez efectuado el procesamiento nodal, el router dirige el paquete a la cola situada antes del enlace que lleva al router B. (En el Capítulo 4 estudiaremos los detalles acerca de cómo funciona un router.)

### *Retardo de cola*

En la cola, el paquete experimenta un **retardo de cola** mientras espera para ser transmitido a través del enlace. La duración del retardo de cola para un determinado paquete dependerá del número de paquetes que hayan llegado antes a la cola y que estén esperando para ser transmitidos por el enlace. Si la cola está vacía y no se está transmitiendo ningún paquete actualmente, entonces el retardo de cola de nuestro paquete será cero. Por el contrario, si hay mucho tráfico y muchos paquetes están esperando también para ser transmitidos, el retardo de cola será grande. Como veremos en breve, el número de paquetes con que un paquete entrante puede esperar encontrarse es una función de la intensidad y de la naturaleza del tráfico que llega a la cola. En la práctica, los retardos de cola pueden ser del orden de microsegundos a milisegundos.

### *Retardo de transmisión*

Suponiendo que los paquetes se transmiten de manera que el primero que llega es el primero que sale, como suele ser común en las redes de conmutación de paquetes, nuestro paquete solo puede ser transmitido después de que todos los paquetes que hayan llegado antes que él hayan sido transmitidos. Sea la longitud del paquete igual a $L$ bits y la velocidad de transmisión del enlace del router A hasta el router B igual a $R$ bits/segundo. Por ejemplo, para un enlace Ethernet a 10 Mbps, la velocidad es $R = 10$ Mbps; para un enlace Ethernet a 100 Mbps, la velocidad será $R = 100$ Mbps. El **retardo de transmisión** será igual a $L/R$. Este es el tiempo necesario para introducir (es decir, transmitir) todos los bits del paquete en el enlace. Normalmente, en la práctica, los retardos de transmisión son del orden de microsegundos a milisegundos.

### *Retardo de propagación*

Una vez que un bit ha entrado en el enlace, tiene que propagarse hasta el router B. El tiempo necesario para propagarse desde el principio del enlace hasta el router B es el **retardo de propagación**. El bit se propaga a la velocidad de propagación del enlace, que depende del medio físico del enlace (es decir, de que el medio sea fibra óptica, cable de cobre de par trenzado, etc.) y está comprendido en el rango entre

$$2 \cdot 10^8 \text{ m/s y } 3 \cdot 10^8 \text{ m/s}$$

que es igual, o ligeramente inferior, a la velocidad de la luz. El retardo de propagación es igual a la distancia entre dos routers dividida por la velocidad de propagación. Es decir, el retardo de

propagación es igual a *d/s*, donde *d* es la distancia entre el router A y el router B y *s* es la velocidad de propagación del enlace. Una vez que el último bit del paquete se ha propagado hasta el nodo B, este y todos los bits anteriores del paquete se almacenan en el router B. A continuación, el proceso continúa, encargándose el router B de llevar a cabo el reenvío. En las redes de área extensa, los retardos de propagación son del orden de milisegundos.

Nota de vídeo

Exploración de los retardos de propagación y transmisión

### Comparación de los retardos de transmisión y de propagación

Los recién llegados al campo de las redes de computadoras en ocasiones tienen dificultades para comprender la diferencia entre el retardo de transmisión y el de propagación. Esta diferencia es sutil, pero importante. El retardo de transmisión es la cantidad de tiempo necesario para que el router saque fuera el paquete; es una función de la longitud del paquete y de la velocidad de transmisión del enlace, pero no tiene nada que ver con la distancia existente entre los dos routers. Por el contrario, el retardo de propagación es el tiempo que tarda un bit en propagarse de un router al siguiente; es una función de la distancia entre los dos routers, pero no tiene nada que ver con la longitud del paquete ni con la velocidad de transmisión del enlace.

Veamos una analogía que nos va a permitir clarificar los conceptos de retardo de transmisión y de retardo de propagación. Imagine una autopista en la que hay un puesto de peaje cada 100 kilómetros, como se muestra en la Figura 1.17. Podemos imaginar que los segmentos de autopista entre peajes son los enlaces y las casetas de peaje son los routers. Suponga que los automóviles viajan (es decir, se propagan) por la autopista a una velocidad de 100 km/hora (es decir, cuando un coche sale de un peaje, instantáneamente acelera hasta adquirir la velocidad de 100 km/hora y la mantiene entre puestos de peaje sucesivos). Supongamos ahora que hay 10 coches que viajan en caravana unos detrás de otros en un orden fijo. Podemos pensar que cada coche es un bit y que la caravana es un paquete. Supongamos también que cada puesto de peaje da servicio (es decir, transmite) a los coches a una velocidad de un coche cada 12 segundos y que es tarde por la noche, por lo que en la autopista solo se encuentra nuestra caravana de coches. Por último, supongamos que cuando el primer coche de la caravana llega a un peaje, espera en la entrada hasta que los otros nueve coches han llegado y se han detenido detrás de él (es decir, la caravana completa tiene que almacenarse en el peaje antes de poder ser reenviada). El tiempo necesario para que el peaje deje pasar a la caravana completa hacia el siguiente tramo de autopista es igual a (10 coches)/(5 coches/minuto) = 2 minutos. Este tiempo es análogo al retardo de transmisión de un router. El tiempo necesario para que un coche se desplace desde la salida del peaje hasta el siguiente puesto de peaje es 100 km/(100 km/hora) = 1 hora. Este tiempo es análogo al retardo de propagación. Por tanto, el tiempo que transcurre desde que la caravana queda colocada delante de un peaje hasta que vuelve a quedar colocada delante del siguiente peaje es la suma del tiempo de transmisión y el tiempo de propagación (en este caso, dicho tiempo será igual a 62 minutos).

Profundicemos un poco más en esta analogía. ¿Qué ocurriría si el tiempo de servicio del puesto de peaje para una caravana fuera mayor que el tiempo que tarda un coche en viajar de un peaje al siguiente? Por ejemplo, supongamos que los coches viajan a una velocidad de 1.000 km/hora y que los peajes operan a una velocidad de un coche por minuto. Entonces, el retardo correspondiente al hecho de desplazarse entre dos puestos de peaje será de 6 minutos y el tiempo que tarda el puesto de peaje en dar servicio a una caravana será de 10 minutos. En este caso, los primeros coches de la

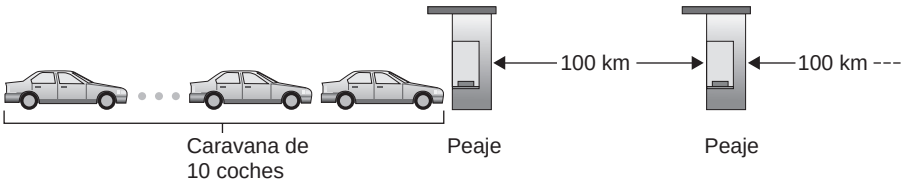



**Figura 1.17** ♦ Analogía de la caravana.

caravana llegarán al segundo puesto de peaje antes de que los últimos coches de la caravana hayan salido del primer peaje. Esta situación también se produce en las redes de conmutación de paquetes: los primeros bits de un paquete pueden llegar a un router mientras que muchos de los bits restantes del paquete todavía están esperando a ser transmitidos por el router anterior.

Si una imagen vale más que mil palabras, entonces una animación vale más que un millón de palabras. En el sitio web de este libro de texto se proporciona un applet Java interactivo que ilustra y compara convenientemente los retardos de transmisión y de propagación. Animamos a los lectores a visitar este applet. [Smith 2009] también proporciona una explicación bastante comprensible de los retardos de propagación, de cola y de transmisión.

Sean $d_{proc}$, $d_{cola}$, $d_{trans}$ y $d_{prop}$ los retardos de procesamiento, de cola, de transmisión y de propagación, respectivamente. Entonces el retardo nodal total estará dado por:

$$d_{nodal} = d_{proc} + d_{cola} + d_{trans} + d_{prop}$$

Las contribuciones de estos componentes de retardo pueden variar significativamente. Por ejemplo, $d_{prop}$ puede ser despreciable (digamos que un par de microsegundos) para un enlace que conecte dos routers del mismo campus universitario; sin embargo, $d_{prop}$ será de cientos de milisegundos para dos routers interconectados mediante un enlace vía satélite geoestacionario y puede llegar a ser el término dominante en $d_{nodal}$. Del mismo modo, $d_{trans}$ puede ser despreciable o significativo. Su contribución normalmente es despreciable para velocidades de transmisión de 10 Mbps y superiores (por ejemplo, para las redes LAN); sin embargo, puede ser igual a cientos de milisegundos para paquetes grandes de Internet enviados a través de enlaces de acceso telefónico que usen modems de baja velocidad. El retardo de procesamiento, $d_{proc}$, suele ser despreciable; sin embargo, tiene una gran influencia sobre la tasa de transferencia máxima del router, que es la velocidad máxima a la que un router puede reenviar los paquetes.

## 1.4.2 Retardo de cola y pérdida de paquetes

El componente más complejo e interesante del retardo nodal es el retardo de cola, $d_{cola}$. De hecho, el retardo de cola es tan importante e interesante en las redes de computadoras que se han escrito miles de artículos y numerosos libros sobre él [Bertsekas 1991; Daigle 1991; Kleinrock 1975, Kleinrock 1976; Ross 1995]. Aquí solo vamos a abordarlo de forma intuitiva y panorámica; los lectores más curiosos pueden echar un vistazo a algunos de los libros que se ocupan de este tema (¡o incluso pueden escribir una tesis doctoral sobre el asunto!). A diferencia de los otros tres retardos ($d_{proc}$, $d_{trans}$ y $d_{prop}$), el retardo de cola puede variar de un paquete a otro. Por ejemplo, si llegan 10 paquetes a una cola vacía al mismo tiempo, el primer paquete transmitido no sufrirá retardo de cola, mientras que el último paquete transmitido sufrirá un retardo de cola relativamente largo (mientras espera a que los restantes nueve paquetes sean transmitidos). Por tanto, al caracterizar el retardo de cola, suelen emplearse medidas estadísticas, como el retardo medio de cola, la varianza del retardo de cola y la probabilidad de que el retardo de cola exceda un cierto valor especificado.

¿En qué casos el retardo de cola es grande y en qué casos es insignificante? La respuesta a esta pregunta depende de la velocidad a la que llega el tráfico a la cola, de la velocidad de transmisión del enlace y de la naturaleza del tráfico entrante, es decir, de si el tráfico llega periódicamente o a ráfagas. Vamos a profundizar en este punto. Sea $a$ la velocidad media a la que llegan los paquetes a la cola ($a$ se expresa en paquetes/segundo). Recuerde que $R$ es la velocidad de transmisión; es decir, es la velocidad (en bits/segundo) a la que los bits salen de la cola. Con el fin de simplificar, supongamos también que todos los paquetes constan de $L$ bits. Entonces, la velocidad media a la que llegan los bits a la cola es igual a $La$ bits/segundo. Supongamos por último que la cola es muy grande, por lo que podemos decir que puede almacenar un número infinito de bits. El cociente $La/R$, denominado **intensidad de tráfico**, suele desempeñar un papel importante a la hora de estimar la magnitud del retardo de cola. Si $La/R > 1$, entonces la velocidad media a la que los bits llegan a la cola excede la velocidad a la que los bits pueden ser transmitidos desde la cola. En esta desafortunada situación, la cola tenderá a aumentar sin límite ¡y el retardo de cola se aproximará a infinito! Por tanto, una de las

reglas de oro en la ingeniería de tráfico es: *diseñe su sistema de modo que la intensidad de tráfico no sea mayor que 1*.

Veamos ahora el caso en que $La/R \leq 1$. Aquí, la naturaleza del tráfico entrante influye sobre el retardo de cola. Por ejemplo, si los paquetes llegan periódicamente —es decir, si llega un paquete cada $L/R$ segundos—, entonces todos los paquetes llegarán a una cola vacía y no habrá retardo de cola. Por el contrario, si los paquetes llegan a ráfagas pero de forma periódica, puede haber un retardo medio de cola significativo. Por ejemplo, supongamos que llegan simultáneamente $N$ paquetes cada $(L/R)N$ segundos. En este caso, el primer paquete transmitido no tiene asociado ningún retardo de cola, el segundo paquete transmitido presentará un retardo de cola de $L/R$ segundos y, de forma más general, el $n$-ésimo paquete transmitido presentará un retardo de cola de $(n-1)L/R$ segundos. Dejamos como ejercicio para el lector el cálculo del retardo medio de cola para este ejemplo.

Los dos ejemplos de llegada periódica de paquetes que acabamos de describir se corresponden con casos teóricos. Normalmente, el proceso de llegada a una cola es *aleatorio*; es decir, las llegadas no siguen ningún patrón y los paquetes quedan separados por periodos de tiempo aleatorios. En este caso más realista, la cantidad $La/R$ normalmente no es suficiente para caracterizar completamente las estadísticas del retardo de cola. Aun así, resulta útil para tener una idea intuitiva de la magnitud del retardo de cola. En particular, si la intensidad de tráfico es próxima a cero, entonces las llegadas de paquetes serán pocas y estarán bastante espaciadas entre sí, por lo que será improbable que un paquete que llegue a la cola se encuentre con que hay otro paquete en la cola. Por tanto, el retardo medio de cola será próximo a cero. Por el contrario, cuando la intensidad de tráfico es próxima a 1, habrá intervalos de tiempo en los que la velocidad de llegada exceda a la capacidad de transmisión (a causa de las variaciones en la velocidad de llegada de los paquetes), por lo que se formará una cola durante estos periodos de tiempo; cuando la velocidad de llegada sea menor que la capacidad de transmisión, la longitud de la cola disminuirá. Sin embargo, a medida que la intensidad de tráfico se aproxime a 1, la longitud media de la cola será cada vez mayor. La dependencia cualitativa del retardo medio de cola con respecto a la intensidad de tráfico se muestra en la Figura 1.18.

Un aspecto importante de la Figura 1.18 es el hecho de que, cuando la intensidad de tráfico se aproxima a 1, el retardo medio de cola aumenta rápidamente. Un pequeño porcentaje de aumento en la intensidad dará lugar a un incremento porcentual muy grande del retardo. Es posible que el lector haya experimentado este fenómeno en una autopista. Si viaja regularmente por una autopista que habitualmente está congestionada, quiere decir que la intensidad de tráfico en esa autopista es próxima a 1. En el caso de que se produzca un suceso que dé lugar a una cantidad de tráfico solo ligeramente mayor que la usual, los retardos que se experimenten pueden llegar a ser enormes.

Con el fin de que entienda bien lo que son los retardos de cola, animamos de nuevo al lector a visitar el sitio web dedicado a este libro, donde se proporciona un applet de Java interactivo que

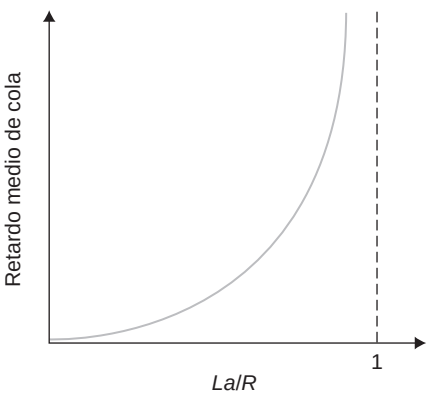



**Figura 1.18** ♦ Dependencia del retardo medio de cola con respecto a la intensidad de tráfico.

muestra una cola. Si establece una velocidad de llegada de los paquetes lo suficientemente alta como para que la intensidad de tráfico sea mayor que 1, comprobará que la cola aumenta lentamente con el tiempo.

### Pérdida de paquetes

En las explicaciones hasta ahora, hemos supuesto que la cola es capaz de almacenar un número infinito de paquetes. En la práctica, una cola para acceder a un enlace tiene una capacidad finita, aunque la capacidad de la cola depende fundamentalmente del diseño y del coste del router. Puesto que la capacidad de cola es finita, los retardos de los paquetes realmente no se aproximan a infinito a medida que la intensidad de tráfico se aproxima a 1. En su lugar, un paquete puede llegar y encontrarse con que la cola está llena. Al no tener sitio para almacenar dicho paquete, el router lo **elimina**; es decir, el paquete se **pierde**. Este desbordamiento de una cola puede verse también en el applet de Java, cuando la intensidad de tráfico es mayor que 1.

Desde el punto de vista de un sistema terminal, un paquete perdido es un paquete que ha sido transmitido al núcleo de la red pero que nunca sale de la red para llegar a su destino. El porcentaje de paquetes perdidos aumenta a medida que crece la intensidad de tráfico. Por tanto, el rendimiento de un nodo suele medirse no solo en función del retardo, sino también en función de la probabilidad de pérdida de paquetes. Como veremos en los siguientes capítulos, un paquete perdido puede retransmitirse de extremo a extremo para garantizar que todos los datos sean transferidos desde el origen hasta el destino.

## 1.4.3 Retardo extremo a extremo

Hasta el momento nos hemos centrado en el retardo nodal, es decir, el retardo en un único router. Ahora vamos a ocuparnos del retardo total entre el origen y el destino. Para entender este concepto, suponga que hay $N-1$ routers entre el host de origen y el host de destino. Suponga también, por el momento, que la red no está congestionada (por lo que los retardos de cola son despreciables), que el retardo de procesamiento en cada router y en el host de origen es $d_{\text{proc}}$, que la velocidad de transmisión de salida de cada router y del host de origen es de $R$ bits/segundo y que el retardo de propagación en cada enlace es igual a $d_{\text{prop}}$. Los retardos nodales se suman para proporcionar el retardo extremo a extremo, luego

$$d_{\text{extremo-extremo}} = N\,(d_{\text{proc}} + d_{\text{trans}} + d_{\text{prop}}) \tag{1.2}$$

donde, de nuevo, $d_{\text{trans}} = L/R$, siendo $L$ el tamaño del paquete. Observe que la Ecuación 1.2 es una generalización de la Ecuación 1.1, en la que no se tenían en cuenta los retardos de procesamiento y de propagación. Dejamos para el lector el ejercicio de generalizar esta fórmula para el caso en que los retardos en los nodos sean diferentes y exista un retardo medio de cola en cada nodo.

### Traceroute

Nota de vídeo

Utilización de Traceroute para descubrir rutas de red y medir el retardo de la red.

Para ver el orden de magnitud del retardo extremo a extremo de una red de computadoras, podemos utilizar el programa Traceroute. Se trata de un programa simple que se puede ejecutar en cualquier host de Internet. Cuando el usuario especifica un nombre de host de destino, el programa del host de origen envía al destino varios paquetes especiales. A medida que estos paquetes se dirigen a su destino, pasan a través de una serie de routers. Cuando un router recibe uno de estos paquetes especiales, devuelve al origen un mensaje corto que contiene el nombre y la dirección del router.

Más concretamente, suponga que hay $N-1$ routers entre el origen y el destino. Entonces el origen enviará $N$ paquetes especiales a la red, todos ellos dirigidos al destino final. Estos $N$ paquetes especiales se marcan de 1 (el primero) a $N$ (el último). Cuando el router $n$-ésimo recibe el paquete $n$-ésimo marcado como $n$, el router no reenvía el paquete hacia su destino, sino que devuelve un

mensaje al origen. Cuando el host de destino recibe el paquete *N*-ésimo, también devuelve un mensaje al origen. El origen registra el tiempo transcurrido entre el momento en que envió un paquete y el momento en que recibe el correspondiente mensaje de respuesta; también registra el nombre y la dirección del router (o del host de destino) que devuelve el mensaje. De esta forma, el origen puede reconstruir la ruta seguida por los paquetes que fluyen desde el origen hasta el destino, y puede también determinar los retardos de ida y vuelta para todos los routers que intervienen en el proceso. En la práctica, Traceroute repite el proceso que acabamos de describir tres veces, de modo que el origen realmente envía 3 • *N* paquetes al destino. El documento RFC 1393 describe en detalle el programa Traceroute.

He aquí un ejemplo de la salida proporcionada por el programa Traceroute, en el que se ha trazado la ruta desde el host de origen gaia.cs.umass.edu (en la Universidad de Massachusetts) al host cis.poly.edu (en la Universidad Politécnica de Brooklyn). La salida consta de seis columnas: la primera de ellas contiene el valor *n* descrito anteriormente, es decir, el número del router a lo largo de la ruta; la segunda columna especifica el nombre del router; la tercera indica la dirección del router (con el formato xxx.xxx.xxx.xxx); las tres últimas columnas especifican los retardos de ida y vuelta correspondientes a los tres experimentos. Si el origen recibe menos de tres mensajes de cualquier router (debido a la pérdida de paquetes en la red), Traceroute incluye un asterisco justo después del número de router y proporciona menos de tres tiempos de ida y vuelta para dicho router.

```
1  cs-gw (128.119.240.254) 1.009 ms 0.899 ms 0.993 ms
2  128.119.3.154 (128.119.3.154) 0.931 ms 0.441 ms 0.651 ms
3  -border4-rt-gi-1-3.gw.umass.edu (128.119.2.194) 1.032 ms 0.484 ms 0.451 ms
4  -acr1-ge-2-1-0.Boston.cw.net (208.172.51.129) 10.006 ms 8.150 ms 8.460 ms
5  -agr4-loopback.NewYork.cw.net (206.24.194.104) 12.272 ms 14.344 ms 13.267 ms
6  -acr2-loopback.NewYork.cw.net (206.24.194.62) 13.225 ms 12.292 ms 12.148 ms
7  -pos10-2.core2.NewYork1.Level3.net (209.244.160.133) 12.218 ms 11.823 ms 11.793 ms
8  -gige9-1-52.hsipaccess1.NewYork1.Level3.net (64.159.17.39) 13.081 ms 11.556 ms 13.297 ms
9  -p0-0.polyu.bbnplanet.net (4.25.109.122) 12.716 ms 13.052 ms 12.786 ms
10 cis.poly.edu (128.238.32.126) 14.080 ms 13.035 ms 12.802 ms
```

Podemos ver en esta traza que existen nueve routers entre el origen y el destino. La mayor parte de estos routers tiene un nombre y todos ellos tienen direcciones. Por ejemplo, el nombre del Router 3 es `border4-rt-gi-1-3.gw.umass.edu` y su dirección es `128.119.2.194`. Si observamos los datos proporcionados para este mismo router, vemos que en la primera de las tres pruebas el retardo de ida y vuelta entre el origen y el router ha sido de 1,03 milisegundos. Los retardos de ida y vuelta para las dos pruebas siguientes han sido 0,48 y 0,45 milisegundos, respectivamente. Estos retardos de ida y vuelta contienen todos los retardos que acabamos de estudiar, incluyendo los retardos de transmisión, de propagación, de procesamiento del router y de cola. Puesto que el retardo de cola varía con el tiempo, el retardo de ida y vuelta del paquete *n* envíado al router *n* puede, en ocasiones, ser mayor que el retardo de ida y vuelta del paquete $n + 1$ enviado al router $n + 1$. De hecho, puede observar este fenómeno en el ejemplo anterior: ¡los retardos correspondientes al Router 6 son mayores que los correspondientes al Router 7!

¿Desea probar el programa Traceroute? Le recomendamos *vivamente* que visite el sitio http://www.traceroute.org, donde se proporciona una interfaz web a una extensa lista de orígenes para el trazado de rutas. Seleccione un origen y especifique el nombre de host de cualquier destino. El programa Traceroute hará entonces todo el trabajo. Hay disponibles diversos programas software gratuitos que proporcionan una interfaz gráfica para Traceroute; uno de nuestros programa favoritos es PingPlotter [PingPlotter 2016].

### Retardos de los sistemas terminales, de las aplicaciones y otros

Además de los retardos de procesamiento, de transmisión y de propagación, en los sistemas terminales pueden existir retardos adicionales significativos. Por ejemplo, un sistema terminal que quiera transmitir un paquete a través de un medio compartido (por ejemplo, como sucede en un escenario que incluya WiFi o un módem por cable) puede retardar su transmisión *a propósito* como parte de su protocolo, para compartir el medio con otros sistemas terminales; en el Capítulo 6 analizaremos en detalle tales protocolos. Otro retardo importante es el retardo de empaquetamiento del medio, que aparece en las aplicaciones de Voz sobre IP (VoIP, *Voice-over-IP*). En VoIP, el lado emisor debe, en primer lugar, rellenar un paquete con voz digitalizada codificada antes de pasar el paquete a Internet. El tiempo que se tarda en rellenar un paquete (denominado retardo de empaquetamiento) puede ser significativo y puede repercutir en la calidad percibida por el usuario de una llamada VoIP. Este problema se abordará más detalladamente en uno de los problemas del final del capítulo.

## 1.4.4 Tasa de transferencia en las redes de computadoras

Además de los retardos y la pérdida de paquetes, otra medida crítica de rendimiento de las redes de computadoras es la tasa de transferencia de extremo a extremo. Para definir la tasa de transferencia, consideremos la transferencia de un archivo de gran tamaño desde el host A al host B a través de una red. Por ejemplo, esta transferencia podría consistir en transferir un clip de vídeo de gran tamaño desde un par (*peer*) a otro en un sistema de compartición de archivos P2P. La **tasa de transferencia instantánea** en cualquier instante de tiempo es la velocidad (en bits/segundo) a la que el host B recibe el archivo. (Muchas aplicaciones, incluyendo muchos sistemas de compartición de archivos P2P, muestran la tasa de transferencia instantánea durante las descargas en la interfaz de usuario; seguro que el lector ya se ha fijado anteriormente en este detalle.) Si el archivo consta de $F$ bits y la transferencia dura $T$ segundos hasta que el host B recibe los $F$ bits, entonces la **tasa media de transferencia** del archivo es igual a $F/T$ bits/segundo. En algunas aplicaciones, tales como la telefonía por Internet, es deseable tener un retardo pequeño y una tasa de transferencia instantánea que esté constantemente por encima de un cierto umbral (por ejemplo, por encima de 24 kbps para ciertas aplicaciones de telefonía por Internet y por encima de 256 kbps para las aplicaciones de vídeo en tiempo real). Para otras aplicaciones, entre las que se incluyen aquéllas que implican la transferencia de archivos, el retardo no es crítico, pero es deseable que la tasa de transferencia sea lo más alta posible.

Con el fin de comprender mejor el importante concepto de tasa de transferencia, vamos a ver algunos ejemplos. La Figura 1.19(a) muestra dos sistemas terminales, un servidor y un cliente, conectados mediante dos enlaces de comunicaciones y un router. Consideremos la tasa de transferencia para transmitir un archivo desde el servidor al cliente. Sea $R_s$ la velocidad del enlace entre el servidor y el router, y sea $R_c$ la velocidad del enlace entre el router y el cliente. Supongamos que los únicos bits que están siendo enviados a través de la red son los que van desde el servidor al cliente. En este escenario ideal, ¿cuál es la tasa de transferencia del servidor al cliente? Para responder a esta pregunta, podemos pensar en los bits como en un *fluido* y en los enlaces de comunicaciones como si fueran *tuberías*. Evidentemente, el servidor no puede bombear los bits a través de su enlace a una velocidad mayor que $R_s$ bps; y el router no puede reenviar los bits a una velocidad mayor que $R_c$ bps. Si $R_s < R_c$, entonces los bits bombeados por el servidor "fluirán" a través del router y llegarán al cliente a una velocidad de $R_s$ bps, obteniéndose una tasa de transferencia de $R_s$ bps. Si, por el contrario, $R_c < R_s$, entonces el router no podrá reenviar los bits tan rápidamente como los recibe. En este caso, los bits abandonarán el router a una velocidad de solo $R_c$, dando lugar a una tasa de transferencia de extremo a extremo igual a $R_c$. (Observe también que si continúan llegando bits al router a una velocidad $R_s$, y siguen abandonando el router a una velocidad igual a $R_c$, la cantidad de bits en el router que están esperando a ser transmitidos hacia el cliente aumentará constantemente, lo que es una situación nada deseable.) Por tanto, en esta sencilla red de dos enlaces, la tasa de transferencia es mín$\{R_c, R_s\}$, es decir, es igual a la velocidad de transmisión del **enlace cuello**

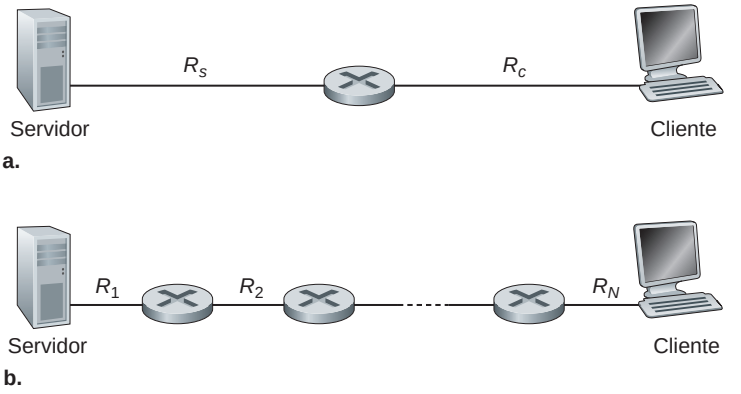



**Figura 1.19** ♦ Tasa de transferencia para la transmisión de un archivo desde un servidor a un cliente.

**de botella**. Una vez determinada la tasa de transferencia, podemos aproximar el tiempo que se tardará en transferir un archivo de gran tamaño de $F$ bits desde el servidor al cliente, mediante la fórmula $F/\text{mín}\{R_s, R_c\}$. Veamos un ejemplo concreto. Suponga que está descargando un archivo MP3 de $F = 32$ millones de bits, que el servidor tiene una velocidad de transmisión $R_s = 2$ Mbps y que disponemos de un enlace de acceso con $R_c = 1$ Mbps. El tiempo necesario para transferir el archivo será igual a 32 segundos. Por supuesto, estas expresiones para la tasa de transferencia y el tiempo de transferencia son únicamente aproximaciones, ya que no se han tenido en cuenta los retardos de almacenamiento y reenvío y de procesamiento, ni las cuestiones relativas al protocolo.

La Figura 1.19(b) muestra una red con $N$ enlaces entre el servidor y el cliente, siendo las velocidades de transmisión de los $N$ enlaces iguales a $R_1, R_2,..., R_N$. Aplicando el mismo análisis que para la red de dos enlaces, podemos determinar que la tasa de transferencia para enviar un archivo desde el servidor al cliente es $\text{mín}\{R_1, R_2,...,R_N\}$, que es de nuevo la velocidad de transmisión del enlace cuello de botella existente en la ruta entre el servidor y el cliente.

Veamos ahora otro ejemplo inspirado en la red Internet de hoy día. La Figura 1.20(a) muestra dos sistemas terminales, un servidor y un cliente, conectados a una red de computadoras. Considere la tasa de transferencia para transmitir un archivo desde el servidor al cliente. El servidor está conectado a la red mediante un enlace de acceso cuya velocidad es $R_s$ y el cliente está conectado a la red mediante un enlace de acceso de velocidad $R_c$. Supongamos ahora que todos los enlaces existentes en el núcleo de la red de comunicaciones tienen velocidades de transmisión muy altas, muy por encima de $R_s$ y $R_c$. Ciertamente, hoy en día, el núcleo de Internet está sobredimensionado, con enlaces de alta velocidad que experimentan una congestión muy baja. Supongamos también que únicamente se están enviando a través de la red esos bits que se transfieren desde el servidor al cliente. Dado que el núcleo de la red es como una tubería ancha en este ejemplo, la velocidad a la que los bits pueden fluir desde el origen hasta el destino es, de nuevo, el mínimo de $R_s$ y $R_c$, es decir, tasa de transferencia $= \text{mín}\{R_s, R_c\}$. Por tanto, en la actualidad, el factor de restricción de la tasa de transferencia en Internet es, normalmente, la red de acceso.

Veamos un último ejemplo. Vamos a utilizar la Figura 1.20(b); en ella vemos que hay 10 servidores y 10 clientes conectados al núcleo de la red de computadoras. En este ejemplo, tienen lugar 10 descargas simultáneas, lo que implica a 10 pares cliente-servidor. Supongamos que estas 10 descargas son el único tráfico existente en la red en este momento. Como se muestra en la figura, hay un enlace en el núcleo que es atravesado por las 10 descargas. Sea $R$ la velocidad de transmisión de este enlace $R$. Supongamos que los enlaces de acceso de todos los servidores tienen la misma velocidad $R_s$, que los enlaces de acceso de todos los clientes tienen la misma velocidad $R_c$ y que las velocidades de transmisión de todos los enlaces del núcleo (excepto el enlace común de velocidad $R$) son mucho mayores que $R_s$, $R_c$ y $R$. Ahora, deseamos saber cuáles son las tasas de transferencia de las descargas. Evidentemente, si la velocidad del enlace común, $R$,

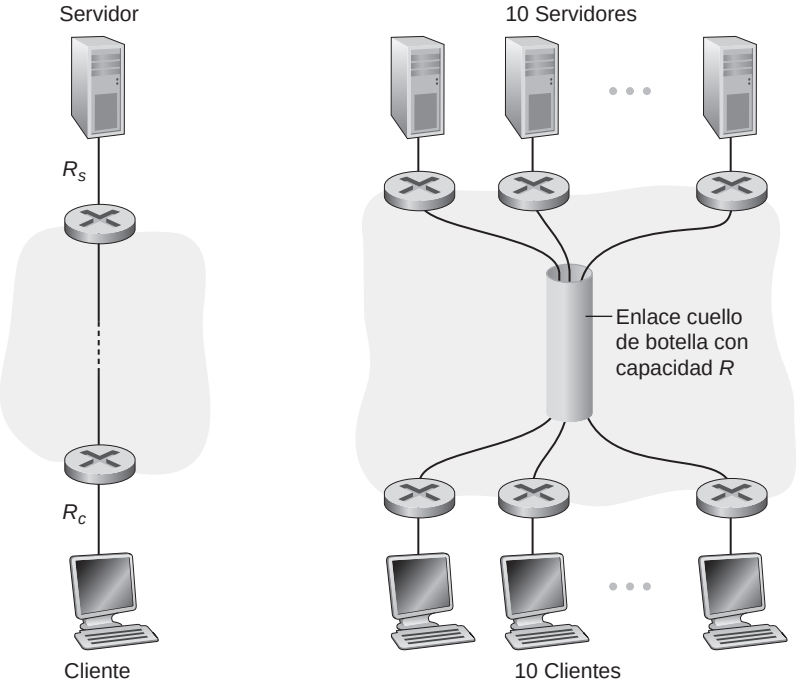



**Figura 1.20** ♦ Tasa de transferencia terminal a terminal: (a) un cliente descarga un archivo de un servidor; (b) 10 clientes descargando con 10 servidores.

es grande (por ejemplo, cien veces mayor que $R_s$ y $R_c$), entonces la tasa de transferencia de cada descarga será igual, de nuevo, a mín$\{R_s, R_c\}$. Pero, ¿qué ocurre si la velocidad del enlace común es del mismo orden que $R_s$ y $R_c$? ¿Cuál será la tasa de transferencia en este caso? Veamos un ejemplo concreto. Supongamos que $R_s = 2$ Mbps, $R_c = 1$ Mbps, $R = 5$ Mbps y que el enlace común divide su velocidad de transmisión en partes iguales entre las 10 descargas. Entonces, el cuello de botella para cada descarga ya no se encuentra en la red de acceso, sino en el enlace compartido del núcleo, que solo proporciona una tasa de transferencia de 500 kbps a cada descarga. Luego la tasa de transferencia extremo a extremo para cada descarga ahora se habrá reducido a 500 kbps.

Los ejemplos de las Figuras 1.19 y 1.20(a) demuestran que la tasa de transferencia depende de las velocidades de transmisión de los enlaces a través de los que fluyen los datos. Hemos visto que cuando no existe ningún otro tráfico, la tasa de transferencia puede simplemente aproximarse mediante la velocidad mínima de transmisión a lo largo de la ruta entre el origen y el destino. El ejemplo de la Figura 1.20(b) demuestra que, en el caso más general, la tasa de transferencia depende no solo de las velocidades de transmisión de los enlaces a lo largo de la ruta, sino también del tráfico existente. En particular, un enlace con una velocidad de transmisión alta puede ser, de todos modos, el enlace cuello de botella para la transferencia de un archivo si hay muchos otros flujos de datos atravesando también ese enlace. Examinaremos más detalladamente la tasa de transferencia de las redes de computadoras en los problemas de repaso y en los capítulos siguientes.

## 1.5 Capas de protocolos y sus modelos de servicio

Después de lo que hemos visto hasta aquí, resulta evidente que Internet es un sistema *extremadamente* complicado. Hemos visto que son muchas las piezas que conforman Internet: numerosas aplicaciones y protocolos, distintos tipos de sistemas terminales, dispositivos de conmutación de

paquetes y diversos tipos de medios para los enlaces. Dada esta enorme complejidad, ¿tenemos alguna esperanza de poder organizar una arquitectura de red o al menos nuestra exposición sobre la misma? Afortunadamente, la respuesta a ambas preguntas es sí.

### 1.5.1 Arquitectura en capas

Antes de intentar organizar nuestras ideas sobre la arquitectura de Internet, vamos a ver una analogía humana. Realmente, de forma continua estamos tratando con sistemas complejos en nuestra vida cotidiana. Imagine que alguien le pide que describa, por ejemplo, cómo funciona el sistema de líneas aéreas. ¿Qué estructura utilizaría para describir este complejo sistema que emplea vendedores de billetes, personal de facturación de equipajes, personal para las puertas de embarque, pilotos, aviones, control de tráfico aéreo y un sistema de ámbito mundial para dirigir los aviones? Una forma de describir este sistema podría consistir en describir la serie de acciones que usted realiza (o que otros realizan para usted) cuando vuela en un avión. En primer lugar, usted compra un billete, luego factura el equipaje, se dirige a la puerta de embarque y por último sube al avión. El avión despega y se dirige a su destino. Una vez que el avión ha tomado tierra, usted desembarca y recoge su equipaje. Si el viaje ha sido malo, se quejará de ello en el mostrador de venta de billetes (lo que, por supuesto, no le servirá de nada). Este escenario se muestra en la Figura 1.21.

Podemos ver inmediatamente algunas analogías con las redes de computadoras: la compañía área le traslada desde un origen hasta un destino, al igual que Internet trasporta un paquete desde un host de origen hasta un host de destino. Pero esta no es la analogía que estábamos buscando. Lo que queremos es encontrar una cierta *estructura* en la Figura 1.21. Si nos fijamos en esta figura, observaremos que hay una función Billete en cada extremo; también existe una función Equipaje para los pasajeros que tienen un billete y una función Embarque para los pasajeros que tienen billete y han facturado su equipaje. Para los pasajeros que han embarcado (es decir, que han adquirido un billete, han facturado las maletas y han embarcado), existe una función de despegue y aterrizaje y, durante el vuelo, hay una función de control del avión. Esto sugiere que podemos fijarnos en las funcionalidades señaladas en la Figura 1.21 de forma *horizontal*, como se muestra en la Figura 1.22.

En la Figura 1.22 se han separado las distintas funciones de la compañía aérea en capas, proporcionando un marco de trabajo en el que podemos explicar cómo se realiza un viaje en avión. Observe que cada capa, combinada con las capas que tiene por debajo, implementa una cierta funcionalidad, un cierto *servicio*. En las capas Billete e inferiores se lleva a cabo la transferencia de una persona de un mostrador de línea aérea a otro. En las capas Equipaje e inferiores se realiza la transferencia de una persona y su equipaje desde el punto de facturación hasta la recogida del equipaje. Observe que la capa Equipaje solo proporciona este servicio a las personas que ya han

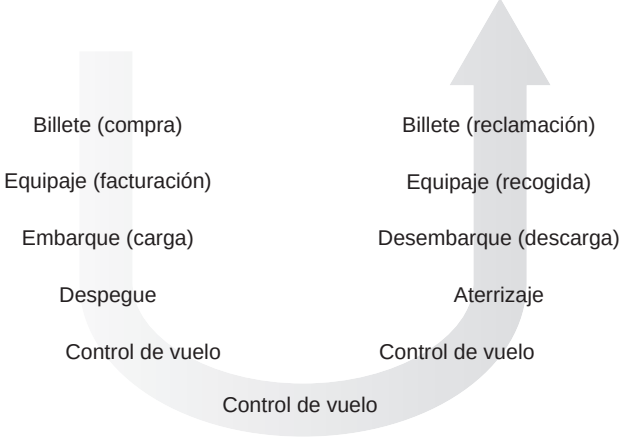



**Figura 1.21** ♦ Acciones para realizar un viaje en avión.

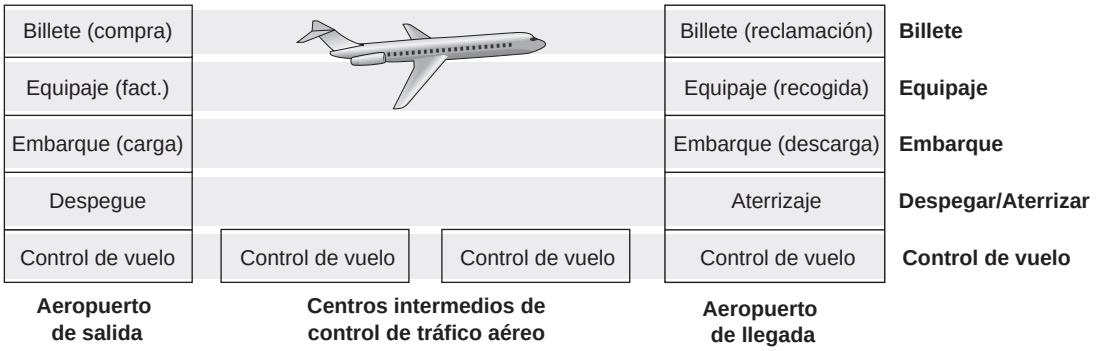



**Figura 1.22** ♦ Disposición horizontal de capas para las funcionalidades de una compañía área.

adquirido un billete. En la capa Embarque, se realiza la transferencia embarque/desembarque de una persona y su equipaje. En la capa Despegue/ Aterrizaje, se realiza la transferencia pista a pista de personas y equipajes. Cada capa proporciona su servicio (1) llevando a cabo determinadas acciones dentro de dicha capa (por ejemplo, en la capa Embarque, se hace subir y bajar al pasaje del avión) y (2) utilizando los servicios de la capa que tiene directamente debajo de ella (por ejemplo, en la capa Embarque, utilizando el servicio de transferencia de pasajeros pista a pista de la capa Despegue/ Aterrizaje).

Una arquitectura de capas nos permite estudiar una parte específica y bien definida de un sistema más grande y complejo. Esta simplificación tiene un valor considerable por sí misma, al proporcionar modularidad, haciendo mucho más fácil modificar la implementación del servicio suministrado por la capa. Mientras que la capa proporcione el mismo servicio a la capa que tiene por encima de ella y emplee los mismos servicios de la capa que tiene por debajo, el resto del sistema permanecerá invariable cuando se modifique la implementación de una capa. (¡Tenga en cuenta que cambiar la implementación de un servicio es muy diferente a cambiar el propio servicio!) Por ejemplo, si se modificara la función Embarque para que las personas embarcaran y desembarcaran por alturas, el resto del sistema de la compañía aérea no se vería afectado, ya que la capa Embarque continuaría llevando a cabo la misma función (cargar y descargar personas); simplemente, implementa dicha función de una forma diferente después de realizar el cambio. En sistemas complejos de gran tamaño que se actualizan constantemente, la capacidad de modificar la implementación de un servicio sin afectar al resto de los componentes del sistema es otra importante ventaja de la disposición en capas.

### Capas de protocolos

Pero ya hemos hablado suficiente de compañías aéreas. Dirijamos ahora nuestra atención a los protocolos de red. Para proporcionar una estructura al diseño de protocolos de red, los diseñadores de redes organizan los protocolos (y el hardware y el software de red que implementan los protocolos) en **capas**. Cada protocolo pertenece a una de las capas, del mismo modo que cada función en la arquitectura de la compañía aérea de la Figura 1.22 pertenecía a una capa. De nuevo, estamos interesados en los **servicios** que ofrece una capa a la capa que tiene por encima, lo que se denomina **modelo de servicio** de la capa. Como en el caso del ejemplo de la compañía aérea, cada capa proporciona su servicio (1) llevando a cabo ciertas acciones en dicha capa y (2) utilizando los servicios de la capa que tiene directamente debajo de ella. Por ejemplo, los servicios proporcionados por la capa $n$ pueden incluir la entrega fiable de mensajes de un extremo de la red al otro. Esto podría implementarse mediante un servicio no fiable de entrega de mensajes extremo a extremo de la capa $n - 1$, y añadiendo una funcionalidad de la capa $n$ para detectar y retransmitir los mensajes perdidos.

Una capa de protocolo puede implementarse por software, por hardware o mediante una combinación de ambos. Los protocolos de la capa de aplicación, como HTTP y SMTP, casi siempre

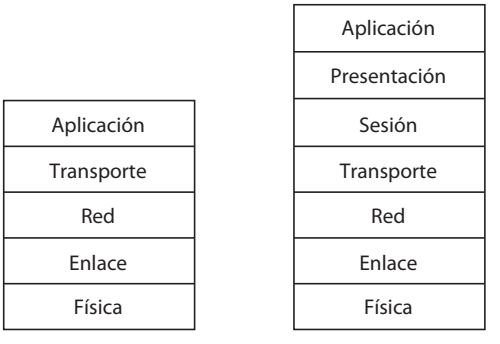
se implementan por software en los sistemas terminales, al igual que los protocolos de la capa de transporte. Puesto que la capa física y las capas de enlace de datos son responsables de manejar la comunicación a través de un enlace específico, normalmente se implementan en las tarjetas de interfaz de red (por ejemplo, tarjetas de interfaz Ethernet o WiFi) asociadas con un determinado enlace. La capa de red a menudo es una implementación mixta de hardware y software. Observe también que, al igual que las funciones de la arquitectura de la compañía aérea estaban distribuidas entre los distintos aeropuertos y centros de control de vuelo que formaban el sistema, también un protocolo de capa *n* está *distribuido* entre los sistemas terminales, los dispositivos de conmutación de paquetes y los restantes componentes que conforman la red. Es decir, suele haber una parte del protocolo de capa *n* en cada uno de estos componentes de red.

Las capas de protocolos presentan ventajas conceptuales y estructurales [RFC 3439]. Como hemos visto, las capas proporcionan una forma estructurada de estudiar los componentes del sistema. Además, la modularidad facilita la actualización de los componentes del sistema. Sin embargo, es preciso señalar que algunos investigadores e ingenieros de redes se oponen vehementemente a la estructura de capas [Wakeman 1992]. Un potencial inconveniente de la estructura de capas es que una capa puede duplicar la funcionalidad de la capa inferior. Por ejemplo, muchas pilas de protocolos proporcionan una función de recuperación de errores tanto para cada enlace, como extremo a extremo. Un segundo inconveniente potencial es que la funcionalidad de una capa puede precisar información (por ejemplo, un valor de una marca temporal) que solo existe en otra capa, y esto viola el objetivo de la separación en capas.

Cuando los protocolos de las distintas capas se toman en conjunto se denominan **pila de protocolos**. La pila de protocolos de Internet consta de cinco capas: capa física, capa de enlace, capa de red, capa de transporte y capa de aplicación, como se muestra en la Figura1.23(a). Si examina el Contenido, comprobará que hemos organizado a grandes rasgos el libro utilizando las capas de la pila de protocolos de Internet. Vamos a aplicar un **enfoque descendente**, abordando en primer lugar la capa de aplicación y continuando hacia abajo por la pila.

### Capa de aplicación

La capa de aplicación es donde residen las aplicaciones de red y sus protocolos de nivel de aplicación. La capa de aplicación de Internet incluye muchos protocolos, tales como el protocolo HTTP (que permite la solicitud y transferencia de documentos web), SMTP (que permite la transferencia de mensajes de correo electrónico) y FTP (que permite la transferencia de archivos entre dos sistemas terminales). Tendremos oportunidad de ver que determinadas funciones de red, como la traducción de los nombres legibles que utilizamos las personas para los sistemas terminales de Internet (por ejemplo, www.ietf.org) en direcciones de red de 32 bits se realiza también con la ayuda de un




**a. Pila de protocolos de Internet de cinco capas** **b. Modelo de referencia OSI de ISO de siete capas**

**Figura 1.23** ♦ Pila de protocolos de Internet (a) y modelo de referencia OSI (b).

protocolo específico de la capa de aplicación: en concreto, mediante el Sistema de nombres de dominio (DNS, *Domain Name System*). En el Capítulo 2 veremos que es muy fácil crear e implantar nuestros propios protocolos nuevos de la capa de aplicación.

Un protocolo de la capa de aplicación está distribuido entre varios sistemas terminales, con la aplicación en un sistema terminal utilizando el protocolo para intercambiar paquetes de información con la aplicación que se ejecuta en otro sistema terminal. A este paquete de información de la capa de aplicación lo denominaremos **mensaje**.

### Capa de transporte

La capa de transporte de Internet transporta los mensajes de la capa de aplicación entre los puntos terminales de la aplicación. En Internet, existen dos protocolos de transporte, TCP y UDP, pudiendo cada uno de ellos transportar los mensajes de la capa de aplicación. TCP ofrece a sus aplicaciones un servicio orientado a la conexión. Este servicio incluye el suministro garantizado de los mensajes de la capa de aplicación al destino y un mecanismo de control del flujo (es decir, una adaptación de las velocidades del emisor y el receptor). TCP también divide los mensajes largos en segmentos más cortos y proporciona un mecanismo de control de congestión, de manera que un emisor regula su velocidad de transmisión cuando la red está congestionada. El protocolo UDP proporciona a sus aplicaciones un servicio sin conexión. Es un servicio básico que no ofrece ninguna fiabilidad, ni control de flujo, ni control de congestión. En este libro, denominaremos a los paquetes de la capa de transporte **segmentos**.

### Capa de red

La capa de red de Internet es responsable de trasladar los paquetes de la capa de red, conocidos como **datagramas**, de un host a otro. El protocolo de la capa de transporte Internet (TCP o UDP) de un host de origen pasa un segmento de la capa de transporte y una dirección de destino a la capa de red, al igual que nosotros damos al servicio de correo postal una carta con una dirección de destino. Luego, la capa de red proporciona el servicio de suministrar el segmento a la capa de transporte del host de destino.

La capa de red de Internet incluye el conocido protocolo IP, que define los campos del datagrama, así como la forma en que actúan los sistemas terminales y los routers sobre estos campos. Existe un único protocolo IP y todos los componentes de Internet que tienen una capa de red deben ejecutar el protocolo IP. La capa de red de Internet también contiene protocolos de enrutamiento, que determinan las rutas que los datagramas siguen entre los orígenes y los destinos. Internet dispone de muchos protocolos de enrutamiento. Como hemos visto en la Sección 1.3, Internet es una red de redes y, dentro de una red, el administrador de la red puede ejecutar cualquier protocolo de enrutamiento que desee. Aunque la capa de red contiene tanto el protocolo IP como numerosos protocolos de enrutamiento, suele hacerse referencia a ella simplemente como la capa IP, lo que refleja el hecho de que IP es el pegamento que une toda la red Internet.

### Capa de enlace

La capa de red de Internet encamina un datagrama a través de una serie de routers entre el origen y el destino. Para trasladar un paquete de un nodo (host o router) al siguiente nodo de la ruta, la capa de red confía en los servicios de la capa de enlace. En concreto, en cada nodo, la capa de red pasa el datagrama a la capa de enlace, que entrega el datagrama al siguiente nodo existente a lo largo de la ruta. En ese siguiente nodo, la capa de enlace pasa el datagrama a la capa de red.

Los servicios proporcionados por la capa de enlace dependen del protocolo concreto de la capa de enlace que se emplee en el enlace. Por ejemplo, algunos protocolos de la capa de enlace proporcionan una entrega fiable desde el nodo transmisor, a través de un enlace y hasta el nodo receptor.

Observe que este servicio de entrega fiable es diferente del servicio de entrega fiable de TCP, que lleva a cabo una entrega fiable desde un sistema terminal a otro. Entre los ejemplos de protocolos de la capa de enlace se incluyen Ethernet, WiFi y el protocolo DOCSIS de la red de acceso por cable. Puesto que normalmente los datagramas necesitan atravesar varios enlaces para viajar desde el origen hasta el destino, un datagrama puede ser manipulado por distintos protocolos de la capa de enlace en los distintos enlaces disponibles a lo largo de la ruta. Por ejemplo, un datagrama puede ser manipulado por Ethernet en un enlace y por PPP en el siguiente enlace. La capa de red recibirá un servicio diferente por parte de cada uno de los distintos protocolos de la capa de enlace. En este libro, denominaremos a los paquetes de la capa de enlace **tramas**.

### Capa física

Mientras que el trabajo de la capa de enlace es mover tramas completas de un elemento de la red hasta el elemento de red adyacente, el trabajo de la capa física consiste en mover de un nodo al siguiente los *bits individuales* que forman la trama. Los protocolos de esta capa son, de nuevo, dependientes del enlace y, por tanto, dependen del medio de transmisión del enlace (por ejemplo, cable de cobre de par trenzado o fibra óptica monomodo). Por ejemplo, Ethernet dispone de muchos protocolos de la capa física: uno para cable de cobre de par trenzado, otro para cable coaxial, otro para fibra, etc. En cada caso, los bits se desplazan a través del enlace de forma diferente.

### El modelo OSI

Una vez vista en detalle la pila de protocolos de Internet, deberíamos mencionar que no es la única pila de protocolos existente. En concreto, a finales de la década de 1970, la Organización Internacional de Estandarización (ISO, *International Organization for Standardization*) propuso que las redes de computadoras fueran organizadas utilizando siete capas, en lo que se vino a denominar modelo OSI (*Open Systems Interconnection*, Interconexión de sistemas abiertos) [ISO 2016]. El modelo OSI tomó forma cuando los protocolos que se convertirían en los protocolos de Internet estaban en su infancia y eran simplemente uno de los muchos conjuntos de protocolos diferentes que estaban en desarrollo; de hecho, probablemente los inventores del modelo OSI original no estaban pensando en Internet cuando lo crearon. No obstante, a partir de finales de la década de 1970, muchos cursos universitarios y de formación, siguiendo las recomendaciones de ISO, organizaron cursos sobre el modelo de siete capas. A causa de su temprano impacto sobre la formación en redes, el modelo de siete capas todavía perdura en algunos libros de texto y cursos de formación sobre redes.

Las siete capas del modelo de referencia OSI, mostrado en la Figura 1.23(b), son: capa de aplicación, capa de presentación, capa de sesión, capa de transporte, capa de red, capa de enlace de datos y capa física. La funcionalidad de cinco de estas capas es básicamente la misma que sus contrapartidas del mismo nombre de Internet. Por tanto, vamos a centrarnos en las dos capas adicionales del modelo de referencia OSI: la capa de presentación y la capa de sesión. La función de la capa de presentación es la de proporcionar servicios que permitan a las aplicaciones que se comunican interpretar el significado de los datos intercambiados. Estos servicios incluyen la compresión y el cifrado de los datos (funciones cuyos nombres son autoexplicativos), así como la descripción de los datos (lo que libera a la aplicación de tener que preocuparse por el formato interno en el que los datos se representan y almacenan, formatos que pueden diferir de una computadora a otra). La capa de sesión permite delimitar y sincronizar el intercambio de datos, incluyendo los medios para crear un punto de restauración y un esquema de recuperación.

El hecho de que en Internet falten dos de las capas existentes en el modelo de referencia OSI plantea un par de cuestiones interesantes: ¿acaso los servicios proporcionados por estas dos capas no son importantes? ¿Qué ocurre si una aplicación *necesita* uno de estos servicios? La respuesta de Internet a ambas preguntas es la misma: es problema del desarrollador de la aplicación. El desarrollador de la aplicación tiene que decidir si un servicio es importante y *si lo es*, será su problema el incorporar dicha funcionalidad a la aplicación.

### 1.5.2 Encapsulación

La Figura 1.24 muestra la ruta física que siguen los datos al descender por la pila de protocolos de un sistema terminal emisor, al ascender y descender por las pilas de protocolos de un switch de la capa de enlace y de un router, para finalmente ascender por la pila de protocolos del sistema terminal receptor. Como veremos más adelante en el libro, los routers y los switches de la capa de enlace operan como dispositivos de conmutación de paquetes. De forma similar a los sistemas terminales, los routers y los switches de la capa de enlace organizan su hardware y software de red en capas. Pero estos dispositivos no implementan *todas* las capas de la pila de protocolos; habitualmente solo implementan las capas inferiores. Como se muestra en la Figura 1.24, los switches de la capa de enlace implementan las capas 1 y 2; y los routers implementan las capas 1 a 3. Esto significa, por ejemplo, que los routers de Internet son capaces de implementar el protocolo IP (un protocolo de la capa 3), mientras que los switches de la capa de enlace no. Veremos más adelante que aunque los switches de la capa de enlace no reconocen las direcciones IP, pueden reconocer las direcciones de la capa 2, como por ejemplo las direcciones Ethernet. Observe que los hosts implementan las cinco capas, lo que es coherente con la idea de que la arquitectura de Internet es mucho más compleja en las fronteras de la red.

La Figura 1.24 también ilustra el importante concepto de **encapsulación**. En el host emisor, un **mensaje de la capa de aplicación** (M en la Figura 1.24) se pasa a la capa de transporte. En el caso más simple, la capa de transporte recibe el mensaje y añade información adicional (denominada información de cabecera de la capa de transporte, $H_t$ en la Figura 1.24) que será utilizada por la capa de transporte del lado receptor. El mensaje de la capa de aplicación y la información de cabecera de la capa de transporte constituyen el **segmento de la capa de transporte**. El segmento de la capa de transporte encapsula, por tanto, el mensaje de la capa de aplicación. La información añadida podría incluir información que permita a la capa de transporte del lado receptor entregar el mensaje a la aplicación apropiada, así como bits de detección de errores que permitan al receptor determinar si los bits del mensaje han cambiado a lo largo de la ruta. A continuación, la capa de transporte pasa el segmento a la capa de red, que añade información de cabecera de la capa de red ($H_n$ en la Figura 1.24), como por ejemplo las direcciones de los sistemas terminales de origen y de destino, creando un **datagrama de la capa de red**. Este datagrama se pasa entonces a la capa de enlace, que (¡por supuesto!) añadirá su propia información de cabecera dando lugar a una **trama de la capa de enlace**. Así, vemos que en cada capa, un paquete está formado por dos tipos de campos: campos de cabecera y un **campo de carga útil**. Normalmente, la carga útil es un paquete de la capa superior.

Una buena analogía para ilustrar este tema sería el envío de un informe interno de empresa desde una sucursal a otra a través del servicio postal público. Supongamos que Alicia, que se encuentra en una sucursal, quiere enviar un informe a Benito, que se encuentra en la otra sucursal. El *informe* es análogo al *mensaje de la capa de aplicación*. Alicia introduce el informe en un sobre de correo interno de la empresa y escribe en él el nombre y el departamento de Benito. El *sobre de correo interno* es análogo a un *segmento de la capa de transporte*: contiene información de cabecera (el nombre y el departamento de Benito) y encapsula el mensaje de la capa de aplicación (el informe). Cuando el departamento de correo de la sucursal emisora recibe este sobre, lo mete en otro sobre adecuado para enviar el informe mediante el servicio público de correos. En la sala de correo de la empresa también se escriben en el segundo sobre las direcciones postales tanto de la sucursal emisora como de la sucursal receptora. Aquí, el *sobre postal* es análogo al *datagrama*: encapsula el segmento de la capa de transporte (el sobre de correo interno), que a su vez encapsula el mensaje original (el informe). El servicio postal entrega el sobre postal al departamento de correo de la sucursal receptora, donde comienza el proceso de desencapsulación. En ese departamento se extrae el sobre de correo interno y se envía a Benito. Por último, Benito abre el sobre de correo interno y saca el informe.

El proceso de encapsulación puede ser más complejo que el que acabamos de describir. Por ejemplo, un mensaje largo puede dividirse en varios segmentos de la capa de transporte (los

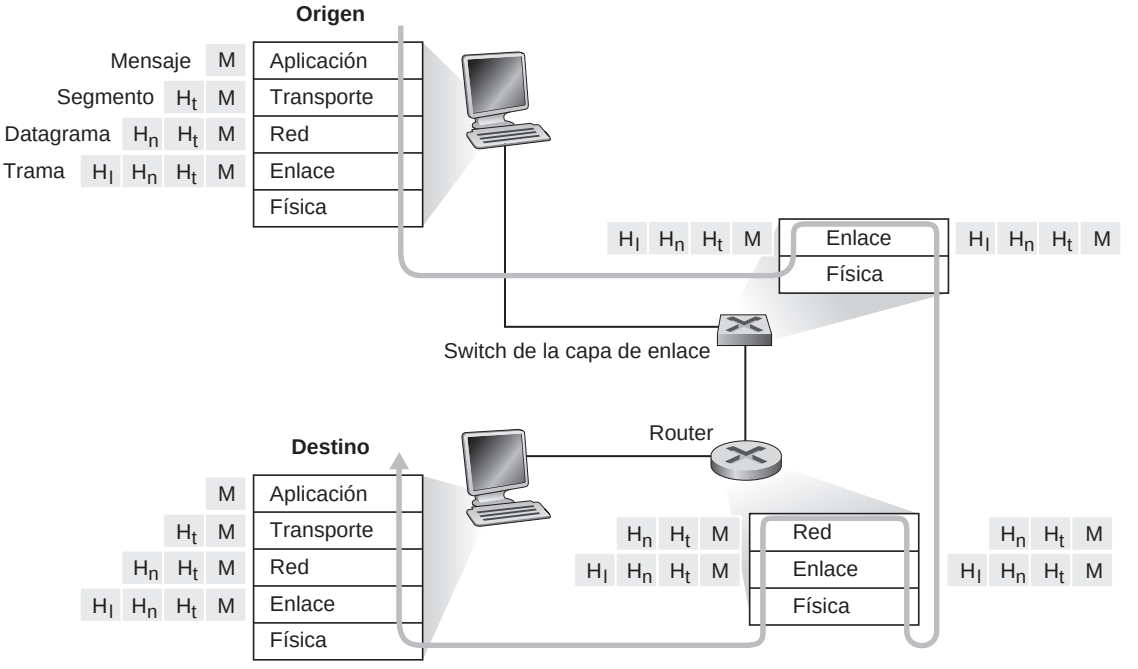



**Figura 1.24** ♦ Hosts, routers y switches de la capa de enlace. Cada uno de ellos contiene un conjunto distinto de capas, lo que refleja sus distintas funcionalidades.

cuales a su vez pueden dividirse en varios datagramas de la capa de red). En el extremo receptor, cada segmento deberá entonces ser reconstruido a partir de sus datagramas constituyentes.

## 1.6 Ataques a las redes

Internet se ha convertido en una herramienta crítica para muchas instituciones actuales, incluyendo empresas pequeñas y medianas, universidades y organismos gubernamentales. Muchas personas individuales también confían en Internet para llevar a cabo muchas de sus actividades profesionales, sociales y personales. Miles de millones de "cosas" (incluyendo dispositivos corporales y electrodomésticos) se conectan hoy en día a Internet. Pero detrás de todas estas utilidades y toda esta excitación, hay un lado oscuro, un lado donde "los malos" intentan hacer estragos en nuestras vidas diarias, dañando nuestras computadoras conectadas a Internet, violando nuestra intimidad y volviendo inoperables los servicios de Internet de los que dependemos.

El campo de la seguridad de red se ocupa de ver cómo "los malos" pueden atacar a las redes de computadoras y de cómo nosotros, que pronto seremos expertos en redes, podemos defendernos frente a estos ataques; o mejor todavía, de cómo diseñar nuevas arquitecturas que sean inmunes a tales ataques. Dada la frecuencia y variedad de ataques existentes, así como la amenaza de nuevos y más destructivos ataques futuros, la seguridad de red se ha convertido en un tema crucial en el campo de las redes de computadoras. Una de las características de este libro de texto es que lleva las cuestiones sobre la seguridad en las redes a primer plano.

Puesto que todavía no tenemos experiencia ni en redes ni en los protocolos de Internet, comenzaremos haciendo un repaso de algunos de los problemas de seguridad que predominan en la actualidad, con el fin de abrir boca para las explicaciones más sustanciosas que proporcionaremos en los capítulos siguientes. Así que empezaremos preguntándonos simplemente, ¿qué es lo que puede ir mal? ¿En qué sentido son vulnerables las redes de computadoras? ¿Cuáles son los principales tipos de ataques hoy día?

### Los "malos" pueden introducir software malicioso en nuestro host a través de Internet

Conectamos nuestros dispositivos a Internet porque deseamos enviar y recibir datos hacia/desde Internet, lo que incluye todo tipo de cosas legítimas, como publicaciones Instagram, resultados de búsquedas en Internet, flujos de música, videoconferencias, películas de cine, etc. Pero, lamentablemente, junto con todos estos elementos legítimos también existen elementos maliciosos, lo que se conoce de forma colectiva como software malicioso o **malware**, que puede también acceder a nuestros dispositivos e infectarlos. Una vez que el malware ha infectado un dispositivo, puede hacer todo tipo de maldades, como por ejemplo borrar nuestros archivos o instalar software espía que recopile nuestra información personal, como el número de la seguridad social, contraseñas y pulsaciones de teclas, y luego envíe esos datos (¡a través de Internet, por supuesto!) a "los malos". Nuestro host comprometido también puede ser reclutado como parte de una red de miles de dispositivos comprometidos de forma similar, lo que se conoce de forma colectiva como **botnet** (red robot), que los atacantes controlan y aprovechan para la distribución de correo electrónico basura (*spam*) o para llevar a cabo ataques distribuidos de denegación de servicio (que pronto explicaremos) contra los hosts objetivo.

Gran parte del malware que existe actualmente es **auto-replicante**: una vez que infecta un host, busca cómo acceder desde dicho host a otros hosts a través de Internet, y desde esos hosts que acaba de infectar, busca cómo acceder a otros. De esta forma, el malware auto-replicante puede extenderse rápidamente de forma exponencial. El malware puede extenderse en forma de virus o de gusano. Un **virus** es un malware que requiere cierta interacción del usuario para infectar el dispositivo. El ejemplo clásico es un adjunto de correo electrónico que contiene código ejecutable malicioso. Si un usuario recibe y abre un adjunto de este tipo, ejecutará inadvertidamente el malware en el dispositivo. Normalmente, tales virus enviados en los mensajes de correo electrónico son auto-replicantes: una vez que se ha ejecutado, el virus puede enviar un mensaje idéntico con el mismo adjunto malicioso a, por ejemplo, todos los contactos de nuestra libreta de direcciones. Un **gusano** es un tipo de malware que puede entrar en un dispositivo sin ninguna interacción explícita por parte del usuario. Por ejemplo, un usuario puede estar ejecutando una aplicación de red vulnerable a la que un atacante puede enviar software malicioso. En algunos casos, sin que el usuario intervenga, la aplicación puede aceptar el malware de Internet y ejecutarlo, creando un gusano. El gusano instalado ahora en el dispositivo recién infectado explora entonces Internet, buscando otros hosts que ejecuten la misma aplicación de red vulnerable. Cuando encuentra otros hosts vulnerables, envía una copia de sí mismo a esos hosts. Por último, un caballo de Troya es un malware que está oculto dentro de otro software que es útil. Hoy día, el malware está generalizado y es costoso defenderse de él. A lo largo del libro, le animaremos a que piense en la siguiente cuestión: ¿qué pueden hacer los diseñadores de redes de computadoras para defender a los dispositivos conectados a Internet de los ataques de malware?

### Los "malos" pueden atacar a los servidores y a la infraestructura de red

Otra amplia clase de amenazas de seguridad son los que se conocen como **ataques de denegación de servicio (DoS, *Denial-of-Service*)**. Como su nombre sugiere, un ataque DoS vuelve inutilizable una red, un host o cualquier otro elemento de la infraestructura para los usuarios legítimos. Los servidores web, los servidores de correo electrónico, los servidores DNS (que se estudian en el Capítulo 2) y las redes institucionales pueden ser, todos ellos, objeto de ataques DoS. Los ataques DoS son muy comunes en Internet, produciéndose miles de ataques de este tipo cada año [Moore 2001]. El sitio web Digital Attack Map permite a los visitantes visualizar los principales ataques DoS de ese día, a nivel mundial [DAM 2016]. La mayoría de los ataques DoS en Internet pueden clasificarse dentro de una de las tres categorías siguientes:

- *Ataque de vulnerabilidad.* Este ataque implica el envío de unos pocos mensajes especialmente diseñados a una aplicación o sistema operativo vulnerable que esté ejecutándose en un host

objetivo. Si se envía la secuencia de paquetes correcta a una aplicación o un sistema operativo vulnerables, el servicio puede detenerse o, lo que es peor, el host puede sufrir un fallo catastrófico.

- *Inundación del ancho de banda.* El atacante envía una gran cantidad de paquetes al host objetivo, de modo que se inunda el enlace de acceso del objetivo, impidiendo que los paquetes legítimos puedan alcanzar al servidor.
- *Inundación de conexiones.* El atacante establece un gran número de conexiones TCP completamente abiertas o semi-abiertas (las conexiones TCP se estudian en el Capítulo 3) con el host objetivo. El host puede llegar a atascarse con estas conexiones fraudulentas, impidiéndose así que acepte las conexiones legítimas.

Vamos a ver a continuación más detalladamente el ataque por inundación del ancho de banda. Recordando el análisis que hemos realizado en la Sección 1.4.2 sobre los retardos y la pérdida de paquetes, es evidente que si el servidor tiene una velocidad de acceso de *R* bps, entonces el atacante tendrá que enviar el tráfico a una velocidad de aproximadamente *R* bps para causar daños. Si *R* es muy grande, es posible que un único origen de ataque no sea capaz de generar el tráfico suficiente como para dañar al servidor. Además, si todo el tráfico procede de un mismo origen, un router situado en un punto anterior de la ruta podría detectar el ataque y bloquear todo el tráfico procedente de ese origen, antes de que llegue a aproximarse al servidor. En un ataque **DoS distribuido (DDoS, *Distributed DoS*)**, como el mostrado en la Figura 1.25, el atacante controla varios orígenes y hace que cada uno de ellos bombardee el objetivo con tráfico. Con este método, la tasa acumulada de tráfico para todos los orígenes controlados tiene que ser aproximadamente igual a *R* para inutilizar el servicio. Actualmente, se producen de forma común ataques DDoS que utilizan botnets con miles de hosts comprometidos [DAM 2016]. Los ataques DDoS son mucho más difíciles de detectar y contrarrestar que los ataques DoS procedentes de un único host.

Le animamos a que piense en la siguiente pregunta según vaya leyendo el libro: ¿qué pueden hacer los diseñadores de redes de computadoras para defenderlas de los ataques DoS? Veremos que son necesarias diferentes defensas para cada uno de los tres tipos de ataques DoS.

### Los "malos" pueden examinar y analizar los paquetes

Actualmente, muchos usuarios acceden a Internet a través de dispositivos inalámbricos, tales como computadoras portátiles con conexión WiFi o dispositivos de mano con conexiones Internet

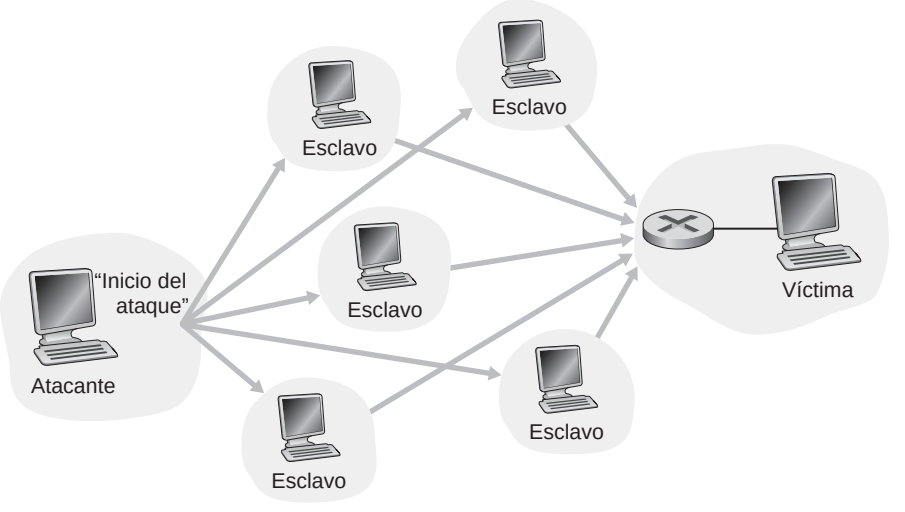



**Figura 1.25** ♦ Ataque de denegación de servicio distribuido.

móviles (de lo que hablaremos en el Capítulo 7). Aunque el omnipresente acceso a Internet es extremadamente útil y hace posibles maravillosas aplicaciones nuevas para los usuarios móviles, también provoca una importante vulnerabilidad: ¡colocando un receptor pasivo en las vecindades del transmisor inalámbrico, se puede recibir una copia de todos los paquetes que se están transmitiendo! Estos paquetes pueden contener todo tipo de información confidencial, incluyendo contraseñas, números de la seguridad social, secretos comerciales y mensajes personales privados. Un receptor pasivo que registra una copia de todos los paquetes que pasan por él se denomina ***sniffer*** (husmeador de paquetes).

Los sniffers también pueden implantarse en entornos cableados. En entornos cableados de multidifusión, como en muchas redes LAN Ethernet, un sniffer puede obtener copias de todos los paquetes enviados a través de la LAN. Como se ha descrito en la Sección 1.2, las tecnologías de acceso por cable también difunden paquetes y son, por tanto, vulnerables a la monitorización y el análisis. Además, un atacante que consiga acceder al router de acceso o al enlace de acceso a Internet de una institución, puede colocar un sniffer que haga una copia de todos los paquetes entrantes y salientes de la organización. Los paquetes así monitorizados pueden ser analizados después en busca de información confidencial.

Hay software sniffer disponible de forma gratuita en varios sitios web y en forma de productos comerciales. Hay profesores que imparten cursos sobre redes y realizan prácticas de laboratorio que implican escribir un programa sniffer y un programa de reconstrucción de datos de la capa de aplicación. ¡De hecho, las prácticas de laboratorio con Wireshark [Wireshark 2016] asociadas con este texto (véase la práctica de laboratorio introductoria de Wireshark al final del capítulo) utilizan precisamente un programa sniffer de ese tipo!

Puesto que los programas sniffer son pasivos (es decir, no inyectan paquetes en el canal), son difíciles de detectar. Por tanto, cuando envíamos paquetes a un canal inalámbrico, tenemos que aceptar que existe la posibilidad de que algún atacante pueda registrar copias de nuestros paquetes. Como habrá adivinado, una de las mejores formas de defenderse frente a los programas sniffer son las técnicas criptográficas. En el Capítulo 8 veremos cómo se aplica la criptografía a la seguridad de la red.

### Los "malos" pueden suplantar la identidad de nuestros conocidos

Es sorprendentemente fácil (¡y *el lector* tendrá los conocimientos necesarios para hacerlo muy pronto, a medida que vaya leyendo este texto!) crear un paquete con una dirección de origen, un contenido de paquete y una dirección de destino arbitrarios y luego transmitir dicho paquete a Internet, que reenviará el paquete a su destino. Imagine que el receptor confiado (por ejemplo, un router de Internet) que recibe tal paquete, toma la dirección de origen (falsa) como buena y luego ejecuta algún comando integrado en el contenido del paquete (por ejemplo, modifica la tabla de reenvío). La capacidad de inyectar paquetes en Internet con una dirección de origen falsa se conoce como **suplantación IP** y es una de las muchas formas en las que un usuario puede hacerse pasar por otro.

Para resolver este problema, necesitaremos aplicar un medio de *autenticación en el punto terminal*, es decir, un mecanismo que nos permita determinar con seguridad si un mensaje tiene su origen donde creemos que lo tiene. De nuevo, animamos a los lectores a que, a medida que avanzan por los capítulos del libro, piensen en cómo pueden hacer esto las aplicaciones y protocolos de red. En el Capítulo 8 exploraremos los mecanismos de autenticación en el punto terminal.

Para terminar con esta sección, vale la pena comentar cuál es la razón de que Internet se haya convertido en un lugar inseguro. Básicamente, la respuesta es que Internet fue diseñada originalmente de esa manera, ya que se basaba en el modelo de un "grupo de usuarios que confiaban entre sí, conectados a una red transparente" [Blumenthal 2001] —un modelo en el que (por definición) no había necesidad de pensar en la seguridad. Muchos aspectos de la arquitectura de Internet original reflejan profundamente esta idea de confianza mutua. Por ejemplo, la posibilidad de que un usuario envíe un paquete a cualquier otro usuario es la opción predeterminada, en lugar de ser una capacidad

solicitada/concedida, al igual que lo normal es creer que la identidad del usuario es la que declara, en lugar de autenticarle por defecto.

Pero actualmente Internet no implica realmente "usuarios de confianza mutua". Sin embargo, los usuarios de hoy día siguen necesitando comunicarse aunque no necesariamente confíen entre sí, pueden querer comunicarse de forma anónima, pueden comunicarse indirectamente a través de terceros (por ejemplo, cachés web, que estudiaremos en el Capítulo 2, o asistentes de movilidad, que veremos en el Capítulo 7) y pueden desconfiar del hardware, del software e incluso del aire a través del que se comunican. A lo largo del libro vamos a encontrarnos con muchos retos relacionados con la seguridad: buscaremos formas de defendernos frente a los sniffer, frente a la suplantación de identidades de los puntos terminales, frente a los ataques por interposición, frente a los ataques DDoS, frente al malware, etc. Tenemos que tener presente que la comunicación entre usuarios de mutua confianza es la excepción, más que la regla. ¡Bienvenido al mundo de las redes modernas de comunicaciones!

## 1.7 Historia de Internet y de las redes de computadoras

En las Secciones 1.1 a 1.6 se ha hecho una presentación de las tecnologías utilizadas en las redes de comunicaciones e Internet. ¡Ahora ya sabe lo suficiente como para impresionar a sus familiares y amigos! Sin embargo, si realmente desea causar una gran impresión en la siguiente fiesta a la que asista, debería salpicar su discurso con algunos detalles interesantes acerca de la fascinante historia de Internet [Segaller 1998].

### 1.7.1 El desarrollo de la conmutación de paquetes: 1961–1972

El campo de las redes de computadoras y de la actual Internet tiene sus inicios a principios de la década de 1960, cuando la red telefónica era la red de comunicaciones dominante en el mundo. Recordemos de la Sección 1.3 que la red telefónica utiliza mecanismos de conmutación de circuitos para transmitir la información entre un emisor y un receptor, una opción apropiada en la que la voz se transmite a una velocidad constante entre el emisor y el receptor. Debido a la creciente importancia de las computadoras en los primeros años de la década de 1960 y a la aparición de las computadoras de tiempo compartido, quizá fue natural considerar la cuestión de cómo enlazar las computadoras con el fin de que pudieran ser compartidas entre usuarios geográficamente distribuidos. El tráfico generado por esos usuarios tenía una tendencia a ser a *ráfagas*, compuesto por periodos de actividad como el envío de un comando a una computadora remota, seguidos de periodos de inactividad mientras se espera a obtener una respuesta o mientras se contempla la respuesta recibida.

Tres grupos de investigación repartidos por el mundo, cada uno de ellos ignorante de la existencia de los otros [Leiner 1998], comenzaron a trabajar en la conmutación de paquetes como alternativa eficiente y robusta a la conmutación de circuitos. El primer trabajo publicado sobre las técnicas de conmutación de paquetes fue el de Leonard Kleinrock [Kleinrock 1961; Kleinrock 1964], por aquel entonces un estudiante de posgrado en el MIT. Basándose en la teoría de colas, el trabajo de Kleinrock demostraba de forma elegante la efectividad de la técnica de conmutación de paquetes para las fuentes que generaban tráfico a ráfagas. En 1964, Paul Baran [Baran 1964], en el Rand Institute, había comenzado a investigar el uso de la conmutación de paquetes para comunicaciones de voz seguras en redes militares y, en el National Physical Laboratory (NPL) de Inglaterra, Donald Davies y Roger Scantlebury también estaban desarrollando sus ideas acerca de la conmutación de paquetes.

Los trabajos realizados en el MIT, en el instituto Rand y en los laboratorios NPL establecieron las bases de la red Internet actual. Pero Internet también tiene una larga tradición de "hagámoslo y demostremos que funciona" que también se remonta a la década de 1960. J. C. R. Licklider

[DEC 1990] y Lawrence Roberts, ambos colegas de Kleinrock en el MIT, dirigieron el programa de Ciencias de la Computación en la Agencia de Proyectos de Investigación Avanzada (ARPA, Advanced Research Projects Agency) de Estados Unidos. Roberts publicó un plan global para la red ARPAnet [Roberts 1967], la primera red de computadoras de conmutación de paquetes y un ancestro directo de la red Internet pública actual. El Día del Trabajo de 1969, se instaló el primer dispositivo de conmutación de paquetes en UCLA bajo la supervisión de Kleinrock y poco después se instalaron tres dispositivos más de conmutación de paquetes en el Instituto de Investigación de Stanford (SRI), en la Universidad de California en Santa Barbara y en la Universidad de Utah (Figura 1.26). Hacia finales de 1969, la naciente red precursora de Internet estaba formada por cuatro nodos. Kleinrock recuerda la primera vez que utilizó la red para llevar a cabo un inicio de sesión remoto desde UCLA en el SRI, consiguiendo que el sistema fallara [Kleinrock 2004].

Hacia 1972, la red ARPAnet había crecido aproximadamente hasta 15 nodos y la primera demostración pública ya había sido realizada por Robert Kahn. También se completó por aquel entonces el primer protocolo host a host entre sistemas terminales de ARPAnet, conocido como el protocolo de control de red (NCP, *Network Control Protocol*) [RFC 001]. Disponiendo de un protocolo extremo a extremo, ahora podían escribirse aplicaciones. Ray Tomlinson escribió el primer programa de correo electrónico en 1972.

## 1.7.2 Redes propietarias e interredes: 1972–1980

La red inicial ARPAnet era una única red cerrada. Para establecer una comunicación con un host de la red ARPAnet, había que estar físicamente conectado a otro dispositivo de conmutación de

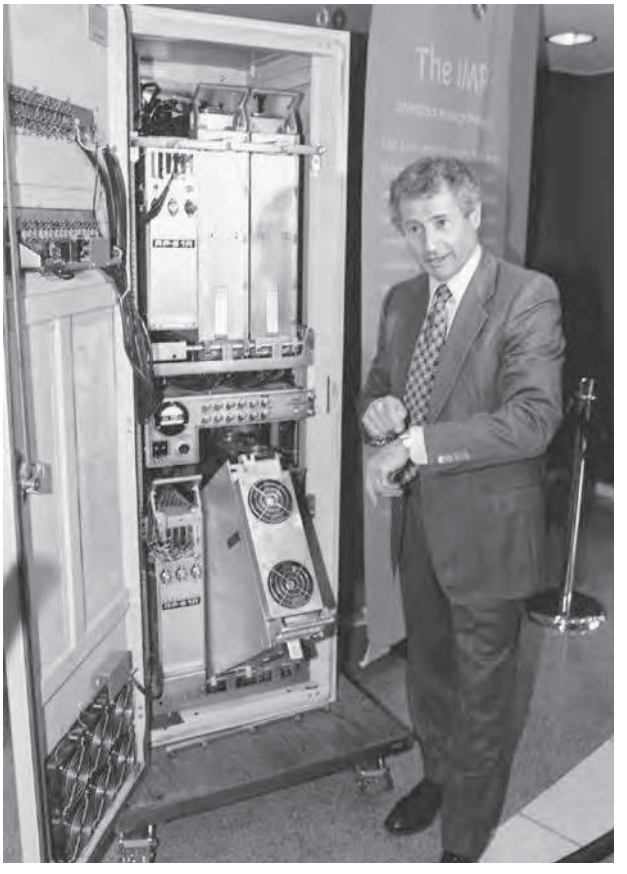



**Figura 1.26** ♦ Uno de los primeros dispositivos de conmutación de paquetes.

paquetes de ARPAnet. A mediados de la década de 1970, comenzaron a surgir otras redes de conmutación de paquetes autónomas además de ARPAnet: ALOHANet, una red de microondas que enlazaba universidades de las islas Hawai [Abramson 1970], así como redes de conmutación de paquetes vía satélite [RFC 829] y vía radio [Kahn 1978] de DARPA; Telenet, una red de conmutación de paquetes comercial de BBN basada en la tecnología de ARPAnet; Cyclades, una red de conmutación de paquetes francesa dirigida por Louis Pouzin [Think 2009]; redes de tiempo compartido tales como Tymnet y la red de Servicios de información de GE, entre otras, a finales de la década de 1960 y principios de la década de 1970 [Schwartz 1977] y la red SNA de IBM (1969-1974), que constituía un desarrollo paralelo al de ARPAnet [Schwartz 1977].

El número de redes estaba creciendo. Retrospectivamente, podemos ver que había llegado el momento de desarrollar una arquitectura completa para la interconexión de redes. El trabajo pionero sobre interconexión de redes (realizado bajo el patrocinio de la Agencia DARPA, *Defense Advanced Research Projects Agency*), que consistía en esencia en la creación de una *red de redes*, fue realizado por Vinton Cerf y Robert Kahn [Cerf 1974]; el término *internetting* (interredes o interconexión de redes) fue acuñado para describir este trabajo.

Estos principios arquitectónicos fueron integrados en TCP. Sin embargo, las primeras versiones de TCP eran bastante distintas al TCP de hoy día. Esas primeras versiones combinaban una entrega fiable en secuencia de los datos mediante retransmisiones por parte del sistema terminal (esta función todavía la realiza TCP hoy día) con funciones de reenvío (que actualmente son realizadas por IP). Los primeros experimentos con TCP, combinados con el reconocimiento de la importancia de disponer de un servicio de transporte extremo a extremo no fiable y sin control de flujo para aplicaciones tales como voz empaquetada, llevaron a separar IP de TCP y al desarrollo del protocolo UDP. Los tres protocolos clave de Internet que se emplean actualmente (TCP, UDP e IP) ya habían sido concebidos a finales de la década de 1970.

Además de las investigaciones relativas a Internet de la agencia DARPA, también estaban en marcha muchos otros importantes proyectos relacionados con las redes. En Hawai, Norman Abramson estaba desarrollando ALOHAnet, una red de paquetes vía radio que permitía a múltiples sitios remotos de las islas Hawai comunicarse entre sí. El protocolo ALOHA [Abramson 1970] fue el primer protocolo de acceso múltiple, que permitió a usuarios geográficamente distribuidos compartir un mismo medio de comunicación de difusión (una frecuencia de radio). Metcalfe y Boggs se basaron en el protocolo de acceso múltiple de Abramson para desarrollar el protocolo Ethernet [Metcalfe 1976], para redes de difusión compartidas basadas en cable; véase la Figura 1.27. Es interesante comentar que el desarrollo del protocolo Ethernet de Metcalfe y Boggs fue motivado por la necesidad de conectar varios PC, impresoras y discos compartidos [Perkins 1994]. Hace veinticinco años, bastante antes de la revolución de los PC y de la explosión de las redes, Metcalfe y Boggs estaban sentando las bases para las redes LAN de computadoras PC actuales.

### 1.7.3 Proliferación de las redes: 1980–1990

A finales de la década de 1970, había aproximadamente unos doscientos hosts conectados a la red ARPAnet. A finales de la década de 1980, el número de hosts conectados a la red Internet pública, una confederación de redes similar a la Internet actual, llegaría a los cien mil. La década de 1980 fue una época de enorme crecimiento.

Gran parte de este crecimiento fue el resultado de varios y distintos esfuerzos por crear redes de computadoras que enlazaran universidades. BITNET proporcionaba servicios de correo electrónico y de transferencia de archivos entre varias universidades del noreste de los Estados Unidos. CSNET (Computer Science Network) se formó para que investigadores universitarios que no tenían acceso a la red ARPAnet pudieran comunicarse. En 1986 se creó NSFNET para proporcionar acceso a los centros de supercomputación patrocinados por la NSF. Con una velocidad en la red troncal de 56 kbps inicialmente, la red troncal de NSFNET llegaría a operar a 1,5 Mbps a finales de la década y serviría como una red troncal primaria para enlazar redes regionales.

En la comunidad ARPAnet, muchas de las piezas finales de la arquitectura de Internet actual fueron encajando. El 1 de enero de 1983 se llevó a cabo el lanzamiento oficial de TCP/IP como el nuevo protocolo de host estándar para ARPAnet (reemplazando al protocolo NCP). La transición [RFC 801] de NCP a TCP/IP fue un suceso señalado: a todos los hosts se les requirió pasar a utilizar TCP/IP ese día. A finales de la década de 1980, se realizaron importantes extensiones en TCP con el fin de implementar un control de congestión basado en host [Jacobson 1988]. También se desarrolló el sistema DNS, que se emplea para establecer la correspondencia entre los nombres de Internet que usamos las personas (por ejemplo, gaia.cs.umass.edu) y sus direcciones IP de 32 bits [RFC 1034].

En paralelo con este desarrollo de ARPAnet (realizado fundamentalmente en Estados Unidos), a principios de la década de 1980, los franceses lanzaron el proyecto Minitel, un ambicioso plan para llevar las redes de datos a todos los hogares. Patrocinado por el gobierno francés, el sistema Minitel consistía en una red pública de conmutación de paquetes (basada en la serie de protocolos X.25), servidores Minitel y terminales económicos que incorporaban modems de baja velocidad. Minitel alcanzó un gran éxito en 1984 cuando el gobierno francés proporcionó un terminal Minitel gratuito a todo aquel que deseara tener uno en su casa. Los sitios Minitel incluían sitios gratuitos, como por ejemplo el directorio telefónico, y sitios privados, que cobraban sus tarifas basándose en la utilización del servicio. Minitel alcanzó su pico de uso a mediados de la década de 1990, llegándose a ofrecer más de 20.000 servicios, que iban desde servicios de banca hasta bases de datos de investigación especializadas. Minitel se utilizaba en un gran porcentaje de hogares de Francia 10 años antes de que la mayoría de los americanos hubiera oído siquiera hablar de Internet.

### 1.7.4 La explosión de Internet: década de 1990

La década de 1990 estuvo marcada por una serie de acontecimientos que simbolizaron la continua evolución y la pronta llegada de la comercialización de Internet. ARPAnet, el progenitor de Internet, dejó de existir. En 1991, NSFNET retiró sus restricciones sobre el uso de NSFNET para propósitos comerciales. La propia NSFNET fue eliminada del servicio activo en 1995, pasando el tráfico troncal de Internet a ser transportado por proveedores de servicios Internet (ISP) comerciales.

Sin embargo, el principal acontecimiento de la década de 1990 fue la aparición de la aplicación World Wide Web, que llevaría Internet a los hogares y negocios de millones de personas de todo el mundo. La Web sirvió como plataforma para posibilitar e implantar cientos de nuevas aplicaciones, que hoy damos por sentadas, incluyendo las búsquedas (por ejemplo, Google y Bing), el comercio Internet (por ejemplo, Amazon y eBay) y las redes sociales (por ejemplo, Facebook).

La Web fue inventada en el CERN por Tim Berners-Lee entre 1989 y 1991 [Berners- Lee 1989], basándose en las ideas de los primeros trabajos acerca del hipertexto desarrolladas en la década de 1940 por Vannevar Bush [Bush 1945] y luego a partir de la década de 1960 por Ted Nelson [Xanadu 2009]. Berners-Lee y sus colegas desarrollaron las versiones iniciales de HTML, HTTP, un servidor web y un navegador (los cuatro componentes clave de la Web). Hacia finales del año 1993 estaban operativos aproximadamente doscientos servidores web. Esta colección de servidores solo sería un presagio de lo que estaba por venir. Al mismo tiempo, había varios investigadores desarrollando navegadores web con interfaces GUI, entre los que se encontraba Marc Andreessen, que fundó junto con Jim Clark la empresa Mosaic Communications, que más tarde se convertiría en Netscape Communications Corporation [Cusumano 1998; Quittner 1998]. Hacia 1995, los estudiantes universitarios empleaban los navegadores Netscape para navegar por la Web diariamente. También por esa época, las empresas, grandes y pequeñas, comenzaron a trabajar con servidores web y a realizar transacciones comerciales a través de la Web. En 1996, Microsoft empezó a desarrollar navegadores y comenzaría la guerra de los navegadores entre Netscape y Microsoft, que terminaría ganando Microsoft unos pocos años después [Cusumano 1998].

La segunda mitad de la década de 1990 fue un periodo de gran crecimiento e innovación para Internet, con grandes corporaciones y miles de empresas de nueva creación desarrollando productos y servicios Internet. A finales del milenio, Internet daba soporte a cientos de aplicaciones populares, entre las que se incluyen las cuatro de más éxito:

- Correo electrónico, incluyendo los adjuntos y el correo electrónico accesible a través de la Web.
- La Web, incluyendo la navegación web y el comercio por Internet.
- La mensajería instantánea, con listas de contactos.
- La compartición igualitaria de archivos MP3, de la que Napster fue la pionera.

Es interesante saber que las dos primeras de estas aplicaciones estrella fueron creadas por la comunidad de investigadores, mientras que las dos últimas lo fueron por unos pocos jóvenes emprendedores.

El periodo comprendido entre 1995 y 2001 fue un paseo en montaña rusa para Internet en los mercados financieros. Antes de que fueran incluso rentables, cientos de nuevas empresas de Internet realizaron ofertas públicas de venta y comenzaron a cotizar en los mercados bursátiles. Muchas empresas fueron valoradas en miles de millones de dólares sin tener ingresos significativos. Las acciones de Internet se hundieron en 2000-2001 y muchas de esas empresas de nueva creación cerraron. No obstante, algunas de ellas emergieron como los grandes ganadores en el espacio Internet, entre las que se incluyen Microsoft, Cisco, Yahoo, e-Bay, Google y Amazon.

### 1.7.5 El nuevo milenio

La innovación en el campo de las redes de computadoras continúa a gran velocidad. Se están haciendo avances en todos los frentes, incluyendo la implantación de routers más rápidos y el uso de mayores velocidades de transmisión, tanto en redes de acceso como en las redes troncales. Pero los siguientes desarrollos merecen especial atención:

- Desde el principio del milenio, hemos sido testigos de la agresiva implantación de accesos Internet de banda ancha en las viviendas —no solo modems de cable y líneas DSL, sino también enlaces de fibra óptica—, como se explica en la Sección 1.2. Este acceso a Internet de alta velocidad ha preparado el terreno para una amplia variedad de aplicaciones de vídeo, incluyendo la distribución de vídeos generados por el usuario (por ejemplo, YouTube), la distribución de películas y programas de televisión a la carta (por ejemplo, Netflix) y videoconferencias multipersona (por ejemplo, Skype, Facetime y Google Hangouts).
- La creciente ubicuidad de las redes WiFi públicas de alta velocidad (54 Mbps y superiores) y de los accesos a Internet a velocidad media (decenas de Mbps) a través de redes de telefonía móvil 4G, no solo está haciendo posible que permanezcamos continuamente conectados mientras nos movemos, sino que también permite nuevas aplicaciones específicas de la ubicación, como Yelp, Tinder, Yik Yak y Waz. El número de dispositivos inalámbricos conectados a Internet sobrepasó en 2011 al número de dispositivos cableados. Este acceso inalámbrico de alta velocidad ha hecho posible la rápida aparición de computadoras de mano (iPhones, Androids, iPads, etc.), que disfrutan de un acceso a Internet constante y sin ataduras.
- Las redes sociales en línea —como Facebook, Instagram, Twitter y WeChat (inmensamente popular en China)— han dado lugar a la aparición de enormes redes de personas basadas en Internet. Muchas de estas redes sociales son ampliamente utilizadas tanto para mensajería, como para compartición de fotografías. Muchos usuarios actuales de Internet "viven" principalmente dentro de una o más redes sociales. A través de sus APIs, las redes sociales en línea crean plataformas para nuevas aplicaciones en red y juegos distribuidos.
- Como hemos explicado en la Sección 1.3.3, los proveedores de servicios en línea, como Google y Microsoft, han implantado sus propias redes privadas de gran envergadura, que no solo conectan entre sí sus centros de datos globalmente distribuidos, sino que se utilizan también para puentear Internet en la medida de lo posible, comunicándose directamente con los ISP de nivel inferior. Como resultado, Google proporciona resultados de búsquedas y acceso al correo electrónico casi instantáneos, como si sus centros de datos se estuvieran ejecutando en nuestras propias computadoras.

- Muchas empresas de comercio electrónico a través de Internet ejecutan ahora sus aplicaciones en la "nube" —como en el EC2 de Amazon, en el Application Engine de Google o en el Azure de Microsoft—. Muchas empresas y universidades han migrado también sus aplicaciones Internet (por ejemplo, el correo electrónico o el hospedaje web) a la nube. Las empresas de servicios de computación en la nube no solo proporcionan entornos de almacenamiento y ejecución escalables para las aplicaciones, sino que también permiten a estas un acceso implícito a sus redes privadas de altas prestaciones.

## 1.8 Resumen

¡En este capítulo hemos presentado una gran cantidad de material! Hemos hablado de los distintos componentes hardware y software que forman Internet, en particular, y las redes de computadoras en general. Hemos partido de la frontera de la red, fijándonos en los sistemas terminales y las aplicaciones, y en el servicio de transporte proporcionado a las aplicaciones que se ejecutan en los sistemas terminales. También hemos hablado de las tecnologías de la capa de enlace y de los medios físicos que pueden encontrarse normalmente en la red de acceso. A continuación, nos hemos adentrado en el interior de la red, en su núcleo, y hemos identificado los mecanismos de conmutación de paquetes y de conmutación de circuitos como los dos métodos básicos utilizados para el transporte de datos a través de una red de telecomunicaciones, y hemos examinado las fortalezas y las debilidades de cada método. También hemos examinado la estructura global de Internet, y hemos aprendido que es una red de redes. Hemos visto que la estructura jerárquica de Internet, formada por ISP de nivel superior e inferior, ha permitido que Internet crezca hasta incluir miles de redes.

En la segunda parte de este capítulo de introducción, hemos abordado varios temas fundamentales del campo de las redes. En primer lugar, hemos estudiado las causas de los retardos, la tasa de transferencia y la pérdida de paquetes en una red de conmutación de paquetes. Hemos desarrollado modelos cuantitativos simples para determinar los retardos de transmisión, de propagación y de cola, así como para la tasa de transferencia; haremos un uso exhaustivo de estos modelos de retardo en los problemas de repaso incluidos a lo largo del libro. A continuación, hemos visto las capas de protocolos y los modelos de servicio, las principales claves arquitectónicas de las redes a las que se volverá a hacer referencia a lo largo del libro. Asimismo, hemos repasado los ataques de seguridad más habituales actualmente en Internet. Hemos terminado nuestra introducción con un breve repaso a la historia de las redes de computadoras. Este primer capítulo en sí mismo constituye un minicurso sobre redes de computadoras.

Por tanto, ¡hemos cubierto una enorme cantidad de conceptos básicos en este primer capítulo! Si se siente un poco abrumado, no se preocupe: en los siguientes capítulos revisaremos todas estas ideas, viéndolas con mucho más detalle (¡lo que es una promesa, no una amenaza!). En este punto, esperamos que termine el capítulo teniendo ya una idea sobre las piezas que forman una red, teniendo un conocimiento (que todavía deberá desarrollar) del vocabulario del campo de las redes de computadoras (no se asuste por tener que volver a consultar este capítulo) y habiendo incrementado su deseo por aprender más acerca de las redes. Esta es la tarea que tenemos por delante durante el resto del libro.

### Mapa de este libro

Antes de iniciar cualquier viaje, siempre debería echarse un vistazo a un mapa de carreteras, para familiarizarse con las principales carreteras e intersecciones con los que nos encontraremos más adelante. En el viaje en el que nos hemos embarcado nosotros, el destino final es conocer en profundidad el cómo, el qué y el por qué de las redes de computadoras, y nuestro mapa son los capítulos de este libro:

1. Redes de computadoras e Internet
2. La capa de aplicación
3. La capa de transporte
4. La capa de red: plano de datos
5. La capa de red: plano de control
6. La capa de enlace y las redes de área local
7. Redes inalámbricas y móviles
8. Seguridad en las redes de computadoras
9. Redes multimedia

Los Capítulos 2 a 6 son los cinco capítulos centrales del libro. Observará que están organizados según las cuatro capas superiores de la pila de protocolos de Internet de cinco capas. Observará también que nuestro viaje va a comenzar por la parte superior de la pila de protocolos de Internet, es decir, por la capa de aplicación, y luego continuaremos nuestro trabajo descendiendo por la pila. La razón de hacer este recorrido de arriba a abajo es porque, una vez que conozcamos las aplicaciones, estaremos en condiciones de comprender los servicios de red necesarios para dar soporte a esas aplicaciones. Podremos así examinar a continuación las distintas formas en que tales servicios pueden ser implementados por una arquitectura de red. Ocuparnos de las aplicaciones en primer lugar nos va a proporcionar la motivación necesaria para abordar el resto del texto.

En la segunda mitad del libro, Capítulos 7 a 9, se abordan tres temas enormemente importantes (y en cierto modo independientes) en las redes de comunicaciones modernas. En el Capítulo 7 examinaremos las redes inalámbricas y celulares, incluyendo las redes LAN inalámbricas (lo que incluye las tecnologías WiFi y Bluetooth), las redes de telefonía móvil (lo que incluye las redes GSM, 3G y 4G) y la movilidad (tanto en redes IP como GSM). En el Capítulo 8, dedicado a la seguridad en las redes de computadoras, veremos en primer lugar los fundamentos de los mecanismos de cifrado y de seguridad de red y después examinaremos cómo está siendo aplicada la teoría básica a un amplio rango de contextos de Internet. El último capítulo, dedicado a las redes multimedia, examina aplicaciones de audio y de vídeo tales como la telefonía Internet, la videoconferencia y los flujos de información multimedia almacenada. También veremos cómo pueden diseñarse redes de conmutación de paquetes para proporcionar una calidad de servicio coherente a las aplicaciones de audio y de vídeo.

# Problemas y cuestiones de repaso

## Capítulo 1 Cuestiones de repaso

### SECCIÓN 1.1

R1. ¿Cuál es la diferencia entre un host y un sistema terminal? Enumere distintos tipos de sistemas terminales. ¿Es un servidor web un sistema terminal?

R2. El término *protocolo* a menudo se emplea para describir las relaciones diplomáticas. ¿Cómo describe Wikipedia un protocolo diplomático?

R3. ¿Por qué son importantes los estándares para los protocolos?

### SECCIÓN 1.2

R4. Enumere seis tecnologías de acceso. Clasifíquelas como de acceso residencial, acceso empresarial o acceso inalámbrico de área extensa.

R5. ¿La velocidad de transmisión en un sistema HFC es dedicada o compartida entre los usuarios? ¿Pueden producirse colisiones en un canal de descarga HFC? ¿Por qué o por qué no?

R6. Enumere las tecnologías de acceso residencial disponibles en su ciudad. Para cada tipo de acceso, detalle la velocidad de descarga ofrecida, la velocidad de carga y el precio mensual.

R7. ¿Cuál es la velocidad de transmisión en las redes LAN Ethernet?

R8. Cite algunos de los medios físicos sobre los que se puede emplear la tecnología Ethernet.

R9. Para el acceso residencial se emplean modems de acceso telefónico, sistemas HFC, DSL y FTTH. Para cada una de estas tecnologías de acceso, detalle el rango de velocidades de transmisión e indique si la velocidad de transmisión es dedicada o compartida.

R10. Describa las tecnologías de acceso inalámbrico a Internet más populares hoy día. Compárelas e indique sus diferencias.

SECCIÓN 1.3

R11. Suponga que hay un único dispositivo de conmutación de paquetes entre un host emisor y un host receptor. Las velocidades de transmisión entre el host emisor y el dispositivo de conmutación y entre este y el host receptor son $R_1$ y $R_2$, respectivamente. Suponiendo que el switch utiliza el mecanismo de conmutación de paquetes de almacenamiento y reenvío, ¿cuál es el retardo total extremo a extremo si se envía un paquete de longitud $L$? (Ignore los retardos de cola, de propagación y de procesamiento.)

R12. ¿Qué ventaja presenta una red de conmutación de circuitos frente a una red de conmutación de paquetes? ¿Qué ventajas tiene la multiplexación TDM frente a la multiplexación FDM en una red de conmutación de circuitos?

R13. Suponga que los usuarios comparten un enlace de 2 Mbps y que cada usuario transmite a una velocidad constante de 1 Mbps, pero solo durante el 20 por ciento del tiempo. (Véase la comparación entre conmutación de circuitos y conmutación de paquetes de la Sección 1.3.)

a. Si se utiliza la conmutación de circuitos, ¿a cuántos usuarios se puede dar soporte?

b. Para el resto del problema, suponga que se utiliza la conmutación de paquetes. ¿Por qué prácticamente no habrá retardo de cola antes del enlace si dos o menos usuarios transmiten a un mismo tiempo? ¿Por qué existirá retardo de cola si tres usuarios transmiten simultáneamente?

c. Calcule la probabilidad de que un usuario dado esté transmitiendo.

d. Suponga ahora que hay tres usuarios. Calcule la probabilidad de que en un instante determinado los tres usuarios estén transmitiendo simultáneamente. Halle la fracción de tiempo durante la que la cola crece.

R14. ¿Por qué dos ISP en un mismo nivel de la jerarquía a menudo se interconectan entre sí? ¿Cómo obtiene sus ingresos un IXP?

R15. Algunos proveedores de contenido han creado sus propias redes. Describa la red de Google. ¿Qué es lo que lleva a los proveedores de contenido a crear estas redes?

SECCIÓN 1.4

R16. Considere el envío de un paquete desde un host emisor a un host receptor a través de una ruta fija. Enumere los componentes del retardo extremo a extremo. ¿Cuáles de estos retardos son constantes y cuáles son variables?

R17. Visite el applet *Transmission Versus Propagation Delay* (retardo de transmisión frente a retardo de propagación) disponible en el sitio web del libro. Utilizando las velocidades, retardos de propagación y tamaños de paquete disponibles, determine una combinación para la cual el emisor termine la operación de transmisión antes de que el primer bit del paquete haya llegado al receptor. Halle otra combinación para la que el primer bit del paquete llegue al receptor antes de que el emisor haya terminado de transmitir.

R18. ¿Cuánto tiempo tarda un paquete cuya longitud es de 1.000 bytes en propagarse a través de un enlace a una distancia de 2.500 km, siendo la velocidad de propagación igual a $2{,}5 \cdot 10^8$ m/s y la velocidad de transmisión de 2 Mbps? De forma más general, ¿cuánto tiempo tarda un paquete de longitud $L$ en propagarse a través de un enlace a una distancia $d$, con una

velocidad de propagación *s* y una velocidad de transmisión de *R* bps? ¿Depende este retardo de la longitud del paquete? ¿Depende este retardo de la velocidad de transmisión?

R19. Suponga que el host A desea enviar un archivo de gran tamaño al host B. La ruta desde el host A al host B está formada por tres enlaces, cuyas velocidades son $R_1 = 500$ kbps, $R_2 = 2$ Mbps y $R_3 = 1$ Mbps.

a. Suponiendo que no hay ningún otro tráfico en la red, ¿cuál es la tasa de transferencia para el archivo?

b. Suponga que el tamaño del archivo es de 4 millones de bytes. Dividiendo el tamaño del archivo entre la tasa de transferencia, ¿cuánto tiempo tardará aproximadamente en transferirse el archivo al host B?

c. Repita los apartados (a) y (b), pero ahora con $R_2$ reducida a 100 kbps.

R20. Suponga que el sistema terminal A desea enviar un archivo de gran tamaño al sistema terminal B. Sin entrar en detalles, describa cómo crea el sistema terminal A los paquetes a partir del archivo. Cuando uno de estos paquetes llega a un conmutador de paquetes, ¿qué información del paquete utiliza el conmutador para determinar el enlace por el que debe ser reenviado el paquete? ¿Por qué la conmutación de paquetes en Internet es análoga a viajar de una ciudad a otra preguntando por la dirección a la que nos dirigimos?

R21. Visite el applet *Queuing and Loss* (colas y pérdida de paquetes) en el sitio web del libro. ¿Cuáles son las velocidades de transmisión máxima y mínima? Para esas velocidades, ¿cuál es la intensidad de tráfico? Ejecute el applet con esas velocidades y determine cuánto tiempo tiene que transcurrir para que tenga lugar una pérdida de paquetes. A continuación, repita el experimento una segunda vez y determine de nuevo cuánto tiempo pasa hasta producirse una pérdida de paquetes. ¿Son diferentes los valores obtenidos? ¿Por qué o por qué no?

SECCIÓN 1.5

R22. Enumere cinco tareas que puede realizar una capa. ¿Es posible que una (o más) de estas tareas pudieran ser realizadas por dos (o más) capas?

R23. ¿Cuáles son las cinco capas de la pila de protocolos Internet? ¿Cuáles son las responsabilidades principales de cada una de estas capas?

R24. ¿Qué es un mensaje de la capa de aplicación? ¿Y un segmento de la capa de transporte? ¿Y un datagrama de la capa de red? ¿Y una trama de la capa de enlace?

R25. ¿Qué capas de la pila de protocolos de Internet procesa un router? ¿Qué capas procesa un switch de la capa de enlace? ¿Qué capas procesa un host?

SECCIÓN 1.6

R26. ¿Cuál es la diferencia entre un virus y un gusano?

R27. Describa cómo se puede crear una red robot (botnet) y cómo se puede utilizar en un ataque DDoS.

R28. Suponga que Alicia y Benito están enviándose paquetes entre sí a través de una red. Imagine que Victoria se introduce en la red de modo que puede capturar todos los paquetes enviados por Alicia y que luego envía lo que ella quiere a Benito. Además, también puede capturar todos los paquetes enviados por Benito y luego enviar a Alicia lo que le parezca. Enumere algunos de los daños que Victoria puede ocasionar desde su posición.

## Problemas

P1. Diseñe y describa un protocolo de nivel de aplicación que será utilizado entre un cajero automático y la computadora central de un banco. El protocolo deberá permitir verificar la tarjeta y la contraseña del usuario, consultar el saldo de la cuenta (que se almacena en la

computadora central) y retirar dinero de la cuenta (entregándose el dinero al usuario). Las entidades del protocolo deben poder manejar el caso, bastante común, de que no haya suficiente saldo en la cuenta como para cubrir el reembolso. Especifique el protocolo enumerando los mensajes intercambiados y las acciones realizadas por el cajero automático o la computadora central del banco al transmitir y recibir mensajes. Haga un boceto del funcionamiento del protocolo para el caso de una retirada de efectivo sin errores, utilizando un diagrama similar al mostrado en la Figura 1.2. Indique explícitamente las suposiciones hechas por el protocolo acerca del servicio de transporte extremo a extremo subyacente.

P2. La Ecuación 1.1 proporciona una fórmula para calcular el retardo extremo a extremo al enviar un paquete de longitud $L$ a través de $N$ enlaces de velocidad de transmisión $R$. Generalice esa fórmula para el caso de enviar $P$ paquetes seguidos a través de los $N$ enlaces.

P3. Considere una aplicación que transmite datos a una velocidad constante (por ejemplo, el emisor genera una unidad de datos de $N$ bits cada $k$ unidades de tiempo, donde $k$ es un valor pequeño y fijo). Además, cuando esta aplicación se inicia, continúa ejecutándose durante un periodo de tiempo relativamente largo. Responda a las siguientes cuestiones, justificando de forma breve su respuesta:

   a. ¿Qué sería más apropiado para esta aplicación, una red de conmutación de circuitos o una red de conmutación de paquetes? ¿Por qué?

   b. Suponga que se utiliza una red de conmutación de paquetes y que el único tráfico que existe en la misma procede de la aplicación que acabamos de describir. Además, suponga que la suma de las velocidades de datos de la aplicación es menor que las capacidades de cada uno de los enlaces. ¿Será necesario algún mecanismo de control de congestión? ¿Por qué?

P4. Considere la red de conmutación de circuitos de la Figura 1.13. Recuerde que hay 4 circuitos en cada enlace. Etiquete los cuatro dispositivos de conmutación con los nombres A, B, C y D, yendo en el sentido de las agujas del reloj.

   a. ¿Cuál es el número máximo de conexiones simultáneas que pueden estar en curso en un determinado instante de tiempo en esta red?

   b. Suponga que todas las conexiones se encuentran entre los dispositivos de conmutación A y C. ¿Cuál será el número máximo de conexiones simultáneas que puede haber en curso?

   c. Suponga que queremos realizar cuatro conexiones entre los dispositivos de conmutación A y C y otras cuatro entre los dispositivos B y D. ¿Podemos enrutar estas llamadas a través de los cuatro enlaces, de forma que se de soporte a las ocho conexiones?

P5. Repase la analogía de la caravana de coches de la Sección 1.4. Suponga una velocidad de propagación de 100 km/hora.

   a. Suponga que la caravana se mueve a una velocidad de 150 km, empezando delante de una caseta de peaje, pasando por una segunda caseta de peaje y terminando justo después de una tercera caseta de peaje. ¿Cuál es el retardo extremo a extremo?

   b. Repita el apartado (a) suponiendo ahora que en la caravana hay ocho coches en lugar de diez.

P6. En este problema elemental comenzamos a explorar los retardos de propagación y de transmisión, dos conceptos fundamentales en las redes de datos. Considere dos hosts, A y B, conectados por un único enlace cuya velocidad es de $R$ bps. Suponga que los dos hosts están separados $m$ metros y que la velocidad de propagación a lo largo del enlace es igual a $s$ metros/segundo. El host A tiene que enviar un paquete de tamaño $L$ bits al host B.

   a. Exprese el retardo de propagación, $d_{prop}$, en función de $m$ y $s$.

   b. Determine el tiempo de transmisión del paquete, $d_{trans}$, en función de $L$ y $R$.

   c. Ignorando los retardos de procesamiento y de cola, obtenga una expresión para el retardo extremo a extremo.

Nota de vídeo

Exploración de los retardos de propagación y transmisión.

d. Suponga que el host A comienza a transmitir el paquete en el instante $t = 0$. En el instante $t = d_{trans}$, ¿dónde estará el último bit del paquete?

e. Suponga que $d_{prop}$ es mayor que $d_{trans}$. En el instante $t = d_{trans}$, ¿dónde estará el primer bit del paquete?

f. Suponga que $d_{prop}$ es menor que $d_{trans}$. En el instante $t = d_{trans}$, ¿dónde estará el primer bit del paquete?

g. Suponga que $s = 2{,}5 \cdot 10^8$ m/s, $L = 120$ bits y $R = 56$ kbps. Determine la distancia $m$ de modo que $d_{prop}$ sea igual a $d_{trans}$.

P7. En este problema vamos a considerar la transmisión de voz en tiempo real desde el host A al host B a través de una red de conmutación de paquetes (VoIP). El host A convierte sobre la marcha la voz analógica en un flujo de bits digital a 64 kbps. A continuación, el host A agrupa los bits en paquetes de 56 bytes. Entre el host A y el host B existe un enlace, cuya velocidad de transmisión es de 2 Mbps y cuyo retardo de propagación es igual a 10 milisegundos. Tan pronto como el host A forma un paquete, lo envía al host B. Cuando el host B recibe un paquete completo, convierte los bits del paquete en una señal analógica. ¿Cuánto tiempo transcurre desde el momento en que se crea un bit (a partir de la señal analógica en el host A) hasta que se decodifica (como parte de la señal analógica en el host B)?

P8. Suponga que varios usuarios comparten un enlace de 3 Mbps. Suponga también que cada usuario requiere 150 kbps para transmitir y que solo transmite durante el 10 por ciento del tiempo. (Véase la comparación entre conmutación de circuitos y conmutación de paquetes en la Sección 1.3.)

a. Si se utiliza la conmutación de circuitos, ¿a cuántos usuarios se puede dar soporte?

b. Para el resto de este problema, suponga que se utiliza una red de conmutación de paquetes. Halle la probabilidad de que un determinado usuario esté transmitiendo.

c. Suponga que hay 120 usuarios. Determine la probabilidad de que en un instante determinado haya exactamente $n$ usuarios transmitiendo simultáneamente. (*Sugerencia*: utilice la distribución binomial.)

d. Calcule la probabilidad de que haya 21 o más usuarios transmitiendo simultáneamente.

P9. Consulte la comparación entre conmutación de circuitos y conmutación de paquetes en la Sección 1.3, en la que se proporciona un ejemplo con un enlace a 1 Mbps. Los usuarios están generando datos a una velocidad de 100 kbps cuando están ocupados, pero solo lo están con una probabilidad de $p = 0{,}1$. Suponga que el enlace a 1 Mbps se sustituye por un enlace a 1 Gbps.

a. ¿Cuál es el valor de $N$, el número máximo de usuarios a los que se les puede dar soporte simultáneamente, cuando se emplea una red de conmutación de circuitos?

b. Considere ahora que se utiliza una red conmutación de paquetes y que el número de usuarios es $M$. Proporcione una fórmula (en función de $p$, $M$, $N$) para determinar la probabilidad de que más de $N$ usuarios estén enviando datos.

P10. Considere un paquete de longitud $L$ que tiene su origen en el sistema terminal A y que viaja a través de tres enlaces hasta un sistema terminal de destino. Estos tres enlaces están conectados mediante dos dispositivos de conmutación de paquetes. Sean $d_i$, $s_i$ y $R_i$ la longitud, la velocidad de propagación y la velocidad de transmisión del enlace $i$, para $i = 1, 2, 3$. El dispositivo de conmutación de paquetes retarda cada paquete $d_{proc}$. Suponiendo que no se producen retardos de cola, ¿cuál es el retardo total extremo a extremo del paquete, en función de $d_i$, $s_i$, $R_i$, ($i = 1, 2, 3$) y $L$? Suponga ahora que la longitud del paquete es de 1.500 bytes, que la velocidad de propagación en los tres enlaces es igual a $2{,}5 \cdot 10^8$ m/s, que la velocidad de transmisión en los tres enlaces es de 2 Mbps, que el retardo de procesamiento en el conmutador de paquetes es de 3 milisegundos, que la longitud del primer enlace es de 5.000 km, que la del

segundo es de 4.000 km y que la del último enlace es de 1.000 km. Para estos valores, ¿cuál es el retardo extremo a extremo?

P11. En el problema anterior, suponga que $R_1 = R_2 = R_3 = R$ y $d_{proc} = 0$. Suponga también que el conmutador de paquetes no almacena los paquetes y los reenvía, sino que transmite inmediatamente cada bit que recibe, sin esperar a que llegue el paquete completo. ¿Cuál será el retardo extremo a extremo?

P12. Un conmutador de paquetes recibe un paquete y determina el enlace saliente por el que deberá ser reenviado. Cuando el paquete llega, hay otro paquete que ya ha sido transmitido hasta la mitad por el mismo enlace de salida y además hay otros cuatro paquetes esperando para ser transmitidos. Los paquetes se transmiten según el orden de llegada. Suponga que todos los paquetes tienen una longitud de 1.500 bytes y que la velocidad del enlace es de 2 Mbps. ¿Cuál será el retardo de cola para el paquete? En sentido más general, ¿cuál es el retardo de cola cuando todos los paquetes tienen una longitud $L$, la velocidad de transmisión es $R$, $x$ bits del paquete que se está transmitiendo actualmente ya han sido transmitidos y hay $n$ paquetes en la cola esperando a ser transmitidos?

P13. (a) Suponga que $N$ paquetes llegan simultáneamente a un enlace en el que actualmente no se está transmitiendo ningún paquete, ni tampoco hay ningún paquete en cola. Cada paquete tiene una longitud $L$ y el enlace tiene una velocidad de transmisión $R$. ¿Cuál es el retardo medio de cola para los $N$ paquetes?

(b) Ahora suponga que llegan al enlace $N$ de esos paquetes cada $LN/R$ segundos. ¿Cuál será el retardo medio de cola de cada paquete?

P14. Considere el retardo de cola en el buffer de un router. Sea $I$ la intensidad de tráfico; es decir, $I = La/R$. Suponga que el retardo de cola puede expresarse como $IL/R\ (1 - I)$ para $I < 1$.

a. Determine una fórmula para calcular el retardo total, es decir, el retardo de cola más el retardo de transmisión.

b. Dibuje el retardo total en función de $L/R$.

P15. Sea $a$ la velocidad de llegada de los paquetes a un enlace, en paquetes/segundo, y sea $\mu$ la velocidad de transmisión del enlace en paquetes/segundo. Basándose en la fórmula del retardo total (es decir, el retardo de cola más el retardo de transmisión) obtenida en el problema anterior, deduzca una fórmula para el retardo total en función de $a$ y $\mu$.

P16. Considere el buffer de un router que precede a un enlace de salida. En este problema utilizaremos la fórmula de Little, una fórmula famosa en la teoría de colas. Sea $N$ el número medio de paquetes que hay en el buffer más el paquete que está siendo transmitido. Sea $a$ la velocidad a la que los paquetes llegan al enlace. Sea $d$ el retardo medio total (es decir, el retardo de cola más el retardo de transmisión) experimentado por un paquete. La fórmula de Little es $N = a \cdot d$. Suponga que, como media, el buffer contiene 10 paquetes y que el retardo medio de cola de un paquete es igual a 10 ms. La velocidad de transmisión del enlace es igual a 100 paquetes/segundo. Utilizando la fórmula de Little, ¿cuál es la velocidad media de llegada de los paquetes, suponiendo que no se produce pérdida de paquetes?

P17. a. Generalice la Ecuación 1.2 dada en la Sección 1.4.3 para velocidades de procesamiento, velocidades de transmisión y retardos de propagación heterogéneos.

b. Repita el apartado (a), pero suponiendo ahora que existe un retardo medio de cola $d_{cola}$ en cada nodo.

P18. Realice un trazado de la ruta (Traceroute) entre un origen y un destino situados en un mismo continente para tres horas del día diferentes.

a. Determine la media y la desviación estándar de los retardos de ida y vuelta para cada una de las horas.

b. Determine el número de routers existente en la ruta para cada una de las horas. ¿Ha variado la ruta para alguna de las horas?

Nota de vídeo

Uso de Traceroute para descubrir rutas de red y medir los retardos de red.

c. Intente identificar el número de redes de ISP que los paquetes de Traceroute atraviesan desde el origen hasta el destino. Los routers con nombres similares y/o direcciones IP similares deben considerarse como parte del mismo ISP. En sus experimentos, ¿los retardos más largos se producen en las interfaces situadas entre proveedores ISP adyacentes?

d. Repita el apartado anterior para un origen y un destino situados en diferentes continentes. Compare los resultados para ubicaciones en un mismo continente y en distintos continentes.

P19. (a) Visite el sitio www.traceroute.org y realice sendos trazados de ruta desde dos ciudades diferentes de Francia a un mismo host de destino en Estados Unidos. ¿Cuántos enlaces coinciden en los dos trazados de ruta? ¿Coincide el enlace trasatlántico?

(b) Repita el apartado (a), pero esta vez seleccione una ciudad en Francia y otra en Alemania.

(c) Seleccione una ciudad en los Estados Unidos y realice sendos trazados de ruta a dos hosts situados en ciudades diferentes de China. ¿Cuántos enlaces tienen en común los dos trazados de ruta? ¿Divergen los dos trazados de ruta antes de alcanzar China?

P20. Considere el ejemplo sobre la tasa de transferencia correspondiente a la Figura 1.20(b). Suponga ahora que hay $M$ parejas cliente-servidor en lugar de 10. Sean $R_s$, $R_c$ y $R$ las velocidades de los enlaces de servidor, de los enlaces de cliente y del enlace de red. Suponga que todos los restantes enlaces tienen la capacidad suficiente y que no existe otro tráfico en la red que el generado por las $M$ parejas cliente-servidor. Deduzca una expresión general para la tasa de transferencia en función de $R_s$, $R_c$, $R$ y $M$.

P21. Considere la Figura 1.19(b). Suponga ahora que existen $M$ rutas entre el servidor y el cliente. No hay dos rutas que compartan ningún enlace. La ruta $k$ ($k = 1, \ldots, M$) consta de $N$ enlaces con velocidades de transmisión iguales a $R_1^k, R_2^k, \ldots, R_N^k$. Si el servidor solo puede utilizar una ruta para enviar datos al cliente, ¿cuál será la máxima tasa de transferencia que puede alcanzar dicho servidor? Si el servidor puede emplear las M rutas para enviar datos, ¿cuál será la máxima tasa de transferencia que puede alcanzar el servidor?

P22. Considere la Figura 1.19(b). Suponga que cada enlace entre el servidor y el cliente tiene una probabilidad de pérdida de paquetes $p$ y que las probabilidades de pérdida de paquetes de estos enlaces son independientes. ¿Cuál es la probabilidad de que un paquete (enviado por el servidor) sea recibido correctamente por el receptor? Si un paquete se pierde en el camino que va desde el servidor hasta el cliente, entonces el servidor volverá a transmitir el paquete. Como media, ¿cuántas veces tendrá que retransmitir el paquete el servidor para que el cliente lo reciba correctamente?

P23. Considere la Figura 1.19(a). Suponga que sabemos que el enlace cuello de botella a lo largo de la ruta entre el servidor y el cliente es el primer enlace, cuya velocidad es $R_s$ bits/segundo. Suponga que envíamos un par de paquetes uno tras otro desde el servidor al cliente y que no hay más tráfico que ese en la ruta. Suponga que cada paquete tiene un tamaño de $L$ bits y que ambos enlaces presentan el mismo retardo de propagación $d_{\text{prop}}$.

a. ¿Cuál es el periodo entre llegadas de paquetes al destino? Es decir, ¿cuánto tiempo transcurre desde que el último bit del primer paquete llega hasta que lo hace el último bit del segundo paquete?

b. Suponga ahora que el enlace cuello de botella es el segundo enlace (es decir, $R_c < R_s$). ¿Es posible que el segundo paquete tenga que esperar en la cola de entrada del segundo enlace? Explique su respuesta. Suponga ahora que el servidor envía el segundo paquete $T$ segundos después de enviar el primero. ¿Qué valor debe tener $T$ para garantizar que el segundo paquete no tenga que esperar en la cola de entrada del segundo enlace? Explique su respuesta.

P24. Suponga que necesita enviar de forma urgente 40 terabytes de datos de Boston a Los Ángeles. Dispone de un enlace dedicado a 100 Mbps para la transferencia de datos. ¿Qué preferiría,

transmitir los datos a través del enlace o utilizar FedEx para hacer el envío por la noche? Explique su respuesta.

P25. Se tienen dos hosts, A y B, separados 20.000 kilómetros y conectados mediante un enlace directo con $R = 2$ Mbps. Suponga que la velocidad de propagación por el enlace es igual a $2{,}5 \cdot 10^8$ m/s.

a. Calcule el producto ancho de banda-retardo, $R \cdot d_{\text{prop}}$.

b. Supongamos que se envía un archivo cuyo tamaño es de 800.000 bits desde el host A al host B. Suponga que el archivo se envía de forma continua como un mensaje de gran tamaño. ¿Cuál es el número máximo de bits que habrá en el enlace en un instante de tiempo determinado?

c. Haga una interpretación del producto ancho de banda-retardo.

d. ¿Cuál es el ancho (en metros) de un bit dentro del enlace? ¿Es más grande que un campo de fútbol?

e. Deduzca una expresión general para la anchura de un bit en función de la velocidad de propagación *s*, la velocidad de transmisión *R* y la longitud del enlace *m*.

P26. Continuando con el Problema P25, suponga que podemos modificar *R*. ¿Para qué valor de *R* es el ancho de un bit tan grande como la longitud del enlace?

P27. Considere el Problema P25 pero ahora para un enlace con $R = 1$ Gbps.

a. Calcule el producto ancho de banda-retardo, $R \cdot d_{\text{prop}}$.

b. Considere el envío de un archivo de 800.000 bits desde el host A al host B. Suponga que el archivo se envía de forma continua como un mensaje de gran tamaño. ¿Cuál es el número máximo de bits que puede haber en el enlace en cualquier instante de tiempo dado?

c. ¿Cuál es el ancho (en metros) de un bit dentro del enlace?

P28. Haciendo referencia de nuevo al problema P25.

a. ¿Cuánto tiempo tarda en enviarse el archivo, suponiendo que se envía de forma continua?

b. Suponga ahora que el archivo se divide en 20 paquetes, conteniendo cada uno de ellos 40.000 bits. Suponga también que el receptor confirma la recepción de cada paquete y que el tiempo de transmisión de un paquete de confirmación es despreciable. Por último, suponga que el emisor no puede transmitir un paquete hasta que el anterior haya sido confirmado. ¿Cuánto tiempo se tardará en enviar el archivo?

c. Compare los resultados obtenidos en los apartados (a) y (b).

P29. Suponga que existe un enlace de microondas a 10 Mbps entre un satélite geoestacionario y su estación base en la Tierra. El satélite toma una fotografía digital por minuto y la envía a la estación base. Supongo que la velocidad de propagación es $2{,}4 \cdot 10^8$ m/s.

a. ¿Cuál es el retardo de propagación del enlace?

b. ¿Cuál es el producto ancho de banda-retardo, $R \cdot d_{\text{prop}}$?

c. Sea *x* el tamaño de la fotografía. ¿Cuál es el valor mínimo de x para que el enlace de microondas esté transmitiendo continuamente?

P30. Considere la analogía de la compañía aérea utilizada en la Sección 1.5 al hablar sobre las capas, y la adición de cabeceras a las unidades de datos del protocolo a medida que fluyen en sentido descendente por la pila de protocolos. ¿Existe algún concepto equivalente a la información de cabecera que se añada a los pasajeros y al equipaje a medida que descienden por la pila de protocolos de la compañía aérea?

P31. En las redes de conmutación de paquetes modernas, el host de origen segmenta los mensajes largos de la capa de aplicación (por ejemplo, una imagen o un archivo de música) en paquetes más pequeños y los envía a la red. Después, el receptor ensambla los paquetes para formar

el mensaje original. Este proceso se conoce como *segmentación de mensajes*. La Figura 1.27 ilustra el transporte extremo a extremo de un mensaje con y sin segmentación del mensaje. Imagine que se envía un mensaje cuya longitud es de $8 \cdot 10^6$ bits desde el origen hasta el destino mostrados en la Figura 1.27. Suponga que cada enlace de los mostrados en la figura es de 2 Mbps. Ignore los retardos de propagación, de cola y de procesamiento.

a. Suponga que el mensaje se transmite desde el origen al destino *sin* segmentarlo. ¿Cuánto tiempo tarda el mensaje en desplazarse desde el origen hasta el primer conmutador de paquetes? Teniendo en cuenta que cada conmutador de paquetes utiliza el método de conmutación de almacenamiento y reenvío, ¿cuál el tiempo total que invierte el mensaje para ir desde el host de origen hasta el host de destino?

b. Suponga ahora que el mensaje se segmenta en 800 paquetes, siendo la longitud de cada paquete igual a 10.000 bits. ¿Cuánto tiempo tarda el primer paquete en transmitirse desde el origen hasta el primer conmutador de paquetes? Mientras se está enviando el primer paquete del primer conmutador al segundo, el host de origen está enviando un segundo paquete al primer conmutador de paquetes. ¿En qué instante de tiempo habrá recibido el primer conmutador el segundo paquete completo?

c. ¿Cuánto tiempo tarda en transmitirse el archivo desde el host de origen al host de destino cuando se emplea la segmentación de mensajes? Compare este resultado con la respuesta del apartado (a) y coméntelo.

d. Además de reducir el retardo, ¿qué otras razones hay para usar la segmentación de mensajes?

e. Comente los inconvenientes de la segmentación de mensajes.

P32. Experimente con el applet *Message Segmentation* (segmentación de mensajes) disponible en el sitio web del libro. ¿Se corresponden los retardos indicados en el applet con los retardos del problema anterior? ¿Cómo afectan los retardos de propagación del enlace al retardo global extremo a extremo de la conmutación de paquetes (con segmentación de mensajes) y de la conmutación de mensajes?

P33. Se envía un archivo de gran tamaño de $F$ bits desde el host A al host B. Entre los hosts A y B hay tres enlaces (y dos dispositivos de conmutación) y los enlaces no están congestionados (es decir, no existen retardos de cola). El host A divide el archivo en segmentos de $S$ bits y añade 80 bits de cabecera a cada segmento, formando paquetes de $L = 80 + S$ bits. La velocidad de transmisión de cada enlace es de $R$ bps. Calcule el valor de $S$ que minimiza el retardo al transmitir el archivo desde el host A al host B. No tenga en cuenta el retardo de propagación.

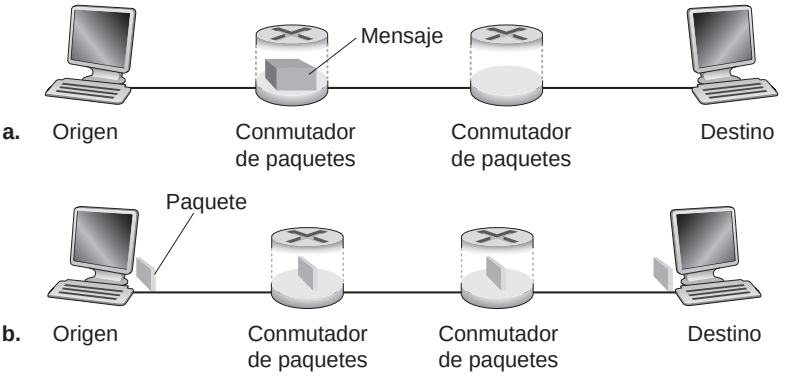



**Figura 1.27** ♦ Transporte de mensajes extremo a extremo: (a) sin segmentación de mensajes; (b) con segmentación de mensajes.

P34. Skype ofrece un servicio que permite realizar una llamada de teléfono desde un PC a un teléfono normal. Esto significa que la llamada de voz debe pasar tanto a través de Internet como a través de una red telefónica. Explique cómo podría hacerse esto.

## Prácticas de laboratorio con Wireshark

*"Dímelo y lo olvidaré. Enséñamelo y lo recordaré. Implícame y lo entenderé."*
Proverbio chino

Se puede comprender de forma más profunda los protocolos de red viéndolos en acción y jugando con ellos —observando la secuencia de mensajes intercambiados entre dos entidades, examinando los detalles de la operación del protocolo, haciendo que los protocolos lleven a cabo determinadas acciones y observando dichas acciones y sus consecuencias. Esto puede hacerse en escenarios simulados o en un entorno de red real, como Internet. Las applets Java disponibles en el sitio web del libro aplican el primero de estos métodos. En las prácticas de laboratorio con Wireshark se aplicará el segundo método. Se ejecutan aplicaciones de red en diversos escenarios utilizando una computadora doméstica, de una oficina o de un laboratorio de prácticas. Podrá observar los protocolos de red en su equipo, interactuando e intercambiando mensajes con entidades que se ejecutan en algún otro lugar de Internet. Así, usted y su computadora serán una parte integral de estas prácticas de laboratorio. Podrá observar practicando y, de ese modo, aprender.

La herramienta básica para observar los mensajes intercambiados entre entidades que ejecutan protocolos es un **husmeador de paquetes (*packet sniffer*)**. Como su nombre sugiere, un husmeador de paquetes copia de forma pasiva los mensajes que están siendo enviados y recibidos por nuestra computadora; también muestra el contenido de los distintos campos de protocolo de los mensajes capturados. En la Figura 1.28 se muestra una captura de pantalla del software Wireshark. Wireshark

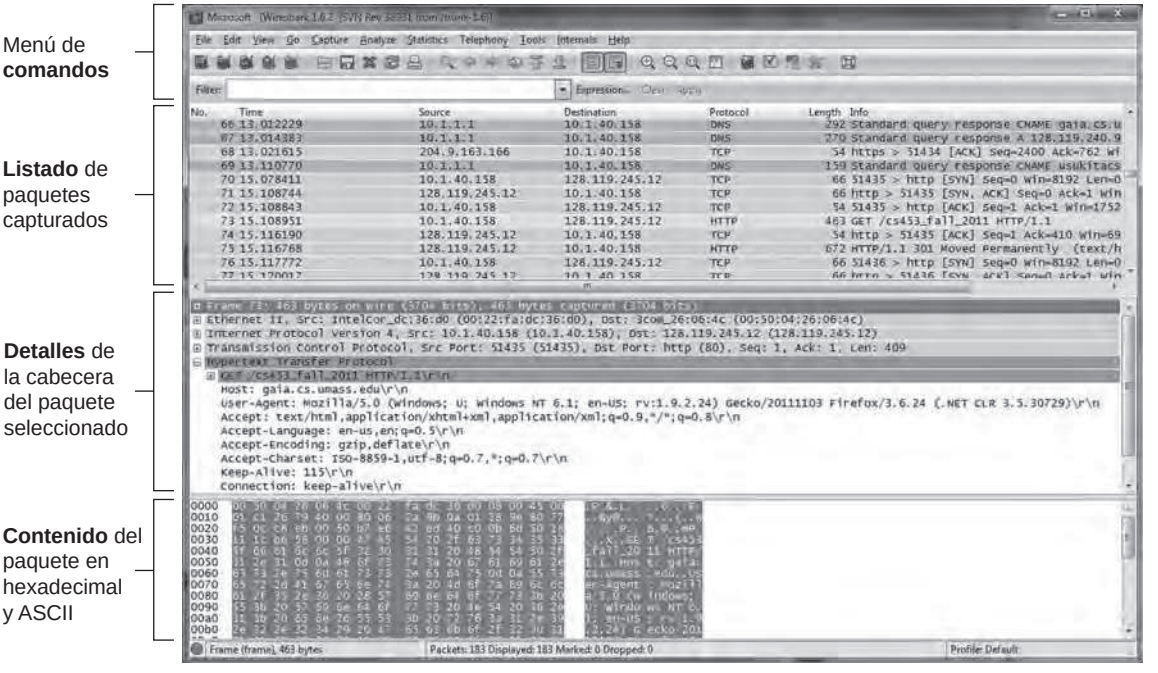



**Figura 1.28** ♦ Una captura de pantalla de Wireshark (reimpresa con permiso de Wireshark Foundation).

es un husmeador de paquetes gratuito que se ejecuta en sistemas Windows, Linux/Unix y Mac. A lo largo del libro, encontrará prácticas de laboratorio con Wireshark que le permitirán explorar algunos de los protocolos estudiados en el capítulo. En la primera práctica de laboratorio con Wireshark, tendrá que conseguir e instalar una copia de Wireshark, acceder a un sitio web y capturar y examinar los mensajes de protocolo que estén siendo intercambiados entre su navegador web y el servidor web.

Puede encontrar todos los detalles acerca de esta primera práctica de laboratorio con Wireshark (incluyendo las instrucciones acerca de cómo obtener e instalar Wireshark) en el sitio web http://www.pearsonhighered.com/cs-resources/.

**UNA ENTREVISTA CON...**

# Leonard Kleinrock

Leonard Kleinrock es catedrático de Ciencias de la Computación en la Universidad de California en Los Ángeles. En 1969, su computadora en UCLA se convirtió en el primer nodo de Internet. Su definición de los principios de la conmutación de paquetes en 1961 se convirtió en la tecnología en la que hoy se basa Internet. Recibió su licenciatura en el City College de Nueva York (CCNY) y sus títulos de máster y doctor en Ingeniería Eléctrica en el MIT.

**¿Qué fue lo que le decidió a especializarse en la tecnología de redes e Internet?**

Como estudiante de doctorado en el MIT en 1959, me di cuenta de que la mayor parte de mis compañeros de clase estaban realizando sus investigaciones en el área de la teoría de la información y de la teoría de la codificación. En el MIT se encontraba el gran investigador Claude Shannon, quien había abierto estos campos de investigación y que ya había resuelto la mayor parte de los problemas importantes. Los problemas de investigación que quedaban eran muy complicados o de consecuencias mucho menos importantes. Así que decidí iniciarme en una nueva área en la que nadie había pensado todavía. Recuerde que, en el MIT, yo estaba rodeado de montones de computadoras y vi claramente que pronto esas máquinas necesitarían comunicarse entre sí. En aquel momento, no existía una forma efectiva de hacerlo, por lo que decidí desarrollar la tecnología que permitiera crear redes de datos eficientes y fiables.

**¿Cuál fue su primer trabajo en la industria informática? ¿Qué significó para usted?**

Entre 1951 y 1957 realicé en el CCNY los estudios de grado en Ingeniería Eléctrica en el turno de tarde. Durante el día, trabajé primero como técnico y luego como ingeniero en una pequeña empresa de electrónica industrial llamada Photobell. Mientras estuve allí, introduje la tecnología digital en sus líneas de productos. Básicamente, utilizábamos dispositivos fotoeléctricos para detectar la presencia de ciertos elementos (cajas, personas, etc.) y el uso de un circuito que por entonces se denominaba *multivibrador biestable* era la clase de tecnología que necesitábamos para llevar el procesamiento digital a este campo de la detección. Estos circuitos resultaron ser la base de las computadoras y ahora se conocen como *flip-flops*, *biestables* o *conmutadores* en la jerga actual.

**¿Qué pasó por su cabeza cuando envió el primer mensaje de un host a otro (desde UCLA al Stanford Research Institute)?**

Francamente, no teníamos ni idea de la importancia de aquel suceso. No teníamos preparado un mensaje especial que pasara a la historia, como tantos otros inventores del pasado (Samuel Morse con "¡Lo que ha hecho Dios!", Alexander Graham Bell con "Watson, ¡ven aquí! Te necesito" o Neal Amstrong con "Un pequeño paso para el hombre, pero un gran paso para la Humanidad"). ¡Aquellos tipos sí eran *inteligentes*! Conocían a los medios de comunicación y sabían lo que eran las relaciones públicas. Todo lo que nosotros queríamos hacer era iniciar una sesión en la computadora del SRI. De modo que escribimos "L", lo que fue correctamente recibido, escribimos luego la letra "o", que fue recibida, y después la letra "g"… ¡que hizo que la computadora host del SRI fallara estrepitosamente! Así que nuestro mensaje fue el más corto de la historia: "Lo".

Anteriormente, aquel año, yo había sido citado en una nota de prensa de UCLA diciendo que una vez que la red estuviera activa y funcionando, sería posible acceder a las utilidades informáticas desde nuestros hogares y oficinas tan fácilmente como ya era disponer de electricidad y teléfono. Por tanto, mi visión en aquel momento era que Internet estaría en todas partes, siempre en funcionamiento, siempre disponible, y que cualquiera con cualquier dispositivo podría conectarse desde cualquier lugar y que sería invisible. Sin embargo, nunca pude prever que mi madre con 99 años utilizaría Internet, ¡y ya lo creo que la utilizó!

**¿Cómo ve el futuro de las redes?**

La parte fácil de la visión es predecir la propia infraestructura. Preveo que veremos una considerable implantación de la computación nómada, los dispositivos móviles y los espacios inteligentes. De hecho, la disponibilidad de dispositivos informáticos y de comunicaciones ligeros, baratos, de altas prestaciones y portátiles (combinada con la omnipresencia de Internet) nos ha permitido convertirnos en nómadas. La computación nómada hace referencia a la tecnología que permite a los usuarios finales ir de un lugar a otro, obteniendo acceso a los servicios de Internet de forma transparente, independientemente de por dónde viajen e independientemente del dispositivo que utilicen o que lleven consigo. La parte más complicada de esta visión de futuro es predecir las aplicaciones y servicios, que siempre nos han sorprendido de forma increíble (correo electrónico, tecnologías de búsqueda, la World Wide Web, los blogs, las redes sociales, la generación y compartición de música, fotografías y vídeos por parte de los usuarios, etc.). Nos encontramos en el amanecer de una nueva era de aplicaciones móviles sorprendentes e innovadoras, que podremos utilizar con nuestros dispositivos de mano.

La siguiente ola nos permitirá pasar del mundo virtual del ciberespacio al mundo físico de los espacios inteligentes. Nuestros entornos (mesas, paredes, vehículos, relojes, etc.) cobrarán vida con la tecnología, mediante actuadores, sensores, lógica, procesamiento, almacenamiento, cámaras, micrófonos, altavoces, pantallas y comunicación. Esta tecnología integrada permitirá a nuestro entorno proporcionar los servicios IP que deseemos. Cuando accedamos a una habitación, la habitación sabrá que hemos entrado. Podremos comunicarnos con nuestro entorno de forma natural, hablando en nuestra lengua materna; nuestras solicitudes generarán respuestas que nos presentarán páginas web en pantallas de pared, en los cristales de las gafas, en forma de texto hablado, de hologramas, etc.

Mirando un poco más lejos, preveo un futuro en red que incluya los siguientes componentes adicionales fundamentales. Veo agentes software inteligentes por toda la red, cuya función será hacer minería de datos, actuar de acuerdo con ellos, detectar tendencias y llevar a cabo tareas de forma dinámica y adaptativa. Preveo que habrá una cantidad de tráfico de red considerablemente mayor, generada no tanto por los seres humanos, sino por estos dispositivos integrados y esos agentes software inteligentes. Preveo que habrá grandes conjuntos de sistemas dotados de auto-organización y encargados de controlar esa red tan enorme y tan rápida. Preveo que habrá enormes cantidades de información viajando instantáneamente a través de esa red y viéndose sometida en el camino a enormes cantidades de procesamiento y de filtrado. Internet será, esencialmente, un sistema nervioso global que llegará a todas partes. Preveo que sucederán todas estas cosas y muchas más a medida que nos vayamos adentrando en el siglo XXI.

**¿Qué personas le han inspirado profesionalmente?**

Con diferencia, el que más me ha inspirado ha sido Claude Shannon del MIT, un brillante investigador que tenía la capacidad de relacionar sus ideas matemáticas con el mundo físico de una forma extremadamente intuitiva. Estuvo en el tribunal de mi tesis doctoral.

**¿Tiene algún consejo para los estudiantes que se inician ahora en el campo de las redes y de Internet?**

Internet y todo lo que esa red hace posible constituye una nueva frontera de grandes dimensiones, llena de desafíos asombrosos. Hay grandes oportunidades para la innovación y no hay que sentirse restringido por la tecnología actual. Lo que hay que hacer es abrir la mente e imaginar cómo podrían ser las cosas, y luego hacer que eso suceda.

CAPÍTULO 2

# La capa de aplicación

Las aplicaciones de red son la *razón de ser* de una red de computadoras: si no pudiéramos concebir ninguna aplicación útil, no existiría la necesidad de disponer de una infraestructura de red y de unos protocolos para darlas soporte. Desde la aparición de Internet, se han creado muchas aplicaciones de red útiles y entretenidas. Estas aplicaciones han sido la fuerza motriz del éxito de Internet, motivando a las personas en sus domicilios, escuelas, gobiernos y empresas a hacer de Internet una parte integrante de sus actividades cotidianas.

Entre las aplicaciones Internet se incluyen las clásicas aplicaciones basadas en texto que se hicieron populares en las décadas de 1970 y 1980, como son el correo electrónico de texto, el acceso remoto a computadoras, la transferencia de archivos y los grupos de noticias. También se incluye la aplicación por excelencia de mediados de la década de 1990: la World Wide Web, que incluye la navegación web, las búsquedas y el comercio electrónico. También se incluyen la mensajería instantánea y la compartición de archivos P2P, las dos aplicaciones estrella introducidas a finales del pasado milenio. En este siglo que comienza continúan apareciendo aplicaciones nuevas y atractivas, incluyendo voz sobre IP y sistemas de videoconferencia como Skype, Facetime y Google Hangouts; sistemas de vídeo generado por el usuario, como YouTube, y de películas a la carta, como Netflix; y juegos en línea multijugador como Second Life y World of Warcraft. Durante este mismo periodo, hemos visto el nacimiento de una nueva generación de aplicaciones de redes sociales (como Facebook, Instagram, Twitter y WeChat) que han creado adictivas redes de personas por encima de la red de routers y enlaces de comunicaciones de Internet. Y más recientemente, junto con la llegada del teléfono inteligente, ha surgido una profusión de aplicaciones móviles basadas en la ubicación, incluyendo populares aplicaciones de reservas, de citas y de previsiones de tráfico, como Yelp, Tinnder, Waz y Yik Yak. Claramente, no se ha ralentizado el ritmo de aparición de nuevas y excitantes aplicaciones Internet. ¡Quizá algunos de los lectores de este texto sean los que desarrollen la siguiente generación de aplicaciones que reinen en Internet!

En este capítulo vamos a estudiar los aspectos conceptuales y de implementación de las aplicaciones de red. Comenzaremos definiendo los conceptos fundamentales relativos a la capa de aplicación, incluyendo los servicios de red requeridos por las aplicaciones, los clientes y servidores, los procesos y las interfaces de la capa de transporte. Examinaremos en detalle varias aplicaciones

de red, como la Web, el correo electrónico, el sistema DNS, la distribución de archivos en redes entre pares (P2P, *Peer-to-Peer*) y los flujos de vídeo. (En el Capítulo 9 se examinarán más en detalle las aplicaciones multimedia, incluyendo los flujos de vídeo y VoIP.) A continuación, nos ocuparemos del desarrollo de las aplicaciones de red, tanto sobre TCP como UDP. En particular, estudiaremos la interfaz de sockets y echaremos un vistazo a algunas aplicaciones cliente-servidor simples implementadas en Python. También proporcionaremos al final del capítulo varias divertidas e interesantes tareas de programación de sockets.

La capa de aplicación es un lugar particularmente bueno para comenzar nuestro estudio de los protocolos, ya que es un terreno conocido. Estamos familiarizados con muchas de las aplicaciones que se basan en los protocolos que vamos a estudiar. Nos dará una idea adecuada de qué son los protocolos y nos servirá para introducir muchas de las cuestiones que tendremos que volver a ver cuando estudiemos los protocolos de las capas de transporte, de red y de enlace.

## 2.1 Principios de las aplicaciones de red

Imagine que se le ha ocurrido una idea para desarrollar una nueva aplicación de red. Es posible que esa aplicación llegue a prestar un gran servicio a la Humanidad, o que le guste a su profesor, o que le haga ganar una fortuna o que, simplemente, le divierta desarrollarla. Sea cual sea la motivación, a continuación vamos a ver cómo transformar la idea en una aplicación de red real.

Básicamente, el desarrollo de una aplicación de red implica escribir programas que se ejecuten en distintos sistemas terminales y que se comuniquen entre sí a través de la red. Por ejemplo, en la aplicación Web se emplean dos programas diferentes que se comunican entre sí: el navegador que se ejecuta en el host del usuario (una computadora de escritorio, un portátil, una tableta, un teléfono inteligente, etc.) y el programa del servidor web que se ejecuta en el host que actúa como servidor web. Otro ejemplo sería el caso de un sistema de compartición de archivos P2P, en el que se emplea un programa en cada host que participa en la comunidad de compartición de archivos. En este caso, los programas instalados en los distintos hosts pueden ser similares o idénticos.

Por tanto, al desarrollar su nueva aplicación tendrá que escribir software que se ejecutará en múltiples sistemas terminales. Este software podría escribirse, por ejemplo, en C, Java o Python. Una cuestión importante es que no es necesario escribir software que se ejecute en los dispositivos del núcleo de la red, como por ejemplo los routers o los switches de la capa de enlace. Incluso aunque deseara escribir software de aplicación para estos dispositivos del núcleo de la red, no podría hacerlo. Como hemos visto en el Capítulo 1 y se ilustra en la Figura 1.24, los dispositivos del núcleo de la red no operan en la capa de aplicación, sino en las capas situadas mas abajo: específicamente, en la capa de red e inferiores. Este diseño básico (que confina el software de aplicación a los sistemas terminales), como que se muestra en la Figura 2.1, ha facilitado el rápido desarrollo e implementación de una gran variedad de aplicaciones de red.

### 2.1.1 Arquitecturas de las aplicaciones de red

Antes de profundizar en la codificación del software, deberíamos disponer de una visión general de la arquitectura de la aplicación. Tenga en cuenta que la arquitectura de una aplicación es muy distinta de la arquitectura de la red (es decir, de la arquitectura de Internet de cinco capas vista en el Capítulo 1). Desde la perspectiva del desarrollador de aplicaciones, la arquitectura de la red es fija y proporciona un conjunto específico de servicios a las aplicaciones. Por otro lado, el desarrollador de aplicaciones diseña la **arquitectura de la aplicación**, que establece cómo debe estructurarse la aplicación en los distintos sistemas terminales. Al seleccionar la arquitectura de la aplicación, el desarrollador probablemente utilizará uno de los dos paradigmas arquitectónicos predominantes en las aplicaciones de red modernas: la arquitectura cliente-servidor o la arquitectura P2P.

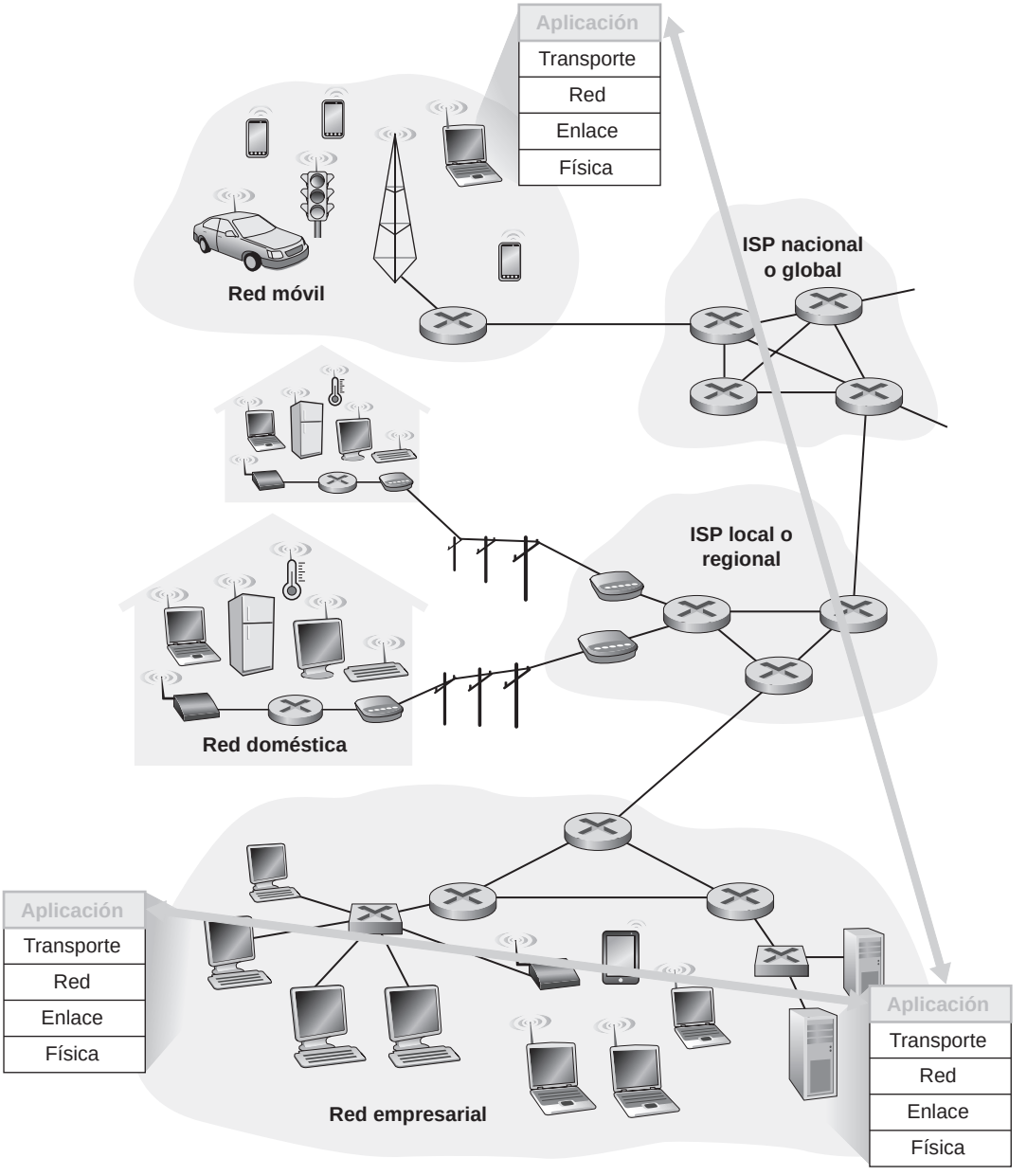


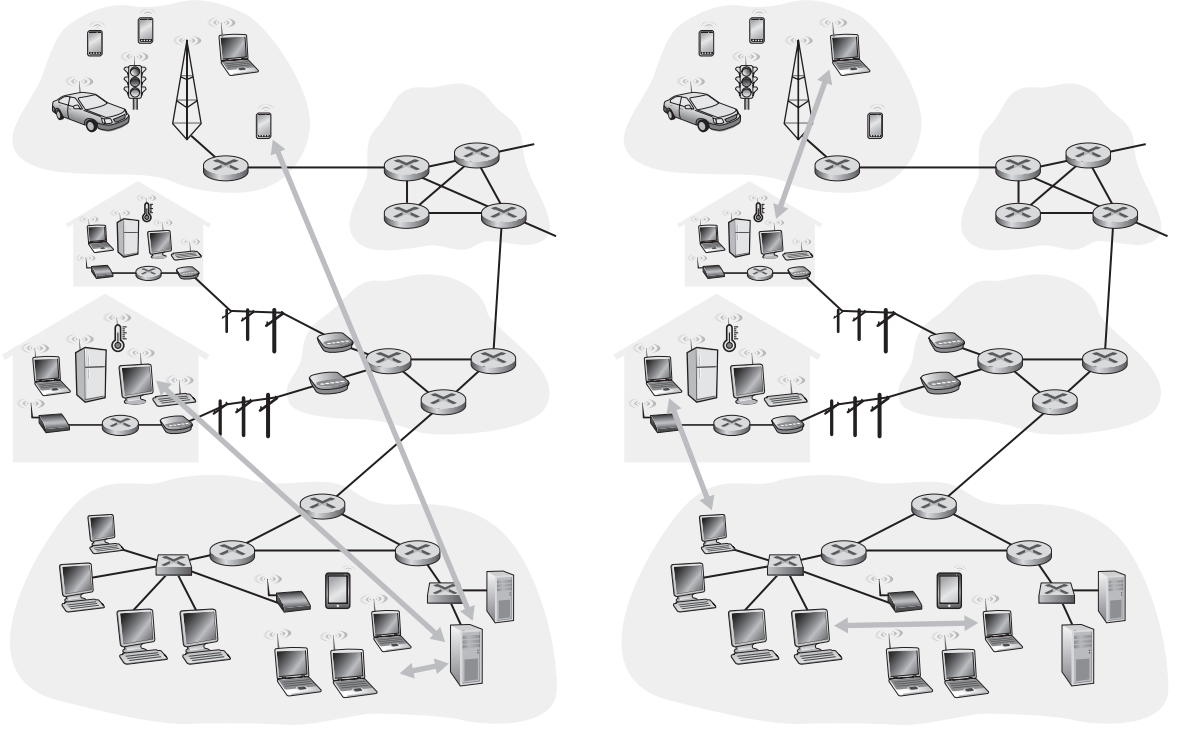

**Figura 2.1** ♦ La comunicación de una aplicación de red tiene lugar entre sistemas terminales en la capa de aplicación.

En una **arquitectura cliente-servidor** existe un host siempre activo, denominado *servidor*, que da servicio a las solicitudes de muchos otros hosts, que son los *clientes*. Un ejemplo clásico es la Web, en la que un servidor web siempre activo sirve las solicitudes de los navegadores que se ejecutan en los hosts clientes. Cuando un servidor web recibe una solicitud de un objeto de un host cliente, responde enviándole el objeto solicitado. Observe que, con la arquitectura cliente-servidor, los clientes no se comunican directamente entre sí; por ejemplo, en la aplicación web, dos navegadores no se comunican de forma directa. Otra característica de la arquitectura cliente-servidor es que el servidor tiene una dirección fija y conocida, denominada dirección IP (de la que hablaremos enseguida). Puesto que el servidor tiene una dirección fija y conocida, y siempre está

activo, un cliente siempre puede contactar con él enviando un paquete a su dirección IP. Entre las aplicaciones más conocidas que utilizan la arquitectura cliente-servidor se encuentran la Web, FTP, Telnet y el correo electrónico. En la Figura 2.2(a) se muestra la arquitectura cliente-servidor.

A menudo, en una aplicación cliente-servidor un único host servidor es incapaz de responder a todas las solicitudes de sus clientes. Por ejemplo, el sitio de una red social popular puede verse rápidamente desbordado si sólo dispone de un servidor para gestionar todas las solicitudes. Por esta razón, en las arquitecturas cliente-servidor suele utilizarse un **centro de datos**, que alberga un gran número de hosts, para crear un servidor virtual de gran capacidad. Los servicios internet más populares –como los motores de búsqueda (p. ej. Google, Bing o Baidu), los sitios de comercio por Internet (p. ej. Amazon, e-Bay o Alibaba), el correo electrónico basado en la Web (p. ej. Gmail y Yahoo Mail) o las redes sociales (como Facebook, Instagram, Twitter y WeChat)– emplean uno o más centros de datos. Como se explica en la Sección 1.3.3, Google dispone de entre 30 y 50 centros de datos distribuídos por todo el mundo, que se encargan de gestionar colectivamente las búsquedas, YouTube, Gmail y otros servicios. Un centro de datos puede tener cientos de miles de servidores, a los que hay que suministrar energía y mantener. Además, los proveedores del servicio deben pagar los costes recurrentes de interconexión y de ancho de banda para poder enviar datos desde sus centros de datos.

En una **arquitectura P2P** existe una mínima (o ninguna) dependencia de una infraestructura de servidores dedicados situados en centros de datos. En su lugar, la aplicación explota la comunicación directa entre parejas de hosts conectados de forma intermitente, conocidos como pares (*peers*). Los pares no son propiedad del proveedor del servicio, sino que son computadoras de escritorio y portátiles controladas por los usuarios, encontrándose la mayoría de los pares en domicilios, universidades y oficinas. Puesto que los pares se comunican sin pasar por un servidor dedicado, la arquitectura se denomina arquitectura *peer-to-peer* (P2P). Muchas de las aplicaciones actuales más populares y con un mayor nivel de tráfico están basadas en arquitecturas P2P. Entre estas



a. Arquitectura cliente-servidor. b. Arquitectura P2P.

**Figura 2.2** ♦ (a) Arquitectura cliente-servidor; (b) Arquitectura P2P.

aplicaciones se incluyen la compartición de archivos (por ejemplo, BitTorrent), la aceleración de descarga con ayuda de pares (por ejemplo, Xunlei) y la telefonía y videoconferencia por Internet (por ejemplo, Skype). En la Figura 2.2(b) se ilustra la arquitectura P2P. Conviene mencionar que algunas aplicaciones tienen arquitecturas híbridas, que combinan elementos cliente-servidor y P2P. Por ejemplo, en muchas aplicaciones de mensajería instantánea, los servidores se utilizan para hacer un seguimiento de las direcciones IP de los usuarios, pero los mensajes de usuario a usuario se envían directamente entre los hosts de dichos usuarios (sin pasar por servidores intermedios).

Una de las características más convincentes de las arquitecturas P2P es su **auto-escalabilidad**. Por ejemplo, en una aplicación de compartición de archivos P2P, aunque cada par genera una carga de trabajo solicitando archivos, también añade capacidad de servicio al sistema, distribuyendo archivos a otros pares. Las arquitecturas P2P también presentan una buena relación coste-prestaciones, ya que normalmente no requieren una infraestructura de servidores significativa ni un gran ancho de banda de servidor (a diferencia de los diseños cliente-servidor con centros de datos). Sin embargo, las aplicaciones P2P plantean problemas de seguridad, rendimiento y fiabilidad, debido a su naturaleza altamente descentralizada.

## 2.1.2 Comunicación entre procesos

Antes de crear su aplicación de red, también necesita disponer de unos conocimientos básicos sobre cómo se comunican entre sí los programas que se ejecutan en varios sistemas terminales. En la jerga de los sistemas operativos, realmente los que se comunican no son programas, sino **procesos**. Un proceso puede interpretarse como un programa que se ejecuta dentro de un sistema terminal. Cuando los procesos se ejecutan en el mismo sistema terminal, pueden comunicarse entre sí mediante sistemas de comunicación inter-procesos, aplicando reglas gobernadas por el sistema operativo del sistema terminal. Pero, en este libro, no estamos especialmente interesados en cómo se comunican los procesos dentro de un mismo host, sino en cómo se comunican los procesos que se ejecutan en hosts *diferentes* (con sistemas operativos potencialmente diferentes).

Los procesos de dos sistemas terminales diferentes se comunican entre sí intercambiando **mensajes** a través de la red de computadoras. Un proceso emisor crea y envía mensajes a la red; un proceso receptor recibe estos mensajes y posiblemente responde devolviendo otros mensajes. La Figura 2.1 ilustra que los procesos que se comunican entre sí residen en la capa de aplicación de la pila de protocolos de cinco capas.

### Procesos cliente y servidor

Una aplicación de red consta de parejas de procesos que se envían mensajes entre sí a través de una red. Por ejemplo, en la aplicación Web un proceso navegador de un cliente intercambia mensajes con un proceso de un servidor web. En un sistema de compartición de archivos P2P, se transfiere un archivo desde un proceso que se ejecuta en un par, a un proceso de otro par. Normalmente, en una pareja de procesos que están comunicándose, designamos a uno de los procesos como el **cliente** y al otro como el **servidor**. En la Web, un navegador es un proceso cliente y el servidor web es un proceso servidor. En la compartición de archivos P2P, el par que descarga el archivo se designa como el cliente y el host que está cargando el archivo se designa como el servidor.

Es posible que haya observado que en algunas aplicaciones, tales como la compartición de archivos P2P, un proceso puede ser tanto un cliente como un servidor. De hecho, un proceso en un sistema de compartición de archivos P2P puede cargar y descargar archivos. No obstante, en el contexto de cualquier sesión de comunicación específica entre una pareja de procesos, seguimos pudiendo designar a uno de los procesos como el cliente y al otro como el servidor. Definimos los procesos cliente y servidor como sigue:

> *En el contexto de una sesión de comunicación entre una pareja de procesos, el proceso que inicia la comunicación (es decir, que inicialmente se pone en contacto con el otro proceso al*

*principio de la sesión) se designa como el* cliente. *El proceso que espera a ser contactado para comenzar la sesión es el* servidor.

En la Web, un proceso navegador inicia el contacto con un proceso servidor web; por tanto, el proceso navegador es el cliente y el proceso servidor web es el servidor. En la compartición de archivos P2P, cuando un par A pide a un par B que le envíe un determinado archivo, el A es el cliente y el B es el servidor en el contexto de esta sesión de comunicación concreta. En ocasiones, cuando no exista ningún tipo de ambigüedad, también emplearemos la terminología "lado del cliente y lado del servidor de una aplicación". Al final del capítulo, examinaremos paso a paso un código simple tanto para el lado del cliente como para el lado del servidor de las aplicaciones de red.

### Interfaz entre el proceso y la red de computadoras

Como hemos mencionado, la mayoría de las aplicaciones constan de parejas de procesos que se comunican, intercambiándose mensajes. Cualquier mensaje enviado de un proceso al otro debe atravesar la red subyacente. Un proceso envía mensajes a la red y los recibe de la red a través de una interfaz software denominada **socket**. Veamos una analogía que nos ayudará a comprender los conceptos de proceso y socket. Un proceso es análogo a una casa y un socket es análogo a la puerta de la casa. Cuando un proceso desea enviar un mensaje a otro proceso que se está ejecutando en otro host, envía el mensaje a través de su propia puerta (socket). Este proceso emisor supone que existe una infraestructura de transporte al otro lado de la puerta que llevará el mensaje hasta la puerta del proceso de destino. Una vez que el mensaje llega al host de destino, éste pasa a través de la puerta (socket) del proceso receptor, actuando entonces el proceso receptor sobre el mensaje.

La Figura 2.3 ilustra la comunicación mediante sockets entre dos procesos que se comunican a través de Internet. (En la Figura 2.3 se supone que el protocolo de transporte subyacente utilizado por los procesos es el protocolo TCP de Internet.) Como se muestra en la figura, un socket es la interfaz entre la capa de aplicación y la capa de transporte de un host. También se conoce como **interfaz de programación de aplicaciones (API, *Application Programming Interface*)** entre la aplicación y la red, ya que el socket es la interfaz de programación con la que se construyen las aplicaciones de red. El desarrollador de la aplicación tiene el control sobre todos los elementos del lado de la capa de aplicación del socket, pero apenas tiene control sobre el lado de la capa de transporte del socket. El único control que tiene el desarrollador de la aplicación sobre el lado de la capa de transporte es (1) la elección del protocolo de transporte y (2) quizá la capacidad de fijar unos pocos parámetros de la capa de transporte, como por ejemplo los tamaños máximo del buffer

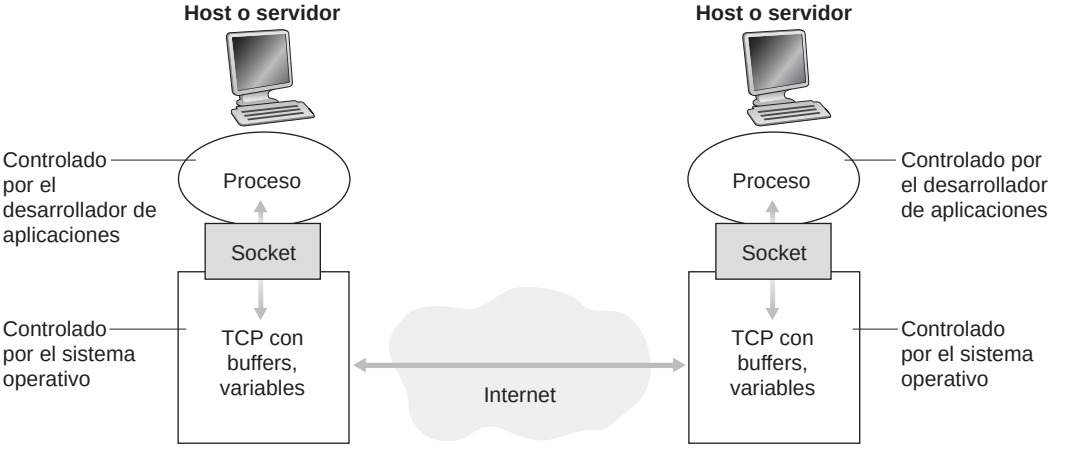



**Figura 2.3** ♦ Procesos de aplicación, sockets y protocolo de transporte subyacente.

y de segmento (lo que veremos en el Capítulo 3). Una vez que el desarrollador de la aplicación ha seleccionado un protocolo de transporte (si hay disponibles varios entre los que elegir), la aplicación se construye utilizando los servicios de la capa de transporte proporcionados por dicho protocolo. En las Secciones 2.7 exploraremos los sockets con un cierto grado de detalle.

### Direccionamiento de procesos

Para enviar una carta por correo postal a un destino concreto, ese destino necesita disponer de una dirección. De forma similar, para que un proceso que se está ejecutando en un host pueda enviar paquetes a otro proceso que se ejecuta en un host distinto, el proceso receptor necesita disponer de una dirección. Para identificar al proceso receptor, tienen que especificarse dos elementos de información: (1) la dirección del host y (2) un identificador que especifique el proceso de recepción en el host de destino.

En Internet, el host se identifica mediante su **dirección IP**. En el Capítulo 4 veremos en detalle las direcciones IP. Por el momento, todo lo que necesitamos saber es que una dirección IP es una magnitud de 32 bits que identifica de forma unívoca a un host. Además de conocer la dirección del host al que está destinado un mensaje, el host emisor también debe identificar el proceso receptor (o, para ser más exactos, el socket receptor) que está ejecutándose en el host. Esta información es necesaria porque, en general, un host podría estar ejecutando muchas aplicaciones de red. Para satisfacer este requisito, se utiliza un **número de puerto** de destino. Las aplicaciones populares tienen asignados números de puerto específicos. Por ejemplo, un servidor web queda identificado por el número de puerto 80. Un proceso servidor de correo (que utilice el protocolo SMTP) se identifica mediante el número de puerto 25. Puede encontrar una lista de números de puerto bien conocidos para todos los protocolos estándar de Internet en http://www.iana.org. Examinaremos los números de puerto en detalle en el Capítulo 3.

## 2.1.3 Servicios de transporte disponibles para las aplicaciones

Recordemos que un socket es la interfaz entre el proceso de la aplicación y el protocolo de la capa de transporte. La aplicación del lado emisor envía los mensajes a través del socket. En el otro lado del socket, el protocolo de la capa de transporte tiene la responsabilidad de llevar los mensajes hasta el socket del proceso receptor.

Muchas redes, incluyendo Internet, proporcionan más de un protocolo de la capa de transporte. Cuando vaya a desarrollar una aplicación, tendrá que elegir uno de los protocolos de la capa de transporte disponibles. ¿Cómo llevar a cabo esta selección? Muy probablemente, tendrá que estudiar los servicios que ofrecen los protocolos de la capa de transporte disponibles y después elegir aquel protocolo que proporcione los servicios que mejor se adapten a las necesidades de la aplicación. La situación es similar a tener que elegir entre viajar en tren o en avión para ir de una ciudad a otra. Tiene que elegir un medio de transporte u otro, y cada uno de ellos ofrece servicios diferentes (por ejemplo, el tren le ofrece partir y llegar al centro de las ciudades, mientras que el avión ofrece un tiempo de viaje más corto).

¿Cuáles son los servicios que puede ofrecer un protocolo de la capa de transporte a las aplicaciones que lo invocan? Podemos clasificar los posibles servicios de forma bastante general según cuatro parámetros: transferencia de datos fiable, tasa de transferencia, temporización y seguridad.

### Transferencia de datos fiable

Como se ha explicado en el Capítulo 1, en una red de computadoras pueden perderse paquetes. Por ejemplo, un paquete puede desbordar el buffer de un router, o podría ser descartado por un host o un router después de comprobar que algunos de sus bits están corrompidos. En muchas aplicaciones (como el correo electrónico, la transferencia de archivos, el acceso remoto a hosts, la transferencia

de documentos web y las aplicaciones financieras) la pérdida de datos puede tener consecuencias catastróficas (¡en el último caso, para el banco o para el cliente!). Por tanto, para dar soporte a estas aplicaciones, es preciso hacer algo para garantizar que los datos enviados desde un terminal de la aplicación sean todos ellos entregados correcta y completamente al otro terminal de la aplicación. Si un protocolo proporciona un servicio de entrega de datos garantizado, se dice que proporciona una **transferencia de datos fiable**. Un servicio importante que un protocolo de la capa de transporte puede potencialmente proporcionar a una aplicación es la transferencia de datos fiable proceso a proceso. Cuando un protocolo de transporte suministra este servicio, el proceso emisor puede pasar sus datos al socket y saber con certidumbre absoluta que los datos llegarán sin errores al proceso receptor.

Si un protocolo de la capa de transporte no proporciona una transferencia de datos fiable, los datos enviados por el proceso emisor pueden no llegar nunca al proceso de recepción. Esto puede ser aceptable para las **aplicaciones tolerantes a pérdidas**; por ejemplo, la mayor parte de las aplicaciones multimedia, como las de audio/vídeo en tiempo real, pueden tolerar que cierta cantidad de datos se pierda. En estas aplicaciones multimedia, la pérdida de datos puede dar lugar a una pequeña interrupción al reproducir el audio/vídeo, lo que no constituye un problema fundamental.

### Tasa de transferencia

En el Capítulo 1 hemos presentado el concepto de tasa de transferencia disponible, que en el contexto de una sesión de comunicaciones entre dos procesos a lo largo de una ruta de red, es la tasa a la que el proceso emisor puede suministrar bits al proceso de recepción. Puesto que otras sesiones compartirán el ancho de banda a lo largo de la ruta de red y puesto que esas otras sesiones se iniciarán y terminarán aleatoriamente, la tasa de transferencia disponible puede fluctuar con el tiempo. Estas observaciones nos llevan de forma natural a otro servicio que un protocolo de la capa de transporte podría proporcionar: una tasa de transferencia disponible garantizada a un cierta velocidad especificada. Con un servicio así, la aplicación podría solicitar una tasa de transferencia garantizada de $r$ bits/segundo y el protocolo de transporte podría entonces garantizar que la tasa de transferencia disponible sea siempre al menos de $r$ bits/segundo. Un servicio que ofreciera una tasa de transferencia garantizada como esta resultaría atractivo para muchas aplicaciones. Por ejemplo, si una aplicación de telefonía por Internet codifica voz a 32 kbps, tendrá que enviar datos a la red y tendrá que entregar los datos a la aplicación receptora a esa velocidad. Si el protocolo de transporte no puede proporcionar esa tasa de transferencia, la aplicación tendrá que realizar la codificación a una velocidad menor (y recibir a una tasa de transferencia adecuada como para mantener esa velocidad de codificación más lenta) o bien tendrá que renunciar, puesto que recibir a la mitad de la tasa de transferencia necesaria no tiene ninguna utilidad para esta aplicación de telefonía por Internet. Las aplicaciones con requisitos de tasa de transferencia se conocen como **aplicaciones sensibles al ancho de banda**. Muchas aplicaciones multimedia actuales son sensibles al ancho de banda, pero algunas de ellas pueden emplear técnicas de codificación adaptativa para codificar la voz o el vídeo digitalizados a una velocidad que se adapte a la tasa de transferencia disponible en cada momento.

Mientras que las aplicaciones sensibles al ancho de banda tienen requisitos específicos de tasa de transferencia, las denominadas **aplicaciones elásticas** pueden hacer uso de la tasa de transferencia, mucha o poca, que haya disponible. El correo electrónico, la transferencia de archivos y las transferencias web son todas ellas aplicaciones elásticas. Por supuesto, cuanto mayor sea la tasa de transferencia, mejor. (¡Como reza el dicho, es imposible ser demasiado rico, demasiado guapo o tener demasiada tasa de transferencia!)

### Temporización

Un protocolo de la capa de transporte también puede proporcionar garantías de temporización. Al igual que con las tasas de transferencia garantizadas, las garantías de temporización también pueden darse de diversas formas. Un ejemplo de garantía podría ser que cada bit que el emisor empuja

por su socket llegue al socket del receptor en no más de 100 milisegundos. Un servicio así sería atractivo para las aplicaciones interactivas en tiempo real, como la telefonía por Internet, los entornos virtuales, la teleconferencia y los juegos multijugador, todas las cuales requieren restricciones de temporización muy estrictas sobre la entrega de datos para ser efectivas. (Véase el Capítulo 9 y [Gauthier 1999; Ramjee 1994].) Por ejemplo, los retardos largos en la telefonía por Internet tienden a dar lugar a pausas antinaturales en una conversación; en un juego multijugador o en un entorno virtual interactivo, un retardo largo entre la realización de una acción y la visualización de la respuesta del entorno (por ejemplo, de otro jugador que se encuentra en el otro extremo de una conexión extremo a extremo) hace que la aplicación parezca menos realista. En las aplicaciones que no se ejecutan en tiempo real, siempre es preferible un retardo pequeño que uno grande, pero no se aplican restricciones estrictas a los retardos extremo a extremo.

### Seguridad

Por último, un protocolo de transporte puede proporcionar a una aplicación uno o más servicios de seguridad. Por ejemplo, en el host emisor, un protocolo de transporte puede cifrar todos los datos transmitidos por el proceso emisor, y en el host receptor puede descifrar los datos antes de entregarlos al proceso receptor. Un servicio así proporcionaría confidencialidad entre los dos procesos, incluso aunque los datos sean observados de alguna manera entre los procesos emisor y receptor. Un protocolo de transporte también puede proporcionar otros servicios de seguridad además del de la confidencialidad, como pueden ser mecanismos para garantizar la integridad de los datos y mecanismos de autenticación en el punto terminal, temas que abordaremos en detalle en el Capítulo 8.

## 2.1.4 Servicios de transporte proporcionados por Internet

Hasta el momento hemos considerado los servicios de transporte que una red de computadoras *podría* proporcionar en general. Seamos ahora un poco más específicos y examinemos los tipos de servicio de transporte que Internet proporciona a las aplicaciones. Internet (y, de forma más general, las redes TCP/IP) pone a disposición de las aplicaciones dos protocolos de transporte: UDP y TCP. Cuando usted, como desarrollador de aplicaciones, cree una nueva aplicación de red para Internet, una de las primeras decisiones que tendrá que tomar es si utilizar UDP o TCP. Cada uno de estos protocolos ofrece un conjunto diferente de servicios a las aplicaciones que los invocan. La Figura 2.4 detalla los requisitos de servicio para algunas aplicaciones seleccionadas.

| Aplicación | Pérdida de datos | Tasa de transferencia | Sensible al tiempo |
|---|---|---|---|
| Transferencia/descarga de archivos | Sin pérdidas | Elástica | No |
| Correo electrónico | Sin pérdidas | Elástica | No |
| Documentos web | Sin pérdidas | Elástica (pocos kbps) | No |
| Telefonía por Internet/ Videoconferencia | Tolerante a las pérdidas | Audio: unos pocos kbps–1 Mbps<br>Vídeo: 10 kbps–5 Mbps | Sí: décimas de segundo |
| Flujos de audio/vídeo almacenado | Tolerante a las pérdidas | Como el anterior | Sí: unos pocos segundos |
| Juegos interactivos | Tolerante a las pérdidas | Unos pocos kbps–10 kbps | Sí: décimas de segundo |
| Mensajería para teléfono inteligente | Sin pérdidas | Elástica | Sí y no |

**Figura 2.4** ♦ Requisitos de algunas aplicaciones de red seleccionadas.

### Servicios TCP

El modelo de servicio TCP incluye un servicio orientado a la conexión y un servicio de transferencia de datos fiable. Cuando una aplicación invoca TCP como su protocolo de transporte, la aplicación recibe ambos servicios de TCP.

- *Servicio orientado a la conexión.* TCP hace que el cliente y el servidor intercambien entre sí información de control de la capa de transporte *antes* de que empiecen a fluir los mensajes del nivel de aplicación. Este procedimiento denominado de negociación, de reconocimiento o de establecimiento de la conexión, alerta al cliente y al servidor, permitiéndoles prepararse para el intercambio de paquetes. Después de esta fase de negociación, se dice que existe una **conexión TCP** entre los sockets de los dos procesos. La conexión es una conexión full-duplex, en el sentido de que los dos procesos pueden enviarse mensajes entre sí a través de la conexión al mismo tiempo. Una vez que la aplicación ha terminado de enviar mensajes, debe terminar la conexión. En el Capítulo 3 examinaremos en detalle los servicios orientados a la conexión y veremos cómo se implementan.
- *Servicio de transferencia de datos fiable.* Los procesos que se están comunicando pueden confiar en TCP para entregar todos los datos enviados sin errores y en el orden correcto. Cuando un lado de la aplicación pasa un flujo de bytes a un socket, puede contar con TCP para entregar el mismo flujo de bytes al socket receptor sin pérdida ni duplicación de bytes.

TCP también incluye un mecanismo de control de congestión, que es un servicio para mejorar el funcionamiento general de Internet, más que para el beneficio directo de los procesos que se comunican. Este mecanismo de control de congestión de TCP regula el proceso emisor (cliente o servidor) cuando la red está congestionada entre el emisor y el receptor. Como se explica en el Capítulo 3, el control de congestión de TCP también intenta limitar cada conexión TCP para que utilice una cuota equitativa de ancho de banda de la red.

## SEGURIDAD

#### TCP SEGURO

Ni TCP ni UDP proporcionan ningún mecanismo de cifrado (los datos que el proceso emisor pasa al socket son los mismos datos que viajan a través de la red hasta el proceso de destino). Luego, por ejemplo, si el proceso emisor envía una contraseña en texto legible (es decir, no cifrada) a su socket, esa contraseña viajará a través de todos los enlaces entre el emisor y el receptor, pudiendo ser "husmeada" y descubierta en cualquiera de los enlaces intervinientes. Puesto que la confidencialidad y otras cuestiones de seguridad son críticas para muchas aplicaciones, la comunidad de Internet ha desarrollado una mejora para TCP, denominada **SSL (*Secure Sockets Layer,* Capa de conectores seguros)**. TCP mejorado con SSL no sólo hace todo lo que hace el protocolo TCP tradicional, sino que también proporciona servicios críticos de seguridad proceso a proceso, entre los que se incluyen mecanismos de cifrado, de integridad de los datos y de autenticación en el punto terminal. Debemos destacar que SSL no es un tercer protocolo de transporte de Internet, al mismo nivel que TCP y UDP, sino que es una mejora de TCP, que se implementa en la capa de aplicación. En concreto, si una aplicación desea utilizar los servicios de SSL, tiene que incluir código SSL (existen clases y librerías enormemente optimizadas) tanto en el lado del cliente como en el del servidor de la aplicación. SSL tiene su propia API de sockets, que es similar a la API de sockets del protocolo TCP tradicional. Cuando una aplicación utiliza SSL, el proceso emisor pasa los datos en texto legible al socket SSL; a continuación, SSL cifra los datos en el host emisor y los pasa al socket TCP. Los datos cifrados viajan a través de Internet hasta el socket TCP del proceso receptor. El socket de recepción pasa los datos cifrados a SSL, que los descifra. Por último, SSL pasa los datos en texto legible a través de su socket al proceso receptor. En el Capítulo 8 se cubre con un cierto detalle SSL.

### Servicios UDP

UDP es un protocolo de transporte ligero y simple que proporciona unos servicios mínimos. No está orientado a la conexión, por lo que no tiene lugar un procedimiento de negociación antes de que los dos procesos comiencen a comunicarse. UDP proporciona un servicio de transferencia de datos no fiable; es decir, cuando un proceso envía un mensaje a un socket UDP, el protocolo UDP no ofrece *ninguna* garantía de que el mensaje vaya a llegar al proceso receptor. Además, los mensajes que sí llegan al proceso receptor pueden hacerlo de manera desordenada.

UDP no incluye tampoco un mecanismo de control de congestión, por lo que el lado emisor de UDP puede introducir datos en la capa inferior (la capa de red) a la velocidad que le parezca. (Sin embargo, tenga en cuenta que la tasa de transferencia extremo a extremo real puede ser menor que esta velocidad, a causa de la capacidad de transmisión limitada de los enlaces intervinientes o a causa de la congestión.)

### Servicios no proporcionados por los protocolos de transporte de Internet

Hemos organizado los posibles servicios del protocolo de transporte según cuatro parámetros: transferencia de datos fiable, tasa de transferencia, temporización y seguridad. ¿Cuáles de estos servicios proporcionan TCP y UDP? Ya hemos mencionado que TCP proporciona transferencia de datos fiable extremo a extremo. Y también sabemos que TCP se puede mejorar fácilmente en la capa de aplicación, con SSL, para proporcionar servicios de seguridad. Pero en esta breve descripción de TCP y UDP hemos omitido notoriamente hacer mención de las garantías relativas a la tasa de transferencia o la temporización, servicios que *no* proporcionan los protocolos de transporte de Internet de hoy día. ¿Significa esto que las aplicaciones sensibles al tiempo, como la telefonía por Internet, no se pueden ejecutar actualmente en Internet? Evidentemente, la respuesta es no: Internet lleva muchos años albergando aplicaciones sensibles al tiempo. Estas aplicaciones suelen funcionar bastante bien, porque han sido diseñadas para hacer frente a esta falta de garantías de la mejor forma posible. En el Capítulo 9 veremos algunos de estos trucos de diseño. No obstante, un diseño inteligente tiene sus limitaciones cuando el retardo es excesivo, o cuando la tasa de transferencia extremo a extremo es limitada. En resumen, actualmente Internet puede ofrecer servicios satisfactorios a las aplicaciones sensibles al tiempo, pero no puede proporcionar ninguna garantía de tasa de transferencia ni de temporización.

La Figura 2.5 enumera los protocolos de transporte utilizados por algunas aplicaciones populares de Internet. Podemos ver que aplicaciones como el correo electrónico, el acceso remoto a terminales, la Web y la transferencia de archivos utilizan TCP. Estas aplicaciones han elegido TCP principalmente porque este protocolo ofrece un servicio de transferencia de datos fiable, garantizando que todos los datos llegarán finalmente a su destino. Como las aplicaciones de telefonía por Internet (como Skype) suelen tolerar cierto grado de pérdidas, pero requieren una tasa de transferencia mínima para ser efectivas, los desarrolladores de aplicaciones de telefonía por Internet suelen preferir ejecutarlas sobre UDP, evitando así el mecanismo de control de congestión de TCP y la mayor sobrecarga (bits que no forman parte de la carga útil) que los paquetes de datos tienen en TCP. Pero como muchos cortafuegos están configurados para bloquear el tráfico UDP (o la mayor parte del mismo), las aplicaciones de telefonía por Internet suelen diseñarse para usar TCP como solución alternativa, cuando falla la comunicación a través de UDP.

## 2.1.5 Protocolos de la capa de aplicación

Acabamos de aprender que los procesos de red se comunican entre sí enviando mensajes a sus sockets. Pero, ¿cómo están estructurados estos mensajes? ¿Cuál es el significado de cada uno de los campos de estos mensajes? ¿Cuándo envían los procesos estos mensajes? Estas preguntas nos llevan al ámbito de los protocolos de la capa de aplicación. Un **protocolo de la capa de aplicación** define cómo los procesos de una aplicación, que se ejecutan en distintos sistemas terminales, se pasan los mensajes entre sí. En particular, un protocolo de la capa de aplicación define:

| Aplicación | Protocolo de la capa de aplicación | Protocolo de transporte subyacente |
|---|---|---|
| Correo electrónico | SMTP [RFC 5321] | TCP |
| Acceso remoto a terminal | Telnet [RFC 854] | TCP |
| Web | HTTP [RFC 2616] | TCP |
| Transferencia de archivos | FTP [RFC 959] | TCP |
| Flujos multimedia | HTTP (p. ej. YouTube) | TCP |
| Telefonía por Internet | SIP [rfc 3261], RTP [RFC 3550] o propietario (p. ej. Skype) | UDP o TCP |

**Figura 2.5** ♦ Aplicaciones populares de Internet, sus protocolos de la capa de aplicación y sus protocolos de transporte subyacentes.

- Los tipos de mensajes intercambiados; por ejemplo, mensajes de solicitud y mensajes de respuesta.
- La sintaxis de los diversos tipos de mensajes, es decir, los campos de los que consta el mensaje y cómo se delimitan esos campos.
- La semántica de los campos, es decir, el significado de la información contenida en los campos.
- Las reglas para determinar cuándo y cómo un proceso envía mensajes y responde a los mismos.

Algunos protocolos de la capa de aplicación están especificados en documentos RFC y, por tanto, son de dominio público. Por ejemplo, el protocolo de la capa de aplicación para la Web, HTTP (*HyperText Transfer Protocol* [RFC 2616]), está disponible como un RFC. Si quien desarrolla un navegador web sigue las reglas dadas en el RFC que se ocupa de HTTP, el navegador podrá recuperar páginas web de cualquier servidor web que también se ajuste a las reglas de dicho RFC. Existen muchos otros protocolos de la capa de aplicación que son propietarios y que intencionadamente no están disponibles para todo el mundo. Por ejemplo, Skype utiliza protocolos de la capa de aplicación propietarios.

Es importante diferenciar entre aplicaciones de red y protocolos de la capa de aplicación. Un protocolo de la capa de aplicación es únicamente un elemento de una aplicación de red (¡aunque uno muy importante, desde nuestro punto de vista!). Veamos un par de ejemplos. La Web es una aplicación cliente-servidor que permite a los usuarios obtener documentos almacenados en servidores web bajo demanda. La aplicación Web consta de muchos componentes, entre los que se incluyen un estándar para los formatos de documentos (es decir, HTML), navegadores web (como Firefox y Microsoft Internet Explorer), servidores web (por ejemplo, servidores Apache y Microsoft) y un protocolo de la capa de aplicación. El protocolo de la capa de aplicación de la Web, HTTP, define el formato y la secuencia de los mensajes que se pasan entre el navegador web y el servidor web. Por tanto, HTTP es sólo una pieza (aunque una pieza importante) de la aplicación Web. Otro ejemplo sería una aplicación de correo electrónico Internet, la cual también está constituida por muchos componentes, entre los que se incluyen los servidores de correo que albergan los buzones de los usuarios; los clientes de correo (como Microsoft Outlook) que permiten a los usuarios leer y crear mensajes; un estándar para definir la estructura de los mensajes de correo electrónico y protocolos de la capa de aplicación que definen cómo se pasan los mensajes entre los servidores, cómo se pasan los mensajes entre los servidores y los clientes de correo y cómo se interpretan los contenidos de las cabeceras de los mensajes. El principal protocolo de la capa de aplicación para el correo electrónico es SMTP (*Simple Mail Transfer Protocol*, Protocolo simple de transferencia de correo) [RFC 5321].

Por tanto, el protocolo principal de la capa de aplicación para correo electrónico, SMTP, sólo es un componente (aunque un componente importante) de la aplicación de correo electrónico.

### 2.1.6 Aplicaciones de red analizadas en este libro

Todos los días se desarrollan nuevas aplicaciones de Internet, tanto de dominio público como propietarias. En lugar de abordar un gran número de aplicaciones de Internet a modo de enciclopedia, hemos decidido centrarnos en unas pocas aplicaciones dominantes e importantes. En este capítulo abordaremos cinco aplicaciones relevantes: la Web, el correo electrónico, el servicio de directorio, los flujos de vídeo y las aplicaciones P2P. En primer lugar veremos la Web, no sólo porque es una aplicación enormemente popular, sino porque también su protocolo de la capa de aplicación, HTTP, es sencillo y fácil de comprender. Después veremos el correo electrónico, que fue la primera aplicación de éxito en Internet. El correo electrónico es más complejo que la Web, en el sentido de que no utiliza uno sino varios protocolos de la capa de aplicación. Después del correo electrónico, abordaremos el sistema DNS, que proporciona un servicio de directorio a Internet. La mayoría de los usuarios no interactúan directamente con DNS; en su lugar, invocan indirectamente a DNS a través de otras aplicaciones (entre las que se incluyen las aplicaciones web, de transferencia de archivos y de correo electrónico). DNS ilustra de forma muy conveniente cómo puede implementarse en la capa de aplicación de Internet un elemento de la funcionalidad de red básica (la traducción entre nombres de red y direcciones de red). Después examinaremos las aplicaciones P2P de compartición de archivos y completaremos nuestro estudio de las aplicaciones analizando los flujos de vídeo a la carta, incluyendo la distribución de vídeo almacenado a través de redes de distribución de contenido. En el Capítulo 9 hablaremos más en detalle de las aplicaciones multimedia, incluyendo la de voz sobre IP y la vídeoconferencia.

## 2.2 La Web y HTTP

Hasta principios de la década de 1990, Internet era utilizada principalmente por investigadores, profesores y estudiantes universitarios para acceder a hosts remotos; para transferir archivos desde los hosts locales a los hosts remotos, y viceversa, y para recibir y enviar noticias y mensajes de correo electrónico. Aunque estas aplicaciones eran (y continúan siendo) extremadamente útiles, Internet era prácticamente desconocida fuera de las comunidades académica y de investigación. Fue entonces, a principios de la década de 1990, cuando una nueva aplicación importante apareció en escena: la World Wide Web [Berners-Lee 1994]. La Web fue la primera aplicación de Internet que atrajo la atención del público general. Cambió de manera dramática, y continúa cambiando, la forma en que las personas interactúan dentro y fuera de sus entornos de trabajo. Hizo que Internet pasará de ser una de entre muchas redes de datos, a ser prácticamente la única red de datos.

Quizá lo que atrae a la mayoría de los usuarios es que la Web opera *bajo demanda*. Los usuarios reciben lo que desean y cuando lo desean. Es muy diferente a la radio y la televisión, que fuerzan a los usuarios a sintonizar los programas cuando el proveedor de contenido tiene el contenido disponible. Además de estar disponible bajo demanda, la Web posee muchas otras características maravillosas que a todo el mundo le gustan y que todos valoran. Para cualquier persona, es tremendamente fácil publicar información en la Web (todo el mundo puede convertirse en editor con unos costes extremadamente bajos). Los hipervínculos y los motores de búsqueda nos ayudan a navegar a través de un océano de sitios web. Las fotografías y los vídeos estimulan nuestros sentidos. Los formularios, JavaScript, los applets de Java y muchos otros mecanismos nos permiten interactuar con las páginas y sitios. Y la Web y sus protocolos sirven como plataforma para YouTube, para el correo electrónico basado en la Web (como Gmail) y para la mayoría de las aplicaciones móviles de Internet, incluyendo Instagram y Google Maps.

### 2.2.1 Introducción a HTTP

El corazón de la Web lo forma el **Protocolo de transferencia de hipertexto** (**HTTP**, ***HyperText Transfer Protocol***), que es el protocolo de la capa de aplicación de la Web. Está definido en los documentos [RFC 1945] y [RFC 2616]. HTTP se implementa mediante dos programas: un programa cliente y un programa servidor. Ambos programas, que se ejecutan en sistemas terminales diferentes, se comunican entre sí intercambiando mensajes HTTP. HTTP define la estructura de estos mensajes y cómo el cliente y el servidor intercambian los mensajes. Antes de explicar en detalle HTTP, vamos a hacer un breve repaso de la terminología Web.

Una **página web** (también denominada documento web) consta de objetos. Un objeto es simplemente un archivo (como por ejemplo un archivo HTML, una imagen JPEG, un applet Java o un clip de vídeo) que puede direccionarse mediante un único URL. La mayoría de las páginas web están constituidas por un **archivo base HTML** y varios objetos referenciados. Por ejemplo, si una página web contiene texto HTML y cinco imágenes JPEG, entonces la página web contiene seis objetos: el archivo base HTML y las cinco imágenes. El archivo base HTML hace referencia a los otros objetos contenidos en la página mediante los URL de los objetos. Cada URL tiene dos componentes: el nombre de host del servidor que alberga al objeto y el nombre de la ruta al objeto. Por ejemplo, en el URL

```
http://www.unaEscuela.edu/unDepartamento/imagen.gif
```

`www.unaEscuela.edu` corresponde a un nombre de host y `/unDepartmento/imagen.gif` es el nombre de una ruta. Puesto que los **navegadores web** (como Internet Explorer y Firefox) implementan el lado del cliente de HTTP, en el contexto de la Web utilizaremos los términos *navegador* y *cliente* de forma indistinta. Los **servidores web**, que implementan el lado del servidor de HTTP, albergan los objetos web, siendo cada uno de ellos direccionable mediante un URL. Entre los servidores web más populares se incluyen Apache y Microsoft Internet Information Server.

HTTP define cómo los clientes web solicitan páginas web a los servidores y cómo estos servidores web transfieren esas páginas a los clientes. Más adelante veremos la interacción entre el cliente y el servidor en detalle, si bien la idea general se ilustra en la Figura 2.6. Cuando un usuario solicita una página web (por ejemplo, haciendo clic en un hipervínculo), el navegador envía al servidor mensajes de solicitud HTTP, pidiendo los objetos contenidos en la página. El servidor recibe las solicitudes y responde con mensajes de respuesta HTTP que contienen los objetos.

HTTP utiliza TCP como su protocolo de transporte subyacente (en lugar de ejecutarse por encima de UDP). El cliente HTTP primero inicia una conexión TCP con el servidor. Una vez que la conexión se ha establecido, los procesos de navegador y de servidor acceden a TCP a través de sus


**Figura 2.6** ♦ Comportamiento solicitud-respuesta de HTTP.

interfaces de socket. Como se ha descrito en la Sección 2.1, en el lado del cliente la interfaz de socket es la puerta entre el proceso cliente y la conexión TCP; en el lado del servidor, es la puerta entre el proceso servidor y la conexión TCP. El cliente envía mensajes de solicitud HTTP a su interfaz de socket y recibe mensajes de respuesta HTTP procedentes de su interfaz de socket. De forma similar, el servidor HTTP recibe mensajes de solicitud a través de su interfaz de socket y envía mensajes de respuesta a través de la misma. Una vez que el cliente envía un mensaje a su interfaz de socket, el mensaje deja de estar en las manos del cliente y pasa "a las manos" de TCP. Recuerde, de la Sección 2.1, que TCP proporciona un servicio de transferencia de datos fiable a HTTP. Esto implica que cada mensaje de solicitud HTTP enviado por un proceso cliente llegará intacto al servidor; del mismo modo, cada mensaje de respuesta HTTP enviado por el proceso servidor llegará intacto al cliente. Esta es una de las grandes ventajas de una arquitectura en capas: HTTP no tiene que preocuparse por las pérdidas de datos o por los detalles sobre cómo TCP recupera los datos perdidos o los reordena dentro de la red. Ése es el trabajo de TCP y de los protocolos de las capas inferiores de la pila de protocolos.

Es importante observar que el servidor envía los archivos solicitados a los clientes sin almacenar ninguna información acerca del estado del cliente. Si un determinado cliente pide el mismo objeto dos veces en un espacio de tiempo de unos pocos segundos, el servidor no responde diciendo que acaba de servir dicho objeto al cliente; en su lugar, el servidor reenvía el objeto, ya que ha olvidado por completo que ya lo había hecho anteriormente. Dado que un servidor HTTP no mantiene ninguna información acerca de los clientes, se dice que HTTP es un **protocolo sin memoria del estado**. Debemos destacar también que la Web utiliza la arquitectura de aplicación cliente-servidor, descrita en la Sección 2.1. Un servidor web siempre está activo, con una dirección IP fija, y da servicio a solicitudes procedentes de, potencialmente, millones de navegadores distintos.

## 2.2.2 Conexiones persistentes y no persistentes

En muchas aplicaciones de Internet, el cliente y el servidor están en comunicación durante un periodo de tiempo amplio, haciendo el cliente una serie de solicitudes y respondiendo el servidor a dichas solicitudes. Dependiendo de la aplicación y de cómo se esté empleando, las solicitudes pueden hacerse una tras otra, periódicamente a intervalos regulares o de forma intermitente. Cuando esta interacción cliente-servidor tiene lugar sobre TCP, el desarrollador de la aplicación tiene que tomar una decisión importante: ¿debería cada par solicitud/respuesta enviarse a través de una conexión TCP *separada* o deberían enviarse todas las solicitudes y sus correspondientes respuestas a través de la *misma* conexión TCP? Si se utiliza el primer método, se dice que la aplicación emplea **conexiones no persistentes**; si se emplea la segunda opción, entonces se habla de **conexiones persistentes**. Con el fin de profundizar en esta cuestión de diseño, vamos a examinar las ventajas y desventajas de las conexiones persistentes en el contexto de una aplicación específica, en concreto HTTP, que puede utilizar ambos tipos de conexión. Aunque HTTP emplea conexiones persistentes de manera predeterminada, los clientes y servidores HTTP se pueden configurar para emplear en su lugar conexiones no persistentes.

### HTTP con conexiones no persistentes

Sigamos los pasos que permiten transferir una página web desde un servidor a un cliente en el caso de conexiones no persistentes. Supongamos que la página consta de un archivo base HTML y de 10 imágenes JPEG, residiendo los 11 objetos en el mismo servidor. Supongamos también que el URL del archivo base HTML es:

```
http://www.unaEscuela.edu/unDepartmento/home.index
```

Lo que ocurre es lo siguiente:

1. El proceso cliente HTTP inicia una conexión TCP con el servidor `www.unaEscuela.edu` en el puerto número 80, que es el número de puerto predeterminado para HTTP. Asociados con la conexión TCP, habrá un socket en el cliente y un socket en el servidor.
2. El cliente HTTP envía un mensaje de solicitud HTTP al servidor a través de su socket. El mensaje de solicitud incluye el nombre de la ruta `/unDepartmento/home.index`. (Más adelante veremos con más detalle los mensajes HTTP.)
3. El proceso servidor HTTP recibe el mensaje de solicitud a través de su socket, recupera el objeto `/unDepartmento/home.index` de su medio de almacenamiento (RAM o disco), encapsula el objeto en un mensaje de respuesta HTTP y lo envía al cliente a través de su socket.
4. El proceso servidor HTTP indica a TCP que cierre la conexión TCP. (Pero TCP realmente no termina la conexión hasta que está seguro de que el cliente ha recibido el mensaje de respuesta en perfecto estado.)
5. El cliente HTTP recibe el mensaje de respuesta. La conexión TCP termina. El mensaje indica que el objeto encapsulado es un archivo HTML. El cliente extrae el archivo del mensaje de respuesta, examina el archivo HTML y encuentra las referencias a los 10 objetos JPEG.
6. Los cuatro primeros pasos se repiten entonces para cada uno de los objetos JPEG referenciados.

Cuando el navegador recibe la página web, la muestra al usuario. Dos navegadores distintos pueden interpretar (es decir, mostrar al usuario) una página web de formas ligeramente distintas. HTTP no tiene nada que ver con cómo un cliente interpreta una página web. Las especificaciones HTTP ([RFC 1945] y [RFC 2616]) únicamente definen el protocolo de comunicación entre el programa cliente HTTP y el programa servidor HTTP.

Los pasos anteriores ilustran el uso de las conexiones no persistentes, donde cada conexión TCP se cierra después de que el servidor envíe el objeto: la conexión no se mantiene (no persiste) para los restantes objetos. Observe que cada conexión TCP transporta exactamente un mensaje de solicitud y un mensaje de respuesta. Por tanto, en este ejemplo, cuando un usuario solicita la página web, se generan 11 conexiones TCP.

En los pasos descritos anteriormente, hemos sido intencionadamente vagos en lo que respecta a si el cliente obtiene las diez imágenes JPEG a través de diez conexiones TCP en serie o si algunas de dichas imágenes se obtienen a través de conexiones TCP en paralelo. De hecho, los usuarios pueden configurar los navegadores modernos para controlar el grado de paralelismo. En sus modos predeterminados, la mayoría de los navegadores abren entre 5 y 10 conexiones TCP en paralelo y cada una de estas conexiones gestiona una transacción solicitud-respuesta. Si el usuario lo prefiere, el número máximo de conexiones en paralelo puede establecerse en uno, en cuyo caso se establecerán diez conexiones en serie. Como veremos en el siguiente capítulo, el uso de conexiones en paralelo reduce el tiempo de respuesta.

Antes de continuar, vamos a realizar un cálculo aproximado para estimar la cantidad de tiempo que transcurre desde que un cliente solicita el archivo base HTML hasta que recibe dicho archivo completo. Para ello, definimos el **tiempo de ida y vuelta** (**RTT, *Round-Trip Time***), que es el tiempo que tarda un paquete pequeño en viajar desde el cliente al servidor y volver de nuevo al cliente. El RTT incluye los retardos de propagación de los paquetes, los retardos de cola en los routers y switches intermedios y los retardos de procesamiento de los paquetes (estos retardos se explican en la Sección 1.4). Consideremos ahora lo que ocurre cuando un usuario hace clic en un hipervínculo. Como se muestra en la Figura 2.7, esto hace que el navegador inicie una conexión TCP entre el navegador y el servidor web, lo que implica un proceso de "acuerdo en tres fases" (el cliente envía un pequeño segmento TCP al servidor, el servidor reconoce la recepción y responde con otro pequeño segmento TCP y, por último, el cliente devuelve un mensaje de reconocimiento al servidor). Las dos primeras partes de este proceso de acuerdo en tres fases tardan un periodo de tiempo igual a RTT. Después de completarse las dos primeras fases de la negociación, el cliente envía a la conexión TCP el mensaje de solicitud HTTP combinado con la tercera parte de la negociación (el mensaje de reconocimiento). Una vez que el mensaje de solicitud llega al servidor, este envía el

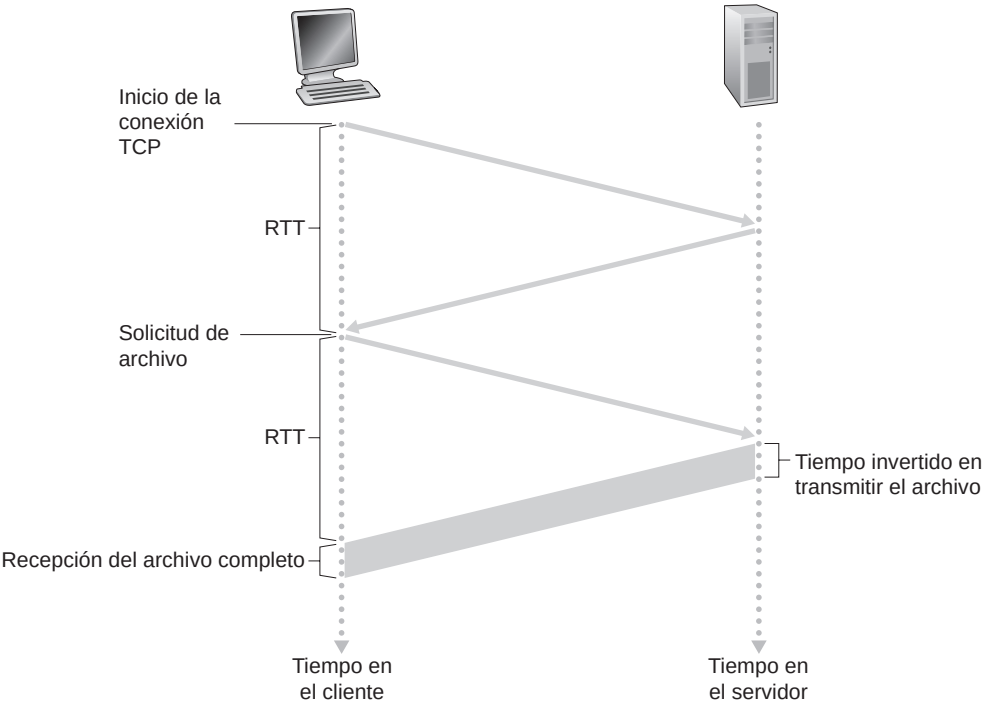



**Figura 2.7** ♦ Cálculo aproximado del tiempo necesario para solicitar y recibir un archivo HTML.

archivo HTML a través de la conexión TCP. Este mensaje de solicitud/respuesta HTTP consume otro periodo de tiempo RTT. Luego el tiempo de respuesta total es aproximadamente igual a dos RTT más el tiempo de transmisión del archivo HTML por parte del servidor.

### HTTP con conexiones persistentes

Las conexiones no persistentes presentan algunos inconvenientes. En primer lugar, tiene que establecerse y gestionarse una conexión completamente nueva *para cada objeto solicitado*. Para cada una de estas conexiones, deben asignarse buffers TCP y tienen que gestionarse variables TCP tanto en el cliente como en el servidor. Esto puede sobrecargar de forma significativa al servidor web, ya que puede estar sirviendo solicitudes de cientos de clientes distintos simultáneamente. En segundo lugar, como ya hemos explicado, cada objeto sufre un retardo de entrega de dos RTT: un RTT para establecer la conexión TCP y otro RTT para solicitar y recibir un objeto.

Con las conexiones persistentes de HTTP 1.1, el servidor deja la conexión TCP abierta después de enviar una respuesta. Las subsiguientes solicitudes y respuestas que tienen lugar entre el mismo cliente y el servidor pueden enviarse a través de la misma conexión. En concreto, una página web completa (en el ejemplo anterior, el archivo base HTML y las 10 imágenes) se puede enviar a través de una misma conexión TCP persistente. Además, varias páginas web que residan en el mismo servidor pueden enviarse desde el servidor a un mismo cliente a través de una única conexión TCP persistente. Estas solicitudes de objetos pueden realizarse una tras otra sin esperar a obtener las respuestas a las solicitudes pendientes (*pipelining*, procesamiento en cadena). Normalmente, el servidor HTTP cierra una conexión cuando no se ha utilizado durante cierto tiempo (un intervalo de fin de temporización configurable). Cuando el servidor recibe las solicitudes una tras otra, envía los objetos uno tras otro. El modo predeterminado de HTTP utiliza conexiones persistentes con procesamiento en cadena. Más recientemente, HTTP/2 [RFC 7540] amplía la funcionalidad

de HTTP 1.1, permitiendo entrelazar múltiples solicitudes y respuestas en la *misma* conexión, y proporcionando también un mecanismo para priorizar los mensajes de solicitud y respuesta HTTP dentro de dicha conexión. En los problemas de repaso de los Capítulos 2 y 3 compararemos cuantitativamente el rendimiento de las conexiones persistentes y no persistentes. Le animamos también a que consulte [Heidemann 1997; Nielsen 1997; RFC 7540].

## 2.2.3 Formato de los mensajes HTTP

Las especificaciones HTTP [RFC 1945; RFC 2616]; RFC 7540) incluyen las definiciones de los formatos de los mensajes HTTP. A continuación vamos a estudiar los dos tipos de mensajes HTTP existentes: mensajes de solicitud y mensajes de respuesta.

### Mensaje de solicitud HTTP

A continuación se muestra un mensaje de solicitud HTTP típico:

```
GET /unadireccion/pagina.html HTTP/1.1
Host: www.unaEscuela.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

Podemos aprender muchas cosas si miramos en detalle este sencillo mensaje de solicitud. En primer lugar, podemos comprobar que el mensaje está escrito en texto ASCII normal, por lo que cualquier persona con conocimientos informáticos puede leerlo. En segundo lugar, vemos que el mensaje consta de cinco líneas, cada una de ellas seguida por un retorno de carro y un salto de línea. La última línea va seguida de un retorno de carro y un salto de línea adicionales. Aunque este mensaje en concreto está formado por cinco líneas, un mensaje de solicitud puede constar de muchas más líneas o tener tan solo una. La primera línea de un mensaje de solicitud HTTP se denomina **línea de solicitud** y las siguientes son las **líneas de cabecera**. La línea de solicitud consta de tres campos: el campo de método, el campo URL y el campo de la versión HTTP. El campo que especifica el método puede tomar diferentes valores, entre los que se incluyen GET, POST, HEAD, PUT y DELETE. La inmensa mayoría de los mensajes de solicitud HTTP utilizan el método GET. Este método se emplea cuando el navegador solicita un objeto, identificándose dicho objeto en el campo URL. En este ejemplo, el navegador está solicitando el objeto `/unadireccion/pagina.html`. El campo correspondiente a la versión se explica por sí mismo; en este ejemplo, el navegador utiliza la versión HTTP/1.1.

Analicemos ahora las líneas de cabecera de este ejemplo. La línea de cabecera `Host: www.unaescuela.edu` especifica el host en el que reside el objeto. Podría pensarse que esta línea de cabecera es innecesaria, puesto que ya existe una conexión TCP activa con el host. Pero, como veremos en la Sección 2.2.5, las cachés proxy web necesitan la información proporcionada por la línea de cabecera del host. Al incluir la línea de cabecera `Connection: close`, el navegador está diciendo al servidor que no desea molestarse en trabajar con conexiones persistentes, sino que desea que el servidor cierre la conexión después de enviar el objeto solicitado. La línea de cabecera `User-agent:` especifica el agente de usuario, es decir, el tipo de navegador que está haciendo la solicitud al servidor. En este caso, el agente de usuario es Mozilla/5.0, un navegador Firefox. Esta línea de cabecera resulta útil porque el servidor puede enviar versiones diferentes del mismo objeto a los distintos tipos de agentes de usuario (todas las versiones tienen la misma dirección URL). Por último, la línea de cabecera `Accept-language:` indica que el usuario prefiere recibir una versión en francés del objeto, si tal objeto existe en el servidor; en caso contrario, el servidor debe enviar la versión predeterminada. La línea de cabecera `Accept-language:` sólo es una de las muchas cabeceras de negociación del contenido disponibles en HTTP.

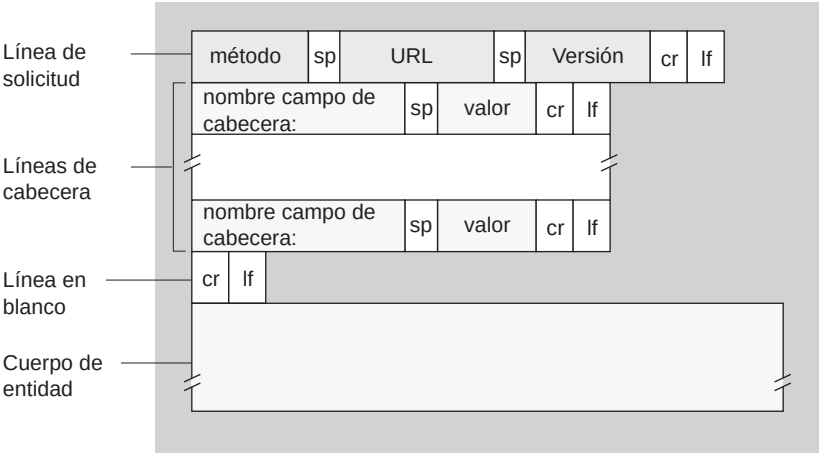



**Figura 2.8** ♦ Formato general de un mensaje de solicitud HTTP.

Una vez visto un ejemplo, vamos a estudiar el formato general de un mensaje de solicitud, ilustrado en la Figura 2.8. Podemos comprobar que el formato general es muy similar al usado en el ejemplo anterior. Sin embargo, fíjese en que después de las líneas de cabecera (y el retorno de carro y el salto de línea adicionales) se incluye un "cuerpo de entidad". Este campo queda vacío cuando se utiliza el método `GET`, pero no cuando se usa el método `POST`. A menudo, un cliente HTTP utiliza el método `POST` cuando el usuario completa un formulario; por ejemplo, cuando especifica términos para realizar una búsqueda utilizando un motor de búsqueda. Con un mensaje `POST`, el usuario solicita también una página web al servidor, pero el contenido concreto de la misma dependerá de lo que el usuario haya escrito en los campos del formulario. Si el valor del campo de método es `POST`, entonces el cuerpo de la entidad contendrá lo que el usuario haya introducido en los campos del formulario.

No podemos dejar de mencionar que una solicitud generada con un formulario no necesariamente utiliza el método `POST`. En su lugar, a menudo los formularios HTML emplean el método `GET` e incluyen los datos de entrada (especificados en los campos del formulario) en el URL solicitado. Por ejemplo, si un formulario emplea el método `GET` y tiene dos campos, y las entradas a esos dos campos son `monos` y `bananas`, entonces el URL tendrá la estructura `www.unsitio.com/busquedadeanimales?monos&bananas`. Probablemente habrá visto direcciones URL ampliadas de este tipo, al navegar por la Web.

El método `HEAD` es similar al método `GET`. Cuando un servidor recibe una solicitud con el método `HEAD`, responde con un mensaje HTTP, pero excluye el objeto solicitado. Los desarrolladores de aplicaciones a menudo utilizan el método `HEAD` para labores de depuración. El método `PUT` suele utilizarse junto con herramientas de publicación web. Esto permite a un usuario cargar un objeto en una ruta específica (directorio) en un servidor web determinado. Las aplicaciones que necesitan cargar objetos en servidores web también emplean el método `PUT`. El método `DELETE` permite a un usuario o a una aplicación borrar un objeto de un servidor web.

### Mensajes de respuesta HTTP

A continuación se muestra un mensaje de respuesta HTTP típico. Este mensaje podría ser la respuesta al ejemplo de mensaje de solicitud que acabamos de ver.

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
```

```
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(datos datos datos datos datos ...)
```

Examinemos detenidamente este mensaje de respuesta. Tiene tres secciones: una **línea de estado** inicial, seis **líneas de cabecera** y después el **cuerpo de entidad**. El cuerpo de entidad es la parte más importante del mensaje, ya que contiene el objeto solicitado en sí (representado por `datos datos datos datos datos ...`). La línea de estado contiene tres campos: el que especifica la versión del protocolo, un código de estado y el correspondiente mensaje explicativo del estado. En este ejemplo, la línea de estado indica que el servidor está utilizando HTTP/1.1 y que todo es correcto (OK); es decir, que el servidor ha encontrado y está enviando el objeto solicitado.

Veamos ahora las líneas de cabecera. El servidor utiliza la línea de cabecera `Connection: close` para indicar al cliente que va a cerrar la conexión TCP después de enviar el mensaje. La línea de cabecera `Date:` indica la hora y la fecha en la que se creó la respuesta HTTP y fue enviada por el servidor. Observe que esta línea no especifica la hora en que el objeto fue creado o modificado por última vez; es la hora en la que el servidor recupera el objeto de su sistema de archivos, inserta el objeto en el mensaje de respuesta y lo envía. La línea de cabecera `Server:` indica que el mensaje fue generado por un servidor web Apache; es análoga a la línea de cabecera `User-agent:` del mensaje de solicitud HTTP. La línea de cabecera `Last-Modified:` especifica la hora y la fecha en que el objeto fue creado o modificado por última vez. Esta línea `Last-Modified:`, que enseguida estudiaremos en detalle, resulta fundamental para el almacenamiento en caché del objeto, tanto en el cliente local como en los servidores de almacenamiento en caché de la red (también conocidos como servidores proxy). La línea de cabecera `Content-Length:` especifica el número de bytes del objeto que está siendo enviado. La línea `Content-Type:` indica que el objeto incluido en el cuerpo de entidad es texto HTML. (El tipo de objeto está indicado oficialmente por la línea de cabecera `Content-Type:` y no por la extensión del archivo.)

Una vez visto un ejemplo, vamos a pasar a examinar el formato general de un mensaje de respuesta, ilustrado en la Figura 2.9. Este formato general de mensaje de respuesta se corresponde con el del ejemplo anterior. Vamos a comentar algunas cosas más acerca de los códigos de estado y sus descripciones. El código de estado y su frase asociada indican el resultado de la solicitud. Algunos códigos de estado comunes y sus frases asociadas son:

- `200 OK:` La solicitud se ha ejecutado con éxito y se ha devuelto la información en el mensaje de respuesta.
- `301 Moved Permanently:` El objeto solicitado ha sido movido de forma permanente; el nuevo URL se especifica en la línea de cabecera `Location:` del mensaje de respuesta. El software cliente recuperará automáticamente el nuevo URL.
- `400 Bad Request:` Se trata de un código de error genérico que indica que la solicitud no ha sido comprendida por el servidor.
- `404 Not Found:` El documento solicitado no existe en este servidor.
- `505 HTTP Version Not Supported:` La versión de protocolo HTTP solicitada no es soportada por el servidor.

Nota de vídeo

Uso de Wireshark para investigar el protocolo HTTP.

¿Le gustaría ver un mensaje de respuesta HTTP real? ¡Esto es muy recomendable y además es muy fácil de hacer! En primer lugar, establezca una conexión Telnet con su servidor web favorito. A continuación, escriba un mensaje de solicitud de una línea para obtener algún objeto que esté almacenado en ese servidor. Por ejemplo, si tiene acceso a la línea de comandos (*prompt*), escriba:

```
telnet gaia.cs.umass.edu 80
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

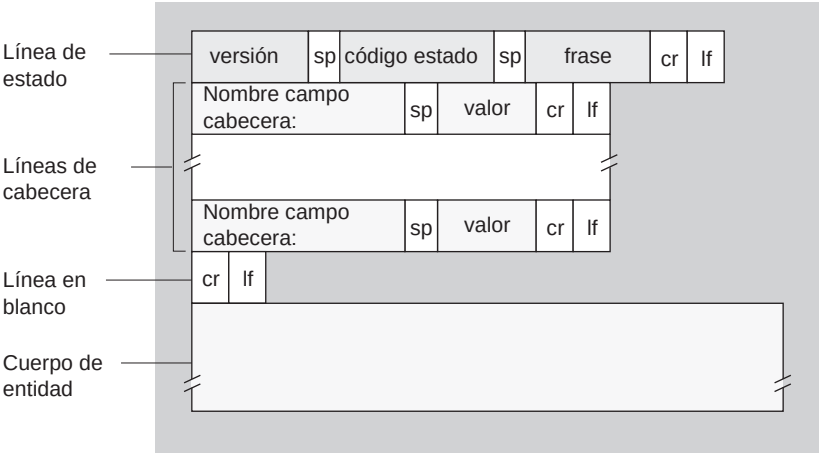



**Figura 2.9** ♦ Formato general de un mensaje de respuesta HTTP.

(Pulse dos veces la tecla retorno de carro después de escribir la última línea.) De este modo se abre una conexión TCP en el puerto 80 del host `gaia.cs.umas.edu` y luego se envía el mensaje de solicitud HTTP. Debería ver un mensaje de respuesta que incluya el archivo base HTML de los problemas interactivos de repaso de este libro. Para ver simplemente las líneas del mensaje HTTP y no recibir el objeto, sustituya `GET` por `HEAD`.

En esta sección hemos visto una serie de líneas de cabecera que pueden utilizarse en los mensajes HTTP de solicitud y respuesta. La especificación de HTTP define un gran número de otras líneas de cabecera que pueden ser insertadas por los navegadores, servidores web y servidores de almacenamiento en caché de la red. Hemos cubierto únicamente una pequeña parte de la totalidad de líneas de cabecera disponibles. A continuación veremos unas pocas más y en la Sección 2.2.5 algunas otras, al hablar del almacenamiento en cachés web de la red. Puede leer una exposición enormemente clara y comprensible acerca del protocolo HTTP, incluyendo sus cabeceras y códigos de estado en [Krishnamurty 2001].

¿Cómo decide un navegador qué líneas de cabecera incluir en un mensaje de solicitud? ¿Cómo decide un servidor web qué líneas de cabecera incluir en un mensaje de respuesta? Un navegador generará líneas de cabecera en función del tipo y la versión del navegador (por ejemplo, un navegador HTTP/1.0 no generará ninguna línea de cabecera correspondiente a la versión 1.1), de la configuración del navegador que tenga el usuario (por ejemplo, el idioma preferido) y de si el navegador tiene actualmente en caché una versión del objeto, posiblemente desactualizada. Los servidores web se comportan de forma similar: existen productos, versiones y configuraciones diferentes, que influyen en las líneas de cabecera que se incluirán en los mensajes de respuesta.

### 2.2.4 Interacción usuario-servidor: cookies

Hemos mencionado anteriormente que un servidor HTTP no tiene memoria del estado de la conexión. Esto simplifica el diseño del servidor y ha permitido a los ingenieros desarrollar servidores web de alto rendimiento que pueden gestionar miles de conexiones TCP simultáneas. Sin embargo, para un sitio web, a menudo es deseable poder identificar a los usuarios, bien porque el servidor desea restringir el acceso a los usuarios o porque desea servir el contenido en función de la identidad del usuario. Para estos propósitos, HTTP utiliza cookies. Las cookies, definidas en [RFC 6265], permiten a los sitios seguir la pista a los usuarios. Actualmente, la mayoría de los sitios web comerciales más importantes utilizan cookies.

Como se muestra en la Figura 2.10, la tecnología de las cookies utiliza cuatro componentes: (1) una línea de cabecera de la cookie en el mensaje de respuesta HTTP; (2) una línea de cabecera de la cookie en el mensaje de solicitud HTTP; (3) el archivo de cookies almacenado en el sistema

terminal del usuario y gestionado por el navegador del usuario; y (4) una base de datos back-end en el sitio web. Basándonos en la Figura 2.10, vamos a ver mediante un ejemplo cómo funcionan las cookies. Suponga que Susana, que accede siempre a la Web utilizando Internet Explorer en el PC de su casa, entra en Amazon.com por primera vez. Supongamos que anteriormente ella había visitado el sitio de eBay. Cuando la solicitud llega al servidor web de Amazon, el servidor genera un número de identificación exclusivo y crea una entrada en su base de datos back-end que está indexada por el número de identificación. El servidor web de Amazon responde entonces al navegador de Susana, incluyendo en la respuesta HTTP una línea de cabecera `Set-cookie:`, que contiene el número de identificación. Por ejemplo, la línea de cabecera podría ser:

```
Set-cookie: 1678
```

Cuando el navegador de Susana recibe el mensaje de respuesta HTTP, ve la cabecera `Set-cookie:`. Entonces el navegador añade una línea a su archivo especial de cookies. Esta línea incluye el nombre de host del servidor y el número de identificación contenido en la cabecera `Set-cookie:`. Observe que el archivo de cookies ya tiene una entrada para eBay, dado que Susana había visitado dicho sitio en el pasado. A medida que Susana continúa navegando por el sitio de Amazon, cada vez que solicita

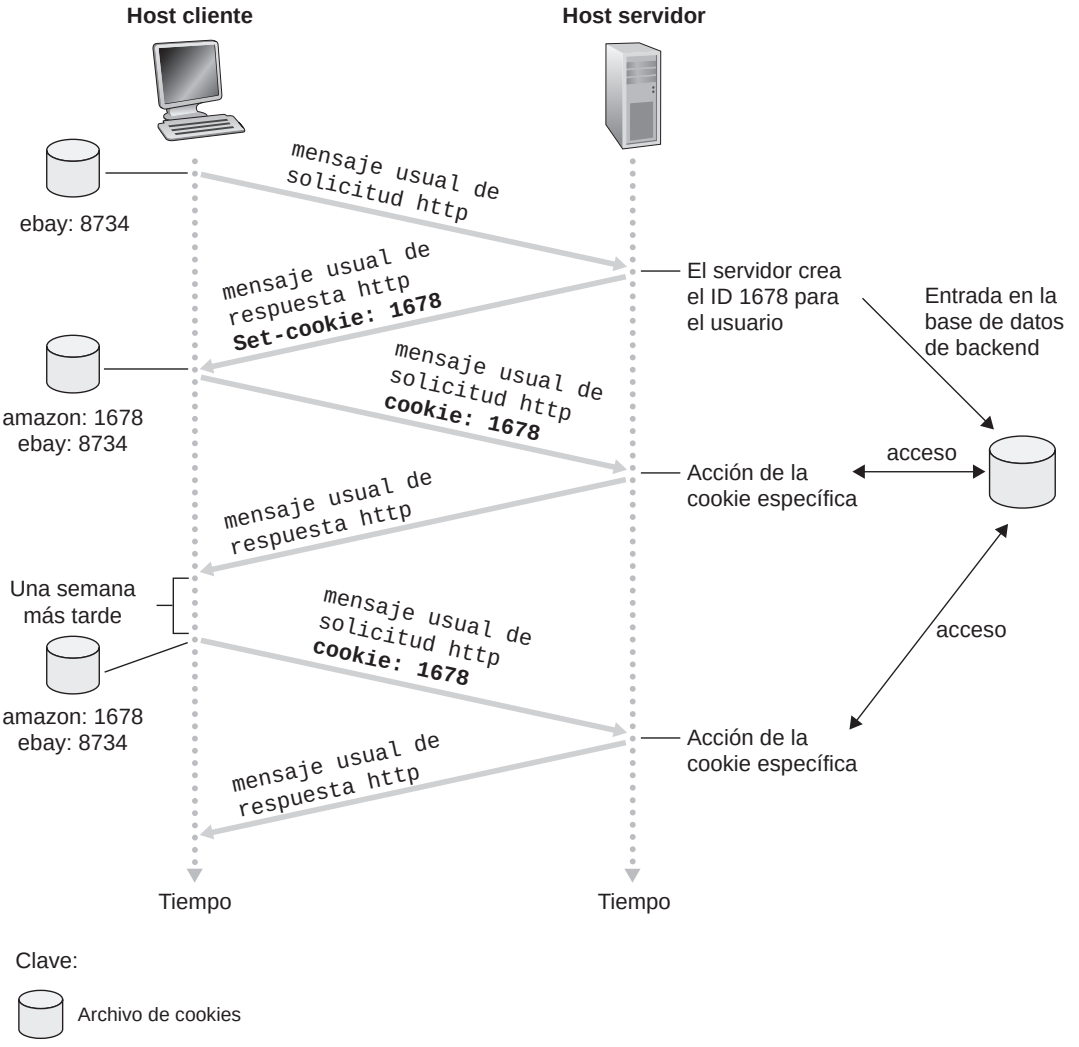



**Figura 2.10** ♦ Mantenimiento del estado del usuario mediante cookies.

una página web su navegador consulta su archivo de cookies, extrae su número de identificación para ese sitio y añade a la solicitud HTTP una línea de cabecera de cookie que incluye el número de identificación. Específicamente, cada una de sus solicitudes HTTP al servidor de Amazon incluye la línea de cabecera:

```
Cookie: 1678
```

De esta forma, el servidor de Amazon puede seguir la actividad de Susana en ese sitio web. Aunque el sitio web de Amazon no necesariamente conoce el nombre de Susana, ¡sabe exactamente qué páginas ha visitado el usuario número 1678, en qué orden y cuántas veces!. Amazon utiliza cookies para proporcionar su servicio de carro de la compra: Amazon puede mantener una lista de todas las compras previstas por Susana, con el fin de que pueda pagarlas todas juntas al final de la sesión.

Si Susana vuelve al sitio de Amazon, por ejemplo una semana más tarde, su navegador continuará incluyendo la línea de cabecera `Cookie: 1678` en los mensajes de solicitud. Amazon también recomienda productos a Susana basándose en las páginas web que ha visitado anteriormente dentro del sitio. Si Susana se registra en Amazon, proporcionando su nombre completo, dirección de correo electrónico, dirección postal y la información de su tarjeta crédito, entonces Amazon podrá incluir esta información en su base de datos, asociando el nombre de Susana con su número de identificación (¡y con todas las páginas que ha visitado dentro del sitio en el pasado!). Así es como Amazon y otros sitios de comercio electrónico proporcionan el servicio de "compra con un clic": cuando Susana desee comprar un producto en una visita posterior, no tendrá que volver a escribir su nombre, su número de la tarjeta de crédito ni su dirección.

Como puede ver, las cookies pueden utilizarse para identificar a un usuario. La primera vez que un usuario visita un sitio, el usuario puede proporcionar una identificación suya (posiblemente su nombre). En las sesiones posteriores, el navegador pasa al servidor una cabecera de cookie, identificando al usuario ante el servidor. Las cookies pueden por tanto utilizarse para crear una capa de sesión por encima del protocolo HTTP sin memoria del estado de la conexión. Por ejemplo, cuando un usuario inicia una sesión en una aplicación de correo electrónico basada en la Web (como por ejemplo Hotmail), el navegador envía al servidor la información de la cookie, permitiendo al servidor identificar al usuario a lo largo de la sesión que el usuario mantenga con la aplicación.

Aunque las cookies a menudo simplifican a los usuarios la realización de compras por Internet, son controvertidas, porque también pueden considerarse como una invasión de la intimidad del usuario. Como acabamos de ver, empleando una combinación de cookies y de la información sobre cuentas suministrada por el usuario, un sitio web puede obtener mucha información de un usuario y, potencialmente, puede vender esa información a otras empresas. Cookie Central [Cookie Central 2016] proporciona mucha información acerca de la controversia de las cookies.

### 2.2.5 Almacenamiento en caché web

Una **caché web**, también denominada **servidor proxy**, es una entidad de red que satisface solicitudes HTTP en nombre de un servidor web de origen. La caché web dispone de su propio almacenamiento en disco y mantiene en él copias de los objetos solicitados recientemente. Como se muestra en la Figura 2.11, el navegador de un usuario se puede configurar de modo que todas sus solicitudes HTTP se dirijan en primer lugar a la caché web. Por ejemplo, suponga que un navegador está solicitando el objeto `http://www.unaEscuela.edu/campus.gif`. Veamos qué es lo que ocurre:

1. El navegador establece una conexión TCP con la caché web y envía una solicitud HTTP para el objeto a la caché web.
2. La caché web comprueba si tiene una copia del objeto almacenada localmente. Si la tiene, la caché web devuelve el objeto dentro de un mensaje de respuesta HTTP al navegador del cliente.
3. Si la caché web no tiene el objeto, abre una conexión TCP con el servidor de origen, es decir, con `www.unaEscuela.edu`. La caché web envía entonces una solicitud HTTP para obtener

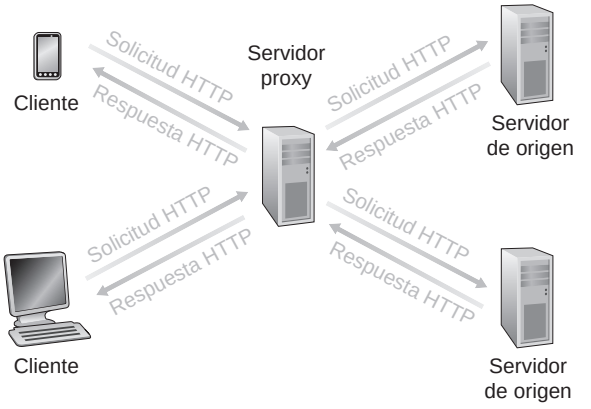



**Figura 2.11** ♦ Clientes que solicitan objetos a través de una caché web.

el objeto a través de la conexión TCP caché-servidor. Después de recibir esta solicitud, el servidor de origen envía el objeto dentro de un mensaje de respuesta HTTP a la caché web.

4. Cuando la caché web recibe el objeto, almacena una copia en su dispositivo de almacenamiento local y envía una copia, dentro de un mensaje de respuesta HTTP, al navegador del cliente (a través de la conexión TCP existente entre el navegador del cliente y la caché web).

Observe que una caché es a la vez un servidor y un cliente. Cuando recibe solicitudes de y envía respuestas a un navegador, se comporta como un servidor. Cuando envía solicitudes a y recibe respuestas de un servidor de origen, entonces actúa como un cliente.

Habitualmente es un ISP quien adquiere e instala una caché web. Por ejemplo, una universidad podría instalar una caché en su red del campus y configurar todos los navegadores del campus apuntando a la caché. O un ISP residencial de gran tamaño (como AOL) podría instalar una o más cachés en su red y preconfigurar los navegadores que suministre para que apunten a las cachés instaladas.

El almacenamiento en caché web se ha implantado en Internet por dos razones. La primera razón es que una caché web puede reducir sustancialmente el tiempo de respuesta a la solicitud de un cliente, especialmente si el ancho de banda cuello de botella entre el cliente y el servidor de origen es mucho menor que el ancho de banda cuello de botella entre el cliente y la caché. Si existe una conexión de alta velocidad entre el cliente y la caché, lo que ocurre a menudo, y si la caché tiene el objeto solicitado, entonces ésta podrá suministrar el objeto rápidamente al cliente. La segunda razón es que, como ilustraremos enseguida con un ejemplo, las cachés web pueden reducir sustancialmente el tráfico en el enlace de acceso a Internet de una institución. Reduciendo el tráfico, la institución (por ejemplo, una empresa o una universidad) no tiene que mejorar el ancho de banda tan rápidamente, lo que implica una reducción de costes. Además, las cachés web pueden reducir mucho el tráfico web global en Internet, mejorando en consecuencia el rendimiento de todas las aplicaciones.

Para adquirir un conocimiento profundo de las ventajas de las cachés, vamos a ver un ejemplo en el contexto ilustrado en la Figura 2.12. Esta figura muestra dos redes: la red institucional y el resto de la red pública Internet. La red institucional es una LAN de alta velocidad. Un router de la red institucional y un router de Internet están conectados mediante un enlace a 15 Mbps. Los servidores de origen están conectados a Internet pero se encuentran distribuidos por todo el mundo. Suponga que el tamaño medio del objeto es de 1 Mbits y que la tasa media de solicitudes de los navegadores de la institución a los servidores de origen es de 15 solicitudes por segundo. Suponga que los mensajes de solicitud HTTP son extremadamente pequeños y que por tanto no crean tráfico en las redes ni en el enlace de acceso (desde el router institucional al router de Internet). Suponga también que el tiempo que se tarda en reenviar una solicitud HTTP (contenida en un datagrama IP) desde el router del lado de Internet del enlace de acceso de la Figura 2.12 hasta que

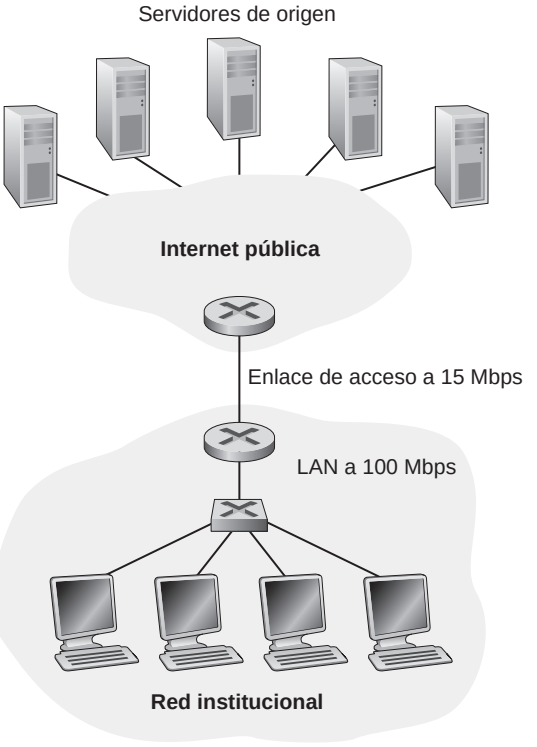



**Figura 2.12** ♦ Cuello de botella entre una red institucional e Internet.

se recibe la respuesta (normalmente contenida en muchos datagramas IP) es igual a dos segundos. Informalmente, vamos a denominar a este último retardo "retardo de Internet".

El tiempo total de respuesta, es decir, el tiempo desde que el navegador solicita un objeto hasta que lo recibe, es la suma del retardo de la LAN, el retardo de acceso (es decir, el retardo entre los dos routers) y el retardo de Internet. Hagamos ahora un burdo cálculo para estimar este retardo. La intensidad de tráfico en la LAN es (véase la Sección 1.4.2):

$$(15 \text{ solicitudes/s}) \cdot (1 \text{ Mbits/solicitud})/(100 \text{ Mbps}) = 0{,}15$$

mientras que la intensidad de tráfico en el enlace de acceso (desde el router de Internet hasta el router de la institución) es:

$$(15 \text{ solicitudes/s}) \cdot (1 \text{ Mbits/solicitud})/(15 \text{ Mbps}) = 1$$

Una intensidad de tráfico de 0,15 en una LAN normalmente da resultados, como máximo, de decenas de milisegundos de retardo; por tanto, podemos despreciar el retardo de la LAN. Sin embargo, como hemos visto en la Sección 1.4.2, cuando la intensidad de tráfico se aproxima a 1 (como es el caso del enlace de acceso de la Figura 2.12), el retardo en el enlace comienza a aumentar y crece sin límite. Por tanto, el tiempo medio de respuesta para satisfacer las solicitudes es del orden de minutos, si no mayor, lo que es inaceptable para los usuarios de la institución. Evidentemente, es necesario hacer algo.

Una posible solución es incrementar la velocidad de acceso de 15 Mbps a, por ejemplo, 100 Mbps. Esto disminuirá la intensidad de tráfico en el enlace de acceso a 0,15, lo que se traduce en retardos despreciables entre los dos routers. En este caso, el tiempo total de respuesta será de unos dos segundos, es decir, el retardo de Internet. Pero esta solución también implica que la institución debe actualizar su enlace de acceso de 15 Mbps a 100 Mbps, lo que resulta ser una solución cara.

Consideremos ahora la solución alternativa de no actualizar el enlace de acceso sino, en su lugar, instalar una caché web en la red institucional. Esta solución se ilustra en la Figura 2.13. Las tasas de acierto, es decir, la fracción de solicitudes que son satisfechas por una caché, suelen estar comprendidas, en la práctica, entre 0,2 y 0,7. Con propósitos ilustrativos, supongamos que la caché proporciona una tasa de acierto de 0,4 para esta institución. Dado que los clientes y la caché están conectados a la misma LAN de alta velocidad, el 40 por ciento de la solicitudes serán satisfechas casi de forma inmediata, es decir, en 10 milisegundos o menos por la caché. No obstante, el restante 60 por ciento de las solicitudes todavía tendrán que ser satisfechas por los servidores de origen. Pero sólo con el 60 por ciento de los objetos solicitados atravesando el enlace de acceso, la intensidad de tráfico en el mismo se reduce de 1,0 a 0,6. Típicamente, una intensidad de tráfico menor que 0,8 se corresponde con un retardo pequeño, digamos de unas decenas de milisegundos, en un enlace de 15 Mbps. Este retardo es despreciable comparado con los dos segundos del retardo de Internet. Teniendo en cuenta todo esto, el retardo medio será entonces:

$$0{,}4 \cdot (0{,}01 \text{ segundos}) + 0{,}6 \cdot (2{,}01 \text{ segundos})$$

lo que es algo más de 1,2 segundos. Por tanto, esta segunda solución proporciona incluso un tiempo de respuesta menor que la primera y no requiere que la institución actualice su enlace con Internet. Por supuesto, la institución tiene que comprar e instalar una caché web, pero este coste es bajo, ya que muchas cachés utilizan software de dominio público que se ejecuta en computadoras PC baratas.

Gracias al uso de las **redes de distribución de contenido (CDN, *Content Distribution Networks*)**, las cachés web están desempeñando un papel cada vez más importante en Internet. Un empresa CDN instala muchas cachés distribuidas geográficamente a través de Internet, localizando la mayor parte del tráfico. Existen redes CDN compartidas (por ejemplo, Akamai y Limelight) y redes CDN dedicadas (como Google y Netflix). Veremos la redes CDN en detalle en la Sección 2.6.

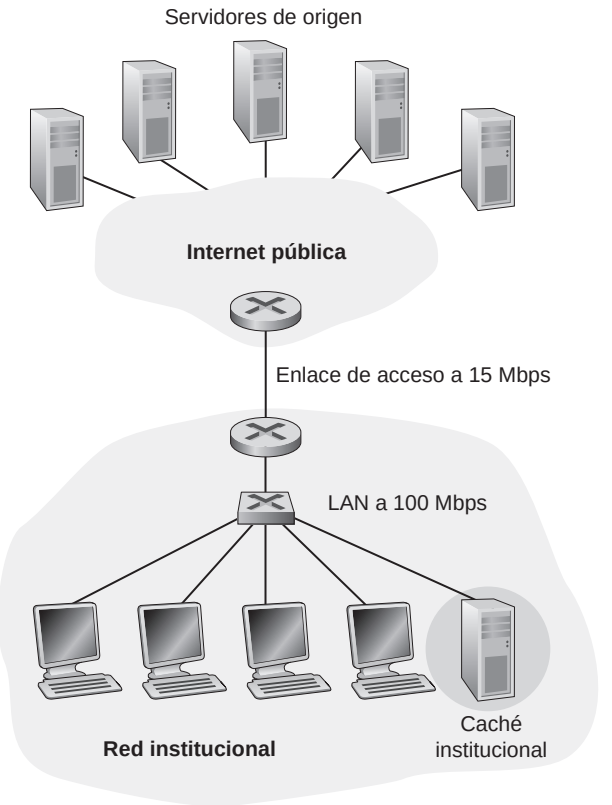



**Figura 2.13** ♦ Adición de una caché a una red institucional.

### GET condicional

Aunque el almacenamiento en caché puede reducir los tiempos de respuesta percibidos por el usuario, introduce un nuevo problema: la copia de un objeto que reside en la caché puede estar desactualizada. En otras palabras, el objeto almacenado en el servidor web puede haber sido modificado desde que la copia fue almacenada en la caché del cliente. Afortunadamente, HTTP dispone de un mecanismo que permite a la caché verificar que sus objetos están actualizados. Este mecanismo se denomina **GET condicional**. Un mensaje de solicitud HTTP se denomina también mensaje GET condicional si (1) el mensaje de solicitud utiliza el método GET y (2) el mensaje de solicitud incluye una línea de cabecera `If- Modified-Since:`.

Para ilustrar cómo opera el GET condicional, veamos un ejemplo. En primer lugar, una caché proxy envía un mensaje de solicitud a un servidor web en nombre de un navegador que realiza una solicitud:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

En segundo lugar, el servidor web envía a la caché un mensaje de respuesta con el objeto solicitado:

```
HTTP/1.1 200 OK
Date: Sat, 3 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 9 Sep 2015 09:23:24
Content-Type: image/gif

(datos datos datos datos datos ...)
```

La caché reenvía el objeto al navegador que lo ha solicitado pero también lo almacena localmente. Y lo que es más importante, la caché también almacena la fecha de la última modificación junto con el objeto. En tercer lugar, una semana después, otro navegador solicita el mismo objeto a través de la caché y el objeto todavía se encuentra almacenado allí. Puesto que este objeto puede haber sido modificado en el servidor web en el transcurso de la semana, la caché realiza una comprobación de actualización ejecutando un GET condicional. Específicamente, la caché envía:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 9 Sep 2015 09:23:24
```

Observe que el valor de la línea de cabecera `If-modified-since:` es exactamente igual al valor de la línea de cabecera `Last-Modified:` que fue enviada por el servidor hace una semana. Este GET condicional le pide al servidor que envíe el objeto sólo si éste ha sido modificado después de la última fecha especificada. Suponga que el objeto no ha sido modificado desde el 9 de septiembre de 2015 a las 09:23:24. Por último y en cuarto lugar, el servidor web envía un mensaje de respuesta a la caché:

```
HTTP/1.1 304 Not Modified
Date: Sat, 10 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)

(cuerpo de entidad vacío)
```

Vemos que en respuesta al GET condicional el servidor web envía un mensaje de respuesta, pero no incluye el objeto solicitado en el mismo. Si incluyera el objeto solicitado sólo conseguiría desperdiciar ancho de banda e incrementar el tiempo de respuesta percibido por el usuario, especialmente si el tamaño del objeto es grande. Observe que este último mensaje de respuesta muestra en

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.

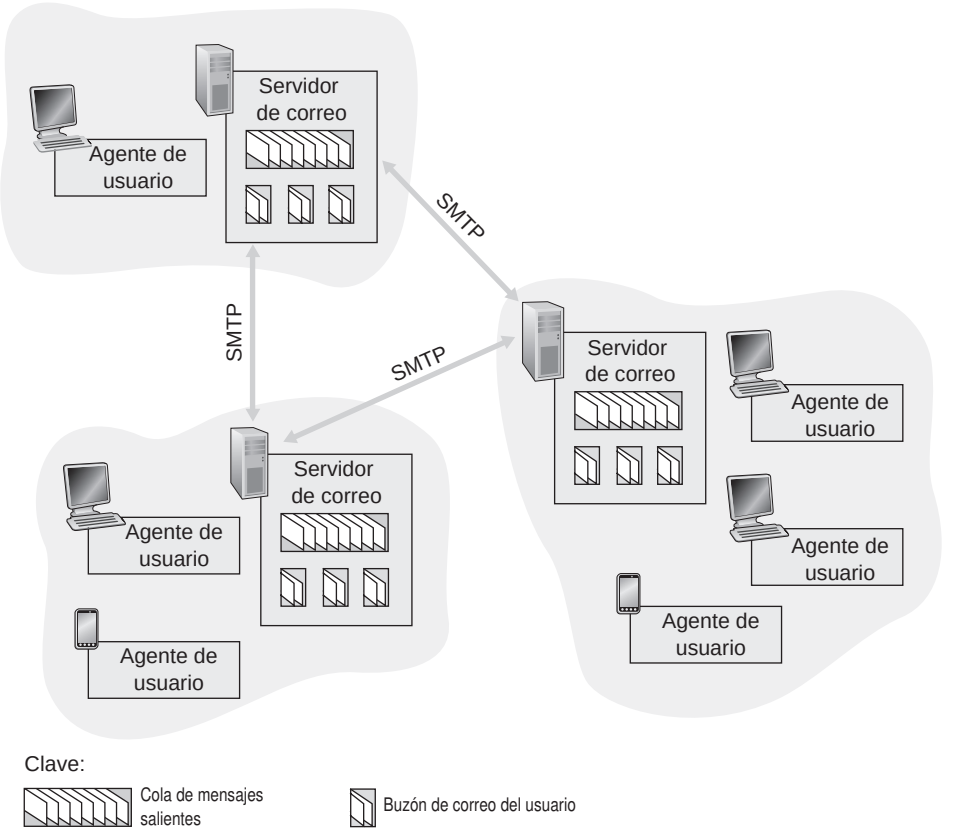



**Figura 2.14** ♦ Esquema general del sistema de correo electrónico de Internet.

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

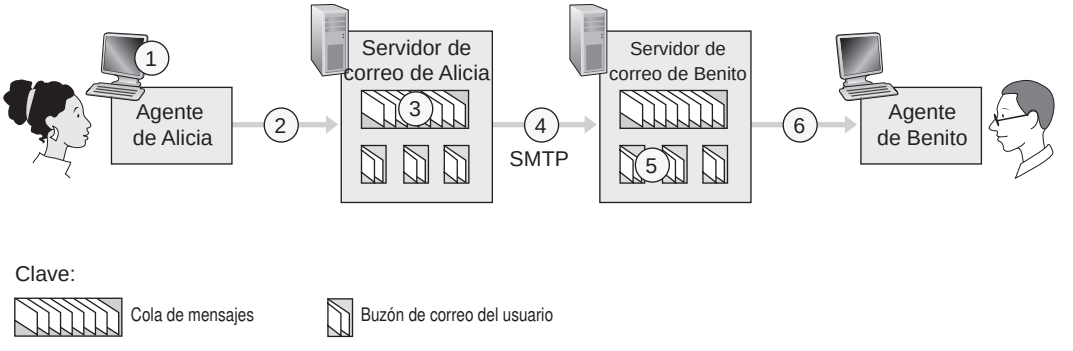



**Figura 2.15** ♦ Alicia envía un mensaje a Benito.

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 accesoria 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. Toda 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 cabecera opcionales. 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 palabra 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 impracticable 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

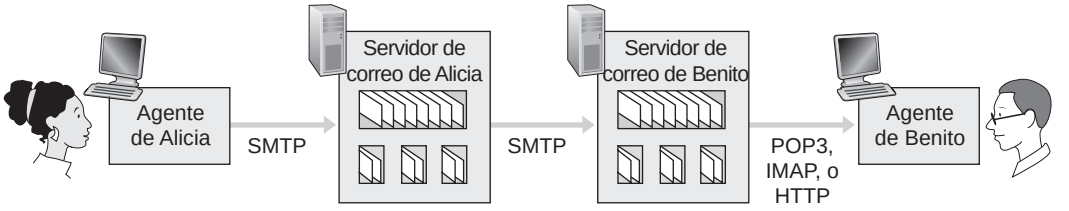



**Figura 2.16** ♦ Protocolos de correo electrónico y entidades que los utilizan.

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 <nombreusuario>` 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`, `retr` 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, pasarlos a carpetas y realizar búsquedas de mensajes (por nombre del emisor o asunto). Pero este paradigma, es decir, las carpetas y los mensajes guardos 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 cumplan 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`, `www.google.com`, `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`, 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, quizá 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*).

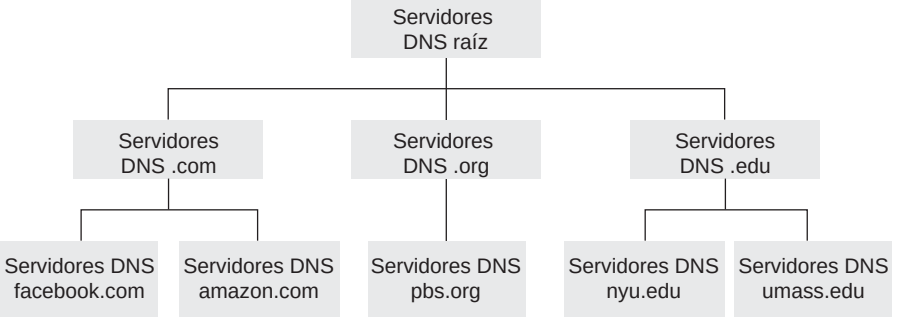



**Figura 2.17** ♦ Parte de la jerarquía de los servidores DNS.


**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 envíado 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.

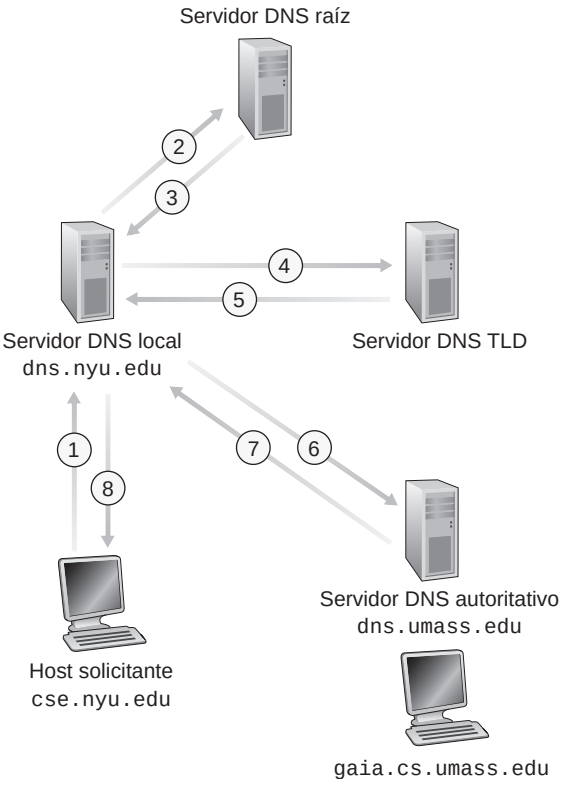



**Figura 2.19** ♦ Interacción de los distintos servidores DNS.

En el ejemplo anterior suponíamos 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 envíado 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.

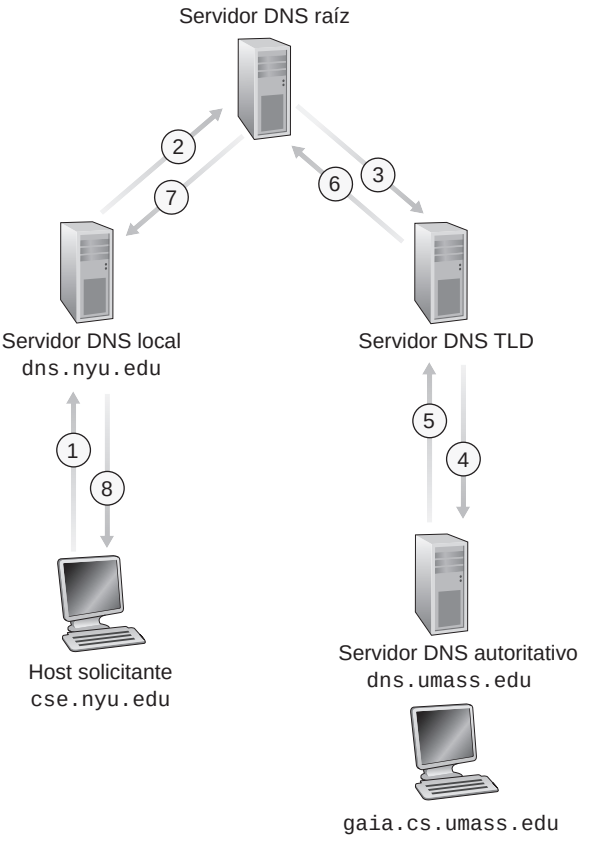



**Figura 2.20** ♦ Consultas recursivas en DNS.

### 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 contendría 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/respuesta 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

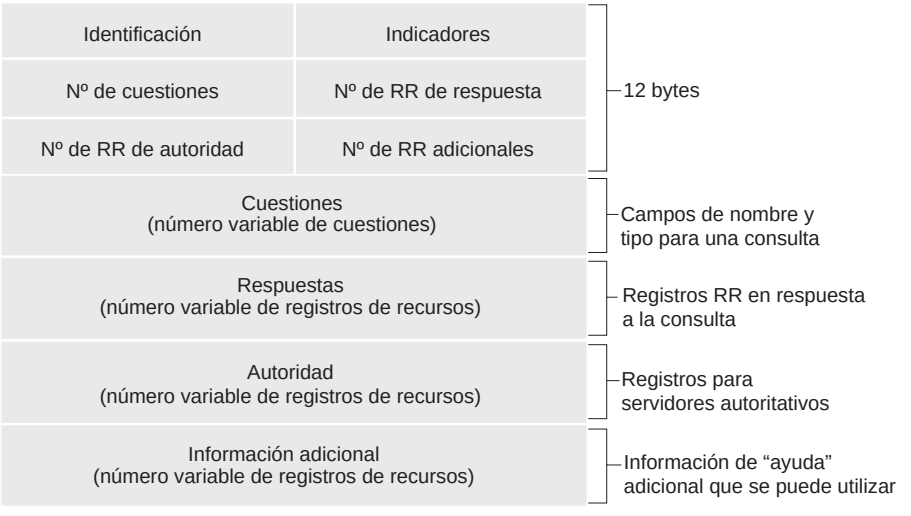



**Figura 2.21** ♦ Formato de los mensajes DNS.

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`. 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`. 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 cualesquiera 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:

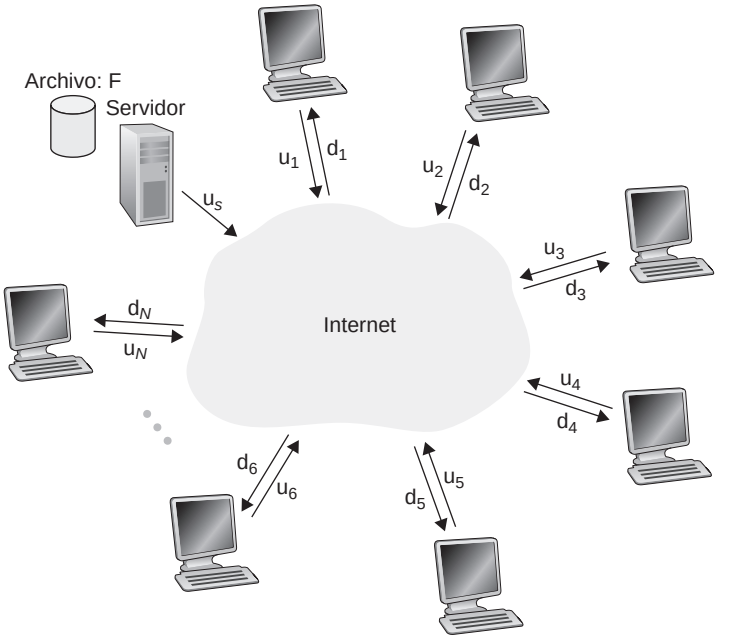



**Figura 2.22** ♦ Problema de distribución de un archivo.

- 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 $d_{mín}$ la velocidad de descarga del par cuya velocidad de descarga sea menor; es decir, $d_{mín} = \text{mín}\{d_1,d_p,...,d_N\}$. El par con la menor velocidad de descarga no puede obtener los $F$ bits del archivo en menos de $F/d_{mín}$ segundos. Por tanto, el tiempo mínimo de distribución es, al menos igual a $F/d_{mín}$.

Teniendo en cuenta estas dos observaciones, se obtiene:

$$D_{cs} \geq \text{máx}\left\{\frac{NF}{u_s}, \frac{F}{d_{mín}}\right\}$$

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,

$$D_{cs} = \text{máx}\left\{\frac{NF}{u_s}, \frac{F}{d_{mín}}\right\} \tag{2.1}$$

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 $F/d_{mín}$ segundos. Por tanto, el tiempo mínimo de distribución es al menos igual a $F/d_{mín}$.
- 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, $u_{total} = u_s + u_1 + \ldots + u_N$. 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 $NF/(u_s + u_1 + \ldots + u_N)$.

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

$$D_{\text{P2P}} \geq \text{máx}\left\{\frac{F}{u_s}, \frac{F}{d_{mín}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\} \tag{2.2}$$

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,

$$D_{\text{P2P}} = \text{máx}\left\{\frac{F}{u_s}, \frac{F}{d_{mín}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\} \tag{2.3}$$

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 $d_{mín} \geq u_s$. 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.

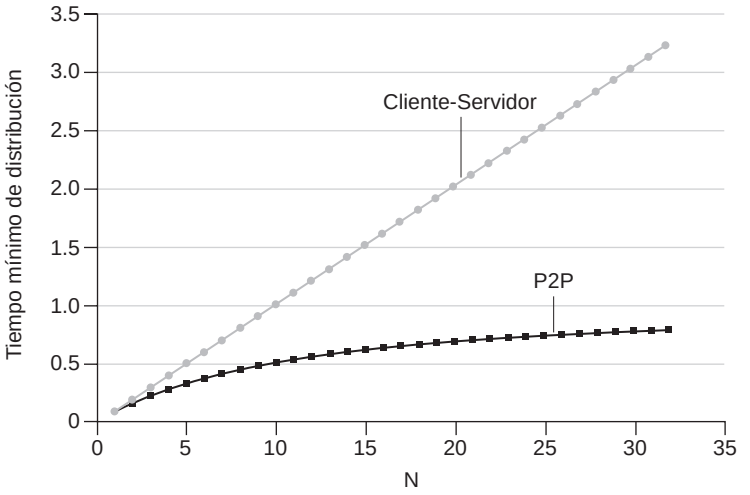



**Figura 2.23** ♦ Tiempo de distribución para las arquitecturas P2P y cliente-servidor.

### 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.

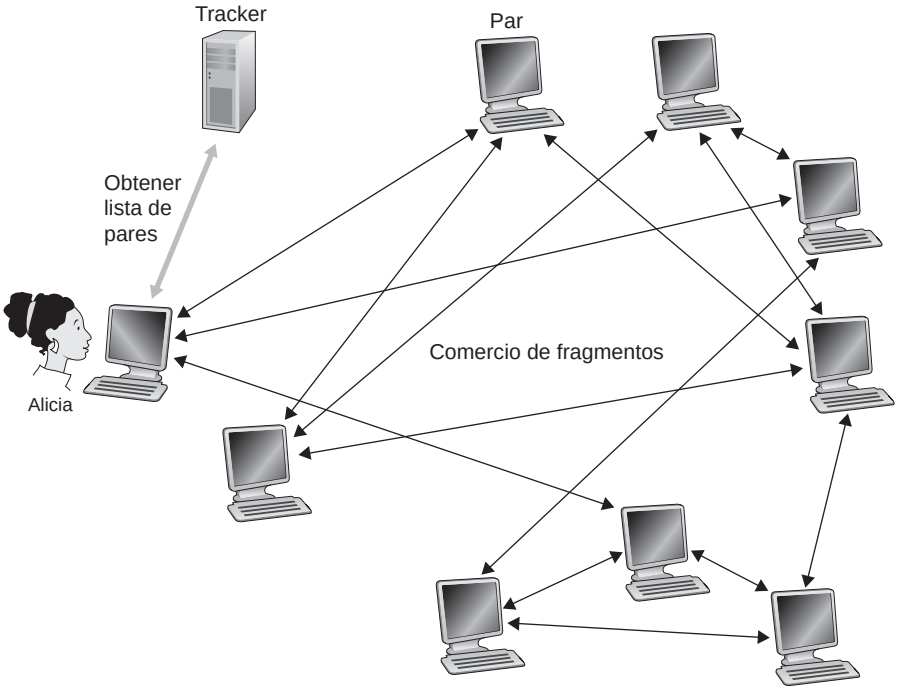



**Figura 2.24** ♦ Distribución de archivos con BitTorrent.

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 vídeo

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, quizá 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 puede 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. E 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, quizá 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 atraviesen 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 actúe 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 unívoco 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.

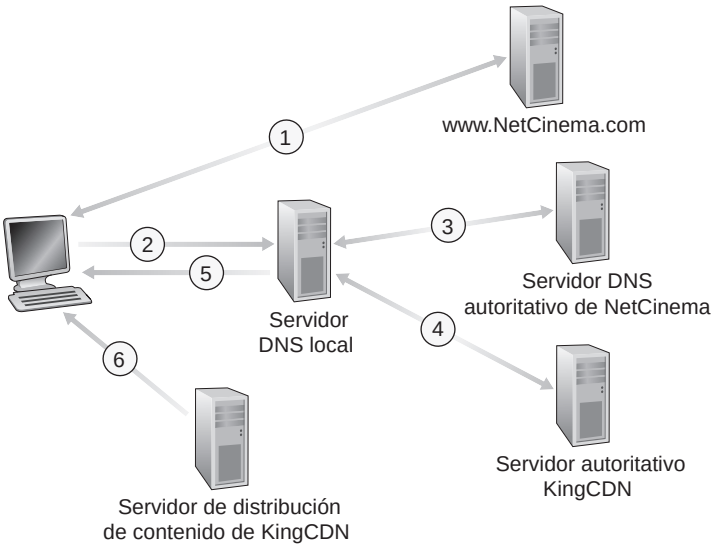



**Figura 2.25** ♦ DNS redirige una solicitud de usuario hacia un servidor CDN.

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 inicios 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.

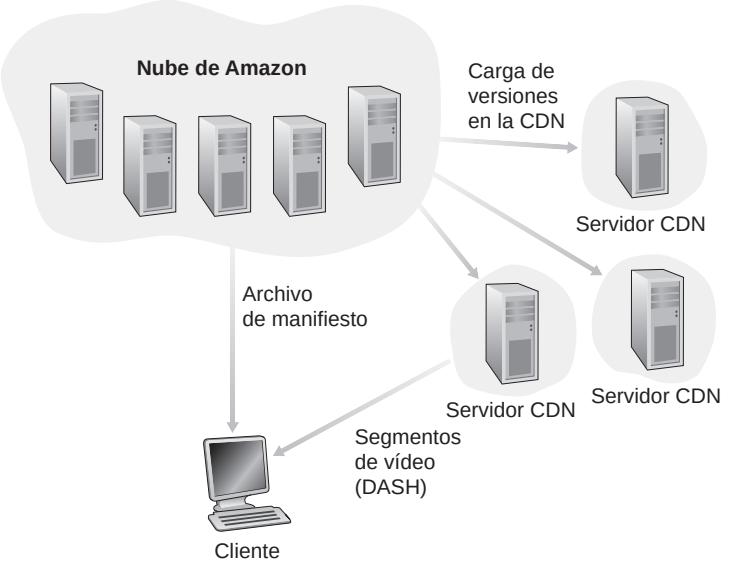



**Figura 2.26** ♦ Plataforma de flujos de vídeo de Netflix.

- ***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 interaccionan 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 cliente 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 sea 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 implemneta 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 tiene 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

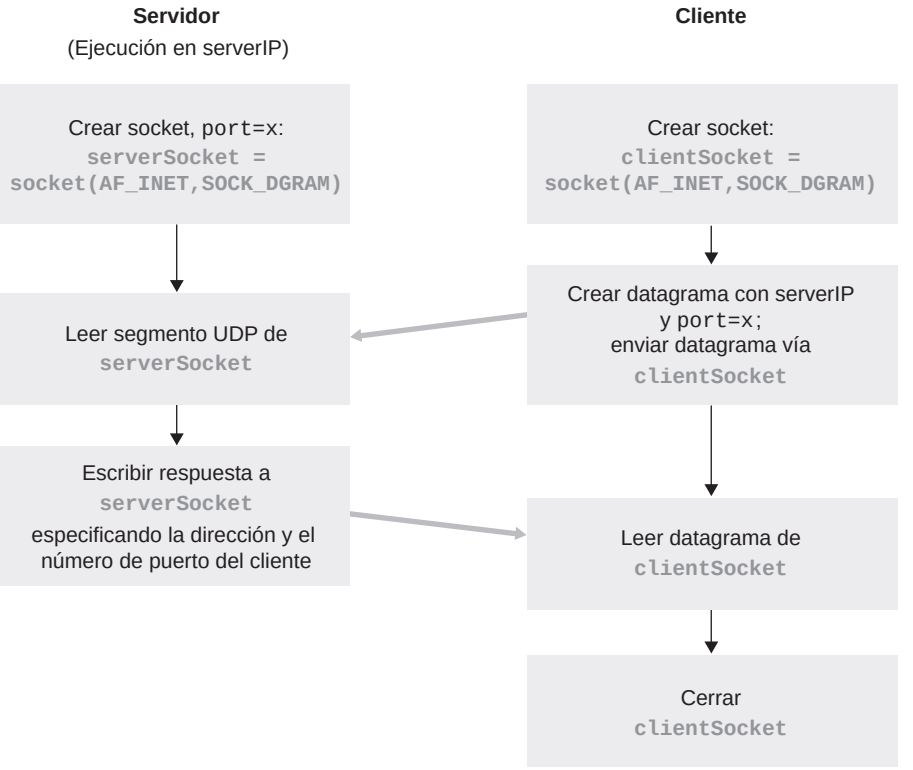



**Figura 2.27** ♦ Aplicación cliente-servidor usando UDP.

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 UDPCliente. 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 UDPCliente 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

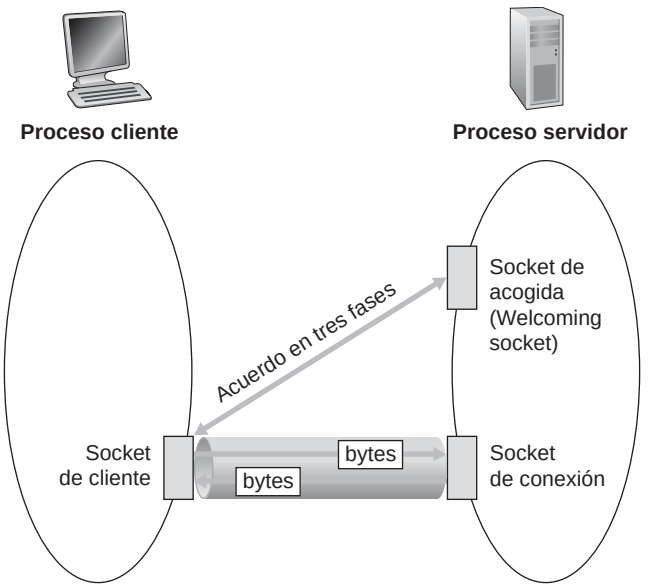



**Figura 2.28** ♦ El proceso TCPServidor tiene dos sockets.

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))
```

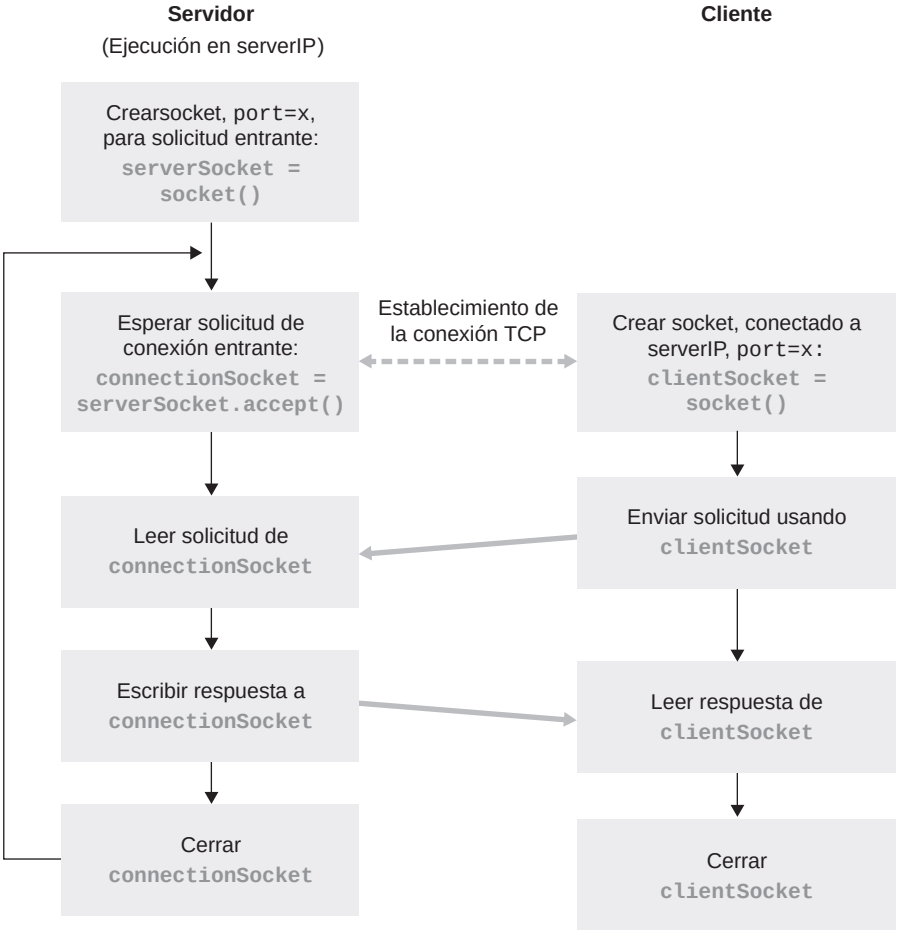



**Figura 2.29** ♦ La aplicación cliente-servidor utilizando TCP.

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

Fijémonos 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 UDPClient:

```
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 UDPClient, 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 UDPClient:

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

La línea anterior envia 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 suminis-

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. Nómbrelas 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?

a. 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.

b. Dos páginas web diferentes (por ejemplo, `www.mit.edu/research.html` y `www.mit.edu/students.html`) se pueden enviar a través de la misma conexión persistente.

c. 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.

d. La línea de cabecera `Date:` del mensaje de respuesta HTTP indica cuándo el objeto fue modificado por última vez.

e. 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) Gec
ko/20040804 Netscape/7.2 (ax) <cr><lf>Accept:ex
t/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 Dec2005 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 4.0transitional//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 $RTT_1$, . . ., $RTT_n$. 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

a. HTTP no persistente sin conexiones TCP en paralelo?

b. HTTP no persistente con el navegador configurado para 5 conexiones paralelo?

c. 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.

a. Calcule el tiempo medio de respuesta total.

b. 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.

a. ¿Le ayudan a Benito las conexiones en paralelo a obtener las páginas más rápidamente? ¿Por qué?

b. 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.exchangeddd.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*?

b. Utilice varias bases de datos whois de Internet para obtener los nombres de dos servidores DNS. Indique qué bases de datos whois ha utilizado.

c. 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.

d. 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?

e. Utilice la base de datos whois ARIN para determinar el rango de direcciones IP utilizado en su universidad.

f. 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.

g. 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.

a. 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.

b. 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 $u_s$.

a. Suponga que $u_s/N \leq d_{mín}$. Especifique un esquema de distribución que tenga un tiempo de distribución de $NF/u_s$.

b. Suponga que $u_s/N \geq d_{mín}$. Especifique un esquema de distribución que tenga un tiempo de distribución de $F/d_{mín}$.

c. Demuestre que, en general, el tiempo mínimo de distribución está dado por máx$\{NF/u_s, F/d_{mín}\}$.

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 $d_{mín}$ es muy grande, por lo que el ancho de banda de descarga de los pares no es nunca un cuello de botella.

a. Suponga que $u_s \leq (u_s + u_1 + ... + u_N)/N$. Especifique un esquema de distribución que tenga un tiempo de distribución de $F/u_s$.

b. Suponga que $u_s \geq (u_s + u_1 + ... + u_N)/N$. Especifique un esquema de distribución que tenga un tiempo de distribución de $NF/(u_s + u_1 + ... + u_N)$.

c. Demuestre que, en general, el tiempo mínimo de distribución está dado por máx$\{F/u_s, NF/(u_s + u_1 + ... + u_N)\}$.

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").

a. Benito afirma que puede recibir una copia completa del archivo compartido por el conjunto de usuarios. ¿Es correcto lo que dice Benito? ¿Por qué?

b. 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.

a. 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)?

b. 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.

a. Suponga que ejecuta TCPCliente antes que TCPServidor. ¿Qué ocurre? ¿Por qué?

b. Suponga que ejecuta UDPCliente antes que UDPServidor. ¿Qué ocurre? ¿Por qué?

c. ¿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.

### 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 incluye 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.

## 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.

**UNA ENTREVISTA CON...**

# Marc Andreessen

Marc Andreessen es el co-creador de Mosaic, el navegador que popularizó la World Wide Web en 1993. Mosaic tenía una interfaz limpia, fácilmente comprensible, y fue el primer navegador en mostrar imágenes intercaladas en el texto. En 1994, Marc Andreessen y Jim Clark fundaron Netscape, cuyo navegador fue con mucho el más popular a mediados de la década de 1990. Netscape desarrolló también el protocolo SSL (Secure Sockets Layer, capa de conectores seguros) y muchos productos de servidor para Internet, incluyendo servidores de correo y servidores web basados en SSL. Ahora es co-fundador y socio de la empresa de capital riesgo Andreessen Horowitz, supervisando la cartera de inversiones en empresas entre las que se incluyen Facebook, Foursquare, Groupon, Jawbone, Twitter y Zynga. Es miembro de numerosos consejos de administración, incluyendo los de Bump, eBay, Glam Media, Facebook y Hewlett-Packard. Es licenciado en Ciencias de la Computación por la Universidad de Illinois en Urbana-Champaign.

**¿Cómo llegó a interesarse por el mundo de la computación? ¿Siempre supo que quería trabajar en tecnologías de la información?**

Las revoluciones de los videojuegos y de las computadoras personales se produjeron siendo yo niño - las computadoras personales representaban la nueva frontera de la tecnología a finales de los 70 y principios de los 80. Y no eran solamente Apple y el IBM PC, sino también cientos de nuevas empresas, como Commodore y Atari. Aprendí yo solo a programar, con un libro titulado “Instant Freeze-Dried BASIC” a los 10 años y conseguí mi primera computadora (una computadora TRS-80 a color, ¡fíjate!) a los 12.

**Describa uno o dos de los proyectos más interesantes en los que haya trabajado durante su carrera. ¿Cuáles fueron los principales desafíos?**

Sin ninguna duda, el proyecto más interesante fue el navegador web Mosaic original en 1992-1993; y el principal desafío fue conseguir que alguien se lo tomara en serio por aquella época. En aquel entonces, todo el mundo creía que el futuro interactivo sería proporcionado en forma de “televisión interactiva” por grandes empresas, no que iba a tomar la forma de Internet gracias a empresas de nueva creación.

**¿Qué es lo que le entusiasma más acerca del futuro de las redes y de Internet? ¿Cuáles son sus principales preocupaciones?**

Lo más excitante es la enorme frontera inexplorada de aplicaciones y servicios que los programadores y los emprendedores tienen en su mano explorar - Internet ha liberado la creatividad a un nivel que no creo que hayamos visto nunca antes. Mi principal preocupación es el principio de las consecuencias no deseadas: no siempre conocemos las implicaciones de lo que hacemos, como por ejemplo que Internet sea usada por los gobiernos para implementar un mayor nivel de vigilancia de los ciudadanos.

**¿Hay algo en concreto de lo que los estudiantes debieran ser conscientes a medida que avanza la tecnología web?**

La tasa de cambio: la lección más importante que hay que aprender es cómo aprender; cómo adaptarse de forma flexible a los cambios en tecnologías específicas y cómo mantener una mente abierta acerca de las nuevas oportunidades y posibilidades a medida que progresamos en nuestra carrera profesional.

**¿Qué personas le han inspirado profesionalmente?**

Vannevar Bush, Ted Nelson, Doug Engelbart, Nolan Bushnell, Bill Hewlett y Dave Packard, Ken Olsen, Steve Jobs, Steve Wozniak, Andy Grove, Grace Hopper, Hedy Lamarr, Alan Turing, Richard Stallman.

**¿Qué le recomendaría a los estudiantes que quieran orientar su carrera profesional hacia la computación y la tecnología de la información?**

Profundiza lo más que puedas en la comprensión de cómo se crea la tecnología, y luego complementa tus conocimientos aprendiendo cómo funciona una empresa.

**¿Puede la tecnología resolver los problemas del mundo?**

No, pero elevamos el nivel de vida de la gente gracias al crecimiento económico, y la mayor parte del crecimiento económico a lo largo de la Historia ha sido consecuencia de la tecnología. Así que eso es lo mejor que podemos hacer.